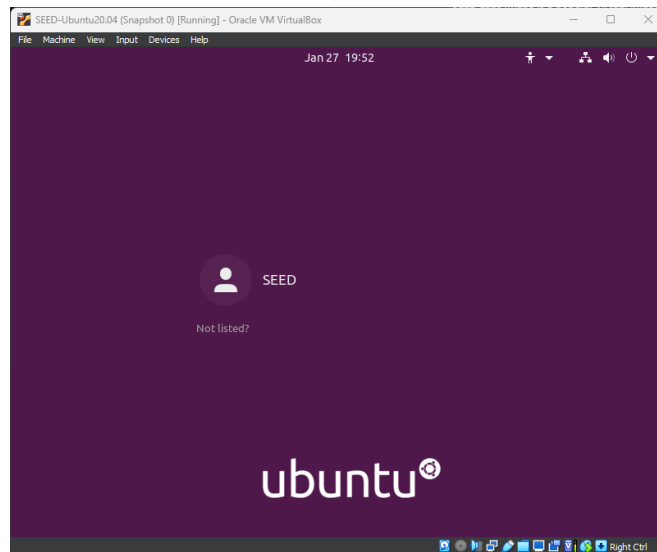


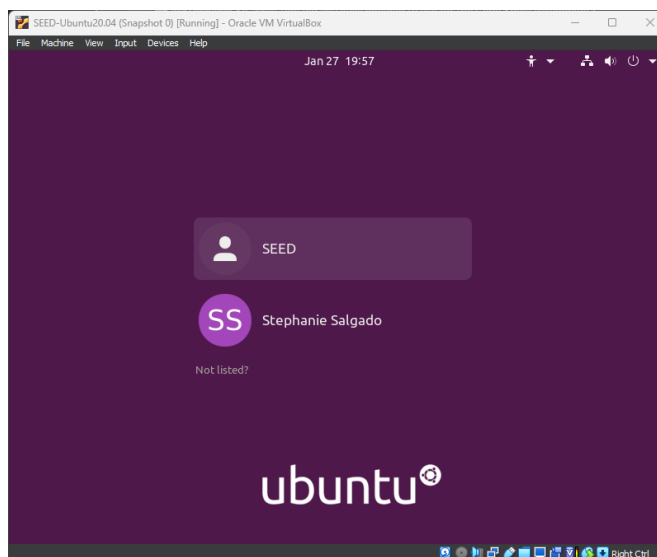
Lab 2 Demo: Shellcode Development

Stephanie Salgado

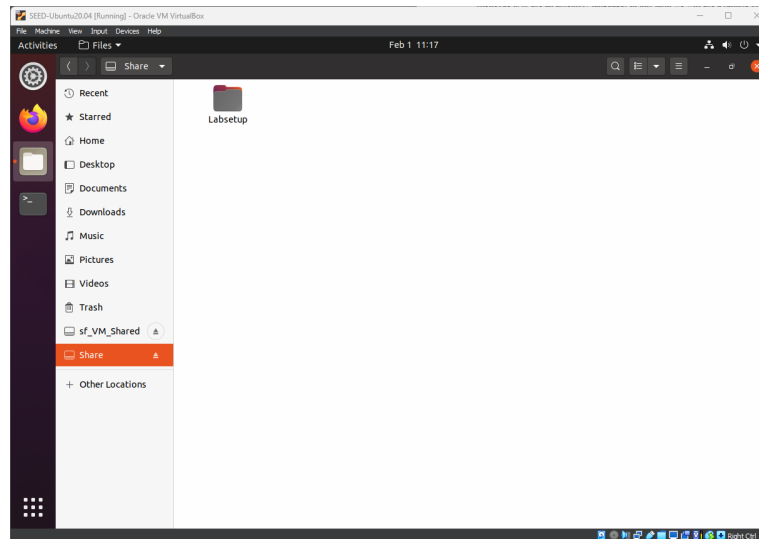
Configured and Launched SEED VM:



Created Username:

