CSE543: Buffer Overflow & ROP Assignment

October 7, 2025

1 Introduction

Welcome! This document will guide you through setting up your virtual machine (VM) and completing the nine security attack assignments. Following these steps carefully is crucial for creating a stable and predictable environment, which will allow you to focus on the core concepts of the attacks.

2 Prerequisites

Before attempting this project, it is advisable to review the basics of stack frames, memory layout of a program, use of the GDB Debugger, and big-endian vs little-endian byte ordering. To quickly brush up on the basics of GDB debugging, we recommend watching this GDB Debugger Tutorial: `https://www.youtube.com/watch?v=J7L2x1ATOgk&t`.

3 Project Platforms and Files

For this project, you must use one of the provided Linux virtual machine (VM) environments. The exploits have been tested on these specific VMs, and using a different environment may cause your attacks to fail.

3.1 Virtual Machine Downloads

- **For Windows and Apple Intel chip platforms:**
  - Download the VM from: `https://drive.google.com/file/d/1mRiSVBgp-pwDRsD-8yF62jCmd7nMCD2I/view?usp=drive_link`
  - You will need to install Oracle VirtualBox to run the `.vbox` file.
- **For Apple M1, M1 Pro, M2 platforms:**
  - Download the VM from: `https://drive.google.com/file/d/13-MY3Zikrx0JlWlSxTRICiurpG9YM4h5/view?usp=sharing`
  - You will need to install UTM to run the `.utm` file.

Note: The password for the VM is '**esc**'.

3.2 Personalized Project Files

Your personalized project files, including victim source code and attack templates, are provided in a folder named after you.

- Download your files from: `https://drive.google.com/drive/folders/1Anra3H5TQ3oZ37LDpqQRtk61nxTPSz-w`

4 Part 1: Virtual Machine Setup

This section covers the configuration of your VirtualBox VM to prepare it for the assignments.

4.1 Enabling VirtualBox Guest Additions (Essential for Usability)

Guest Additions improve the performance and usability of your VM, enabling features like bidirectional copy/paste and drag-and-drop.

4.1.1 Step 1: Update Your VM's Package Manager

First, ensure your VM has the latest list of available software.

Terminal

```
sudo apt-get update
```

4.1.2 Step 2: Install Required Packages

Guest Additions require a few tools to build kernel modules. Install them with this command:

Terminal

```
sudo apt-get install build-essential linux-headers-$(uname -r) dkms
```

4.1.3 Step 3: Insert the Guest Additions CD Image

In the VirtualBox window menu for your running VM, navigate to **Devices → Insert Guest Additions CD Image...**. A dialog box may appear; if so, click "Run".

4.1.4 Step 4: Manually Run the Installation Script

If the installer does not start automatically, run these commands:

Terminal

```
# Mount the CD-ROM
sudo mkdir /media/cdrom
sudo mount /dev/cdrom /media/cdrom

# Run the installation script
sudo /media/cdrom/VBoxLinuxAdditions.run
```

4.1.5 Step 5: Reboot and Configure

The changes will only take effect after you restart the VM.

Terminal

```
sudo reboot
```

After rebooting, go to **Devices → Shared Clipboard → Bidirectional** and **Devices → Drag and Drop → Bidirectional** in the VirtualBox menu.

4.2 Disabling ASLR (Crucial for Predictable Attacks)

Address Space Layout Randomization (ASLR) must be disabled to ensure memory addresses are predictable.

4.2.1 Step 1: Check the Current ASLR Setting

Terminal

```
cat /proc/sys/kernel/randomize_va_space
```

A value of `2` means ASLR is on; `0` means it is off.

4.2.2 Step 2: Disable ASLR

Run the following command. **Note:** This setting is temporary and resets on reboot. You must run this command at the start of each work session.

Terminal

```
sudo sysctl -w kernel.randomize_va_space=0
```

4.3 Compiling Your Personalized Assignment Code

You must compile the provided source code yourself to create your unique victim binaries.

4.3.1 Step 1: Navigate to Your Assignment Folder

Open a terminal and `cd` into the directory created for you.

Terminal

```
cd /path/to/your/assignment_folder/
```

4.3.2 Step 2: Compile the Victim Programs

Use the provided `Makefile` to build all victim programs.

Terminal

```
make
```

This command creates a `bin/` directory and places all compiled executables inside it.

5 Part 2: Project Tasks

Each of the following tasks requires you to exploit a buffer overflow vulnerability in the corresponding victim binary. Analyze the provided source code to understand the vulnerability and craft your attack.

