# CS 5153/5053 Network Security, Spring 2023

## Project 2: Buffer Overflow Attack

## Report

### **Student:** Austin Tyler Conn

Link to Source Code <https://github.com/austinc3030/buffer_m11809075>

## Host Environment Used

Operating System: macOS Ventura 13.0.1

Hardware: Apple M1 Mac Mini (ARM Architecture)

Hypervisor: UTM

## Virtual Machine Environment Used

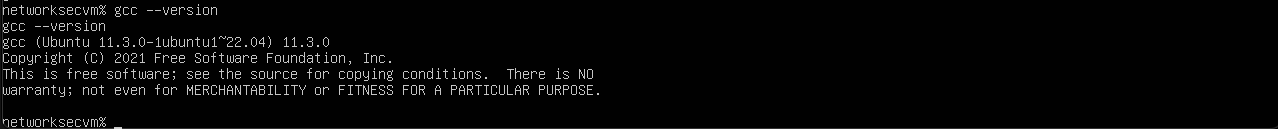
Operating System: Ubuntu Desktop 22.03.2



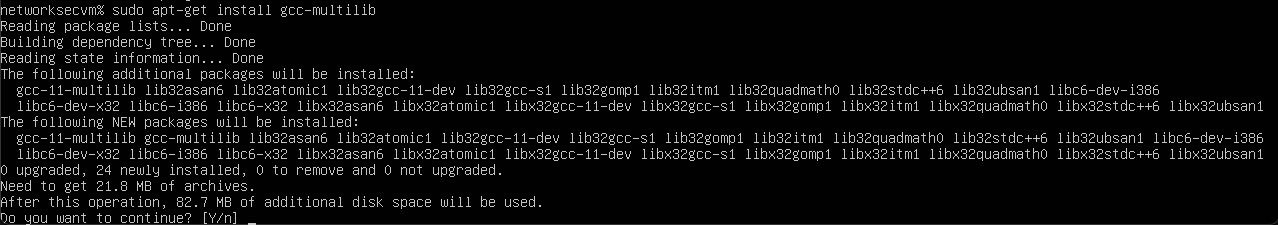
Shell: zsh v5.8.1A black screen with white text

Description automatically generated with low confidence

gcc:



Caveats:

1. Due to differences between 32/64-bit assembly, the `gcc-multilib` package is required. 
   1. Further, when compiling c code in this environment, the gcc flag `-m32` is required to instruct gcc to compile for a 32-bit architecture.
2. The gdb plugin Python Exploit Development Assistance for GDB (PEDA) is installed on this virtual machine for ease of use in gdb

Text

Description automatically generated

## How do you perform the attack in your VM

1. Disable address space randomization



1. Compile *stack.c* with the flags:
   1. `-m32` instructs gcc to compile for 32-bit architectures
   2. `-z execstack` make the stack executable
   3. `-fno-stack-protector` disabled Stack-Guard Protection Scheme

Text

Description automatically generated

***Note:*** *it is assumed here that the steps in the section below, “How do you find the value of ebp” have been completed previously to create the badfile with the proper content*



## How do you find the value of ebp

1. Compile *stack.c* with the flags:
   1. `-m32` instructs gcc to compile for 32-bit architectures
   2. `-g` debug flag
   3. `-z execstack` make the stack executable
   4. `-fno-stack-protector` disabled Stack-Guard Protection Scheme

Text

Description automatically generated

1. Create a temporary, blank file *badfile* (failure to do so leads to a segmentation fault)

Shape

Description automatically generated with medium confidence

1. Run *stack\_debug* with *gdb*

Text

Description automatically generated

1. Set a breakpoint on the *bof* function



1. Continue execution of the program

Text

Description automatically generated

1. Obtain the value of *ebp*  
   

*ebp* = **0xffffd858**

## How do you decide the content of badfile

1. Complete the steps in the section above, “How do you find the value of ebp” as the *ebp* value found will be used in the calculations for the construction of the badfile
2. In addition to the value of *ebp*, the address where *buffer* starts is required



*buffer* starts at **0xffffd7ce**

1. Find the difference between the return address (*ebp*) and the start address of *buffer*



1. The address we want to return should be stored in the badfile at offset *ebp* + 4, thus we should store the address at offset 142 in the badfile

## Whether your attack is successful