**sploit1**

The vulnerability is: strcpy() does not check the number of bits which will be copied into "char buf[96]", so we overflow the return address of “lab\_main()” and redirect the execution of the program to the shellcode we provide which runs a shell terminal.

**sploit2**

The vulnerability was that the len specified in the program was set to a max of 272. This allowed the attack string to overwrite i and len. The i variable was first overwritten such that it would remain small, so just the last byte of its four was overwritten with 0xb. Then len was overwritten 0x0000011c to enable i to index into buf to reach where the return address laid in memory. The environment variables were necessary to split up the attack string whenever a null byte was needed.

**sploit3**

The vulnerability is: "char buf[64]" is only 64 bytes, but "bar ( arg, buf, 88 )" allows us to write 88 bytes into it, therefore, we overflow the return address of “foo()” and redirect the execution of the program to the shellcode we provide which runs a shell terminal.

sploit4

Similar to sploit2, len could be set to a max of 169 and so len and i could be overwritten. The difference for sploit4 is that there is no indexing into the array using i. Thus, len first has be overwritten to allow enough iterations to overwrite i. The new value of i to be written needs to be at least such that len - i = number of iterations needed to fully overwrite all four bytes of the return address. The environment variables were necessary to split up the attack string whenever a null byte was needed.

sploit5

The vulnerability is: passing a certain number of “%x’s” to the “snprintf()” function allows us to change the argument pointer to point to the beginning of the “formatString”. At the beginning of the “formatString” we insert the location of the first 4 bytes of the return address of foo() seperated by NULL bytes. We then pass “%##u%hhn” to “snprintf()” four times to change the first 4 bytes of the return address one at a time. We then let the program continue to run, and once it returns from foo(), it will jump to the beginning of “char buf[1024]” and start executing the shellcode we inserted in there.

sploit6

The solution required two fake chunk tags. The first was located on the 8 bytes before the address of q where the double free was called and the second was placed a short distance away. The first tag’s next pointer pointed to the second fake tag. The second tag’s next pointer had the address of the return address, which was copied to the first’s next. It was then dereferenced and overwritten with a value in tmalloc.c’s “arena”, redirecting program execution. Placing the shellcode somewhere after the q region allowed a new shell to be spawned.