Before Task1:



Turning off address randomization using the above command so the the address stay the same on every execution

Text

Description automatically generated

Changing L to 90 as required by the assignement

Text

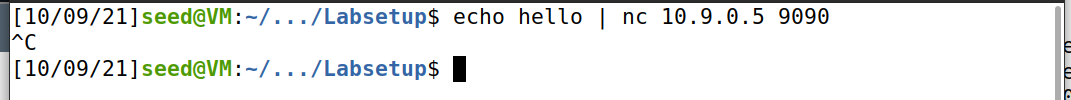
Description automatically generated

Using the make command to compile the vulnerable program. Here we get 2 warnings for printf(the vulnerable line of code as no format specifiers are specified for it) in myprintf function that exists in the code

Text

Description automatically generated

Setting up the server



Text

Description automatically generated

Testing the server using a simple echo command

Task1:

Text

Description automatically generated

Using the given build\_string.py and creating the badfile which is sent to the vulnerable program that exists in the serverText

Description automatically generated

We can see that the program does not print out returned properly which means that the format program has crashed

Task2a

Graphical user interface, text, email

Description automatically generated

We need 61%x format specifiers to print out the first four bytes of the input that we send to the server using the cat command



Executing the modified buildstring.py and sending the created badfile to the server

Text, letter

Description automatically generated

Here we can see that we get the output of the program and can see 41414141 at the 61st position as we have taken “AAAAAA” as out input string for that position

Task2.b

Graphical user interface, text

Description automatically generated

For this task we update the buildstring.py code by changing the number value to the address of the secret message which we get from the server terminal. And for the sting we give the address in the reverse order as mentioned In the pdf with “60%s” specifiers



Then we execute the build\_string.py program to generate a badfile that we send to the server using nc

Text

Description automatically generated

Whe the badfile is received by the server the format.c vulnerable program gets executed and we are able to print out the secret message that is at the location 0x080b4008

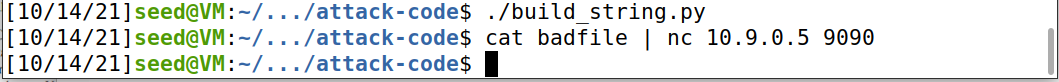
Task3:

3.a:

Graphical user interface, text

Description automatically generated

In this task we again modify the address of the number at line 20 to the target variable address and change the string “s” line according to the task as we need to change the address of the target variable we use a %n specifier at the 60th position



Executing the program to genereate badfile and send it to the server using nc at port 9090

Text, letter

Description automatically generated

Once we execute the command we can see the output on the server terminal where the target variable address is changed in the after value

3.b:

Graphical user interface, text, email

Description automatically generated

In this task we agin modify the python program that creates badfile by adding another variable D(line 19) where we give the address we want that is 0x5000 which is subtracted by all the already used %x specifiers that is 58\*8 and another “-8” as we use 2 locations above each 4 bits for number and string “AAAAAA”

And the we add this as a string to our main string “s” as shown above in the image



Executing the program to generate badfile and sending the badfile to the server

A picture containing text

Description automatically generated

Once we execute the program we can see the target variable’s address is changed to 0x00005000 as we expected

3.c:

Graphical user interface, text

Description automatically generated

In this task we modify the build string .py file bu adding another number variable who has the address but which is +2 of the target variable address and then we modify the content at 0:4 and create another at location 8:12 in lines 10 and 13 as shown in the image.

For the string we need to send we create 2 another variables d1 and d2 which are used on s using a half%n specifier “%hn” which uses only 2 bits of space for d1 we use the same technique as in previous task to get the address and subract 12 because another 4 bits is added in this program using content[8:12] and for d2 we just subtract 0xaabb from 0xccdd.

These both d1 and d2 variables are used in the variable “s” as shown in the image



Then we execute the program and generate badfile which is sent to the server using nc

Background pattern

Description automatically generated

The result is shown in the server terminal where we can see that the target variable address is changed to 0xaabbccdd as we wanted it to change.

Task4:

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

For the number2 and number1 address in the exploit.py file we add 4 and 6 respectively to the frame pointer address

And then for d1 and d2 we add 144 to the input buffer address so that the points jumps to the NOP’s when it returns from the mtprintf function and then executed our desired malicious code

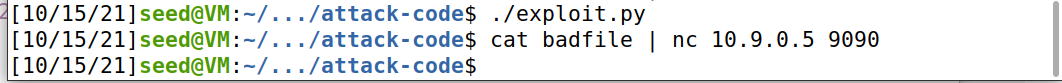
The malicious code is stored at location 3 that is 0xffffd524

Question1:

The memory address for location 2 is ebp value + 4 which is the return address (address of number 2 in the code) and the address of location 3 is 0xffffd524 which is used as d1 and d2(full address split into 2 parts of higher address and lower address) in the code

Question2:

We need to move 58 format specifiers to get to location 3 from location 1



Then we execute exploit.py and generate badfile which is then sent to the server using nc

Text

Description automatically generated with medium confidence

Here we can see the result that we are able to execute the shell command “ls -l” and get the server to display all its files in the current directory

Task6:

Graphical user interface, text, application

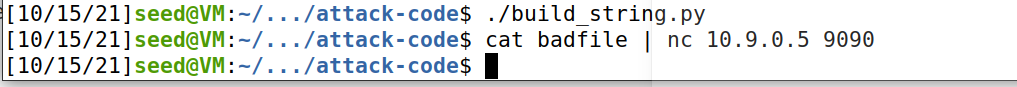
Description automatically generated

In the format.c program in server code file we edit the code a little bit by adding a format specifier in line 44 that is the printf command in myprintf function

Text, Word

Description automatically generated with medium confidence

Then we compile the program and see that we don’t get any warnings this time as the code has a format specifier already when compared to the first task where we get the warning and then we sent he compiled files to the server



Then again we generate a badfile in the attack-code file executing build\_sting.py and send it to the server using nc

Text, letter

Description automatically generated

On the server terminal we can see that the program executes without any vulnerability as it takes all the input as a string and prints it out as a string because a %s format specifier is added to the program