AMINE BADAOUI, LAB SÉCURITÉ INFORMATIQUE INF4420A, HIVER 2020

INTRODUCTION AU BUFFER OVERFLOW

SEGMENTATION DE LA MÉMOIRE D'UN PROCESSUS

Chaque processus en mémoire possède plusieurs sections, chacune d'elles contient des informations pour le fonctionnement du programme.

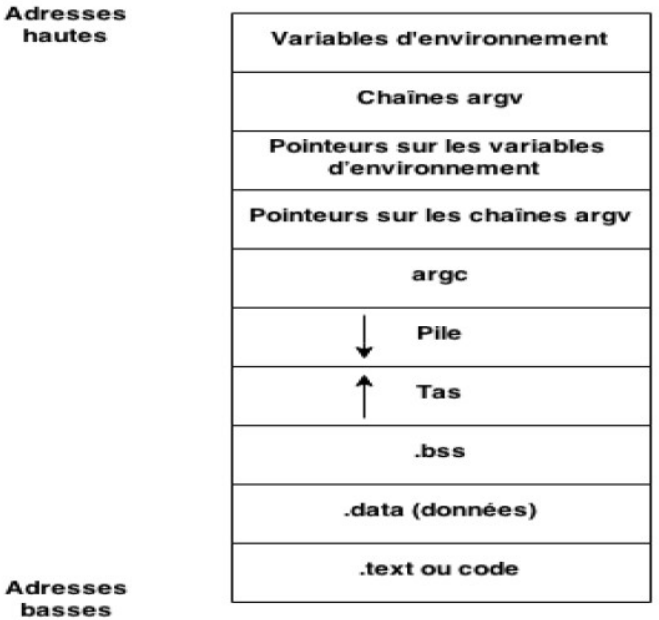
.**Segment text** contient les instructions du programme.

.Segment data contient les variables globales et statiques initialisées.

.Segment Bss contient les variables non initialisées.

.Segment Heap/Tas zone pour l'allocation dynamique de la mémoire.

.Segment Stack/Pile contexte des fonctions et leurs variables.



Organisation simplifiée de la mémoire d'un processus sous Linux

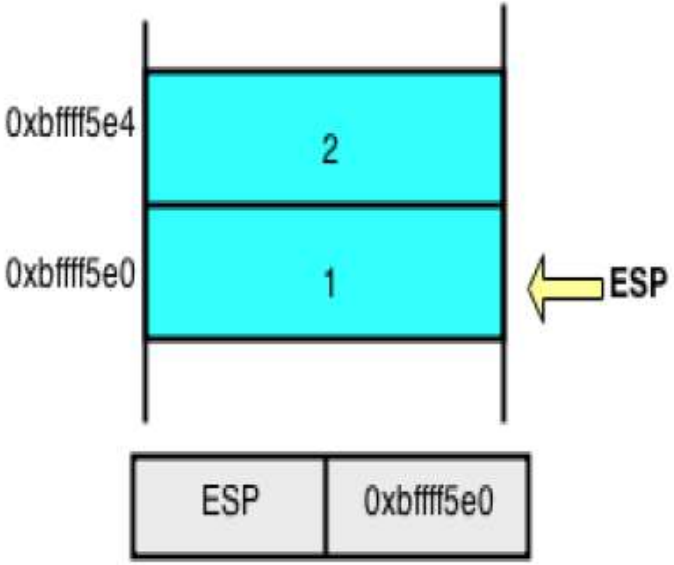
```c
#include <stdio.h>
#include <stdlib.h>

void f(int x, int y)
{   int local1=1;
    char local2[]="buffer";
    return
}
int main (int argc, char **argv)
{
    f(1,2);
    return EXIT_SUCCESS;
}
```

