二、逆向初探

❖从一个简单的程序说起

```
; int __cdecl main(int argc, const char **argv, const char **envp)
                public _main
_main           proc near               ; CODE XREF: ___mingw_CRTStartup

argc            = dword ptr  8
argv            = dword ptr  0Ch
envp            = dword ptr  10h

                push    ebp
                mov     ebp, esp
                and     esp, 0FFFFFFF0h ; Logical AND
                sub     esp, 20h        ; Integer Subtraction
                call    ___main         ; Call Procedure
                mov     dword ptr [esp+1Ch], 5
                lea     eax, [esp+18h]  ; Load Effective Address
                mov     [esp+4], eax
                mov     dword ptr [esp], offset aD ; "%d"
                call    _scanf          ; Call Procedure
                mov     eax, [esp+18h]
                test    eax, eax        ; Logical Compare
                jnz     short loc_40137A ; Jump if Not Zero (ZF=0)
                mov     dword ptr [esp+18h], 8

loc_40137A:                             ; CODE XREF: _main+30↑j
                mov     edx, [esp+18h]
                mov     eax, [esp+1Ch]
                add     eax, edx        ; Add
                leave                   ; High Level Procedure Exit
                retn                    ; Return Near from Procedure
_main           endp
```

```c
#include <stdio.h>

int main(void)
{
    int a, b = 5;
    scanf("%d", &a);

    if(a == 0)
        a = 8;

    return a + b;
}
```

```
; int __cdecl main(int argc, const char **argv, const char **envp)
                public _main
_main           proc near               ; CODE XREF: ___mingw_CRTStartup

argc            = dword ptr  8
argv            = dword ptr  0Ch
envp            = dword ptr  10h

                push    ebp
                mov     ebp, esp
                and     esp, 0FFFFFFF0h ; Logical AND
                sub     esp, 20h        ; Integer Subtraction
                call    ___main         ; Call Procedure
                mov     dword ptr [esp+1Ch], 5
                lea     eax, [esp+18h]  ; Load Effective Address
                mov     [esp+4], eax
                mov     dword ptr [esp], offset aD ; "%d"
                call    _scanf          ; Call Procedure
                mov     eax, [esp+18h]
                test    eax, eax        ; Logical Compare
                jnz     short loc_40137A ; Jump if Not Zero (ZF=0)
                mov     dword ptr [esp+18h], 8

loc_40137A:                             ; CODE XREF: _main+30↑j
                mov     edx, [esp+18h]
                mov     eax, [esp+1Ch]
                add     eax, edx        ; Add
                leave                   ; High Level Procedure Exit
                retn                    ; Return Near from Procedure
_main           endp
```

- 汇编语言（assembly language）
  - 是一种用于电子计算机、微处理器、微控制器或其他可编程器件的低级语言，亦称为符号语言
  - 在汇编语言中，用助记符代替机器指令的操作码，用地址符号或标号代替指令或操作数的地址
  - 在不同的设备中，汇编语言对应着不同的机器语言指令集，通过汇编过程转换成机器指令
  - 普遍地说，特定的汇编语言和特定的机器语言指令集是一一对应的，不同平台之间不可直接移植

汇编语言

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- 汇编语言的主体是汇编指令
- 汇编指令和机器指令的差别在于指令的表示方法上
- 汇编指令是机器指令便于记忆的书写格式
- 汇编指令是机器指令的助记符

- 机器指令： 1000100111011000
- 操作：寄存器BX的内容送到AX中
- 汇编指令： MOV AX, BX
- 这样的写法与人类语言接近，便于阅读和记忆

常用寄存器

- EAX： 累加器，在加法、乘法指令中用到的寄存器，或存放函数返回值
- EBX： 基地址寄存器，在内存寻址时存放基地址
- ECX： 计数器，在循环中一般会使用
- EDX： 存放整数除法产生的余数
- ESI/EDI： 源/目标索引寄存器，在很多字符串操作中ESI指向源，EDI指向目标
- EBP： 基址指针，一般用来存放函数的起始地址
- ESP： 始终指向栈顶
- EIP： 存放下条指令的地址

