



汇编语言与逆向工程

北京邮电大学
2019年3月

北邮网安学院 崔宝江



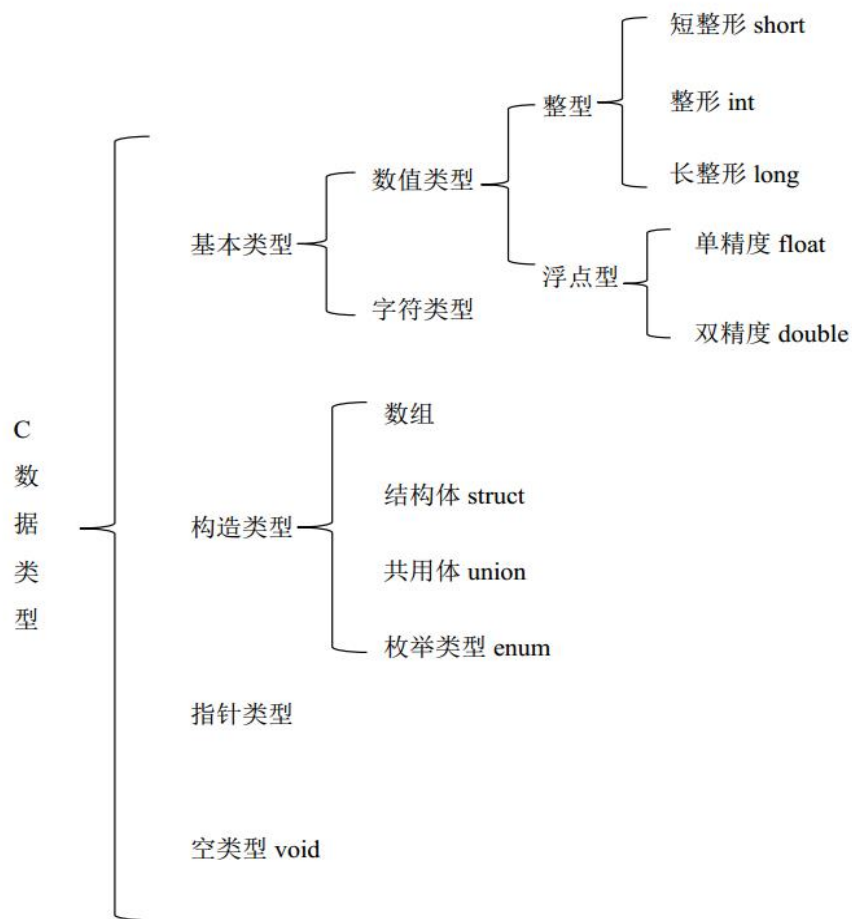
第三章从C语言看汇编

- ① 一. 基本数据类型
- ② 二. 流程控制语句
- ③ 三. 变量表现形式



一. 基本数据类型

@ C语言基本的数据类型



一. 基本数据类型

- 本小节介绍基本类型和指针类型在汇编中的表现形式
- 构造类型将在下一章介绍
- 空类型是需要转化为其他形式的类型



一. 基本数据类型

- 内存中任何类型的变量，都以字节的形式存储和运行于内存中
 - 一个字节也就是由8个二进制数组成
 - 一个十六进制数可用4个二进制数表示
 - 一个字节由两个十六进制数表示
 - \windows中，一个字等于两个字节
 - 32位程序和64位程序，只需要记住计算机在处理时只对8字节，4字节，2字节和1字节进行处理
 - ❖ 计算机的处理“只看字节不看类型”



一. 基本数据类型

- 用一个C语言和汇编对比的例子，从汇编看C语言基本数据类型
 - 定义了基本类型的数组，使用相应的指针指向这些数组
 - 通过指针自加的方式，让大家认识基本类型在汇编中的表现形式



一. 基本数据类型

```
Int main(int argc,char *argv[])
{
    /* define array */
    short shortValue[4]={100,};
    int intValue[4]={200,};
    long longValue[4]={300,};
    float floatValue[4]={400.1,};
    double doubleValue[4]={500.02,};
    char charValue[4]={48,};
    /* define array */
    int i;
    /* define points */
    short *pShortValue=shortValue;
    int *pIntValue=intValue;
    long *pLongValue=longValue;
    float *pFloatValue=floatValue;
    double *pDoubleValue=doubleValue;
    char *pCharValue=charValue;
    /* define points */
```



一. 基本数据类型

```
/* show point in the memory */  
    printf("pShortValue: %p\npIntValue: %p\npLongValue:  
%p\npFloatValue: %p\npDoubleValue: %p\npCharValue: %p\n",  
pShortValue,pIntValue,pLongValue,pFloatValue,pDoubleValue,pChar  
rValue);  
/* show point in the memory */
```



一. 基本数据类型

```
/* show the form of data type in the memory */
printf("shortValue: \n");
for(i=0;i<4;i++)
{
    printf("shortValue%d:
%p\n",i,pShortValue);
    pShortValue++;
}
printf("intValue: \n");
for(i=0;i<4;i++)
{
    printf("intValue%d: %p\n",i,pIntValue);
    pIntValue++;
}
printf("longValue: \n");
for(i=0;i<4;i++)
{
    printf("longValue%d:
%p\n",i,pLongValue);
    pLongValue++;
}
```



一. 基本数据类型

```
printf("floatValue: \n");
for(i=0;i<4;i++)
{
    printf("floatValue%d: %p\n",i,pFloatValue);
    pFloatValue++;
}
printf("doubleValue: \n");
for(i=0;i<4;i++)
{
    printf("doubleValue%d: %p\n",i,pDoubleValue);
    pDoubleValue++;
}
printf("charValue: \n");
for(i=0;i<4;i++)
{
    printf("charValue%d: %p\n",i,pCharValue);
    pCharValue++;
}
/* show data type in the memory */
system("pause");
return 0;
```

}



C:\ "C:\Documents and Settings\Administrator\桌面\3-1 backup\Debug\3-1-1.exe" -

pShortValue: 0012FF78
pIntValue: 0012FF68
pLongValue: 0012FF58
pFloatValue: 0012FF48
pDoubleValue: 0012FF28
pCharValue: 0012FF24
shortValue:
shortValue0: 0012FF78
shortValue1: 0012FF7A
shortValue2: 0012FF7C
shortValue3: 0012FF7E
intValue:
intValue0: 0012FF68
intValue1: 0012FF6C
intValue2: 0012FF70
intValue3: 0012FF74
longValue:
longValue0: 0012FF58
longValue1: 0012FF5C
longValue2: 0012FF60
longValue3: 0012FF64
floatValue:
floatValue0: 0012FF48
floatValue1: 0012FF4C
floatValue2: 0012FF50
floatValue3: 0012FF54
doubleValue:
doubleValue0: 0012FF28
doubleValue1: 0012FF30
doubleValue2: 0012FF38
doubleValue3: 0012FF40
charValue:
charValue0: 0012FF24
charValue1: 0012FF25
charValue2: 0012FF26
charValue3: 0012FF27
请按任意键继续. . .



