This exploit involves using an exploited client to overflow a buffer of the server in order to run malicious commands on the server. When server.c is running, it waits for messages from clients (client.c). These messages are read into a buffer that is npages (n\*4096 bytes) long in the server’s buffer. However, if a message is sent that overflows this buffer, the function pointer s.fp can be overwritten. For this exploit, we overwrite s.fp to an address somewhere in the middle of the buffer. The buffer, which is filled by the message we sent, is mostly filled with NOP (no instruction but increase the program counter) instructions. Our malicious shellcode sits in the high addresses of the buffer, while the middle is made up of NOP instructions. This means that when we overwrite s.fp to somewhere in the middle of the buffer, the counter will keep increasing until it hits and executes our shellcode, meaning that we do not need to know the exact addresses for the shellcode in the server’s buffer to launch the attack. In my case, the data address range for the buffer was 0xdeea0000-0xf7540000, and s.fp was set to 0xf0000000, somewhere in the middle of the range. The shellcode runs shell\_reverse\_tcp, which allows the attacked to listen in on a client port. Thus when the attacker runs nc -I 9010 and the exploit program (in separate terminals), the attacker is able to run their own shell commands on the server, such as ls, pwd, rm, and cat. I was able to run cat server.c and see the source code of the server program, even while outside the server’s directory by listening on port 9010.