常见的汇编指令

- 传送指令（4个）：mov、push、pop、lea
- 转移指令（8个）：call、jmp、je、jne、jb、jnb、ja、jna
- 运算指令（7个）：add、sub、mul、div、adc、sbb、cmp
- 处理机控制指令（1个）：nop

常见的汇编指令

- Add eax,ecx　eax寄存器的值加上ecx寄存器的值，结果保存在eax寄存器
- Sub eax,ecx　eax寄存器的值减去ecx寄存器的值，结果保存在eax寄存器
- cmp eax,ebx　eax寄存器的值与ebx寄存器的值比较，如相等z标志置1否则置0
- Jnz eax　Jnz不为0时跳转，即z标志为0时跳转到eax表示的地址处继续执行
- Call eax　先将下条指令的地址压栈，再跳转到eax表示的地址处执行
- mov eax,ecx　将ecx寄存器的值保存在eax寄存器中
- Ret　将当前栈顶的值取出，存放到EIP中，并继续执行

Instruction	Effect	Examples
Copying Data		
mov *dest,src*	Copy src to dest	mov eax,10 mov eax,[2000]
Arithmetic		
add *dest,src*	dest = dest + src	add esi,10
sub *dest,src*	dest = dest – src	sub eax, ebx
mul *reg*	edx:eax = eax * reg	mul esi
div *reg*	edx = edx:eax **mod** reg eax = edx:eax ÷ reg	div edi
inc *dest*	Increment destination	inc eax
dec *dest*	Decrement destination	dec word [0x1000]
Function Calls		
call *label*	Push eip, transfer control	call format_disk
ret	Pop eip and return	ret
push *item*	Push item (constant or register) to stack. I.e.: esp=esp-4; memory[esp] = item	push dword 32 push eax
pop *[reg]*	Pop item from stack and store to register I.e.: reg=memory[esp]; esp=esp+4	pop eax
Bitwise Operations		
and *dest, src*	dest = src & dest	and ebx, eax
or *dest,src*	dest = src | dest	or eax,[0x2000]
xor *dest, src*	dest = src ^ dest	xor ebx, 0xffffffff
shl *dest,count*	dest = dest << count	shl eax, 2
shr *dest,count*	dest = dest >> count	shr dword [eax],4

Instruction	Effect	Examples
Conditionals and Jumps		
cmp *b,a*	Compare b to a; must immediately precede any of the conditional jump instructions	cmp eax,0
je *label*	Jump to label if b == a	je endloop
jne *label*	Jump to label if b != a	jne loopstart
jg *label*	Jump to label if b > a	jg exit
jge *label*	Jump to label if b ≥ a	jge format_disk
jl *label*	Jump to label if b < a	jl error
jle *label*	Jump to label if b ≤ a	jle finish
test *reg,imm*	Bitwise compare of register and constant; should immediately precede the jz or jnz instructions	test eax,0xffff
jz *label*	Jump to label if bits were **not** set ("zero")	jz looparound
jnz *label*	Jump to label if bits **were** set ("not zero")	jnz error
jmp *label*	Unconditional relative jump	jmp exit
jmp *reg*	Unconditional absolute jump; arg is a register	jmp eax
Miscellaneous		
nop	No-op (opcode 0x90)	nop
hlt	Halt the CPU	hlt

Instructions with no memory references must include 'byte', 'word' or 'dword' size specifier.
Arguments to instructions: Note that it is not possible for **both** src and dest to be memory addresses.

Constant (decimal or hex):	10 or 0xff	Fixed address:	[200] or [0x1000+53]
Register:	eax bl	Dynamic address:	[eax] or [esp+16]

32-bit registers: eax, ebx, ecx, edx, esi, edi, ebp, esp (points to first used location on top of stack)
16-bit registers: ax, bx, cx, dx, si, di, sp, bp
8-bit registers: al, ah, bl, bh, cl, ch, dl, dh