一. 基本数据类型

- ❑ 在看相应的汇编代码之前，先想一想上面的代码会输出什么，一会比较一下，这与你想的是否一致
- ❑ 自己测试编译时选择**debug**版本，不选**release**版本
 - **release**是经过编译器优化的代码
 - 初学时，看**debug**版本才能看到原汁原味的源代码编译链接成的程序



一. 基本数据类型

@ 先看指针

```
lea    edx, [ebp+var_8]
mov     [ebp+var_64], edx
lea    eax, [ebp+var_18]
mov     [ebp+var_68], eax
lea    ecx, [ebp+var_28]
mov     [ebp+var_6C], ecx
lea    edx, [ebp+var_38]
mov     [ebp+var_70], edx
lea    eax, [ebp+var_58]
mov     [ebp+var_74], eax
lea    ecx, [ebp+var_5C]
mov     [ebp+var_78], ecx
mov     edx, [ebp+var_78]
push    edx
mov     eax, [ebp+var_74]
push    eax
mov     ecx, [ebp+var_70]
push    ecx
mov     edx, [ebp+var_6C]
push    edx
mov     eax, [ebp+var_68]
push    eax
mov     ecx, [ebp+var_64]
push    ecx
push    offset aPshortvaluePPI ; "pShortValue: %p\npIntValue: %p\npLongVa"...
```

pShortValue = shortValue

打印指针pShortValue的值



一. 基本数据类型

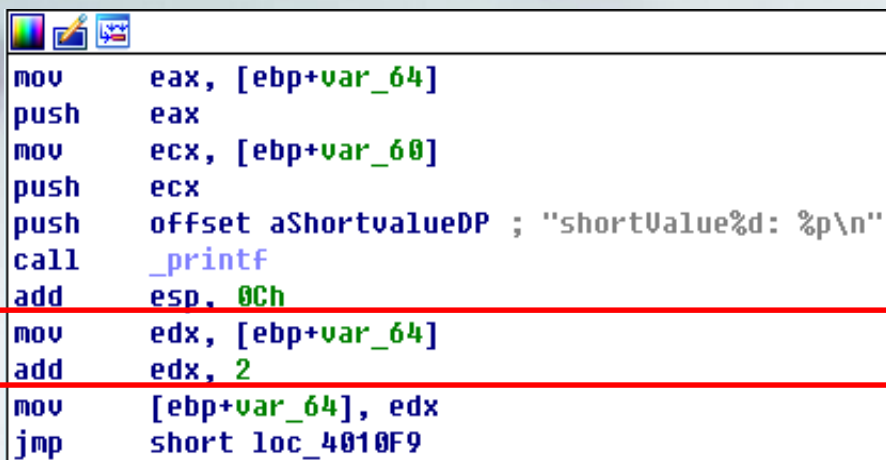
□ 例

- pShortValue 指针
- shortValue 数组是存储在栈上
- 上图中[ebp+var_8], [ebp+var_64]是一个指针，存储的是shortValue数组的地址，指向shortValue数组。



一. 基本数据类型

@打印的部分:



```
mov     eax, [ebp+var_64]
push    eax
mov     ecx, [ebp+var_60]
push    ecx
push    offset aShortvalueDP ; "shortValue%d: %p\n"
call    _printf
add     esp, 0Ch
mov     edx, [ebp+var_64]
add     edx, 2
mov     [ebp+var_64], edx
jmp     short loc_4010F9
```

The assembly code snippet is displayed in a window. The lines 'mov edx, [ebp+var_64]' and 'add edx, 2' are highlighted with a red rectangular box.



一. 基本数据类型

- ❑ 取出了指针所指向的地址，然后依次打印出了数组的内容，其中指针的自加，即**add eax,2**，因为是**short**类型，一个数占用两个字节。
- ❑ 注意看看**double**指针和**int**指针的自加。



一. 基本数据类型

```
mov     ecx, [ebp+var_74]
push    ecx
mov     edx, [ebp+var_60]
push    edx
push    offset aDoublevalueDP ; "doubleValue%d: %p\n"
call    _printf
add     esp, 0Ch
mov     eax, [ebp+var_74]
add     eax, 8
mov     [ebp+var_74], eax
jmp     short loc_40120D
```

```
mov     ecx, [ebp+var_68]
push    ecx
mov     edx, [ebp+var_60]
push    edx
push    offset aIntvalueDP ; "intValue%d: %p\n"
call    _printf
add     esp, 0Ch
mov     eax, [ebp+var_68]
add     eax, 4
mov     [ebp+var_68], eax
jmp     short loc_40113E
```



一. 基本数据类型

- ④ 同样是指针的自加，不同的数据类型在汇编中的表现形式就十分不同，而且在内存中，均是以字节为单位进行运算。



一. 基本数据类型

④ 整型数和浮点数在内存中表示:

```
mov    [ebp+var_18], 0C8h
xor     ecx, ecx
mov     [ebp+var_14], ecx
mov     [ebp+var_10], ecx
mov     [ebp+var_C], ecx
mov     [ebp+var_28], 12Ch
xor     edx, edx
mov     [ebp+var_24], edx
mov     [ebp+var_20], edx
mov     [ebp+var_1C], edx
mov     [ebp+var_38], 43C80CCDh
xor     eax, eax
mov     [ebp+var_34], eax
mov     [ebp+var_30], eax
mov     [ebp+var_2C], eax
mov     [ebp+var_58], 0EB851EB8h
mov     [ebp+var_54], 407F4051h
```

