

Lab 1: Buffer Overflow and Security Consequences

Learning Goals

By the end of this Lab, you should be able to:

- 1) Set up a Raspberry Pi5 and a simple LED circuit with GPIO
- 2) Understand how data and contiguous memory is stored in low-level programming languages such as C
- 3) Implement a simple login system in C with a username, password, and a system password (secret)
- 4) Observe how buffer overflow vulnerabilities compromise memory and security of a system
- 5) Use gdb to analyze memory layout, execution step, and visualize why overflow occurs
- 6) Recognize consequences of insecure code and learn best coding practices to prevent buffer overflows

Introduction

What is buffer overflow?

As you learned in class (refer Lecture 2: Slide 20-26), buffer overflow occurs when a program writes more data into a fixed-size buffer than it can hold.

In C, if bounds are not handled correctly, extra bytes can overwrite adjacent memory leading to buffer overflow.

In this lab, you will design and analyze a simple security system called **The Vault**, running on a Raspberry Pi5. The Vault is meant to protect a **sensitive system password** (such as a root password or API key) while allowing users to log in with their own username and password.

During this lab you will take on multiple roles:

- 1) Developer Role: You will build the Vault application in C based on the instructions provided to you below and configure a raspberry pi5 as your “security indicator”.
- 2) Attacker Role: Once your application is ready, you will try to break the Vault by attempting to do a buffer overflow attack, identify the vulnerabilities, and inspect raw memory using GDB to understand how the attack worked.
- 3) Security Analyst Role: You will attempt to fix the identified vulnerabilities to secure the Vault.

Lab Setup

Hardware

- 1) Raspberry Pi 5
- 2) Breadboard
- 3) 1 x Red LED

- 4) 1 x 330Ω resistor
- 5) Jumper wires





















Part 1 - 5 points

Connect the LED & resistor using GPIO and the wiringPi library (check Raspberry Pi 5 & wiringPi pin layout below) in such a way that the Red LED turns on when the system is compromised.

Use test1.c to ensure that your wiring is correct.

To compile and run test1.c:

```
gcc test1.c -o test1 -lwiringPi
```

Raspberry Pi Model B+ (J8 Header)					
GPIO#	NAME			NAME	GPIO#
	3.3 VDC Power	1		2	5.0 VDC Power
8	GPIO 8 SDA1 (I2C)	3		4	5.0 VDC Power
9	GPIO 9 SCL1 (I2C)	5		6	Ground
7	GPIO 7 GPCLK0	7		8	GPIO 15 TxD (UART)
	Ground	9		10	GPIO 16 RxD (UART)
0	GPIO 0	11		12	GPIO 1 PCM_CLK/PWM0
2	GPIO 2	13		14	Ground
3	GPIO 3	15		16	GPIO 4
	3.3 VDC Power	17		18	GPIO 5
12	GPIO 12 MOSI (SPI)	19		20	Ground
13	GPIO 13 MISO (SPI)	21		22	GPIO 6
14	GPIO 14 SCLK (SPI)	23		24	GPIO 10 CE0 (SPI)
	Ground	25		26	GPIO 11 CE1 (SPI)
30	SDA0 (I2C ID EEPROM)	27		28	SCL0 (I2C ID EEPROM)
21	GPIO 21 GPCLK1	29		30	Ground
22	GPIO 22 GPCLK2	31		32	GPIO 26 PWM0
23	GPIO 23 PWM1	33		34	Ground
24	GPIO 24 PCM_FS/PWM1	35		36	GPIO 27
25	GPIO 25	37		38	GPIO 28 PCM_DIN
	Ground	39		40	GPIO 29 PCM_DOUT

Attention! The GPIO pin numbering used in this diagram is intended for use with WiringPi / Pi4J. This pin numbering is not the raw Broadcom GPIO pin numbers.

<http://www.pi4j.com>

Building the Vault (with Starter Code)

You will now extend the program to simulate a login system with username, password, and a secret system password.

Part 2 - 25 points

You must:

- 1) Initialize system_password with a password of your choice (this is intended to be protected)

- 2) Your application must prompt the user to enter a username and password using gets() [*Do not use any other function, to read input, for this part]
- 3) Print the state of username, password, flag
- 4) If flag is not 0, print the system_password and turn the Red LED ON to show compromise
- 5) Otherwise, print that the password is protected

*Please follow the TODOs in lab1.c closely.

To compile the program & run the program:

```
gcc -O0 -g lab1.c -o lab1 -lwiringPi
./lab1
```

Part 3 - 15 points

1. Run with short inputs (e.g., username ≤ 8 chars, password ≤ 8 chars) (5 Points)

Q. What do you observe about the flag?

ANS:

Q. Did the Red LED switch ON?

ANS:

Q. What happens to the system password?

ANS:

2. Break the vault! (10 Points)

Find ways to break the vault i.e. enter username and password combinations that cause:

- I. Flag to become non-zero
- II. Red LED to switch ON
- III. System password to be compromised (printed)

Q. Write a short answer describing how you managed to compromise the vault (describe at least three different ways and their consequences)

Hint: see what happens when you overflow the username vs overflowing the password and observe different ways in which the application and system password is compromised.

ANS:

Part 4 - 25 points

Live Demo

Make sure your circuit and code works to show how the LED switches ON and how the application vulnerabilities can cause the password system to be compromised.

4.1: Demo at least three different ways you can compromise the system (30 pts) Hint: Search what null pointers are, and how they behave during overflows.

4.2: This part will involve questions during the demo that revolve around your understanding of the code, buffer overflow, and vulnerabilities in the code. (20 pts)

Part 5 - 5 points

Inspecting with GDB

1. Compile your application, so that it can be debugged with GDB:

Eg: gcc -O0 -g lab1.c -o lab1 -lwiringPi

2. Run with gdb

Eg: gdb ./lab1

3. Use breakpoints to pause execution before and after input

Eg: break lab1.c:17 (to break at line 17)

4. Some commands to help you use gdb:

run : run the application with gdb

continue: continue the application after a break point

x/<count><format><size> <address>: to examine raw memory

Eg : x/16bx &data.username ; this command shows 16 bytes of raw memory starting from where username is stored

Quit: to exit gdb

5.1: Attach a screenshot of your gdb console showing

- i. Raw memory when there is no buffer overflow
- ii. Raw memory when there is an overflow in the username
- iii. Raw memory when there is an overflow in the password
- iv. Raw memory when the flag is damaged

SCREENSHOTS:

Securing the Vault

Part 6 - 25 points

Create a copy of the lab1.c code and modify that copy to strengthen the vault and protect against buffer overflow attacks. During the live demo, show how your modified code withstands the previous buffer overflow attacks (from part 4.1) without compromising the system password. Make sure you understand how to mitigate the previously detected vulnerabilities.

Deliverables

- 1) Source Code (lab1.c)
- 2) Live Demo
- 3) Screenshots of gdb output
- 4) Short answers for part 3
*[You can make a copy of this document, and edit it to answer the questions and attach screenshots]
- 5) Source code for the secured vault