FONCTIONNEMENT DE LA PILE

APPEL D'UNE FONCTION

- Préparation des arguments pour la fonction f

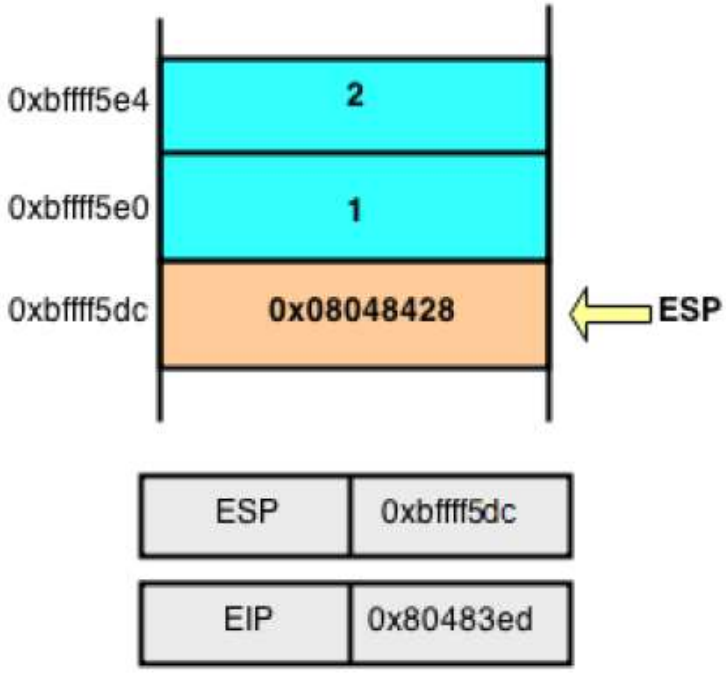
```
james@g1s:~/Desktop$ gdb -q fonction
Reading symbols from fonction...done.
(gdb) set disassembly-flavor intel
(gdb) disassemble main
Dump of assembler code for function main:
   0x0804840e <+0>:  push   ebp
   0x0804840f <+1>:  mov    ebp,esp
   0x08048411 <+3>:  sub    esp,0x8
   0x08048414 <+6>:  mov    DWORD PTR [esp+0x4],0x2
   0x0804841c <+14>: mov    DWORD PTR [esp],0x1
   0x08048423 <+21>: call   0x80483ed <f>
   0x08048428 <+26>: mov    eax,0x0
   0x0804842d <+31>: leave
   0x0804842e <+32>: ret
End of assembler dump.
```



APPEL D'UNE FONCTION

- Sauvegarde l'adresse de retour

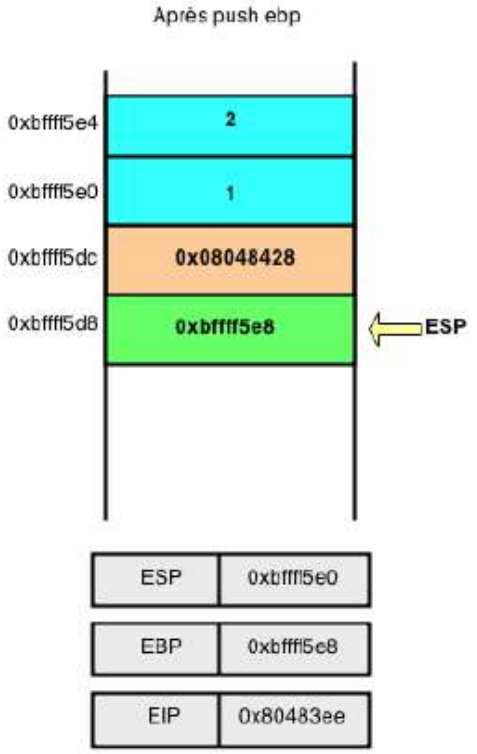
```
james@g1s:~/Desktop$ gdb -q fonction
Reading symbols from fonction...done.
(gdb) set disassembly-flavor intel
(gdb) disassemble main
Dump of assembler code for function main:
   0x0804840e <+0>:  push   ebp
   0x0804840f <+1>:  mov    ebp,esp
   0x08048411 <+3>:  sub    esp,0x8
   0x08048414 <+6>:  mov    DWORD PTR [esp+0x4],0x2
   0x0804841c <+14>: mov    DWORD PTR [esp],0x1
   0x08048423 <+21>: call   0x80483ed <f>
   0x08048428 <+26>: mov    eax,0x0
   0x0804842d <+31>: leave
   0x0804842e <+32>: ret
End of assembler dump.
```



APPEL D'UNE FONCTION, PROLOGUE

- Sauvegarde du stack frame

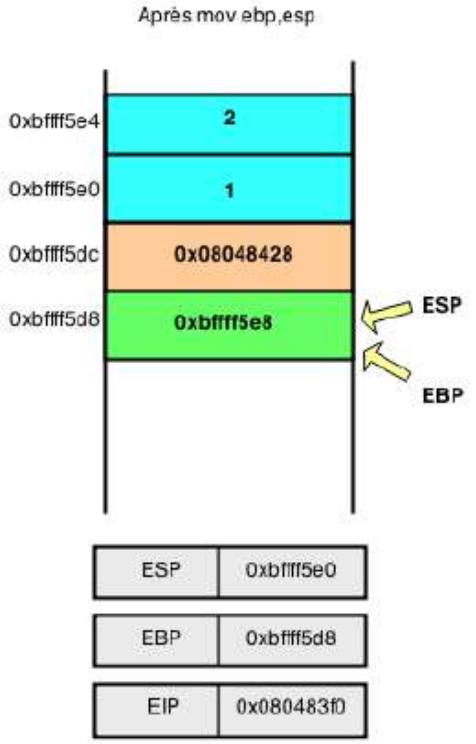
```
(gdb) disassemble f
Dump of assembler code for function f:
      0x080483ed <+0>:  push   ebp
      0x080483ee <+1>:  mov    ebp,esp
      0x080483f0 <+3>:  sub    esp,0x10
      0x080483f3 <+6>:  mov    DWORD PTR [ebp-0x4],0x1
      0x080483fa <+13>: mov    DWORD PTR [ebp-0xb],0x66667562
      0x08048401 <+20>: mov    WORD PTR [ebp-0x7],0x7265
      0x08048407 <+26>: mov    BYTE PTR [ebp-0x5],0x0
      0x0804840b <+30>: nop
      0x0804840c <+31>: leave
      0x0804840d <+32>: ret
End of assembler dump.
```