整形数

浮点数



一. 基本数据类型

□ 浮点数运算

- x86使用的是浮点寄存器，Intel提供了8个128位的寄存器，xmm0~xmm7，每一个寄存器可以存放4个(32位)单精度的浮点数。



一. 基本数据类型

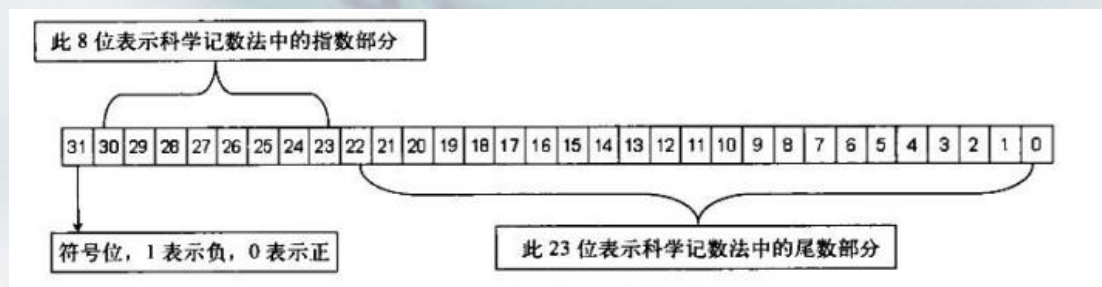
④ 以浮点数**400.1**和**500.02**为例

- ❑ 这两个数在内存中的十六进制表现形式分别为**0x43C80CCD**和**0x407F4051EB851EB8**
- ❑ **float**类型在内存中占**4**字节，需要经过**IEEE**编码
- ❑ 编码方式如下：最高位用于表示符号，剩余**31**位中，**8**位用于表示指数，其余用于表示尾数



一. 基本数据类型

□ 编码方式如下：最高位用于表示符号，剩余31位中，8位用于表示指数，其余用于表示尾数



第三章从C语言看汇编

- @一. 基本数据类型
- @二. 流程控制语句
- @三. 变量表现形式



二. 流程控制语句

④ 流程控制语句在汇编中的表现形式

- C语言基本的选择，循环等流程控制块在汇编中的表现形式



二. 流程控制语句

- ❑ (1) **if,elseif,else**选择控制块
- ❑ (2) **switch case**选择控制块
- ❑ (3) **while/for/do**循环控制块



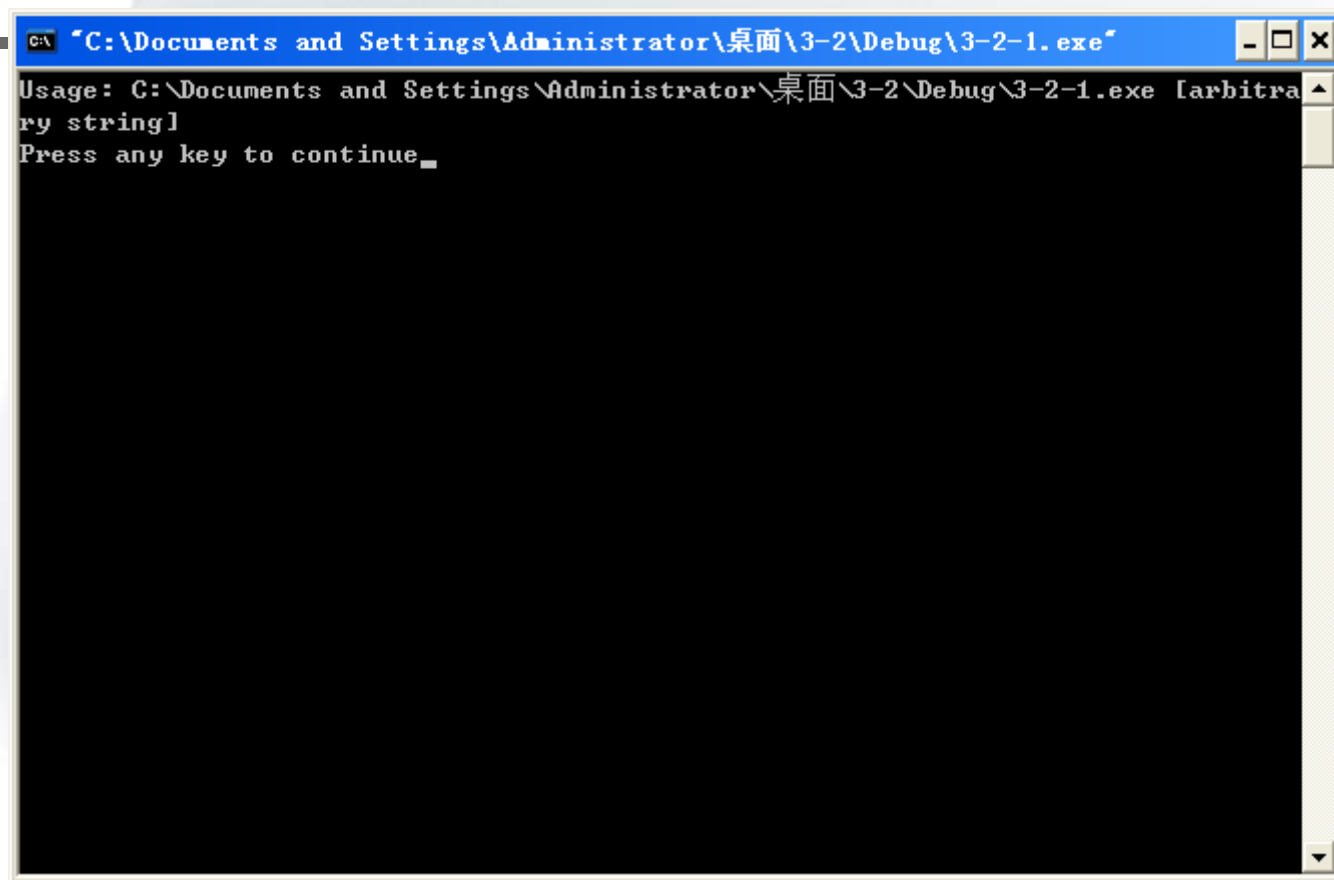
二. 流程控制语句

@if语句 (vc++ 6.0 debug版本)

```
Int main(int argc, char *argv[])
{
    if(argc<2)
    {
        printf("Usage: %s [arbitrary
string]\n", argv[0]);
    }
    return 0;
}
```



二. 流程控制语句



```
C:\ "C:\Documents and Settings\Administrator\桌面\3-2\Debug\3-2-1.exe"
Usage: C:\Documents and Settings\Administrator\桌面\3-2\Debug\3-2-1.exe [arbitra
ry string]
Press any key to continue_
```



