|  |
| --- |
| **实验报告** |

|  |  |  |  |
| --- | --- | --- | --- |
| **实验名称：** | **Lab3** | **日期：** | **2024.12.10** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **班级：** | **U10M21003.05** | **学号：** | **2024302589** | **姓名：** | **苗清皓** |

1. **实验概述：**

Lab3给定我们一个bufbomb文件，要求我们利用bufbomb文件中Gets函数的不安全性对bufbmob程序进行攻击。

实验原理：对目标程序实施缓冲区溢出攻击，通过造成缓冲区溢出来破坏目标程序的栈帧结构，继而执行一些原来程序中没有的行为。包括调用其他函数，改变全局变量，传递特定参数等。

实验目的：加深对IA-32函数调用规则和栈帧结构的理解。

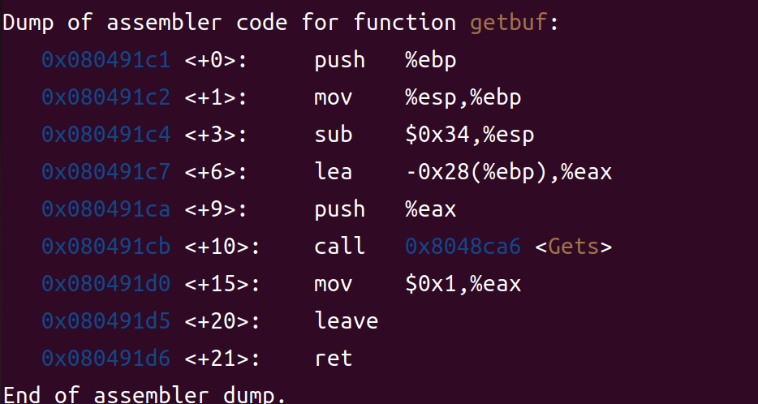
实验任务：构造5个攻击字符串，对目标文件进行5次缓冲区溢出攻击。

1. **实验分析：**

第一次攻击：smoke

目标：让目标程序调用smoke函数

思路：既然Gets函数读取过多字符会改写栈中其他内容，那么只要让Gets函数返回后，让PC寄存器中存放smoke函数的起始地址即可。



Gets函数是由getbuf函数调用的，getbuf传给Get函数一个地址作为参数，即-0x28(%ebp)，Gets函数以此处作为存放字符串的起点。

由于getbuf函数的栈帧大小有限，只要读取超过40个字节就会破坏栈帧的其他内容。我们已经知道在调用getbuf前的call指令会把getbuf调用完成成后要执行的下一条指令压栈，即返回地址。那么，只要我们把返回地址，即第45到第48个字节替换为smoke的起始地址即可。

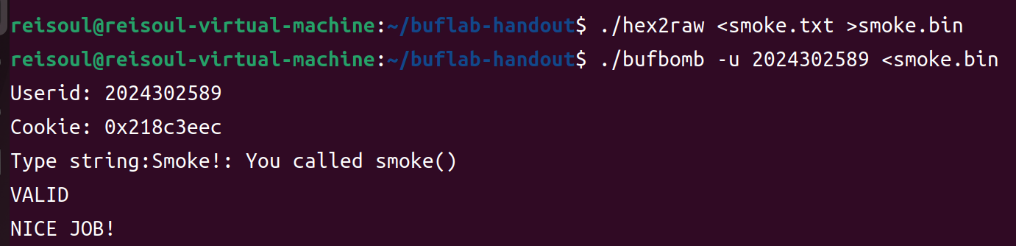
同时，linux系统使用的是小端存放，因此smoke函数的起始地址“08048bde”需要“倒过来”，低有效位放在低地址处。