5.1 Task 1: Collect Your Saber

- **Victim:** `victim1-binary`
- **Goal:** Invoke the `get_saber` function with the correct key.
- **Logic:** The `enter_temple` function contains a buffer overflow vulnerability. Exploit it to overwrite the local `key` variable on the stack with the student-specific `V1_TARGET` value defined in your header file.

5.2 Task 2: Choose Your Kyber Crystal

- **Victim:** `victim2-binary`
- **Goal:** Influence the `crystal` method to print "Purple", "Green", or "Blue" instead of the default "Red".
- **Logic:** Analyze `victim2.c` to identify which variables control the output of the `crystal` method and overwrite them using a buffer overflow.

5.3 Task 3: Escape the Death Star

- **Victim:** `victim3-binary`
- **Goal:** Invoke the `rescue` function to get a shell.
- **Logic:** Exploit a vulnerability to manipulate program flow, ensuring the `rescue` function is called while the `capture` function is bypassed.

5.4 Task 4: Escape Hoth

- **Victim:** `victim4-binary`
- **Goal:** Call the `hyperspace()` method, which is not normally called.
- **Logic:** Use a buffer overflow to overwrite a saved return address on the stack, changing it to the address of the `hyperspace` function.

5.5 Task 5: Complete Your Jedi Training

- **Victim:** `victim5-binary`
- **Goal:** Call three functions in a specific sequence: `round_1()`, then `round_2()`, then `round_3()`.
- **Logic:** Craft a payload that overflows the buffer and overwrites multiple return addresses on the stack to chain the function calls in the correct order.

5.6 Task 6: Rescue Han from Jabba

- **Victim:** `victim6-binary`
- **Goal:** Force the program to print "You rescue Han from Jabba!".
- **Logic:** This task involves a heap-based buffer overflow. You must understand how `malloc` allocates memory to overwrite data on the heap to control program execution.

5.7 Task 7: Befriend the Ewoks

- **Victim:** `victim7-binary`
- **Goal:** Make the program print your name.
- **Logic:** The payload must construct an entirely new, fake stack frame. This frame will set up the arguments for a call to the `system("printf")` library function to print your name.

5.8 Task 8: Shellcode Injection

- **Victim:** `victim8-binary`
- **Goal:** Spawn a shell (`/bin/sh`).
- **Logic:** The stack for this binary is executable. Your payload must contain "shellcode" (raw machine code that executes a shell) and overwrite the return address to jump to the location of your shellcode on the stack.

5.9 Task 9: Return Oriented Programming (ROP)

- **Victim:** `victim9-binary`
- **Goal:** Spawn a shell (`/bin/sh`).
- **Logic:** The stack for this binary is non-executable (DEP is enabled). You cannot inject shellcode directly. Instead, you must use a Return Oriented Programming (ROP) attack. Use a tool like `ROPgadget` to find existing code "gadgets" in the binary and chain them together by manipulating the stack to achieve your goal.

5.10 Report Questions

Your final report must include answers to the following questions:

1. Draw the heap diagram in Task 6 before and after the attack. Use a diagramming tool; do not submit hand-drawn diagrams.
2. Why does Task 7 fail to run from the command line, but succeed when run in the GDB debugger?
3. Draw the function's stack frame in Task 8 to demonstrate the overflow. Use a diagramming tool.
4. Were you able to complete the attack in Task 9? If you used ROP, specify the gadgets you used. Otherwise, explain your method.
5. Research and discuss mitigations for ROP and buffer overflow attacks. Which protections do you think are most effective? How might an attacker bypass common protections? (This question is open-ended).

5.11 Deliverables

Please submit a single zip file containing the following:

- All of your completed `attack*.c` and `attack*.py` files (9 total).
- The corresponding compiled binaries for your victims.
- The corresponding compiled binaries for your C attacks (`attack*-binary`, 6 total).
- The corresponding payload files generated by your C attacks (`attack*-payload`, 6 total).
- A single report in PDF format containing:
  - Screenshots of the successful output for each of the 9 tasks.
  - Your answers to the report questions listed above.

6 Grading

The assignment is worth 100 points, broken down as follows:

- **Successful VM Setup (10 points)**.
- **Answers to Report Questions (10 points).**
- **Correct Packaging (5 points):** Your submitted zip file is correctly structured and all attack programs build without errors.
- **Report Completeness (30 points):** The report is well-organized and includes all required screenshots and answers.
- **Tasks 1-9 (45 points):** 5 points for each successfully completed task.

6.1 Final Note

A demo will be scheduled for each student with the TA, where you will be asked to explain 1-2 randomly selected tasks. Please keep your VM and attack files saved after submission. Your ability to explain your work can affect your final grade. This is an individual project.