一 流程控制语句

IDA - C:\Documents and Settings\Administrator\桌面\3-2\Debug\3-2-1.exe

File Edit Jump Search View Debugger Options Windows Help

Library function Data Regular function Unexplored Instruction External symbol

Functions window

Function name	Segment	S
_main	.text	0
_main_0	.text	0
_printf	.text	0
_chkesp	.text	0
start	.text	0
_msg_exit	.text	0
_fast_error_exit	.text	0
_stbuf	.text	0
_ftbuf	.text	0
sub_4014B0	.text	0
_write_char	.text	0
_write_multi_char	.text	0
_write_string	.text	0
_get_int_arg	.text	0
_get_int64_arg	.text	0
_get_short_arg	.text	0
_initstdio	.text	0
_endstdio	.text	0
sub_402560	.text	0
_CrtSetReportMode	.text	0
_CrtSetReportFile	.text	0
_CrtDbgReport	.text	0
_CrtMessageWindow	.text	0
_cinit	.text	0
_exit	.text	0
_cexit	.text	0
_doexit	.text	0
_initterm	.text	0
_xoptFilter	.text	0
_xoptLookup	.text	0
_setenvp	.text	0
_setargv	.text	0
_parse_cmdline	.text	0
_crtGetEnvironmentStringsA	.text	0
_j0init	.text	0

Line 2 of 177

Graph overview

100.00% (-499,-11) (504,23) 00001010 00401010: _main_0 (Synchronized with Hex View-1)

Output window

IDA Python v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>

Python

AU: idle Up Disk: 35GB

Attributes: bp-based frame

_main_0 proc near

```
var_40= byte ptr -40h
arg_0= dword ptr 8
arg_4= dword ptr 0Ch

push ebp
mov ebp, esp
sub esp, 40h
push ebx
push esi
push edi
lea edi, [ebp+var_40]
mov ecx, 10h
mov eax, 0CCCCCCCCh
rep stosd
cmp [ebp+arg_0], 2
jge short loc_401041
```

mov eax, [ebp+arg_4]
mov ecx, [eax]
push ecx
push offset aUsage\$Arbitrar ; "Usage: %s [arbitrary string]\n"
call _printf
add esp, 8

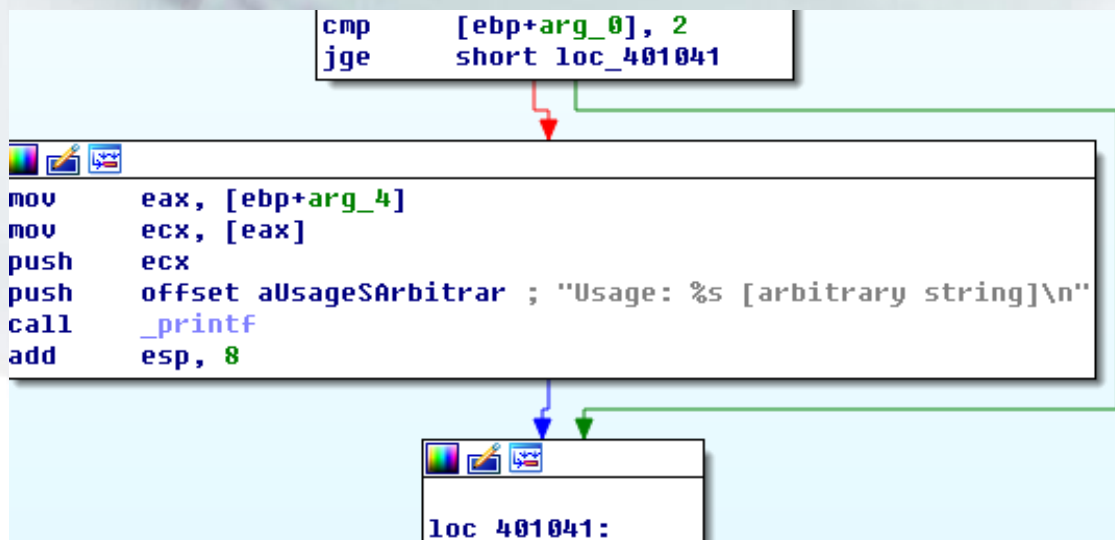
loc_401041:

```
xor eax, eax
pop edi
pop esi
pop ebx
add esp, 40h
mov ebp, esp
call _chkesp
mov esp, ebp
pop ebp
```



二. 流程控制语句

- 可以通过ida清楚地看到if语句控制块的模样，if `argc < 2`，在汇编中是一句 `jge short loc_401041` 代码，如果大于等于2，则跳出，小于，执行 `printf` 函数。



二. 流程控制语句

@if else控制块 (vc++ 6.0 debug版本)

```
Int main(int argc,char *argv[])
{
    if(argc<2)
    {
        printf("Usage: %s [arbitrary string]\n",argv[0]);
    }
    else
    {
        printf("Hello World:%s\n",argv[1]);
    }
    return 0;
}
```



二. 流程控制语句

IDA - C:\Documents and Settings\Administrator\桌面\3-2-2\Debug\3-2-2.exe

File Edit Jump Search View Debugger Options Windows Help

Library function Data Regular function Unexplored Instruction External symbol

Functions window

Function name	Segment	S
__main	.text	0
__main_0	.text	0
__printf	.text	0
__chkesp	.text	0
start	.text	0
__msg_exit	.text	0
__fast_error_exit	.text	0
__stbuf	.text	0
__ftbuf	.text	0
sub_4014F0	.text	0
__write_char	.text	0
__write_multi_char	.text	0
__write_string	.text	0
__get_int_arg	.text	0
__get_int64_arg	.text	0
__get_short_arg	.text	0
__initstdio	.text	0
__endstdio	.text	0
sub_402570	.text	0
__CrtSetReportMode	.text	0
__CrtSetReportFile	.text	0
__CrtDbgReport	.text	0
__CrtMessageWindow	.text	0
__cinit	.text	0
__exit	.text	0
__cexit	.text	0
__c_exit	.text	0
__doexit	.text	0
__initterm	.text	0
__xcpFilter	.text	0
__xcpLookup	.text	0
__setenvp	.text	0
__setargv	.text	0
__parse_cmdline	.text	0
__crtGetEnvironmentStringsA	.text	0
__j0init	.text	0