由此在smoke.txt文件中构造字符串：

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

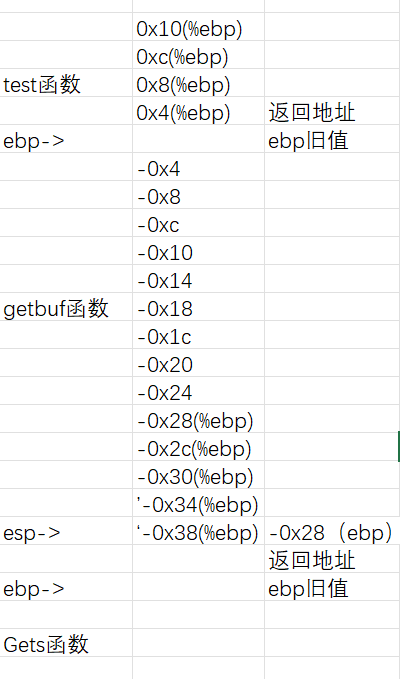
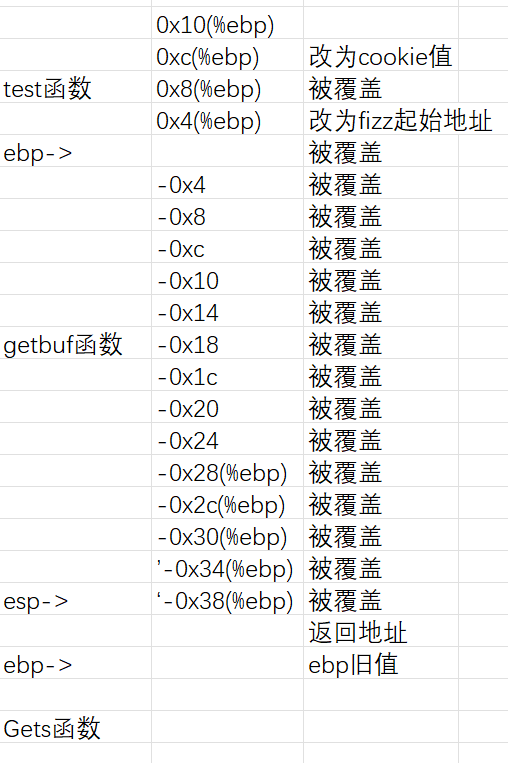
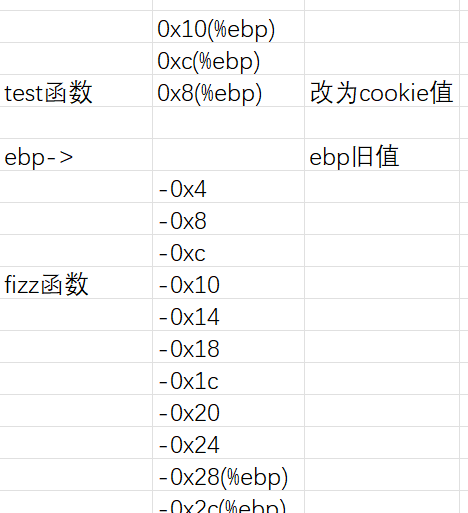
00 00 00 00 00 00 00 00 00 00 00 00 de 8b 04 08

但是还没有结束，因为我们输入的都是字符串，输入计算机后会转换为ASCII码，而我们是想要以二进制形式直接输入栈中。这是就要用到实验包中的hex2raw工具，其作用是把16进制的文本文档中的字符转换为二进制形式，通过新生成的二进制文件输入正确答案。

第二次攻击：fizz

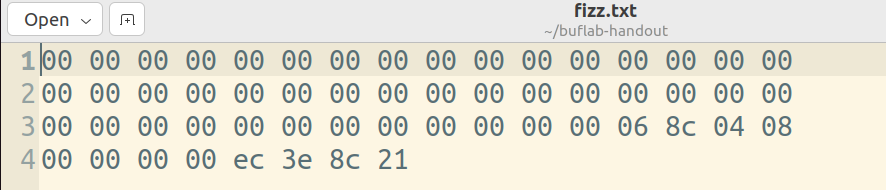
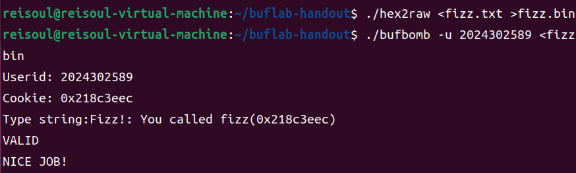
目标：让目标程序使用特定参数cookie调用fizz函数。

Cookie是根据每个人的学号生成的一串数据，如上图所示。

思路：这次攻击时smoke的进阶版，但是思路是一致的，进阶之处在于不仅要改写返回地址，还要改写参数了。

当被调用函数执行完毕后，函数的局部变量和其他上下文信息会被清理出栈，接着，之前存储的返回地址会被弹出栈，pc寄存器接受弹出的返回地址，控制权会返回到调用函数的地方，即返回地址指向的指令处，继续执行。

