CSC 583 Fall 2024 Lab 2

Dr. Si Chen

October 1, 2024

Introduction

Lab Objectives:

- Understand the fundamentals of stack overflows and buffer overflows.
- Learn how to exploit stack buffer overflow vulnerabilities.
- Develop skills to craft and deploy shellcode in the context of buffer overflows.

Upon completing this lab, students should be capable of articulating:

1. The nature and risks associated with stack and buffer overflows.
2. The methodologies for exploiting stack and buffer overflow vulnerabilities.
3. Techniques for crafting and injecting shellcode to execute arbitrary code.
4. Proficiency in using debugging tools like GDB to analyze and exploit vulnerabilities.

Equipped with this understanding, students are expected to successfully execute attacks that exploit buffer overflow vulnerabilities in a given toy program.

Course Webpage: https://www.cs.wcupa.edu/schen/advsec24/

Lab Exercises

This lab comprises two main exercises:

1. **Return Hijack Attack**
2. **Shellcode Attack**

Both exercises utilize the same C source file (‘lab2.c‘). Ensure that you follow the instructions carefully for each part to successfully complete the lab.

1. Return Hijack Attack

The first exercise focuses on understanding and exploiting a stack buffer overflow vulnerability to hijack the return address of a function.

Objective: Overwrite the return address to redirect execution to the 'hacked()' function, resulting in the program outputting **"hacked by [Your First + Last Name]!"**.

Steps:

1. **Access the Badger CTF Platform:**
   - Log in to Badger CTF using your personal account credentials.
2. **Retrieve the Vulnerable C Code:**
   - Download the provided 'lab2.c' file from the course website.
   - Copy the source code to your working directory by executing:

     ```
     cp /workdir/advsec24/lab2/lab2.c /workdir/
     ```
3. **Modify the Source Code:**
   - Open 'lab2.c' using a command-line text editor (e.g., Vim, nano).
   - **Array Size Adjustment:** Set the size of the array ('char array[]') to **the last two digits of your student ID**. For example, if your student ID is '0861339', set the array size to '39'. **Note:** If the last two digits of your ID are less than 10 (e.g., '05'), use the last three digits of your student ID as the array size.
   - **Name Replacement:** Replace the string literal 'YOUNAME' in the 'hacked' function with your actual name (e.g., 'Si Chen').
4. **Compile the C Code:**

   ```
   arm-linux-gnueabi-gcc lab2.c -o lab2 -fno-stack-protector -z execstack -no-pie
   ```
5. **Execute the Program:**
   - Run the compiled program and input a long string of characters. You should observe that 'lab2' crashes due to a memory segmentation fault, indicating that the return address has been overwritten.
6. **Craft the Exploit Script:**
   - Write an exploit script in Python using the pwntools library.
   - The script should overwrite the return address with the address of the 'hacked()' function.

- Utilize GDB to find the correct addresses and to test/debug your exploit.
- Ensure that running your exploit script results in the program outputting **"hacked by [Your First + Last Name]!"**.

7. **Verification:**
   - Provide a screenshot demonstrating the successful exploitation of the program.

2. Shellcode Attack

The second exercise delves deeper into buffer overflow exploitation by crafting and deploying shellcode to execute arbitrary commands.

Objective: Inject shellcode into the vulnerable program to spawn a shell, resulting in the message **"Shell gained by [Your First + Last Name]!"** upon successful execution.

Steps:

1. **Access the Badger CTF Platform:**
   - Log in to Badger CTF using your personal account credentials.
2. **Retrieve the Vulnerable C Code:**
   - Copy the existing 'lab2.c' file to create a separate copy for this exercise by executing:

     $$cp/workdir/lab2.c/workdir/lab2\_shellcode.c$$
3. **Analyze the Vulnerability:**
   - Inspect 'lab2_shellcode.c' to identify the buffer overflow vulnerability.
4. **Develop Shellcode:**
   - Write shellcode that, when injected, will spawn a shell.
   - You can refer to existing shellcode examples to generate appropriate shellcode.
5. **Craft the Exploit Script:**
   - Write a Python script (e.g., 'shellcode_exploit.py') using the pwntools library.
   - The script should inject the crafted shellcode into the program's buffer and manipulate the execution flow to execute the shellcode.
   - Use GDB to determine the correct offsets and addresses required for a successful exploit.
6. **Execute the Exploit:**

- Run your exploit script against the vulnerable program.
- Verify that you obtain an interactive shell and that the message **"Shell gained by [Your First + Last Name]!"** is displayed.

7. **Documentation:**
   - Provide a screenshot demonstrating the successful execution of your exploit.
   - Explain the method used to determine the **Magic Number**—the length of the dummy-character string required to successfully overwrite the return address.
   - Include your shellcode and exploit script (‘shellcode_exploit.py‘) in your lab report.
8. **Ethical Considerations:**
   - Engage with this exercise ethically and responsibly. Ensure that all exploitation is conducted within the controlled environment provided by Badger CTF.

Deliverables

Submit a comprehensive lab report in PDF format that includes:

- The source code of your exploit scripts (‘exploit.py‘ for Return Hijack Attack and ‘shellcode_exploit.py‘ for Shellcode Attack).
- Screenshots demonstrating the successful execution of both exploits.
- Detailed explanations of your methodology, including how you identified vulnerabilities, determined memory offsets, crafted shellcode, and overcame any challenges encountered during the exploitation process.
- An explanation of how you determined the **Magic Number** for the buffer overflow.

Submission

- **Due Date:** Check the lab due date on the course website. Late submissions will not be accepted.
- **Submission Method:** Submit your lab report to D2L directly.
- **Academic Integrity: No copying or cheating is tolerated**. If your work is based on others', please provide clear attribution. Failure to do so will result in failing this course.