Project 3

Buffer Overflow

Due December 11, 2019 at 11:59pm

In class we explored a simple buffer overflow that overwrote a local variable and allowed us access to a login system. Buffer overflow can also be used to take control of an entire system by overwriting return addresses of functions on the stack. Changing the return address of a function allows the malicious actor to return to a function of their choice.

In this project you will follow a tutorial on how to use buffer overflow to overwrite the return address of a function to take control of a program. You will then add a new function to the program and have the flow of control continue to this function. The new program should exit cleanly using the exit function in the C library. The buffer overflow and the take-over of the program will be done through well-constructed input to the program called the payload.

The goals of this project:

- Understanding the virtual address space of a program
- Understanding how stack memory is used
- Understanding the concepts of buffer overflow
- Exploiting a stack buffer overflow vulnerability

Project Setup

For this project, you will need to import a new virtual machine with a 32 bit Ubuntu distribution. The .ova file can be downloaded here:
https://drive.google.com/open?id=160Pck0JP68_PDanLn8RZBaRPBzwHYSjP

If you do not have room on your computer to install a new virtual machine, you may work with a partner on the exploit, but the write up must be done alone.

Buffer Overflow Exploit Instructions

First - Read and follow the buffer overflow tutorial found in the following blog post by Dhaval Kapil. You should go through the tutorial step by step and answer the 2 questions and complete the 2 tasks that follow.

https://dhavalkapil.com/blogs/Buffer-Overflow-Exploit/

After completing the blog post and tutorial continue here…

Question 1

If you are here, you should have completed the tutorial on buffer overflow.
What is the payload to overflow the buffer to enter the secretFunction? Note: your answer will look like the author's, but the memory address will be specific to your machine. It is ok if the memory address is the same as the authors.

Question 2

Memory Architecture. (Diagram(s) would be helpful, but are not required)

Describe the virtual address space. What segments are included in the space?

Describe stack in the address space. Where in memory is the stack located?

Which direction, relative to overall memory, does a stack consume memory when it grows?

Explain how program control flow is implemented using the stack.

Create a diagram that illustrates the following:

What does the stack structure look like when data is pushed onto the stack and popped off the stack? Include everything from the function arguments to the local variables.

How is a buffer overflow and the overwriting of a function's return address achieved?

Note: you may use the blog post or other online resources, but please **cite** the sources. The diagrams you use can be copied from these online resources, but if they are, **explain the diagram thoroughly in your own words**.

Task 1

Continue the exploit - In the tutorial you created a program called vuln.c, which included a main function, an echo function and the secretFunction. By overwriting the return address of echo to return to secretFunction in the tutorial, you have now caused stack misalignment. You can continue to add addresses of functions and these functions will be executed. You are to add a function to vuln.c called anotherSecretFunction that looks like this:

```
void anotherSecretFunction()

{

    printf("Congratulations!\n");

    printf("You made it to the second secret function!\n");


}
```

Recompile vuln.c. Change the original payload to execute this function after executing the original secretFunction. The output of the program after entering text should look like this:

Congratulations!

```
You have entered in the secret function!
Congratulations!
You made it to the second secret function!
Segmentation fault (core dumped)
```

What is the new payload?

Task 2

Return to libc attack – In order to return to a function in a program, the malicious actor needs access to the original program to modify it. This is not always possible. However, one could have the program return to C library functions to do harm to a system. The most popular is to have the program open a shell. The hacker now has a shell to write malicious commands.

In this task, you will have the program execute the exit function in the C library to terminate the program.
The current buffer overflow exploit ends in a segmentation fault. This could alert a system administrator to check the system for foul play. Change the payload so that the exit function in the C library is called after anotherSecretFunction() is called, and the program terminates cleanly without a segmentation fault.
Your output should now look like this:

```
Congratulations!
You have entered in the secret function!
Congratulations!
You made it to the second secret function!
```

What is the new payload?

Submission Instructions:

You should answer the first 2 questions and put the payloads for tasks 1 and 2 in a document. Make sure you cite your sources in question 2. Your final document should include a payload for question 1, task 1 and task 2. And a concise answer to question 2.
Convert the document to pdf and submit to Blackboard.
In addition, submit vuln.c that includes the new anotherSecretFunction() function.

Grading

40 points – Questions 1 and 2
60 points – Payloads for Tasks 1 and 2

Late Policy

There will be no extensions given on the project except under extreme circumstances. A 3 point per day late penalty will be instituted.