CSCD 437
Lab 3
Buffer Overflows

LAB OVERVIEW

Buffer overflow vulnerabilities occur when a program writes more data to a buffer than it can hold. This oversight can allow attackers to overwrite memory, potentially leading to arbitrary code execution or crashing the program. By exploiting these vulnerabilities, students will learn the dangers of insecure code and the importance of validating input and managing memory correctly.

OBJECTIVE

The purpose of this laboratory exercise is to provide students with hands-on experience in understanding and exploiting buffer overflow vulnerabilities. Through this exercise, students will gain a deeper comprehension of the stack's operation and learn how improperly secured code can lead to data corruption, unexpected control transfers, and memory access violations. The lab involves executing three distinct buffer overflow attacks of escalating complexity, thereby reinforcing the concepts of secure coding practices and vulnerability exploitation.

Final Deliverable: Submit a PDF document named according to the convention: [Last Name][First Letter of First Name]Lab3.pdf (e.g., steinersLab3.pdf). Your submission must include:

- Screen captures demonstrating your success in each task. Your screen capture must include the command line with your IP address.
- **Detailed explanations** of how you performed each buffer overflow attack, including any tools or techniques used.
- The flag associated with that buffer overflow attack

SETUP

- In your AWS VM, a lab3 folder was created.
    - If you don't have a lab3 folder send me an email immediately. Waiting till Monday or Tuesday to email me will not get you an extension. You need to be working on this lab starting Friday.
- Inside the lab3 folder is five folders named Task1 through Task5
- Each directory contains the source C file and an exploit.py file that will handle the network connections for you. Your job is to modify the *sendlineafter()* 2nd parameter which is the input that is sent to the executable. You will need to craft an input that creates a buffer overflow, you will get a flag back from the server with the identifier EWU{theFlag}

Here are the lines from exploit.py
# Send string to overflow buffer
io.sendlineafter(b':', b'A')

You will need to add the appropriate number characters inside b'A'

Then execute python3 exploit.py

NOTE: ASLR has already been turned off. You don't need to worry about ASLR.
NOTE: YOU WILL NOT CHANGE THE C CODE IN ANY FASHION. CHANGING THE CODE RESULTS IN A 0 FOR THAT TASK

- Compile Task 1 through Task 4 with the command
  - gcc *.c -o output -fno-stack-protector -z execstack -no-pie -m32

- Compile Task 5 with the command
  - gcc *.c -o output -fno-stack-protector -z execstack -no-pie

Task 1
Buffer Overflow I

Overview

This program prompts the user for a string input and then performs a conditional check based on a predefined variable's value. It demonstrates basic buffer handling and conditional logic in C programming. Look at the code and modify the b'A' in exploit.py to get the flag.

Task 2
Buffer Overflow II

Overview

This program, similar in structure to the first, focuses on user input and conditional checks against a specific variable's value. It introduces a slightly different scenario for the conditional check.

Hints for Success:

- Utilize GDB (GNU Debugger) to inspect the program's execution and memory states.
- Consider the significance of endianness in how data is stored and processed.
- Ensure proper input encoding to effectively manipulate the program's execution flow.

- *printf "AABB\xef\xbe\xad\xde" | ./dead* demonstrates how to pass in bytes as input.

Task 3
Buffer Overflow III

Overview

This program introduces a function that prompts for a name and checks this input against a validation condition. The main focus is on understanding function calls, user input handling, and conditional logic to control access to certain program functionalities.

Hints for Success:

- The EIP (Extended Instruction Pointer) register plays a crucial role in controlling the flow of execution in a program. Why is it important to know about the EIP when dealing with buffer overflows?
- Every function in a program is loaded into memory at a specific address. How can you determine the memory address of a particular function, and why might this information be crucial in the context of buffer overflow exploitation?
- The function *cyclic()* within the PwnTools library is helpful

Task 4
Buffer Overflow IV 32 Bit

Overview

This program is crafted to illustrate advanced buffer overflow techniques, specifically focusing on overwriting function parameters to gain unauthorized access. It challenges users to exploit a buffer overflow vulnerability to manipulate the program's control flow and pass specific

Hints for Success:

- Function parameters are typically stored on the stack in most calling conventions, making them susceptible to being overwritten if proper input validation is not enforced, especially in buffer overflow scenarios.
- You will need a debugging tool.

Task 5
Buffer Overflow V (Difficult Extra Credit) 64 Bit

Overview

This program is crafted to illustrate advanced buffer overflow techniques, specifically focusing on overwriting function parameters to gain unauthorized access. It challenges users to exploit a buffer overflow vulnerability to manipulate the program's control flow and pass specific

Hints for Success:

- Function parameters are typically stored in a specific area of memory reserved for temporary data during function execution, which varies depending on whether the architecture uses a stack-based or register-based calling convention.
- A ROP gadget is a sequence of instructions ending in a return statement, found within existing code. By carefully chaining these gadgets, you can perform arbitrary operations without injecting new code, effectively navigating security restrictions.
- You will need a debugging tool.

TO TURN IN

A single PDF with:

- Appropriate screen captures (Screenshot must include username and host information).
- Detailed explanations of how you performed each buffer overflow attack, including any tools or techniques used.
- The flag associated with that buffer overflow attack,

You will submit a PDF file named your last name first letter of your first name Lab3.pdf. (Example steinersLab3.pdf)