那么，我们把getbuf函数的返回地址改为fizz函数的起始地址，然后特别注意：由于返回地址是要退栈的，我们填入的fizz起始地址会被弹出，在调用fizz函数后，fizz函数会把ebp的旧值压栈，原来fizz函数的起始地址会变成ebp的旧值，而fizz要调用的参数是现在的ebp+0x8。所以，我们在构造攻击字符串时，要在fizz的起始地址后空出4个字节，然后放入cookie，才能让fizz函数正确调用参数。（如上三图所示）

这里是一个易错点，之前我因为没有理解返回地址要退栈，所以不知道要在起始地址和cookie之间留出4字节，导致这题卡关了。

仍然用第一题的方式提交，攻击成功！

第三次攻击：bang

目标：让目标调用bang函数，并修改全局变量。

全局变量不是存放在栈中，而是存放在.bss和.data节中的，因此简单的数据覆盖是不能修改全局变量的。所以我们要编写一段恶意代码，用来达成修改全局变量的操作。

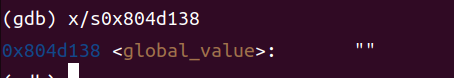
找出bang函数的汇编代码：

0x08048c55 <+0>: push %ebp

0x08048c56 <+1>: mov %esp,%ebp

0x08048c58 <+3>: sub $0x8,%esp

0x08048c5b <+6>: mov 0x804d138,%eax

 查看地址处存的值：

我们找到了全局变量的地址：0x804d138

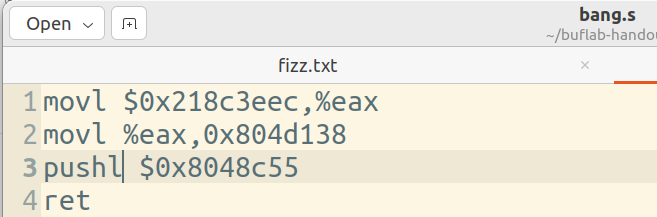
0x08048c60 <+11>: cmp 0x804d140,%eax

再查看一下这里的值：

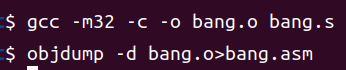
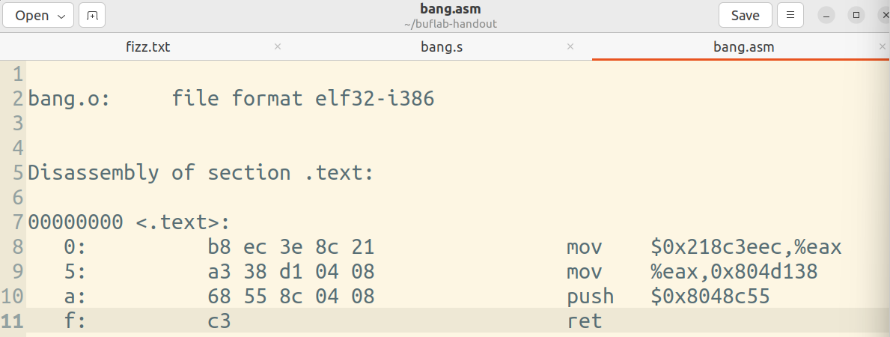
找到了cookie的x地址。