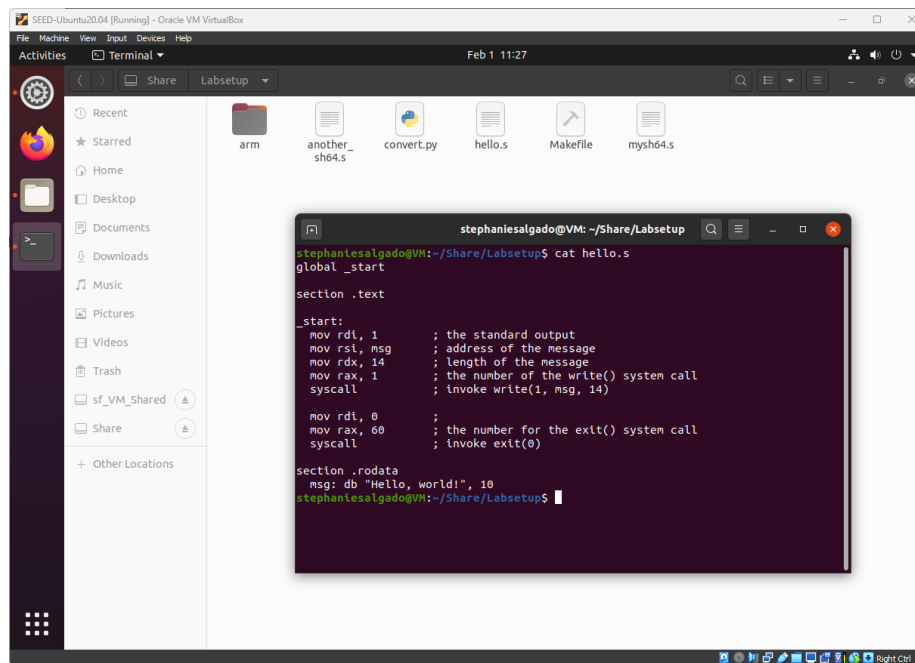


Demo:



Downloaded and unzipped the lab set up files in host then added them to the shared folder.

Task 1: Writing Assembly Code



Located "hello.s" file required for the task.

```
stephaniesalgado@VM: ~/Share/Labsetup
stephaniesalgado@VM:~/Share/Labsetup$ cat hello.s
global _start

section .text

_start:
    mov rdi, 1        ; the standard output
    mov rsi, msg       ; address of the message
    mov rdx, 14        ; length of the message
    mov rax, 1         ; the number of the write() system call
    syscall            ; invoke write(1, msg, 14)

    mov rdi, 0         ;
    mov rax, 60        ; the number for the exit() system call
    syscall            ; invoke exit(0)

section .rodata
    msg: db "Hello, world!", 10
stephaniesalgado@VM:~/Share/Labsetup$ nasm -f elf64 hello.s -o hello.o
stephaniesalgado@VM:~/Share/Labsetup$ ld hello.o -o hello
stephaniesalgado@VM:~/Share/Labsetup$ ./hello
Hello, world!
stephaniesalgado@VM:~/Share/Labsetup$
```

Compiled assembly code using “nasm” with “-f elf64” indicating that it should compile the code to 64-bit ELF binary format. After getting the object code “hello.o”, running the linker program “ld” is required to generate the executable binary. Yielding the final executable code “hello” which when run prints “Hello, world!”.

```
stephaniesalgado@VM: ~/Share/Labsetup
stephaniesalgado@VM:~/Share/Labsetup$ objdump -Mintel -d hello.o

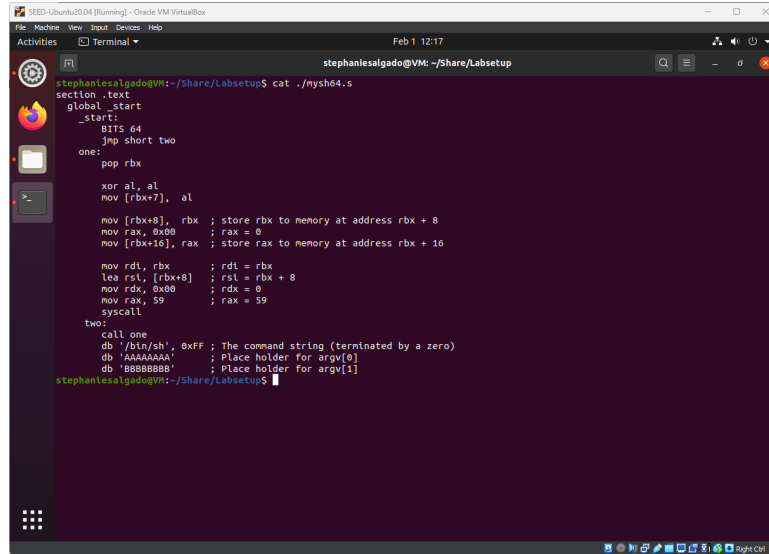
hello.o:      file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <_start>:
 0:  bf 01 00 00 00      mov     edi,0x1
 5:  48 be 00 00 00 00    movabs rsi,0x0
 c:  00 00 00
 f:  ba 0e 00 00 00      mov     edx,0xe
14:  b8 01 00 00 00      mov     eax,0x1
19:  0f 05               syscall
1b:  bf 00 00 00 00      mov     edi,0x0
20:  b8 3c 00 00 00      mov     eax,0x3c
25:  0f 05               syscall
stephaniesalgado@VM:~/Share/Labsetup$
```