IDA View-A

Hex View-1

Structures

Enums

Imports

Exports

```
; Attributes: bp-based frame
__main_0 proc near
var_40= byte ptr -40h
arg_0= dword ptr 8
arg_4= dword ptr 0Ch

push    ebp
mov     ebp, esp
sub     esp, 40h
push    ebx
push    esi
push    edi
lea     edi, [ebp+var_40]
mov     ecx, 10h
mov     eax, 0CCCCCCCCh
rep stosd
cmp     [ebp+arg_0], 2
jge     short loc_401043

mov     eax, [ebp+arg_4]
mov     ecx, [eax]
push    ecx
push    offset aUsageSArbitrar ; "Usage: %s [arbitrary string]\n"
call    __printf
add     esp, 8
jmp     short loc_401057

loc_401043:
mov     edx, [ebp+arg_4]
mov     eax, [edx+4]
push    eax
push    offset aHelloWorldS ; "Hello World:%s\n"
call    __printf
add     esp, 8

loc_401057:
xor     eax, eax
pop     edi
pop     esi
pop     ebx
add     esp, 40h
cmp     ebp, esp
call    __chkesp
```

ne 2 of 177

Graph overview

100.00% (-228,-12) (1337,419) 00001010 00401010: __main_0 (Synchronized with Hex View-1)

Output window

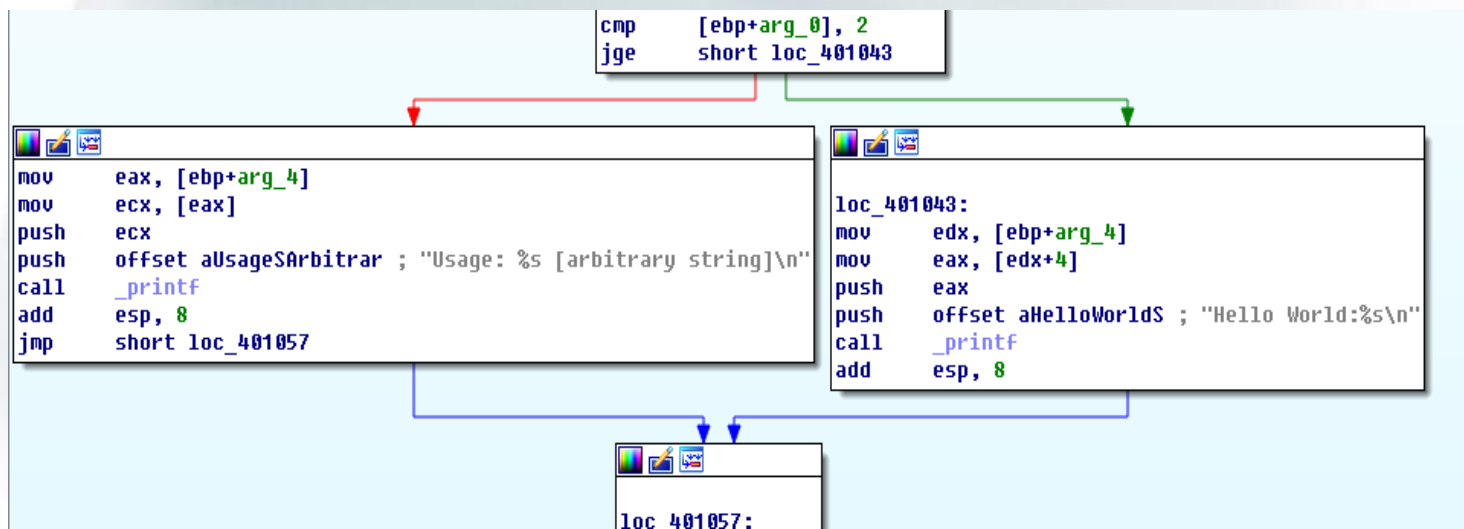
IDA Python v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>

Python

idle Down Disk: 35GB

二. 流程控制语句

□ 用ida打开，进行分析



○ JGE/JNL 大于或等于转移



二. 流程控制语句

- 和刚刚的if控制块相比，**if else**控制块多出来了一个分支，使得**cmp**之后，必须选择其中之一进行执行，而且也没有多余的分支可以选择
 - 大于等于2选择右边的基础块
 - 小于2选择左边的基础块



