Merry Assessment

Rudranil Maity | CS647-Spring 2023

Executive Summary

In the assessment, the objective was to identify any buffer overflow vulnerabilities present in the executable file named "/home/merry/retAddr3". The task involved exploiting the vulnerabilities by crafting a buffer overflow attack to execute the code that was previously unreachable, which would print out the "merryflag.txt" text. The main focus of the assessment was to demonstrate the potential risks of buffer overflow attacks and to expose any vulnerabilities in the program.

Vulnerabilities Identified

During the merry assessment, a buffer overflow vulnerability was discovered in the program. This type of vulnerability occurs when the input to a program exceeds the buffer size, and the programmer did not include bounds for the input. Essentially, the vulnerability is caused by a failure to properly limit the amount of data that can be accepted by the program, which can allow malicious actors to exploit the program and execute unauthorized code.

Recommendations

The information that the file examination process gave; it can be said that the developer should validate the number of user inputs for the code. Regarding this, using of strncpy() could be mentioned. Using of strcpy() is a risky because it does not check the length of the string being copied and can result in a buffer overflow if the destination buffer is not large enough to hold the entire source string. To avoid this vulnerability, programmers should always ensure that the size of the destination buffer is large enough to hold the entire source string plus one additional byte for the null terminator. Additionally, it is recommended to use strncpy() instead of strcpy() when possible to reduce the risk of buffer overflow attacks. Additionally, programmers should always ensure that the destination buffer size is large enough to hold the entire source string plus one additional byte for the null terminator.

Assumptions

The main assumption which was made while examining the retAddr3 file was regarding the input value to the memory stack which overflowed the buffer to exploit it and print the flag. The length of the input string was a major factor towards performing the buffer overflow because the amount of padding which was needed to control the value in EIP, could’ve only be found through assumptions.

Steps to Reproduce the Attack

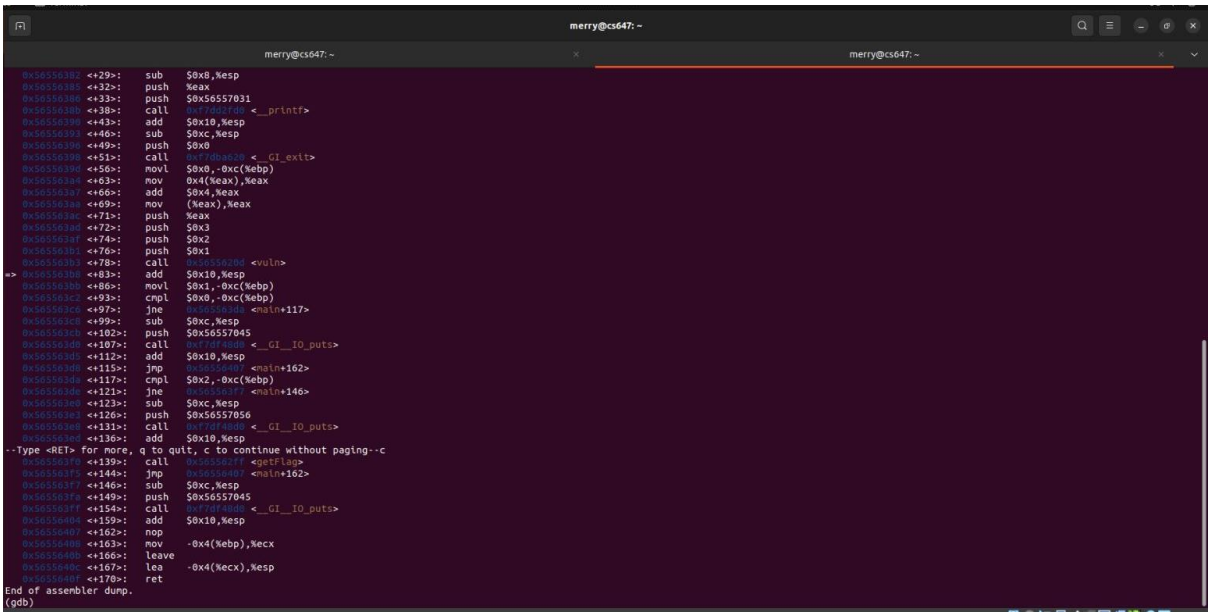
At first the ASLR (Address Space Layout Randomization); which is a security feature implemented in modern operating systems to protect against buffer overflow and other types of attacks; was disabled through the command “`$toggleASLR`”.

Screenshot 1 (Disabling ASLR)

Then the padding value was assumed for the input string and the command "`$./retAddr3 $(perl -e 'print "A"x512')`" was ran. The same command was rerun with different values for input string to get the point through which the final padding value was determined as 265. (Screenshot 2)

Screenshot 2

After that the retAddr3 file was opened in gdb through the command "`$gdb retAddr3`" and then it was run through the command "`(gdb)run retAddr3`". Then the command "`(gdb)disassemble main`" was used to display the assembly code (Screenshot 3) of the file retAddr3, which was being debugged.



Screenshot 3

As seen from Screenshot 3, the command at line <+123> is the one which was giving the output of "`You lose`", as seen on Screenshot 2. So this line needed to be skipped through buffer overflow to get the result as "`You Win`", containing the required text from "merryflag". The operation was ran through the command "`$./retAddr3 $(perl -e 'print "A"x269 . "\xe0\x63\x55\x56"')`" ; 4 more characters were added to the string to overflow the buffer. In the previous command, the address of the memory buffer which was required to be overflown was converted from hexadecimal to little endian format and was being used. The result is shown in Screenshot 4 in the "Findings" section.

A stack frame diagram for buffer size 265 is given below.

```
      High Memory Addresses
      ---------------------
                    |  buffer[260]  |
EBP+12-->           |  buffer[259]  |
                    |     ...       |
                    |  buffer[5]    |
EBP-4 -->           |  buffer[4]    |
EBP   -->           |  buffer[3]    |
                    |  buffer[2]    |
                    |  buffer[1]    |
                    |  buffer[0]    |
                    |---------------|
EBP-16-->           |       y       |
EBP-20-->           |       x       |
                    |---------------|
                    |  input[3]     |
                    |     ...       |
                    |  input[0]     |
                    |---------------|
                    |     EIP       |
                    |---------------|
                    |     EBP       |
                    |---------------|
                    |  saved EIP    |
                    |---------------|
                    |  saved EBP    |
                    |---------------|
                    |   buffer      |
                    |  (unused)     |
                    |---------------|
                    |  padding      |
                    |  (unused)     |
                    |---------------|
      Low Memory Addresses
```

Findings

The findings from the assessment and vulnerabilities, including the contents of “merryflag.txt” and the step where the program was exploited are given below (Screenshot 4):

Screenshot 4

The content of from the file was:

da46d83995802939d9c3942325d788a7496ac8750c416a5c7f8a6deaac7624

57e7c76daec8d5b6cd3cd4c40866C19ae821c9d71cf1ae5dfe8eb4e33d88d94d