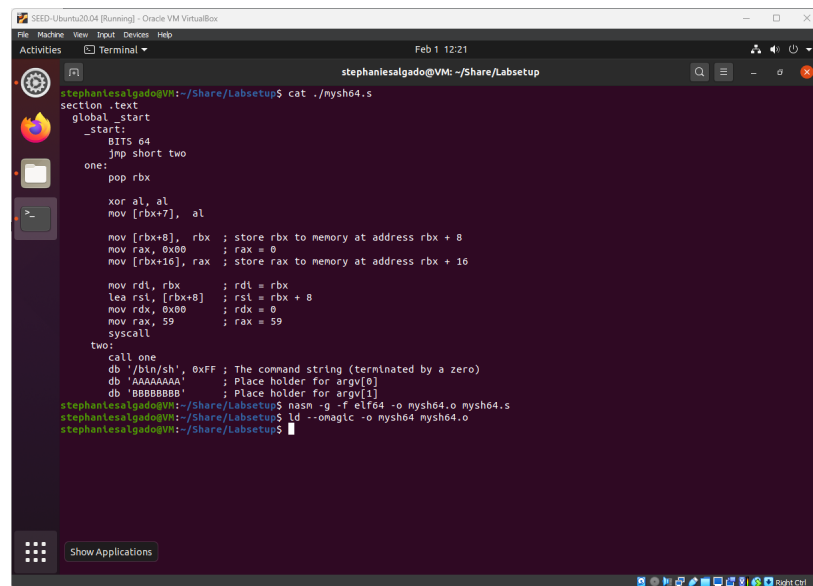
There are different methods to extract the machine code from the executable file or the object file. In this case, we used “objdump” to disassemble the executable or object file. Since “objdump” uses the AT&T mode by default the “-Mintel” flag was used for Intel mode.

Task 2: Writing Shellcode (Approach 1)



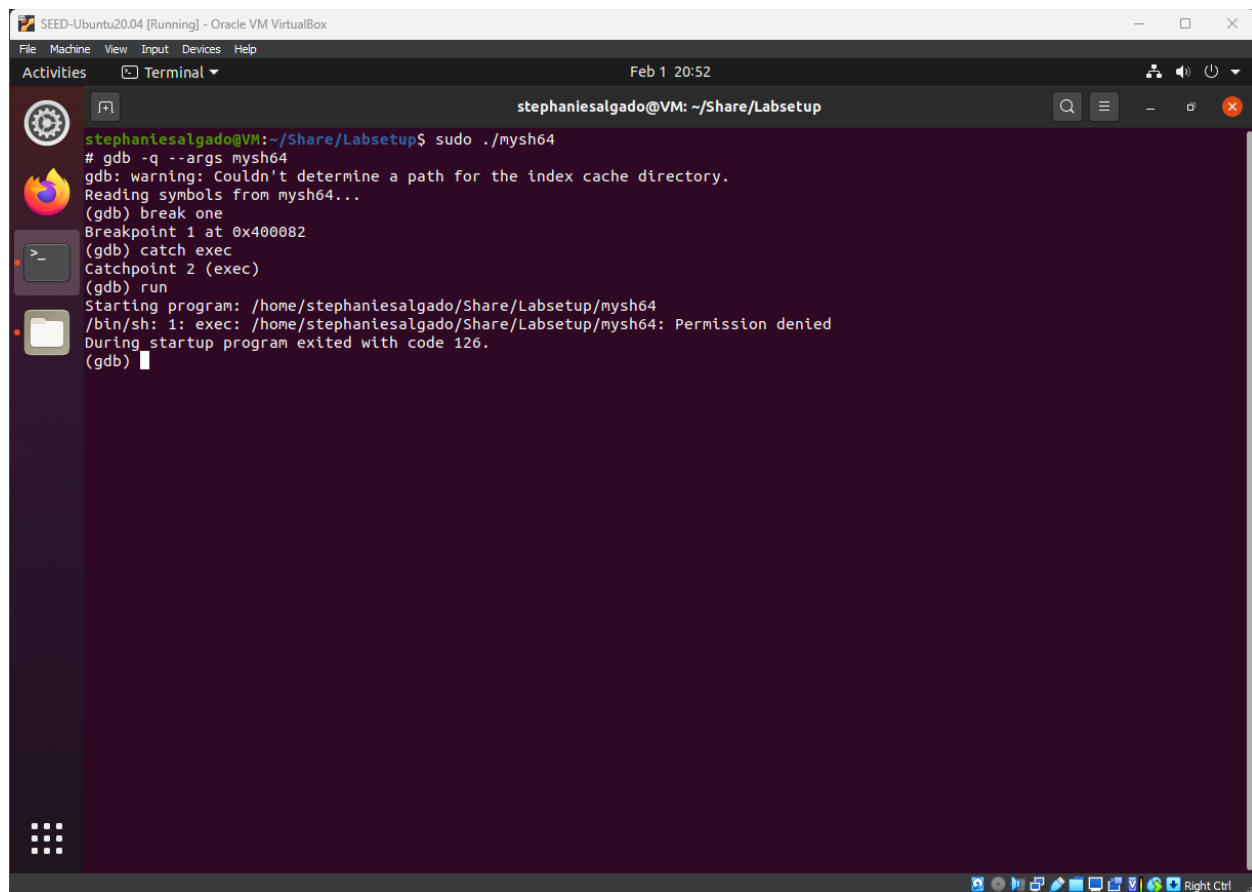
```
stephaniesalgado@VM: ~/Share/Labsetup$ cat ./mysh64.s
section .text
global _start
_start:
    BITS 64
    jmp short two
one:
    pop rbx
    xor al, al
    mov [rbx+7], al
    mov [rbx+8], rbx ; store rbx to memory at address rbx + 8
    mov rax, 0x00 ; rax = 0
    mov [rbx+16], rax ; store rax to memory at address rbx + 16
    mov rdi, rbx ; rdi = rbx
    lea rsi, [rbx+8] ; rsi = rbx + 8
    mov rdx, 0x00 ; rdx = 0
    mov rax, 59 ; rax = 59
    syscall
two:
    call one
    db '/bin/sh', 0xFF ; The command string (terminated by a zero)
    db 'AAAAAAA' ; Place holder for argv[0]
    db 'BBBBBBBB' ; Place holder for argv[1]
stephaniesalgado@VM: ~/Share/Labsetup$
```

