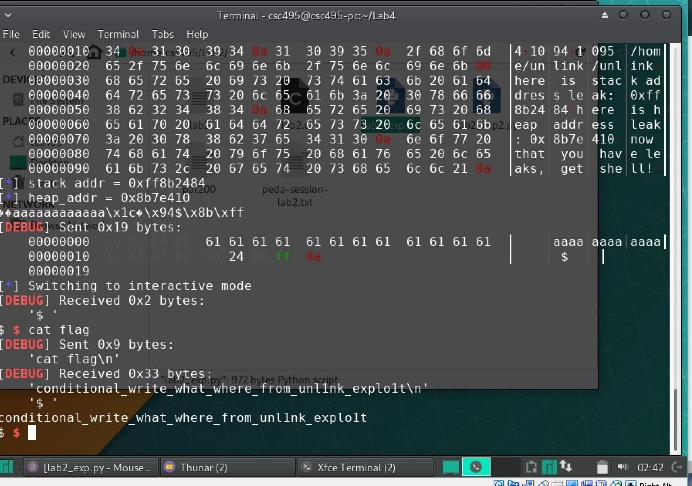
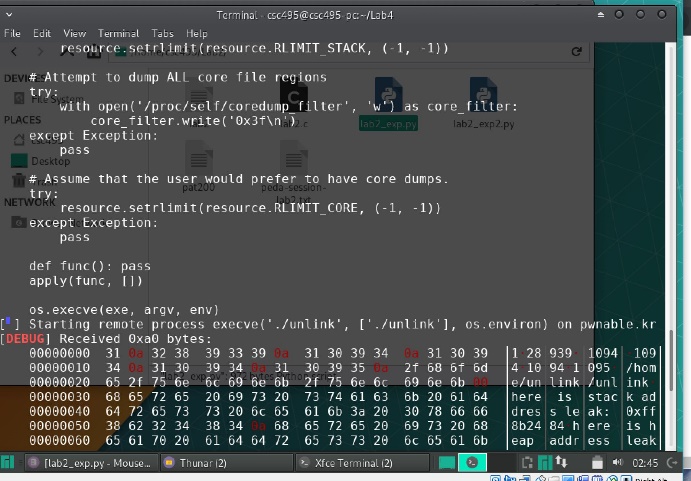
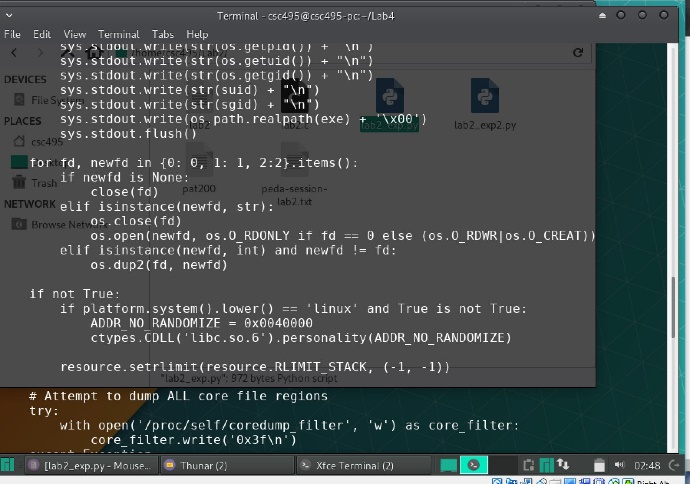
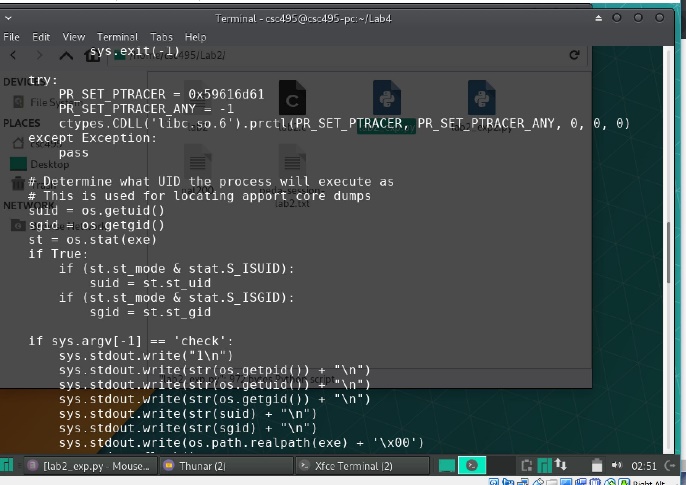
