CMPSC 443: Introduction to Computer Security
Project 2: Buffer Overflow Attacks

Fall 2025

Due: October 20, 2025

1 Introduction

In this assignment, you will produce Buffer overflow attacks. First, you learn some attacks that invoke shared functions with arguments obtained from different places in memory (injected by you, or from environment variables, or from the hard-coded strings in the code, etc). Successful completion of this project heavily relies on the correct understanding of stacks, heaps, program memory layout, and a function's stack frame.

2 Prerequisite

Before attempting this project, it is advisable to review on the basics of stack frame, memory layout of program, use of GDB Debugger and big-endian vs little-endian byte ordering. To quickly brush through the basics of GDB debugging, I'd recommend watching this GDB Debugger Tutorial - `https://youtu.be/-pKu42v_opk?si=Ndzg97MCqZxLJa6k`.

3 Project Platform

This assignment will only work on VMs on x86 platforms.

- For Windows/Linux (x84) users, you can install VirtualBox (`https://www.virtualbox.org/`) to run the VM. Download this VM at `https://drive.google.com/file/d/1Gb9nJcUB5lf2D24tud9k53QwuCx98E69/view?usp=sharing`.
- For Mac (ARM) users, please install UTM (`https://mac.getutm.app/`). And emulate the x86-84 environment. Download this VM at `https://pennstateoffice365-my.sharepoint.com/:u:/g/personal/xzz5503_psu_edu/Ebin8n1U9ulOlmbdsecJMKMBWWo3NAoeBdaO43pQ7Gn9Mg`.

  The system lacks a graphical interface to ensure better performance. Emulation is an inefficient way to run an OS, and a graphical display would make it unacceptably slow.
- Username: `user443`
- Password: `443crypto`

Please specify the architecture you are using clearly in your report.

4 Exercise Tasks

Senario

Recall the secure cloud storage service called 'Crypto443 Company' in Project 1. However, it seems that the company has an old, but vital, server running a piece of proprietary software. You believe this system is riddled with classic vulnerabilities. However, your boss seems to be unfazed by the issues. So you want to show him the potential problems. Your task is to infiltrate this system and showcase what an actual attacker would be able to do.

The project consists of five tasks in total. Out of these tasks, Task No. 4 is a BONUS task and can be treated as optional. Therefore, tasks 1,2,3,5 are enough to provide you with the full grade. Every task/attack follows a similar execution flow at your end. Primarily, the victim-binary has at least 5 buffer overflow vulnerabilities, which you will take advantage of in each attack to generate unexpected and interesting results. To analyse these vulnerabilities, we have provided you with the victim's source code in `old_server.c`.

Overview You are provided with the binary `old_server` and its source code `old_server.c`.

- Do not modify the `old_server` binary or its environment. Your payload will be tested against the original binary on the provided VM. A payload that only works on a modified file or in a different environment will not be considered correct.
- The source code `old_server.c` is for reference only. Do not change it, as doing so will cause discrepancies when debugging with `gdb`.

You also have several python scripts `payload_`i`.py` to craft the payload where $i \in \{1, ..., 5\}$. Fill in those files, and they will put the actual payload in `task`i`-payload`.

You need to turn off ASLR when you execute each program. You can use

```
$ setarch -R your_program [arguments]
```

You will use GDB frequently. You can find the common commands in GDB easily online. One hint is to use `x/20x $sp` to print the stack. And use `print` to find the addresses of certain functions.

Task 1: The Lobby (30 pts) In this task, you need to bypass the authentication and get access to the server. Target the function `task1_variable_overwrite`. You want to overwrite the value of the `is_authenticated`.

Craft your payload in the `payload_1.py` to send in the message. Run `python3 payload_1.py` to store the binary payload in `task1-payload`,

Now, run command `./old_server 1 task1-payload`. You should see the output:

```
---[ TASK 1: Bypass The Lobby ]---
[SUCCESS] Authentication bypassed! Access granted to Server
```

Task 2: The Server (40 pts) Now you are inside the server. Target the function `task2_function_pointer_hijack`. You don't want to execute the log user action. Instead, you want to execute `grant_admin_privileges`. Try to hijack the control flow so that the desired function is executed.

Similarly, you need to create the payload in `payload_2.py`, and feed the payload to the old server binary. You should see the output:

```
---[ TASK 2: Infiltrate Server ]---
[SUCCESS] Administrative privileges granted!
```

Task 3: The Shell (40 pts) Now that you have the administrator privileges, you want to spawn a shell to execute arbitrary code. Target the function `task3_obtain_shell`.

You need to create the payload in `payload_3.py`, and feed the payload to the old server binary. Note: after you have spawned the shell. Please run a few commands like `ls, pwd` to make sure it is running properly. Record it in your report.

You should see the output:

```
---[ TASK 3: Access The Mainframe ]---
$ ls
Makefile payload_1.py payload_4.py task2-payload task5-payload
old_server payload_2.py payload_5.py task3-payload
old_server.c payload_3.py task1-payload task4-payload
$ pwd
/home/user443/project_2
```

Task 4: Obtain IDs (30 BONUS pts) You want to see the impact of the flaws and see if you can steal some IDs from the server. Your task is to obtain 5 IDs and exit the program. You need to obtain at least one ID each for a programmer, an engineer, and an admin. Target the function `task4_chain_attack`.

Note that your program should exit properly after these steps. If it crashes after obtaining the IDs, there would be point deductions. The output should be something similar to this.

```
---[ TASK 4: Steal the IDs ]---
Objective: COLLECT 4 IDs to bypass security.

[!!!] You've obtained programmer id 74876!

[!!!] You've obtained programmer id 15987!

[!!!] You've obtained engineer id 67121!

[!!!] You've obtained engineer id 69418!

[!!!] You've obtained admin id 88197!
[SUCCESS] You have obtained 5 IDs!
```

Task 5: Admin Imposter (50 pts) Finally, you want to print your name when the system prints the admin name. However, for this task, the attack only works in GDB. So you should first run `gdb ./old_server`. Target the function `task5_print_name_to_admin_list`.

The correct output should be something similar to this. Again, your program should not crash after printing your name.

```
(gdb) r 5 task5-payload
---[ TASK 5: Admin Imposter ]---
Admin: John_Joe
[Inferior 1 (process 479102) exited with code 0152]
(gdb)
```

You can replace John_Joe with your own name.

5 Questions

1. Draw the function's stack frame in Task 2 to demonstrate the overflow. Use tools like Paint, Excel or any other online tool to show the stack frame. Refrain from providing diagrams drawn by hand.
2. Why does Task 5 fail to run from the command line, but succeed when run in GDB debugger?
3. Why do Tasks 1-4 run both from the command line and GDB debugger the same?
4. Briefly identify and explain a viable defense mechanism to prevent the attack in Task 4. Precisely explain how this would prevent the attack you have crafted.

10 pts for each question.

6 Deliverables

Please submit a tar ball containing the following:

1. The Python scripts `payload_`*i*`.py` files (4 or 5 files) and payload files `task`*i*`-payload` (4 or 5 files).
2. A report in PDF containing:
   (a) Screenshot of each completed task (e.g., shell invocation from your execution of each case);
   (b) Briefly explain how you completed the task (You don't need to draw the stack frame for all tasks, but need to explain how you obtained the values in the payload);
   (c) Answers to project questions.

Total points are 200 + 30 (BONUS).