The “mysh64.s” seemed to match the file used in the lab pdf.



```
stephaniesalgado@VM: ~/Share/Labsetup$ cat ./mysh64.s
section .text
global _start
_start:
    BITS 64
    jmp short two
one:
    pop rbx
    xor al, al
    mov [rbx+7], al
    mov [rbx+8], rbx ; store rbx to memory at address rbx + 8
    mov rax, 0x00 ; rax = 0
    mov [rbx+16], rax ; store rax to memory at address rbx + 16
    mov rdi, rbx ; rdi = rbx
    lea rsi, [rbx+8] ; rsi = rbx + 8
    mov rdx, 0x00 ; rdx = 0
    mov rax, 59 ; rax = 59
    syscall
two:
    call one
    db '/bin/sh', 0xFF ; The command string (terminated by a zero)
    db 'AAAAAAA' ; Place holder for argv[0]
    db 'BBBBBBBB' ; Place holder for argv[1]
stephaniesalgado@VM: ~/Share/Labsetup$ nasm -g -f elf64 -o mysh64.o mysh64.s
stephaniesalgado@VM: ~/Share/Labsetup$ ld --omagic -o mysh64 mysh64.o
stephaniesalgado@VM: ~/Share/Labsetup$
```

Using “-g” enabled debugging information.



```
SEED-Ubuntu20.04 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal Feb 1 20:52
stephaniesalgado@VM: ~/Share/Labsetup

stephaniesalgado@VM:~/Share/Labsetup$ sudo ./mysh64
# gdb -q --args mysh64
gdb: warning: Couldn't determine a path for the index cache directory.
Reading symbols from mysh64...
(gdb) break one
Breakpoint 1 at 0x400082
(gdb) catch exec
Catchpoint 2 (exec)
(gdb) run
Starting program: /home/stephaniesalgado/Share/Labsetup/mysh64
/bin/sh: 1: exec: /home/stephaniesalgado/Share/Labsetup/mysh64: Permission denied
During startup program exited with code 126.
(gdb)
```

For some reason, the program didn't want to execute. I tried using "chmod +x mysh64" to change permissions and that also didn't seem to work. Although the majority of the content covered in the lab was new to me, I did my best to experiment with different gdb commands. I tried "x/s \$rbx" which from my search, is supposed to be the address where "bin/bash" is stored. I also tried "info registers rbx" which should show the current value in "rbx" register. Whenever I did this, the return was "The program has no registers now." and "No registers." respectively. Around this time, my screen capture tool started quitting whenever I attempted to screenshot the results.

It's very possible I was running the wrong commands or that I might've not fully understood the scope of the lab. Even so, I feel that I learned a lot from it and hopefully I get to advance further in the next lab.