APPEL D'UNE FONCTION, PROLOGUE

- Sauvegarde du stack frame

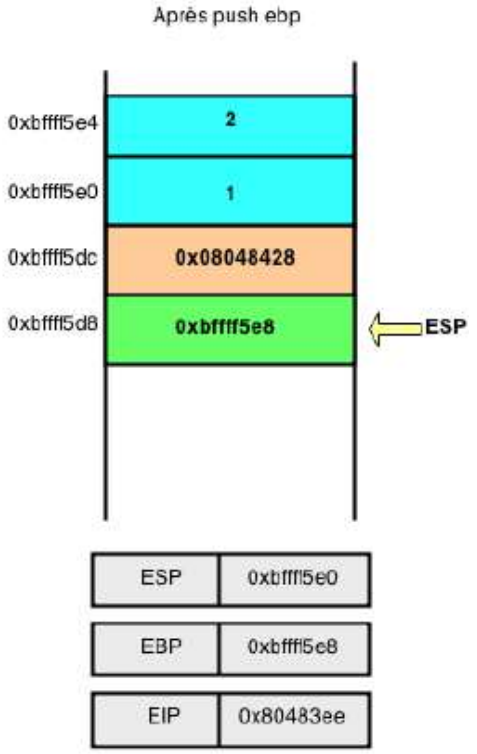
```
(gdb) disassemble f
Dump of assembler code for function f:
   0x080483ed <+0>:  push   ebp
   0x080483ee <+1>:  mov    ebp,esp
   0x080483f0 <+3>:  sub    esp,0x10
   0x080483f3 <+6>:  mov    DWORD PTR [ebp-0x4],0x1
   0x080483fa <+13>: mov    DWORD PTR [ebp-0xb],0x66667562
   0x08048401 <+20>: mov    WORD PTR [ebp-0x7],0x7265
   0x08048407 <+26>: mov    BYTE PTR [ebp-0x5],0x0
   0x0804840b <+30>: nop
   0x0804840c <+31>: leave
   0x0804840d <+32>: ret
End of assembler dump.
```



APPEL D'UNE FONCTION, PROLOGUE

- Sauvegarde du stack frame

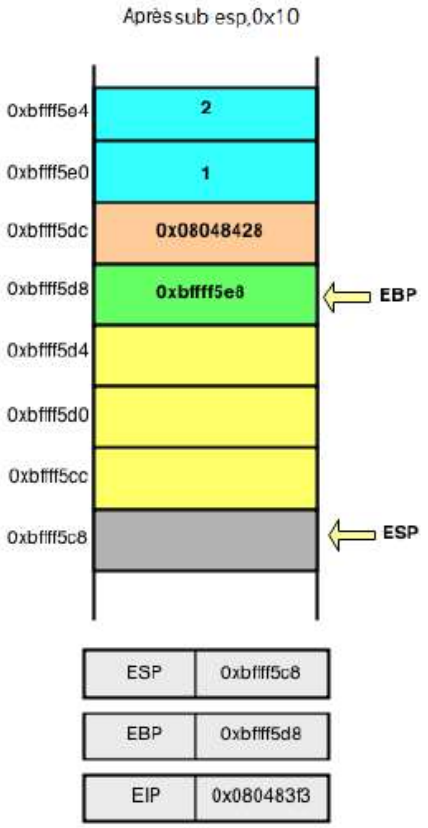
```
(gdb) disassemble f
Dump of assembler code for function f:
   0x080483ed <+0>:  push   ebp
   0x080483ee <+1>:  mov    ebp,esp
   0x080483f0 <+3>:  sub    esp,0x10
   0x080483f3 <+6>:  mov    DWORD PTR [ebp-0x4],0x1
   0x080483fa <+13>: mov    DWORD PTR [ebp-0xb],0x66667562
   0x08048401 <+20>: mov    WORD PTR [ebp-0x7],0x7265
   0x08048407 <+26>: mov    BYTE PTR [ebp-0x5],0x0
   0x0804840b <+30>: nop
   0x0804840c <+31>: leave
   0x0804840d <+32>: ret
End of assembler dump.
```