之后编写恶意代码，把cookie的值给全局变量的地址。

先用汇编写一个bang.s文件



前两步把cookie的值赋给全局变量，第三句把bang函数的返回地址压栈，好调用bang函数。

之后生成机器代码.o文件，然后反汇编.o文件得到机器代码

之后要得到getbuf缓冲区的首地址，对getbuf分析，

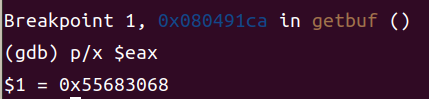
0x080491c1 <+0>: push %ebp

0x080491c2 <+1>: mov %esp,%ebp

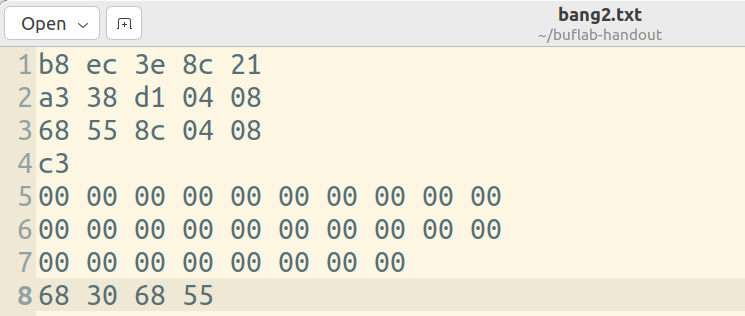
0x080491c4 <+3>: sub $0x34,%esp

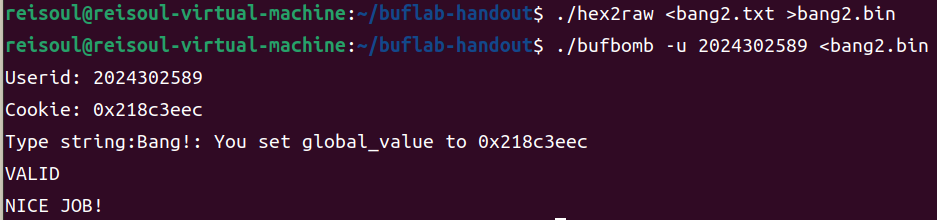
0x080491c7 <+6>: lea -0x28(%ebp),%eax

0x080491ca <+9>: push %eax

这一句将缓冲区首地址压栈，我们需要在这里查看eax寄存器的值，需要设断点运行到此处。

得到首地址。

 构造攻击字符串：

一共48字节，最后4个字节是小端存放的缓冲区首地址，用来跳转到恶意代码执行。

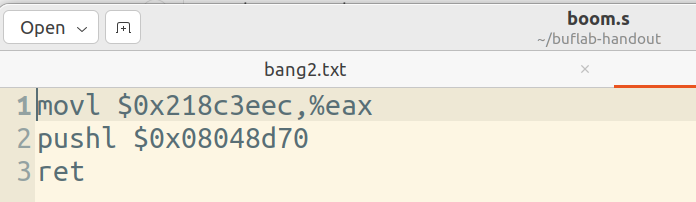
攻击成功！

第四次攻击：boom

目标：让目标程序返回test函数，但修改返回值为cookie

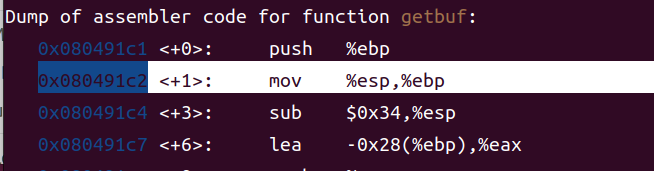
分析：之前几次都是破环了程序的执行顺序，这一次要保证攻击后的程序在表面上要正常运行。让我们偷梁换柱，修改返回值，进行隐秘的攻击。难度大大提高。

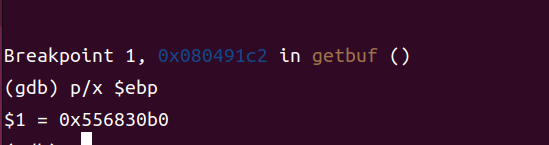
思路：test函数的栈帧空间不能被破坏，所以我们需要在getbuf调用后返回test函数，继续执行，但此时eax（返回值）要放入我们的cookie，那么我们把test的ebp存放在getbuf的对应位置，返回地址存放在缓冲区的首地址，攻击代码为修改eax的值为cookie并返回test。

