Stack Overflow

CSC 472 - 01

Tyler Prehl

10/4/2021

Introduction

The purpose of this lab is to thoroughly understand and successfully execute a stack overflow (also known as “Return Hijack”) attack. This lab was performed so that the experimenter would gain a stronger understanding of the stack overflow attack – how it works and how to implement it – to help prepare for more complex attacks that utilize stack overflow methods, such as running a small program through the use of the stack overflow attack method.

Analysis and Results

To begin the lab, the experimenter downloaded three files – lab2.c, six.py, and exploit.py. lab2.c contains the victim C program that will be hacked in the lab. In it is the following (except YOURNAME has been replaced with “Tyler Prehl”, and the char array was changed to be of size 24):

Text

Description automatically generated

You can see that while there is a main() that calls the return\_input() method, there is also a hacked() method that is never called by main. A successful stack overflow attack will allow the experimenter to use the call of return\_input() to *actually* call hacked(). To do this, the experimenter first compiled the lab2.c program using the following command:



This command turns off the default protection settings that prevent basic stack overflow attacks like the one the experimenter practiced. Then, with lab2 compiled as an executable file, the experimenter ran lab2 using the gdb debugger, and disassembled the hacked() method to identify the location in memory where hacked() begins. Although it is difficult to see, the memory address is 0x08049172.

A screenshot of a computer

Description automatically generated with medium confidence

With this information, the experimenter then created a pattern of 100 characters (saved into a file named “sequence”) to input into the program to identify after how many input characters the EIP register is overwritten. In our case, the EIP register was overridden after 36 characters (designated as “Found at offset 36 (little-endian search):

Text

Description automatically generated

With this information, the experimenter now has all necessary information to successfully execute a stack overflow attack. To create a string input that includes the 36 characters (letters in groups of 5 in this case) and the memory address of hacked() as 4 bytes (which will fit into the EIP register) instead of as a string of length 16, the experimenter used an echo call. This string was then moved into a file named “input” using the ‘string’ > input command, and when lab2 is run with “input” as the input, a successful stack overflow hack is achieved:

Text

Description automatically generated

Now, all that’s left is for the experimenter to create a python script for this attack. The template python file exploit.py is this:

Text, letter

Description automatically generated

To create a successful attack file, the experimenter created a return address variable (converted to bytecode) which gets added to the payload variable (which is 36 A’s in length, also converted to bytecode):

Text

Description automatically generated

Then the experimenter commented out line 16:



And ran the final product:

Text

Description automatically generated

Resulting in a successful attack:

Text

Description automatically generated

Discussion and Conclusion

Through this lab, the experimenter practiced using the gdb debugger to gain specific assembly-code information about programs in order to take advantage of the stack through a buffer overflow attack. The experimenter successfully attacked the victim file and ran the code desired by the hacker. This will lead to future attacks that implement a stack overflow aspect - overwriting the called method’s stack frame with random information and overwriting the EIP register with a different, desired return address. More specifically, the experimenter will soon learn to use stack overflow attack methods to run an entire program of the experimenter’s choice within the overflowed stack frame.