APPEL D'UNE FONCTION, PROLOGUE

- Allocation de l'espace pour ls variable locales de la fonction f

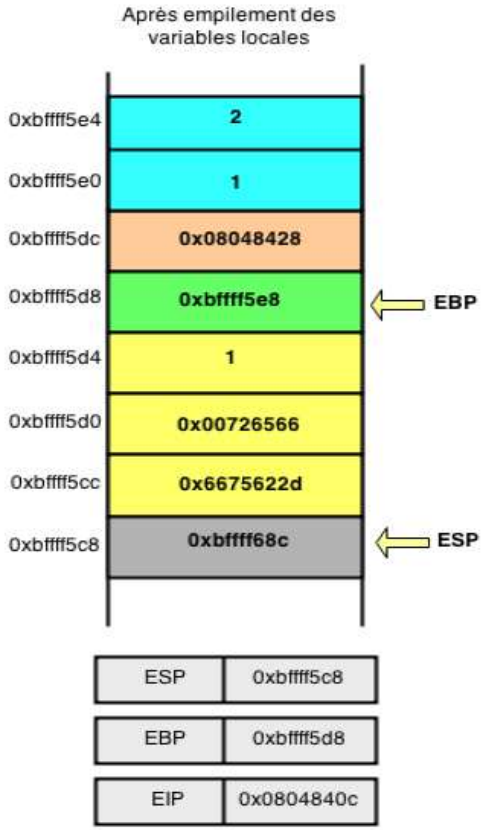
```
(gdb) disassemble f
Dump of assembler code for function f:
   0x080483ed <+0>:  push   ebp
   0x080483ee <+1>:  mov    ebp,esp
   0x080483f0 <+3>:  sub    esp,0x10
   0x080483f3 <+6>:  mov    DWORD PTR [ebp-0x4],0x1
   0x080483fa <+13>: mov    DWORD PTR [ebp-0xb],0x66667562
   0x08048401 <+20>: mov    WORD PTR [ebp-0x7],0x7265
   0x08048407 <+26>: mov    BYTE PTR [ebp-0x5],0x0
   0x0804840b <+30>: nop
   0x0804840c <+31>: leave
   0x0804840d <+32>: ret
End of assembler dump.
```



APPEL D'UNE FONCTION, PROLOGUE

- Allocation de l'espace pour ls variable locales de la fonction f

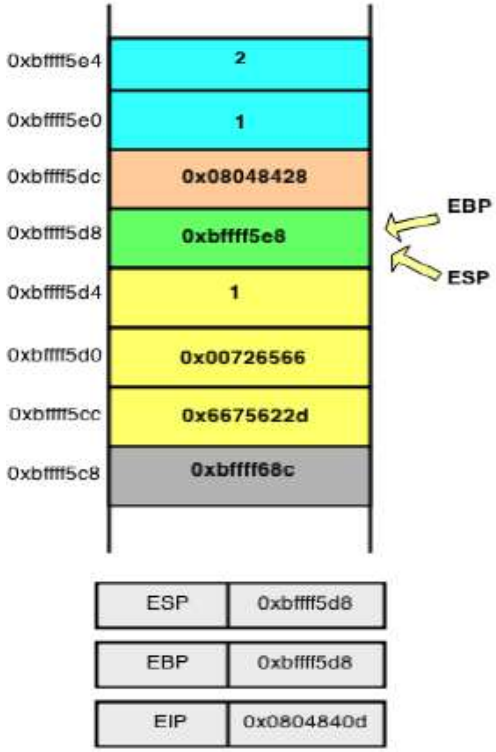
```
(gdb) disassemble f
Dump of assembler code for function f:
   0x080483ed <+0>:  push   ebp
   0x080483ee <+1>:  mov    ebp,esp
   0x080483f0 <+3>:  sub    esp,0x10
   0x080483f3 <+6>:  mov    DWORD PTR [ebp-0x4],0x1
   0x080483fa <+13>: mov    DWORD PTR [ebp-0xb],0x66667562
   0x08048401 <+20>: mov    WORD PTR [ebp-0x7],0x7265
   0x08048407 <+26>: mov    BYTE PTR [ebp-0x5],0x0
   0x0804840b <+30>: nop
   0x0804840c <+31>: leave
   0x0804840d <+32>: ret
End of assembler dump.
```



