Rétro-ingénierie 101

NOP, NOP, NOP, NOP, NOP, NOP, NOP

Objectifs

- Assembleur 101
- Reverse des crackmes faciles

Assembleur 101

```
mov edi, 0
my_procedure:
        mov eax, 0x4
        mov ebx, 0x1
        mov ecx, my_string
        mov edx, 0x7
        int 0x80
        inc edi
        cmp edi,0xf
        jne my_procedure
        mov eax, 1
        int 0x80

my_string:
        dq 0x0a216f6c6c6568
```

Registres

Registres

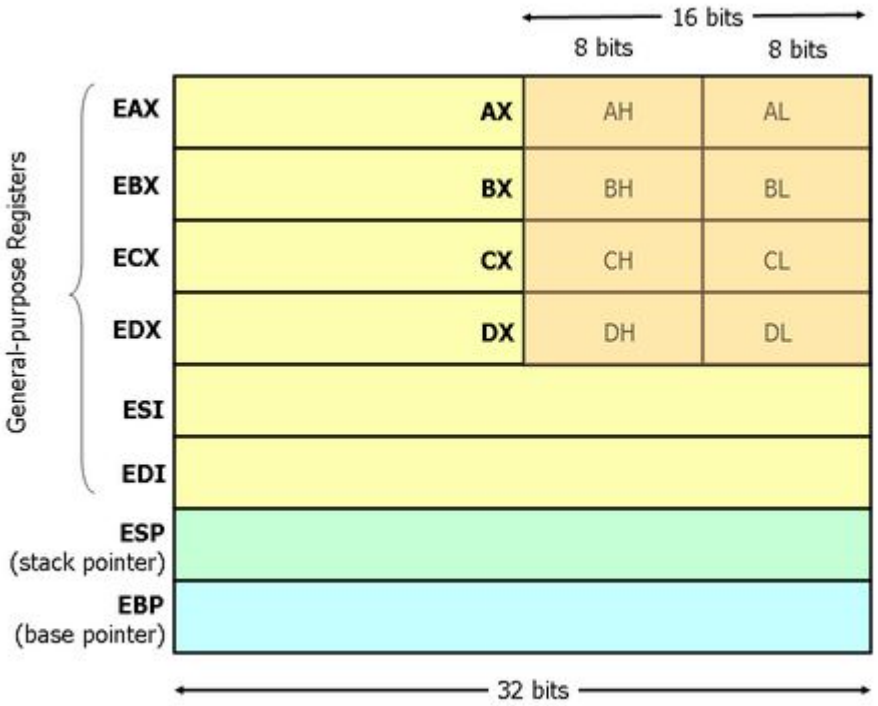
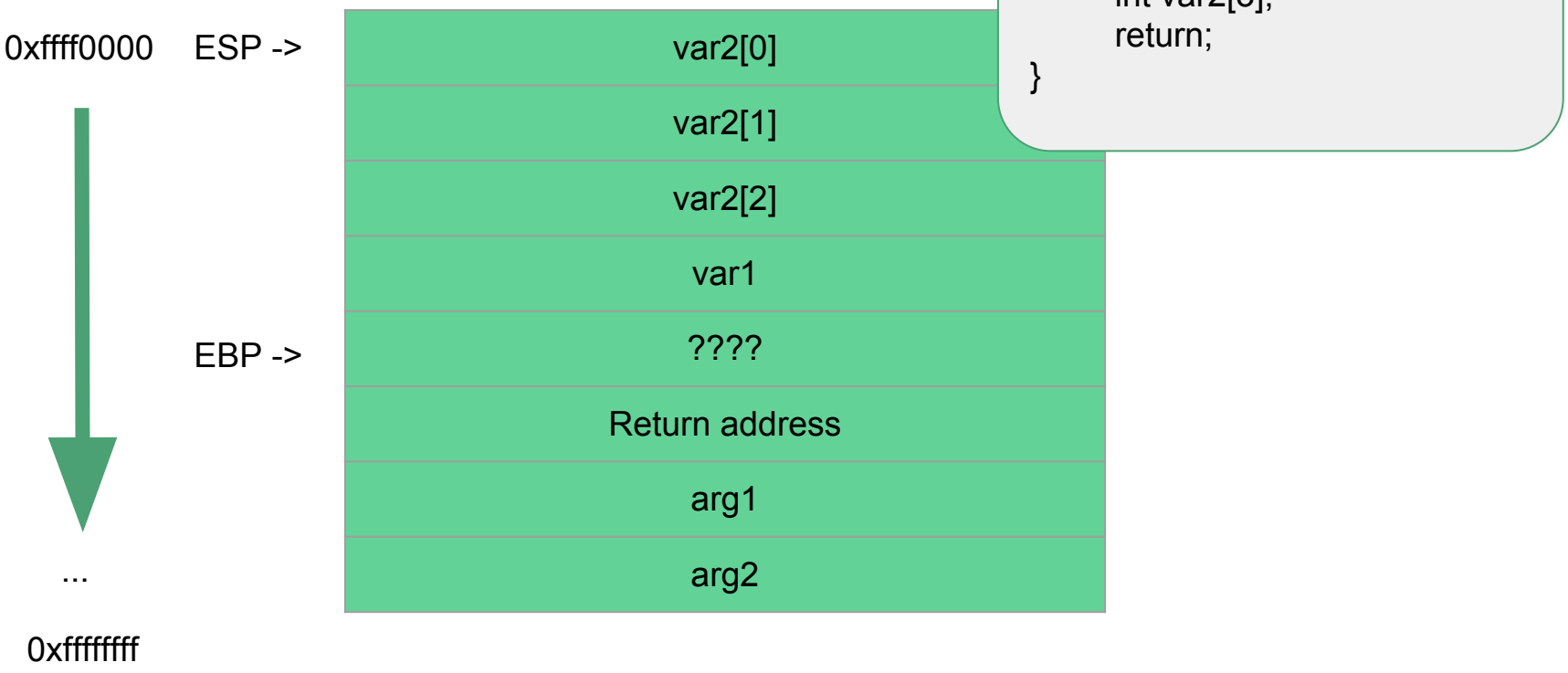