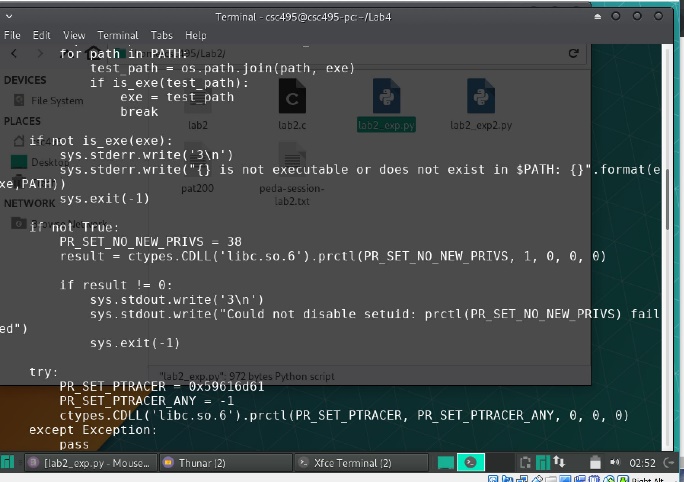
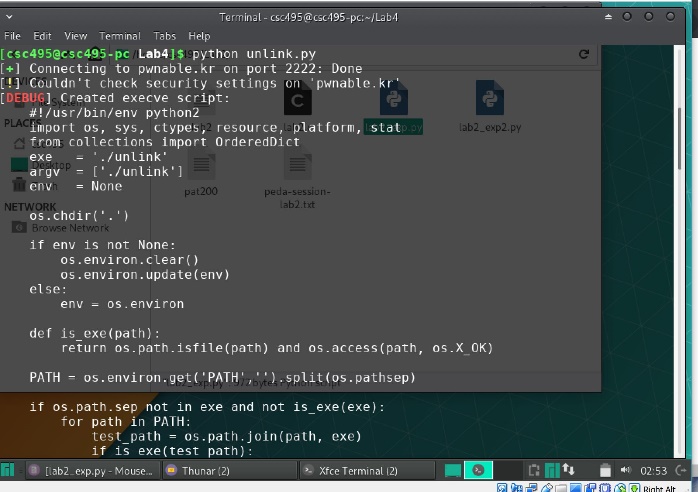
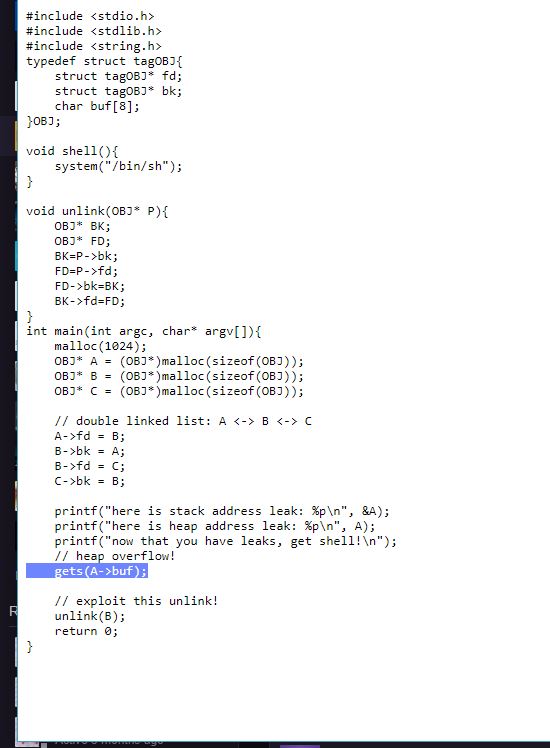
1.