APPEL D'UNE FONCTION, EPILOGUE

- Quitter la fonction f, instruction ***leave***

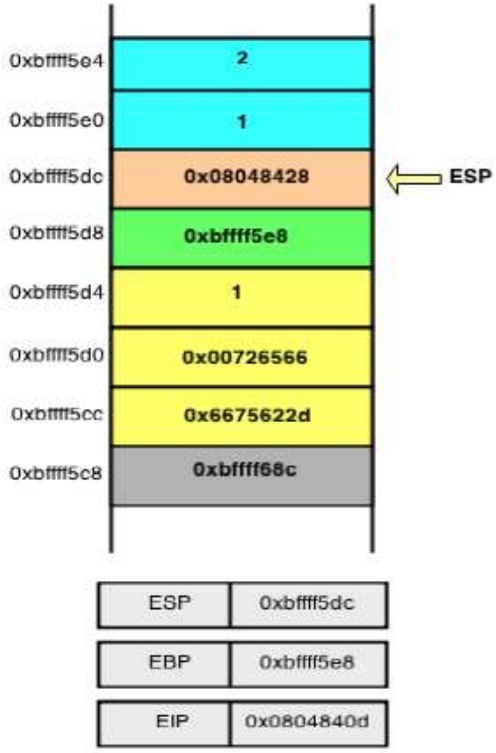
```
(gdb) disassemble f
Dump of assembler code for function f:
   0x080483ed <+0>:  push   ebp
   0x080483ee <+1>:  mov    ebp,esp
   0x080483f0 <+3>:  sub    esp,0x10
   0x080483f3 <+6>:  mov    DWORD PTR [ebp-0x4],0x1
   0x080483fa <+13>: mov    DWORD PTR [ebp-0xb],0x66667562
   0x08048401 <+20>: mov    WORD PTR [ebp-0x7],0x7265
   0x08048407 <+26>: mov    BYTE PTR [ebp-0x5],0x0
   0x0804840b <+30>: nop
   0x0804840c <+31>: leave
   0x0804840d <+32>: ret
End of assembler dump.
```



APPEL D'UNE FONCTION, PROLOGUE

- Quitter la fonction f, instruction ***leave***

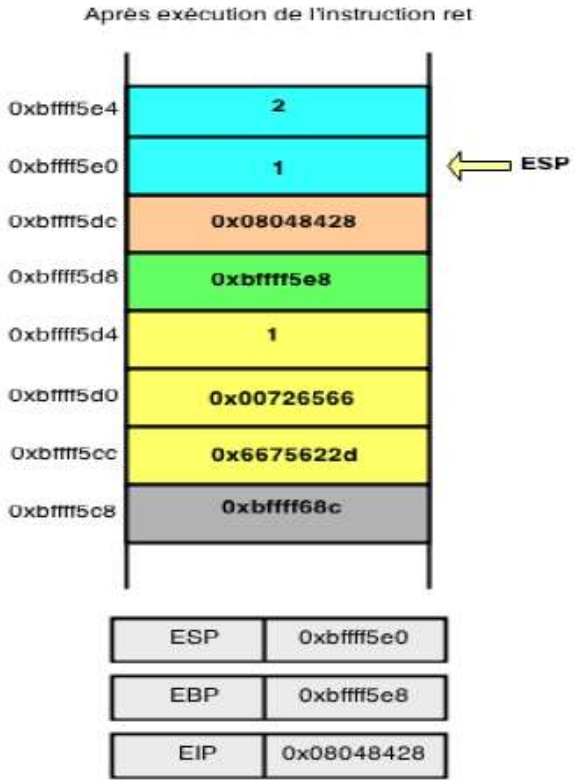
```
(gdb) disassemble f
Dump of assembler code for function f:
   0x080483ed <+0>:  push   ebp
   0x080483ee <+1>:  mov    ebp,esp
   0x080483f0 <+3>:  sub    esp,0x10
   0x080483f3 <+6>:  mov    DWORD PTR [ebp-0x4],0x1
   0x080483fa <+13>: mov    DWORD PTR [ebp-0xb],0x66667562
   0x08048401 <+20>: mov    WORD PTR [ebp-0x7],0x7265
   0x08048407 <+26>: mov    BYTE PTR [ebp-0x5],0x0
   0x0804840b <+30>: nop
   0x0804840c <+31>: leave
   0x0804840d <+32>: ret
End of assembler dump.
```



APPEL D'UNE FONCTION, EPILOGUE

- Retour à la fonction appelante, instruction **ret (pop eip)**

```
(gdb) disassemble f
Dump of assembler code for function f:
   0x080483ed <+0>:  push   ebp
   0x080483ee <+1>:  mov    ebp,esp
   0x080483f0 <+3>:  sub    esp,0x10
   0x080483f3 <+6>:  mov    DWORD PTR [ebp-0x4],0x1
   0x080483fa <+13>: mov    DWORD PTR [ebp-0xb],0x66667562
   0x08048401 <+20>: mov    WORD PTR [ebp-0x7],0x7265
   0x08048407 <+26>: mov    BYTE PTR [ebp-0x5],0x0
   0x0804840b <+30>: nop
   0x0804840c <+31>: leave
   0x0804840d <+32>: ret
End of assembler dump.
```