Figure 1. x86 Registers

Stack

Stack


0xffff0000
ESP ->
var2[0]
var2[1]
var2[2]
var1
EBP ->
????
Return address
arg1
arg2
...
0xffffffff
void do_stuff(int arg1, int arg2) {
int var1;
int var2[3];
return;
}

Instructions

Instructions

NOP

PUSH

POP

MOV

ADD, SUB, XOR

CALL

CMP, JNE, JMP

NOP

- Instruction vide
- Utilisé pour l’alignement
- NOP-sled en exploitation binaire

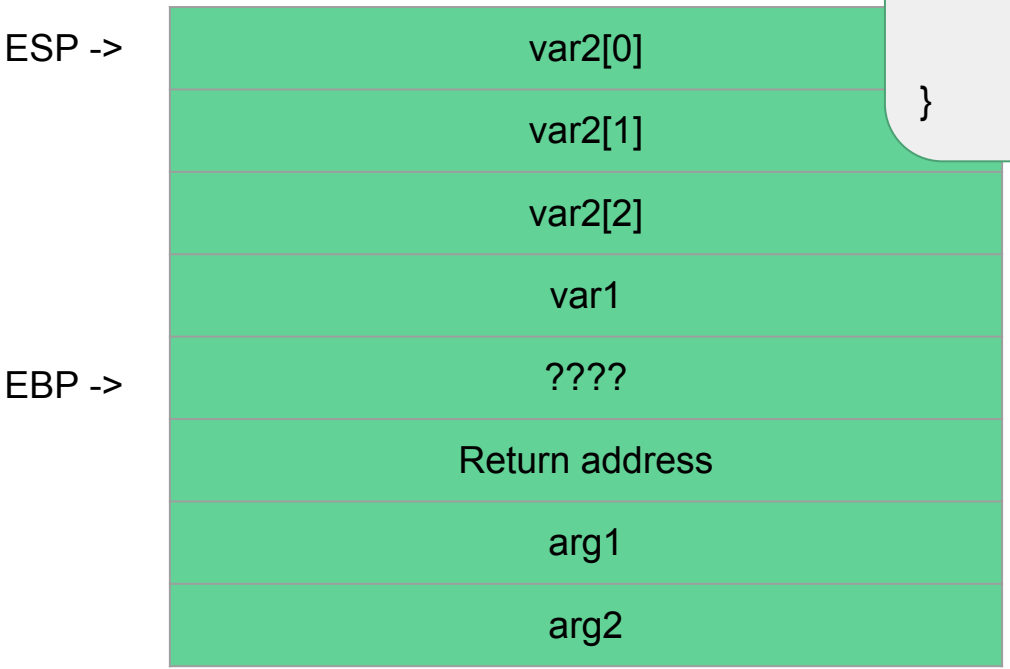
PUSH

- Empiler une valeur sur la stack

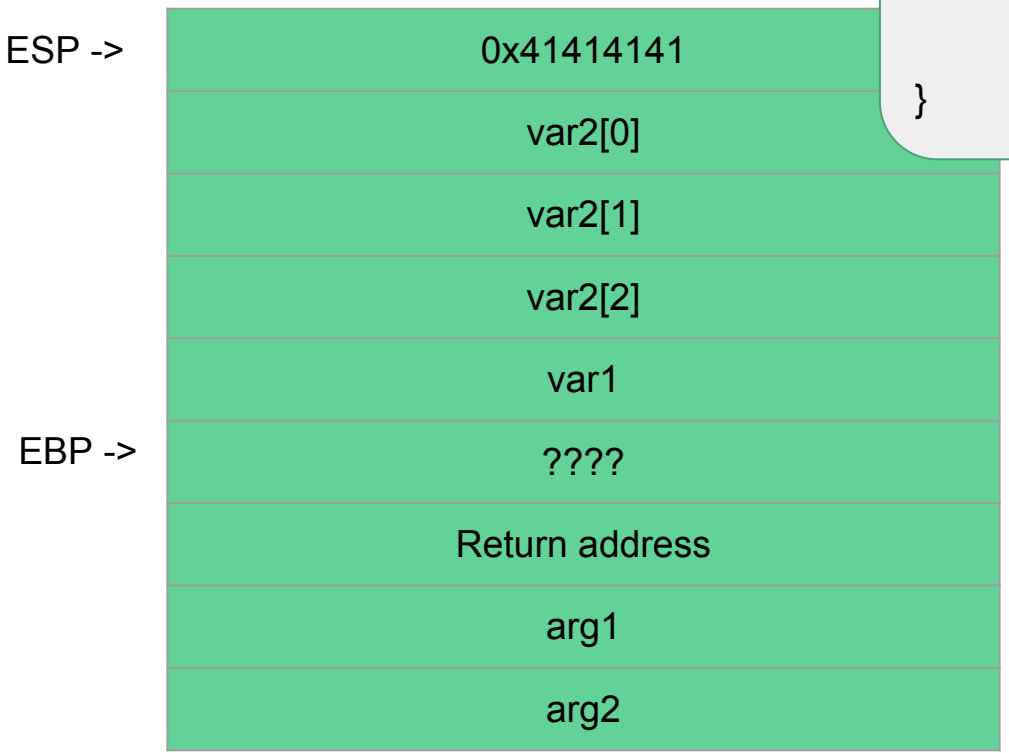
```
push 0x42424242
push ebx
```

Stack

```
void do_stuff(int arg1, int arg2) {
        int var1;
        int var2[3];
        asm("push 0x41414141");
        return;
}
```



Stack




```
void do_stuff(int arg1, int arg2) {
        int var1;
        int var2[3];
        asm("push 0x41414141");
        return;
}
```

POP

- Dépiler une valeur sur la stack et l’insérer dans un registre

```
pop ebx
```

```
main:
  push eax ⇐
  push ebx
  push ecx
  pop ebx