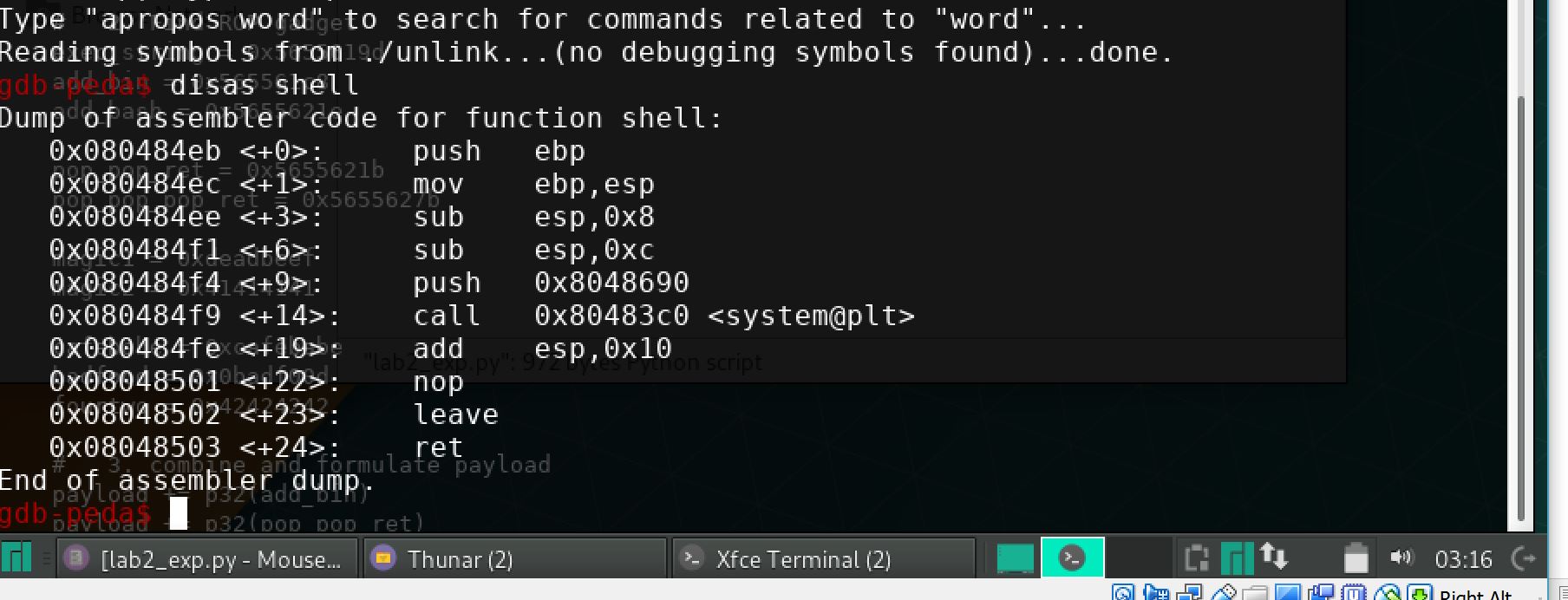


2.



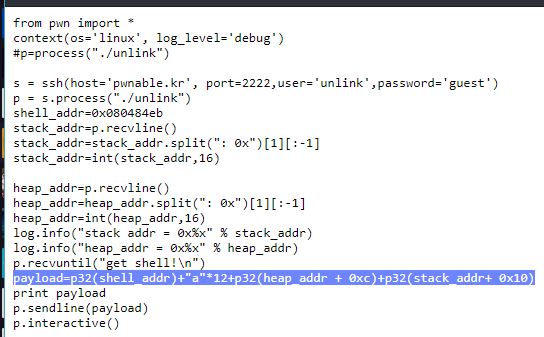
The char buf[8] is specifically the line that creates a buffer that can be exploited, however the gets(A->buf); is what causes the heap overflow to actually be able to occur.

3.



After using GDB to look at unlink, using disas shell reveals the memory address of the function.

4.



The payload consists of the shell address, then 12 a’s, then heap address plus 12 (locations), finally the stack address plus 10 (locations). These addresses allow for the exploit as memory is run though it can will trigger the different parts, as they are ordered, and run the shell at the end.

5.

By looking at the payload we can see how there is an addition of multiple addresses together matching up to full address spaces. With the heap address, there is 0xC added to it, this moves the address 12 spaces, the same amount as the injected a’s. Then there is the stack with 0x10 added to it which is 16 in decimal, or equal to two memory addresses. These allow for positioning and skipping of addresses in memory. The stack address will point to the stack, and so it can be used when unlinked, and the heap address will allow to pick out the space in the heap to be unlinked and overwritten. By using the specific offsets we can accurately pick out where to jump to next and have the exploit work.