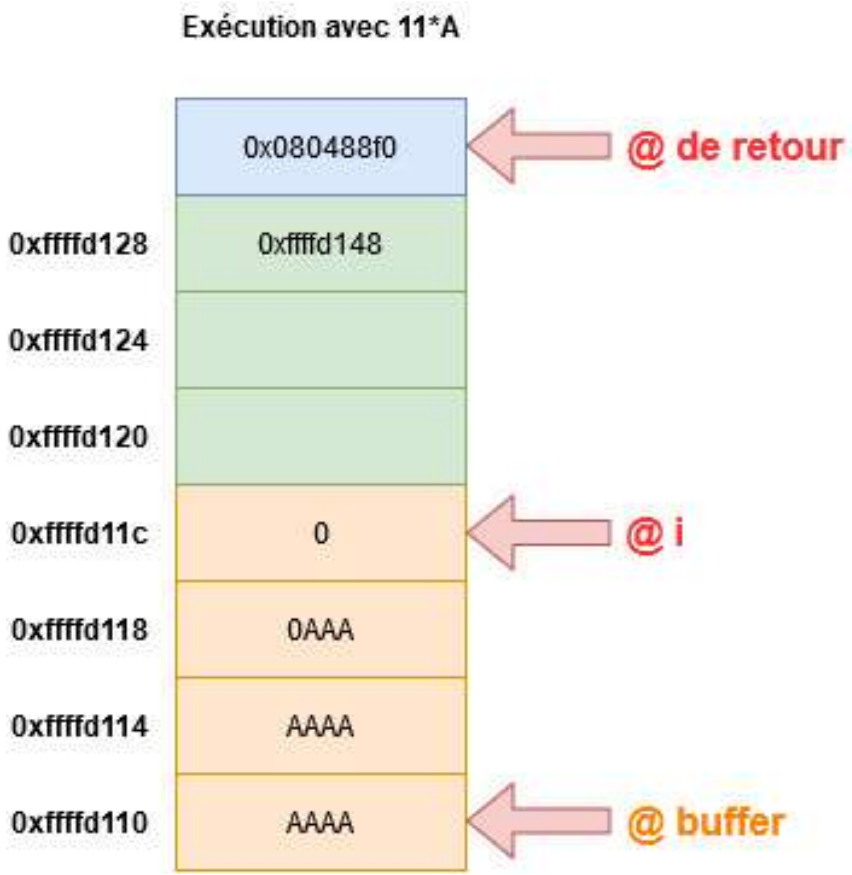


DÉBORDEMENT DE TAMPON DE PILE/ STACK OVERFLOW

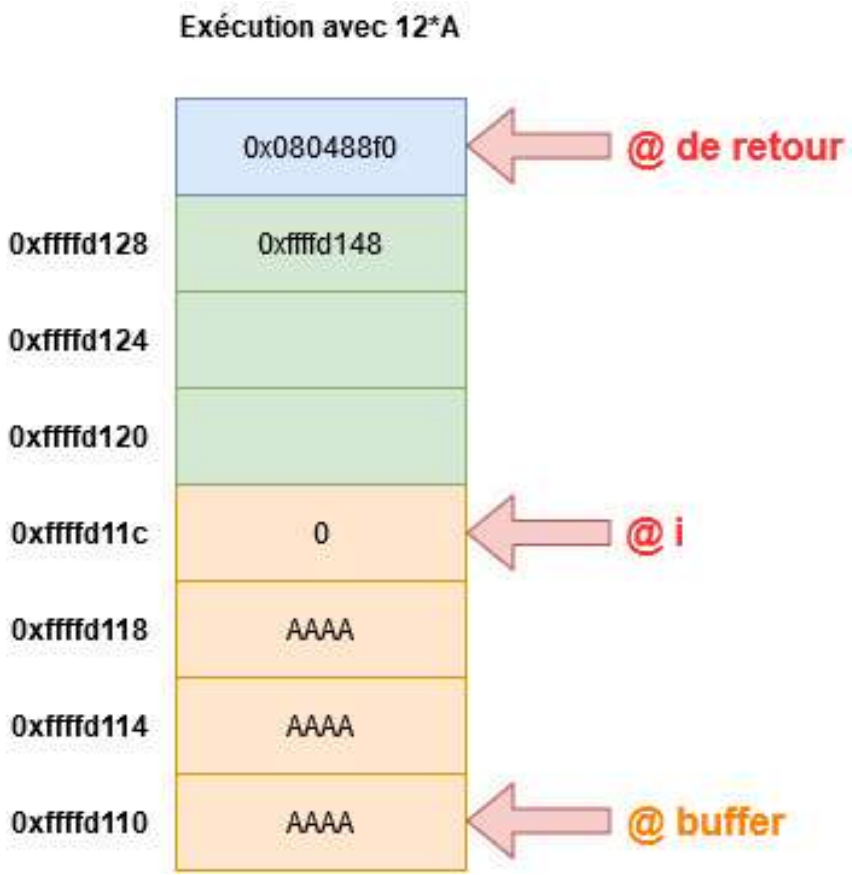
```c
#include <stdio.h>
#include <stdlib.h>

void vuln(char *arg)
{   int i=0;
    char buffer[12];
    strcpy(buffer, arg);
    if (i>0) printf("Cooooooool!");
}
int main (int argc, char **argc)
{
    if (argc <2) exit(0);
    vuln(argv[1]);
    exit(1);
}
```