  pop eax
  pop ecx
```

Registres :
==========
eax => 0x41
ebx => 0x51
ecx => 0x61

Stack :
==========
0xffff0000: ????
0xffff0004: ????
0xffff0008: ????

```
main:
  push eax
  push ebx ⇐
  push ecx
  pop ebx
  pop eax
  pop ecx
```

Registres :
==========
eax => 0x41
ebx => 0x51
ecx => 0x61

Stack :
==========
0xfffefffc : 0x41
0xffff0000: ????
0xffff0004: ????
0xffff0008: ????

```
main:
  push eax
  push ebx
  push ecx ⇐
  pop ebx
  pop eax
  pop ecx
```

Registres :
==========
eax => 0x41
ebx => 0x51
ecx => 0x61

Stack :
==========
0xfffefff8 : 0x51
0xfffefffc : 0x41
0xffff0000: ????
0xffff0004: ????
0xffff0008: ????

```
main:
  push eax
  push ebx
  push ecx
  pop ebx ⇐
  pop eax
  pop ecx
```

Registres :
==========
eax => 0x41
ebx => 0x51
ecx => 0x61

Stack :
==========
0xfffefff4 : 0x61
0xfffefff8 : 0x51
0xfffefffc : 0x41
0xffff0000: ????
0xffff0004: ????
0xffff0008: ????

```
main:
  push eax
  push ebx
  push ecx
  pop ebx
  pop eax ⇐
  pop ecx
```