二. 流程控制语句

□ **if elseif else**控制块（**vc++ 6.0 debug**版本）

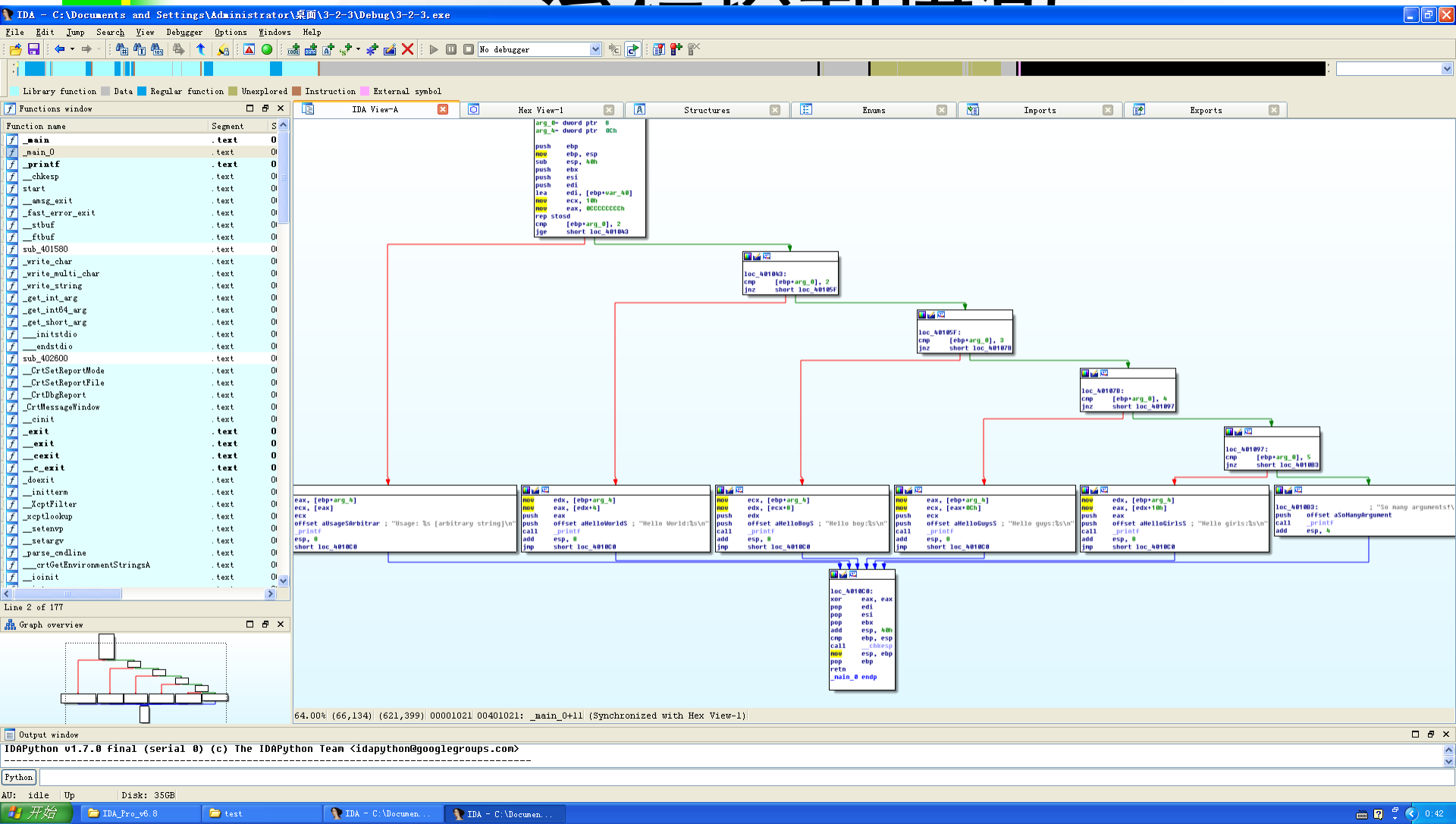


二. 流程控制语句

```
Int main(int argc,char *argv[])
{
    if(argc<2)
    {
        printf("Usage: %s [arbitrary string]\n",argv[0]);
    }
    elseif(argc==2)
    {
        printf("Hello World:%s\n",argv[1]);
    }
    elseif(argc==3)
    {
        printf("Hello boy:%s\n",argv[2]);
    }
    elseif(argc==4)
    {
        printf("Hello guys:%s\n",argv[3]);
    }
    elseif(argc==5)
    {
        printf("Hello girls:%s\n",argv[4]);
    }
    else
    {
        printf("So many arguments!\n");
    }
    return 0;
}
```