Objectif; Exécuter printf("Cooooooool!");

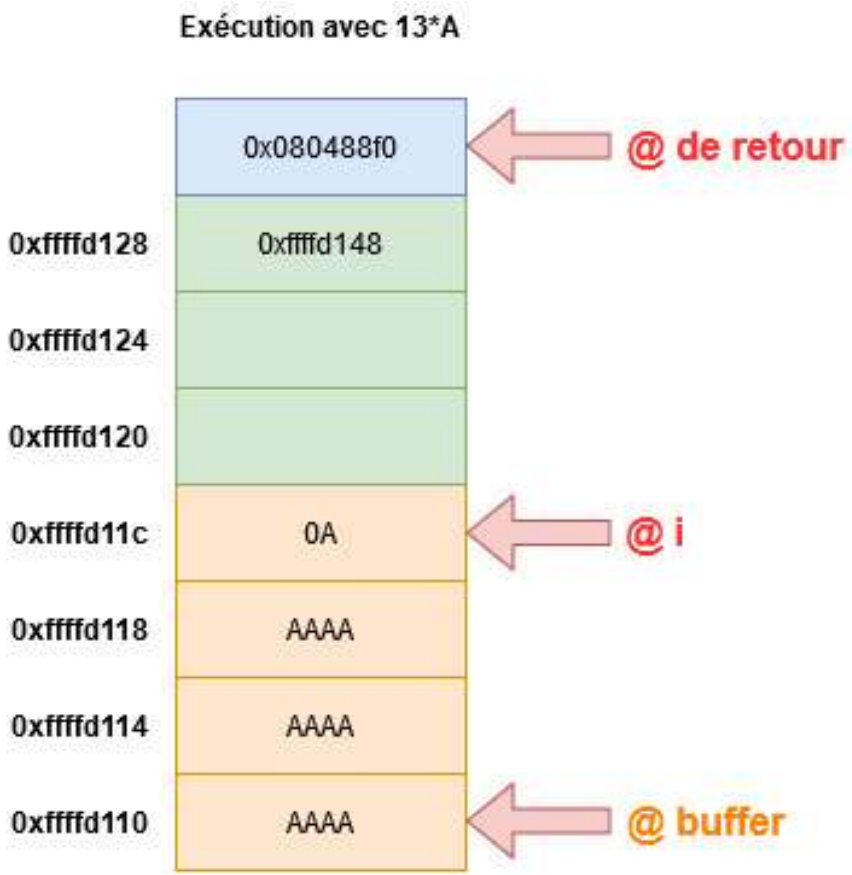
DÉBORDEMENT DE TAMPON DE PILE/ STACK OVERFLOW



DÉBORDEMENT DE TAMPON DE PILE/ STACK OVERFLOW



DÉBORDEMENT DE TAMPON DE PILE/ STACK OVERFLOW



Coooooooool !