Registres :
==========
eax => 0x41
ebx => ~~0x51~~ 0x61
ecx => 0x61

Stack :
==========
0xfffefff8 : 0x51
0xfffefffc : 0x41
0xffff0000: ????
0xffff0004: ????
0xffff0008: ????

```
main:
  push eax
  push ebx
  push ecx
  pop ebx
  pop eax
  pop ecx ⇐
```

Registres :
==========
eax => ~~0x41~~ 0x51
ebx => ~~0x51~~ 0x61
ecx => 0x61

Stack :
==========
0xfffefffc : 0x41
0xffff0000: ????
0xffff0004: ????
0xffff0008: ????

```
main:
  push eax
  push ebx
  push ecx
  pop ebx
  pop eax
  pop ecx
```

Registres :
==========
eax => ~~0x41~~ 0x51
ebx => ~~0x51~~ 0x61
ecx => ~~0x61~~ 0x41

Stack :
==========
0xffff0000: ????
0xffff0004: ????
0xffff0008: ????

MOV

- Assigner une valeur dans un registre

```
mov ebx, 0x23
mov [ebx], 0x23
mov ebx, eax
mov [ebx], eax
mov eax, [edx]
```

```
main:
  push eax ⇐
  mov ebx, esp
  mov ecx, [esp]
```

Registres :
==========
eax => 0x51
ebx => 0x61
ecx => 0x41
esp => 0xffff0000

Stack :
==========
0xffff0000: ????
0xffff0004: ????
0xffff0008: ????

```
main:
  push eax
  mov ebx, esp ⇐
  mov ecx, [esp]
```

Registres :
==========
eax => 0x51
ebx => 0x61
ecx => 0x41
esp => 0xfffefffc

Stack :
==========
0xfffefffc: 0x51
0xffff0000: ????
0xffff0004: ????
0xffff0008: ????

```
main:
  push eax
  mov ebx, esp
  mov ecx, [esp] ⇐
```

Registres :
==========
eax => 0x51
ebx => 0xfffefffc
ecx => 0x41
esp => 0xfffefffc

Stack :
==========
0xfffefffc: 0x51
0xffff0000: ????
0xffff0004: ????
0xffff0008: ????

```
main:
  push eax
  mov ebx, esp
  mov ecx, [esp]
```

Registres :
==========
eax => 0x51
ebx => 0xfffefffc
ecx => 0x51
esp => 0xfffefffc

Stack :
==========
0xfffefffc: 0x51
0xffff0000: ????
0xffff0004: ????
0xffff0008: ????

ADD & SUB & XOR

- Ajouter ou soustraire deux registres

```
add ebx, ecx
sub ebx, ecx
sub ebx, 0x23
add [ebx], 0x23
xor eax, eax
```

CMP

- Comparer un registre à une autre valeur

```
cmp eax, 0xf
cmp ecx, ebx
```

JNE/JB/JA/JMP

- JNE == Jump Not Equal
  JB = Jump Below
  JA = Jump Above
- Souvent utilisé à la suite d'un CMP

```
cmp  eax, 0x23
jne 0x12131415
```

Ce qu’on connaît maintenant

Registres généraux : eax, ebx, ecx, edx, esi, edi
Registres pointant vers la stack : esp, ebp

Stack => Variables locaux et arguments

Plusieurs instructions manipulent la stack et les registres

Procedures et Syscalls

Procedure == un ensemble d’instructions

CALL

- Appeler une procédure

```
call do_stuff
call eax
call [eax]

do_stuff:
    mov eax, ebx
    ...
```

System Calls == Interface entre l’application et le kernel

System Calls

Les registres ont maintenant une signification :

eax => le syscall à exécuter
ebx => argument 1
ecx => argument 2
edx => argument 3
esi => argument 4
edi => argument 5

System calls - write()

Signature : write(int fd, const void *buf, size_t count);

```
mov eax, 4
mov ebx, 1
mov ecx, ma_string
mov edx, 4
int 0x80
ma_string:
    dd 0x41414141
```

```
mov edi, 0
my_procedure:
        mov eax, 0x4
        mov ebx, 0x1
        mov ecx, my_string
        mov edx, 0x7
        int 0x80
        inc edi
        cmp edi,0xf
        jne my_procedure
        mov eax, 1
        int 0x80

my_string:
        dq 0x0a216f6c6c6568
```

Démo + Hopper