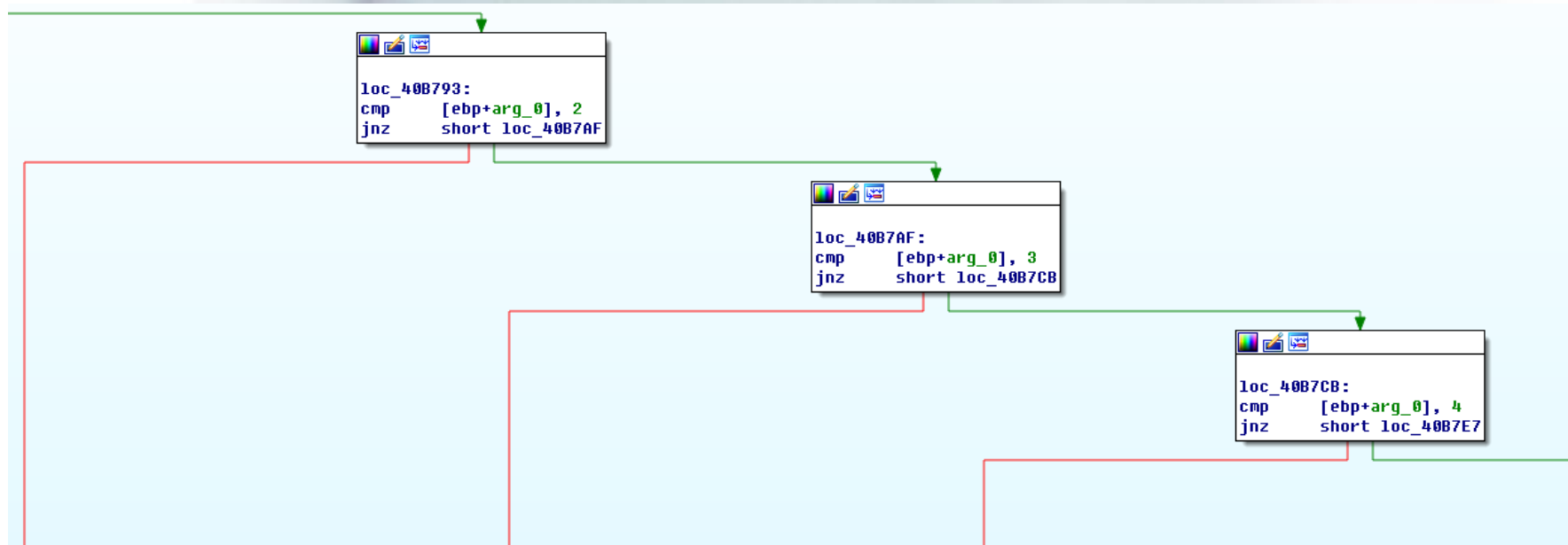


一 法和控制语句



二. 流程控制语句

○ Jnz 条件转移指令。结果不为零（或不相等）则转移



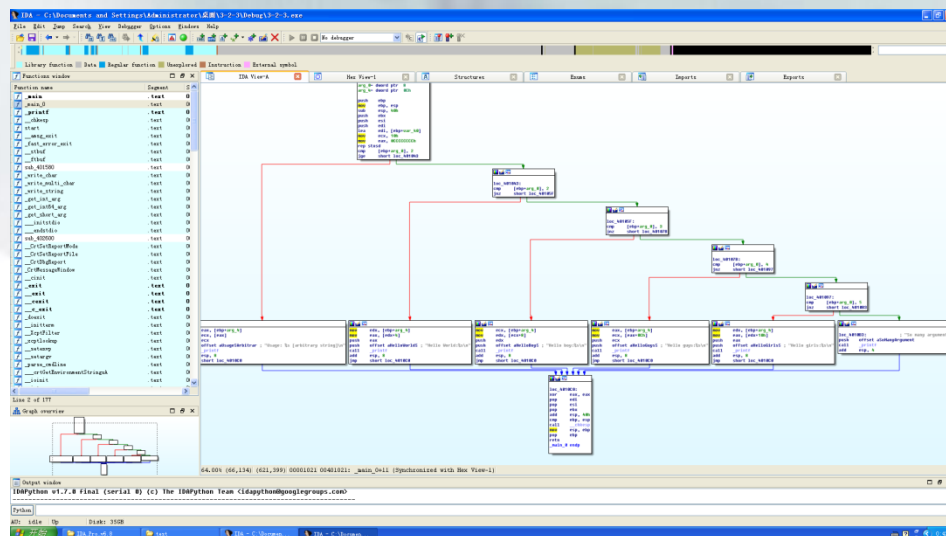
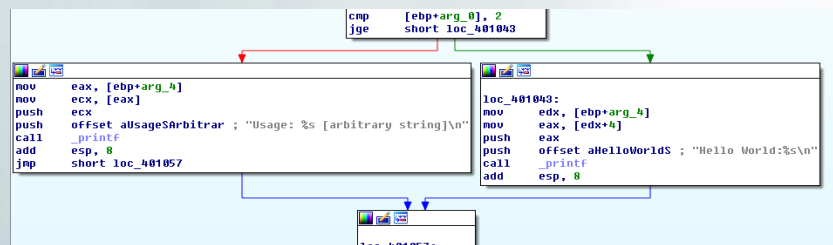
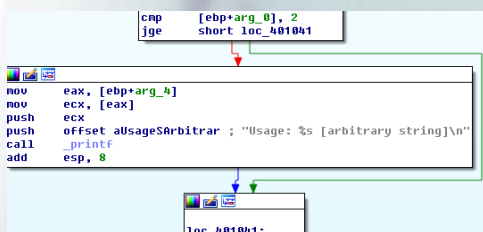
二. 流程控制语句

- 通过if else if 控制块全貌图，可以明显地看到，程序是先判断一个分支，然后依次判断下一个else if分支，这样依次下去，判断语句做为其中的一条分支。



二. 流程控制语句

④ 通过if, if else, if elseif else控制块, 可以从ida的图形上清楚地看到如何去识别一个选择流程控制块, 然后转换为高级语言。



崔宝江



二. 流程控制语句

- ❑ (1) **if,elseif,else**选择控制块
- ❑ (2) **switch case**选择控制块
- ❑ (3) **while/for/do**循环控制块



二. 流程控制语句

□ **switch case控制块 (vc++ 6.0 debug版本)**



二. 流程控制语句

```
Int main(int argc, char *argv[])
{
    switch(argc)
    {
        case1:
            printf("Usage: %s [arbitrary string]\n", argv[0]);
            break;
        case2:
            printf("Hello World:%s\n", argv[1]);
            break;
        case3:
            printf("Hello boy:%s\n", argv[2]);
            break;
        case4:
            printf("Hello guys:%s\n", argv[3]);
            break;
        case5:
            printf("Hello girls:%s\n", argv[4]);
        default:
            printf("So many arguments!\n");
    };
    return 0;
}
```



一 法控控制语句

IDA - C:\Documents and Settings\Administrator\桌面\3-2-4switch case\Debug\3-2-4.exe

File Edit Jump Search View Debugger Options Windows Help

No debugger

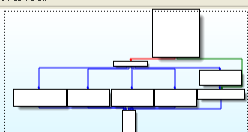
Library function Data Regular function Unexplored Instruction External symbol

Functions window

Function name	Segment	S
_main	.text	0
_main_0	.text	0
_printf	.text	0
_chkexp	.text	0
_start	.text	0
_amsq_exit	.text	0
_fast_error_exit	.text	0
_stbuf	.text	0
_ftbuf	.text	0
sub_401590	.text	0
_write_char	.text	0
_write_multi_char	.text	0
_write_string	.text	0
_get_int_arg	.text	0
_get_int64_arg	.text	0
_get_short_arg	.text	0
_initstdio	.text	0
_endstdio	.text	0
sub_402610	.text	0
_CrtSetReportMode	.text	0
_CrtSetReportFile	.text	0
_CrtDbgReport	.text	0
_CrtMessageBoxW	.text	0
_cinit	.text	0
_exit	.text	0
_exit	.text	0
_c_exit	.text	0
_c_exit	.text	0
_doexit	.text	0
_initterm	.text	0
_XcptFilter	.text	0
_xcplookup	.text	0
_setenvp	.text	0
_setargv	.text	0
_parse_cmdline	.text	0
_crtGetEnvironmentStringsA	.text	0
_iointit	.text	0

Line 2 of 177

Graph overview



IDA View-A

Hex View-1

Structures

Enums

Imports

Exports

Attributes: bp-based frame

_main_0 proc near

var_4h byte ptr -4h

var_8h dword ptr -8h

arg_0 dword ptr 0

arg_4h dword ptr 4h

push ebp

mov ebp, esp

sub esp, 4h

push ebx

push esi

push edi

lea edi, [ebp+var_4h]

mov ecx, 10h

mov ecx, 0CCCCCCC

rep stsd

mov eax, [ebp+arg_0]

mov [ebp+var_4h], eax

mov ecx, [ebp+var_4h]

sub ecx, 1

mov [ebp+var_4h], ecx

cmp [ebp+var_4h], 4 ; switch 5 cases

ja short loc_401007 ; jumtable 00401000 default case

loc_401007: ; jumtable 00401000 case 0
mov edx, [ebp+var_4h]
jmp ds:off_401007[edx*4] ; switch jump

loc_401007: ; jumtable 00401000 case 0
mov eax, [ebp+arg_4]
mov ecx, [eax]
push ecx
offset aUsage\$Arbitrar: "Usage: %s [arbitrary string]\n"
call _printf
add esp, 8
jmp short loc_40100F

loc_40100C: ; jumtable 00401000 case 1
mov edx, [ebp+arg_4]
mov eax, [edx+4]
push eax
offset aHelloWorld\$: "Hello World!\n"
call _printf
add esp, 8
jmp short loc_40100F

loc_40100E: ; jumtable 00401000 case 2
mov ecx, [ebp+arg_4]
mov edx, [ecx+8]
push edx
offset aHelloBy\$: "Hello by:\n"
call _printf
add esp, 8
jmp short loc_40100F

loc_401008: ; jumtable 00401000 case 3
mov eax, [ebp+arg_4]
mov ecx, [eax+0Ch]
push ecx
offset aHelloGuys\$: "Hello guys:\n"
call _printf
add esp, 8
jmp short loc_40100F

loc_40100E: ; jumtable 00401000 case 4
mov edx, [ebp+arg_4]
mov eax, [edx+10h]
push eax
offset aHelloGirls\$: "Hello girls!\n"
call _printf
add esp, 8
jmp short loc_40100F

loc_401002: ; jumtable 00401000 default case
push offset aSetLanguage\$: "Set language:\n"
call _printf
add esp, 4
jmp short loc_40100F

loc_40100F: ; End of function
xor eax, eax
pop edi
pop esi
pop ebx
add esp, 4h
cnp ebp, esp
call _chkexp
mov esp, ebp
pop ebp
ret

64.00% (-88,11) (979,405) 00001010 00401010: _main_0 (Synchronized with Hex View-1)

Output window

IDA Pro v7.0 Final (Serial 0) (C) The IDA Pro Team <ida@pro.com>

Python

AU: idle Up Disk: 350B

开始 IDA_Pro_v6.8 text 3-2 IDA - C:\Documen... IDA - C:\Documen...