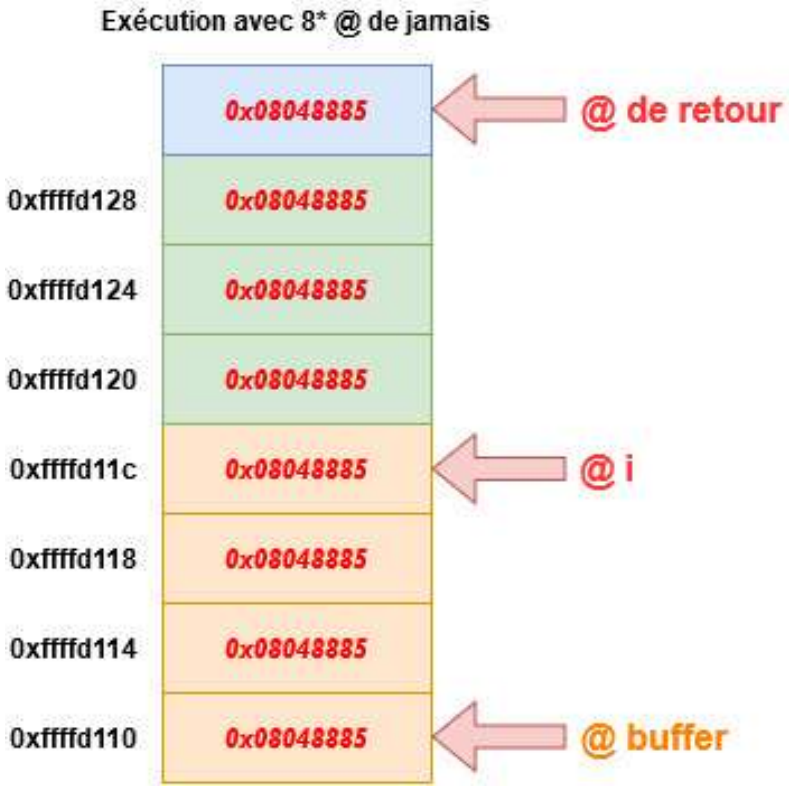
```c
#include <stdio.h>
#include <stdlib.h>

void jamais(){
    printf("Salut tu m'as appelé");
}
void vuln(char *arg)
{   int i=0;
    char buffer[12];
    strcpy(buffer, arg);
    if (i>0) printf("Coooooool!");
}
int main (int argc, char **argv)
{
    if (argc <2) exit(0);
    vuln(argv[1]);
    exit(1);
}
```

Objectif; Exécuter jamais()

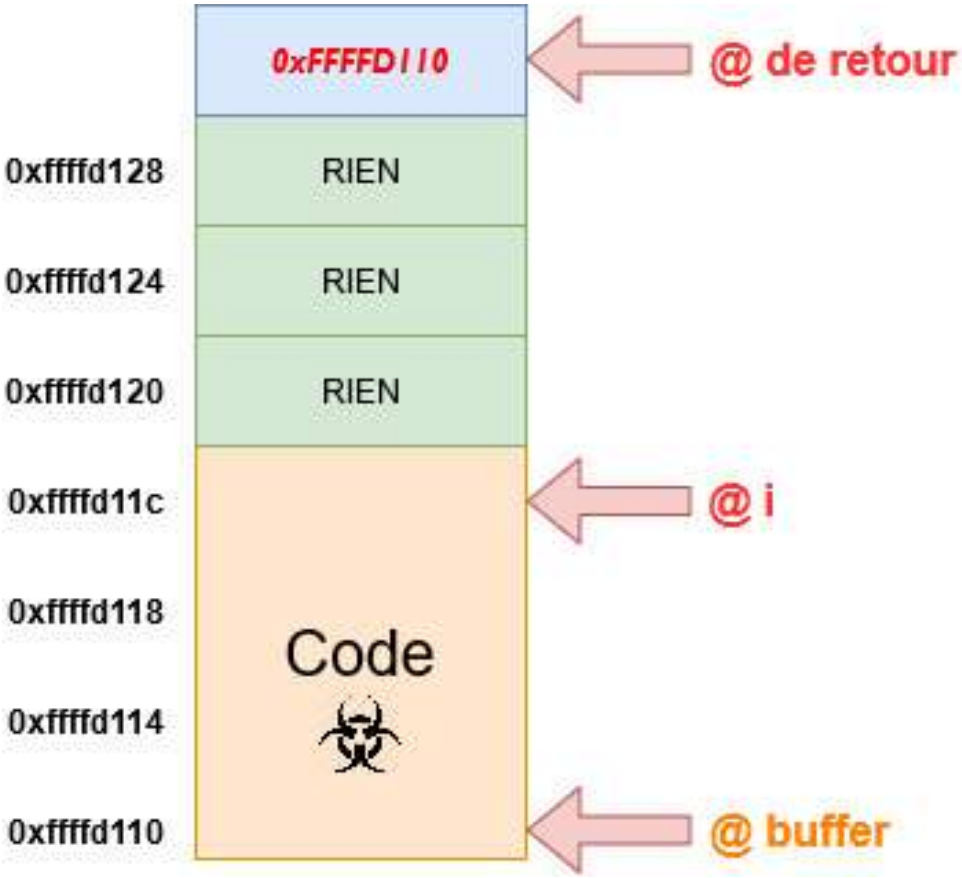
DÉBORDEMENT DE TAMPON DE PILE/ STACK OVERFLOW

DÉBORDEMENT DE TAMPON DE PILE/ STACK OVERFLOW

@ de jamais() 0x08048885



Salut tu m'as appelé



DÉBORDEMENT DE TAMPON DE PILE/ STACK OVERFLOW ET EXECUTION DE CODE