编写攻击代码：

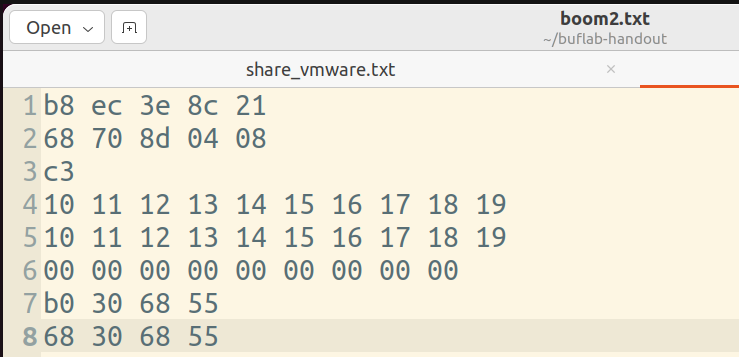
第一句把cookie赋给返回值。

第二句把test调用getbuf后的下一条指令地址压栈。

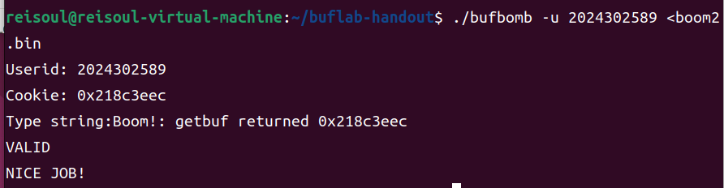
通过getbuf函数刚开始移动esp位置到ebp位置的操作，获得test中ebp的值：

需要在此处打断点然后运行到这里查看：

最后和第三次攻击同理，编写攻击字符串：

第7行时test中ebp的值；第8行是buf首地址。

保证函数调用完成后能进入恶意代码开始执行。

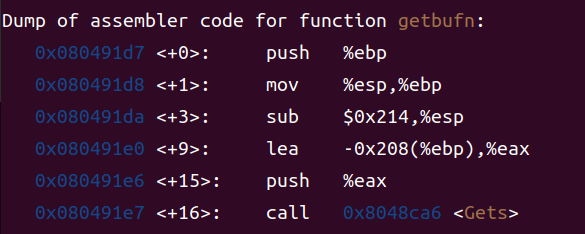
提交方法同理：

第四次攻击成功！

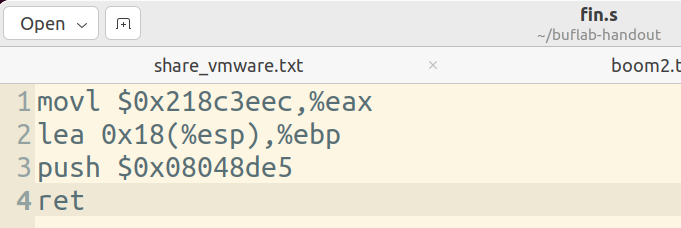
第五次攻击：Nitro

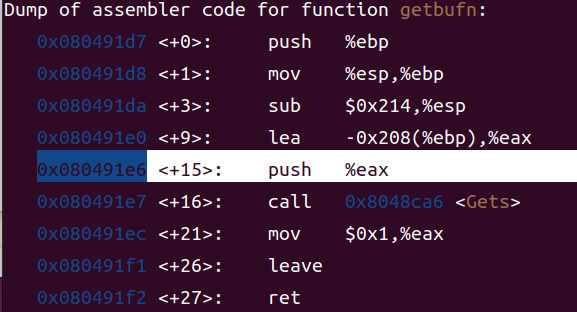
目标：用同一个攻击字符串，执行getbufn函数5次，使getbufn函数每次都把cookie值返回给testn函数。

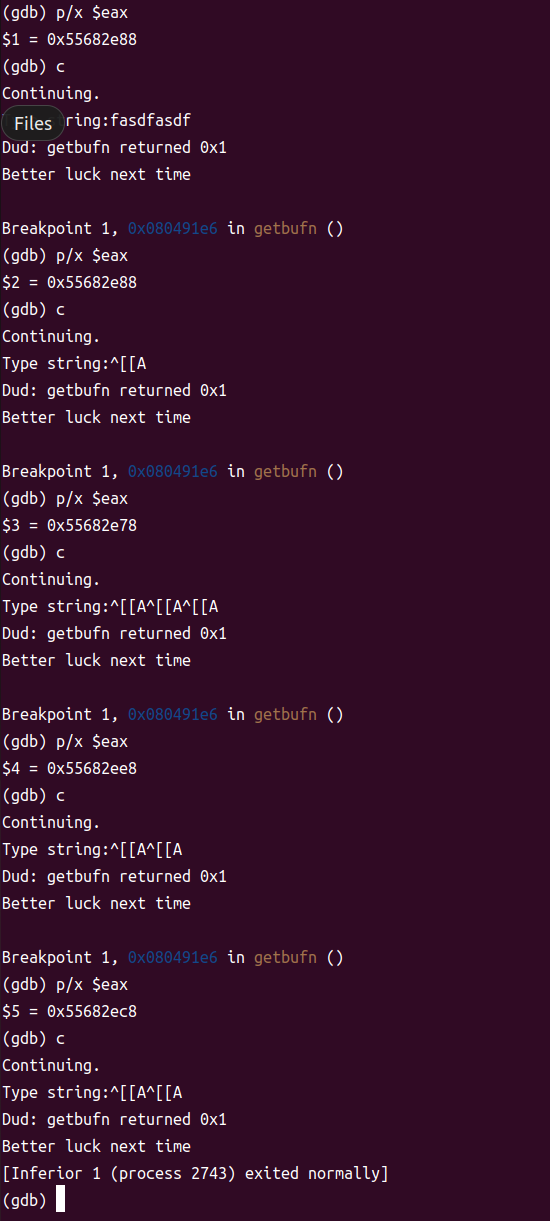
思路：第五次攻击是第四次攻击的进阶版本，在这次攻击中，要用一个攻击字符串实现多次攻击，难点在于getbufn每次循环的栈地址不同。Nitro阶段是为了模拟栈随机化。

为了实现这次攻击，老师给出了新的指令NOP sled指令（空操作指令，机器码为0x90）。把它放在目标代码前面，目的是不执行指令，一直滑到可执行指令处。

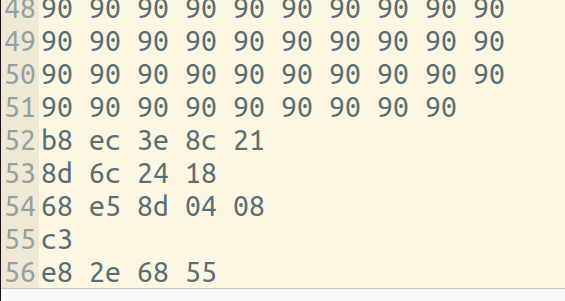
getbufn缓冲区有520个字节，加上ebp有524个字节。

然后构造恶意代码，将cookie传入eax,之后获取ebp地址，然后返回testn函数。

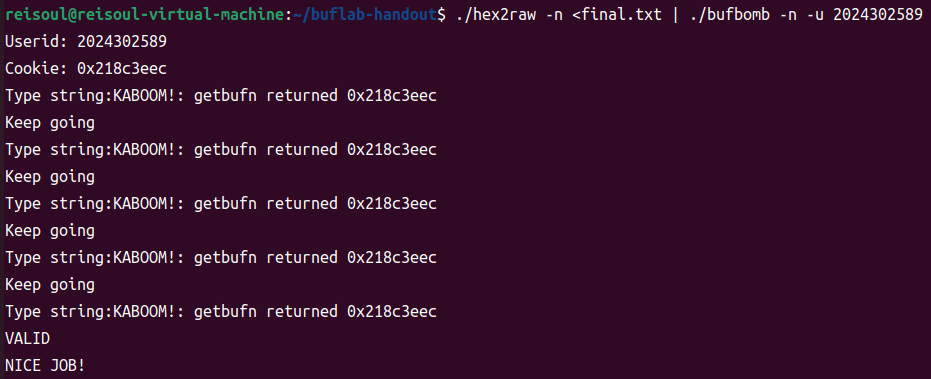
然后我们在buf缓冲区首地址设断点，运行getbufn函数n次，要在运行指令后加上-n。

断点设在这里，然后每次运行到这里查看eax的值，然后输入c继续运行，在让输入字符串的时候随便输入一些，然后重复以上操作，于是我们得到了四个地址，为了保证所有情况都能滑到正确的地址，我们要把最大的地址作为返回地址。

找到最大的地址为0x55682ee8，将其作为返回地址。

52~55行为恶意代码，前面是509个0x90.

注意在转换为二进制文件和提交时也要加上-n。



这里用管道的方法提交，攻击成功！

1. **实验总结:**

这次实验进一步让我了解了计算机中的函数调用规则和栈帧结构，从第一次攻击中简单的返回地址改写，到第二次攻击的改变参数，是对函数调用规则的考验。而第三次和第四次攻击的恶意代码的编写，让我打开了新的思路，更加深入理解了“计算机中信息的储存方式都是一样的二进制”这句话的理解。第五次攻击更是让我学习到了新的指令NOP的作用。除此以外，也学习到了多种gdb调试指令和多种文件格式，以及小端存放原则，让我对linux系统和计算机系统的理解进一步加深。