北邮网安学院 崔宝江



二. 流程控制语句

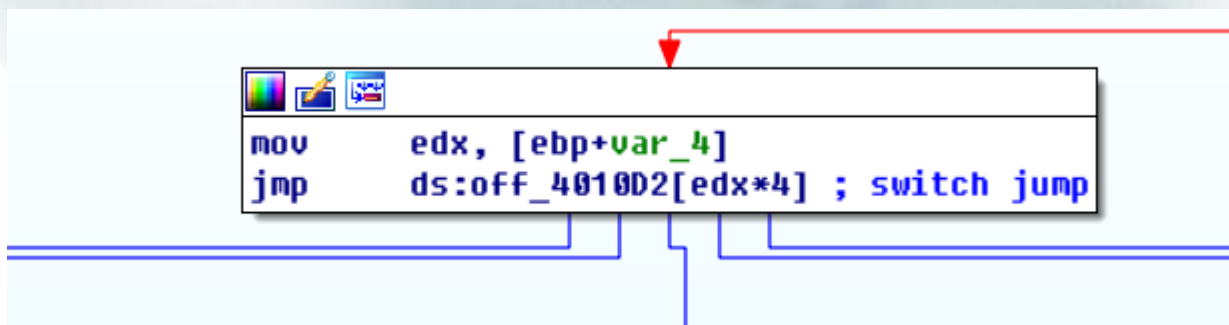
- 从控制流程图上将其与上一小节讲的**if else if**作一下比较
- **switch case**控制块的左边有点类似于分发器，由一个基本块来决定执行哪一块函数功能。



二. 流程控制语句

④ 该基本块的汇编代码如下：

- ❑ 将选择的序号值给了**edx**，根据**edx**，**jmp**到**off_4010D2**这个**table**中的某个地址。
- ❑ 这一处也就是和**if else**语句明显的不同之处，将要执行的地址存放到了一个数组里面，数组的每一个元素都是一个地址。



二. 流程控制语句

□ 减少case分支，控制块全貌图又会发生什么变化

```
Int main(int argc, char *argv[])
{
    switch(argc)
    {
        case1:
            printf("Usage: %s [arbitrary string]\n", argv[0]);
            break;
        case2:
            printf("Hello World:%s\n", argv[1]);
            break;
        default:
            printf("So many arguments!\n");
    };
    return 0;
}
```



一 法控制语句

IDA - C:\Documents and Settings\Administrator\桌面\3-2-5 case2\Debug\3-2-5.exe

File Edit Jump Search View Debugger Options Windows Help

Library function Data Regular function Unexplored Instruction External symbol

Functions window

Function name	Segment	S
_main	.text	0
_main_0	.text	0
_printf	.text	0
_chkexp	.text	0
_start	.text	0
_msg_exit	.text	0
_fast_error_exit	.text	0
_stbuf	.text	0
_ftbuf	.text	0
sub_401520	.text	0
_write_char	.text	0
_write_multi_char	.text	0
_write_string	.text	0
_get_int_arg	.text	0
_get_int64_arg	.text	0
_get_short_arg	.text	0
_initstdio	.text	0
_endstdio	.text	0
sub_4025A0	.text	0
_CrtSetReportMode	.text	0
_CrtSetReportFile	.text	0
_CrtDbgReport	.text	0
_CrMessageWindow	.text	0
_cinit	.text	0
_exit	.text	0
_c_exit	.text	0
_c_exit	.text	0
_doexit	.text	0
_initterm	.text	0
_XcptFilter	.text	0
_xcptlookup	.text	0
_setenvp	.text	0
_setargv	.text	0
_parse_cmdline	.text	0
_crGetEnvironmentStringsA	.text	0
_iointit	.text	0

IDA View-A Hex View-1 Structures Enums Imports Exports

```
push ebp
mov ebp, esp
sub esp, 44h
push ebx
push esi
push edi
lea edi, [ebp+var_44]
mov ecx, 11h
mov eax, 0CCCCCCCCh
rep stosd
mov eax, [ebp+arg_0]
mov [ebp+var_4], eax
cmp [ebp+var_4], 1
jz short loc_40103C
```

```
cmp [ebp+var_4], 2
jz short loc_401051
```

```
jmp short loc_401067
```

```
loc_40103C:
mov ecx, [ebp+arg_4]
mov edx, [ecx]
push edx
push offset aUsageArbitrar ; "Usage: %s [arbitrary string]\n"
call _printf
add esp, 8
jmp short loc_401074
```

```
loc_401051:
mov eax, [ebp+arg_4]
mov ecx, [eax+4]
push ecx
push offset aHelloWorldS ; "Hello World:%s\n"
call _printf
add esp, 8
jmp short loc_401074
```

```
loc_401067:
; "So many arguments!\n"
push offset aSoManyArgument
call _printf
add esp, 4
```

```
loc_401074:
xor eax, eax
pop edi
pop esi
pop ebx
add esp, 44h
cmp ebp, esp
```

Line 2 of 177

Graph overview

Output window

Python v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>

Python

AU: idle Up Disk: 350B

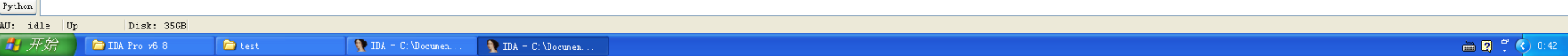
开始 IDA_Pro_v6.8 test 3-2 IDA - C:\Documen... IDA - C:\Documen...

100.00% (-128,198) [1316,639] 00001010 00401010: _main_0 (Synchronized with Hex View-1)

1:01

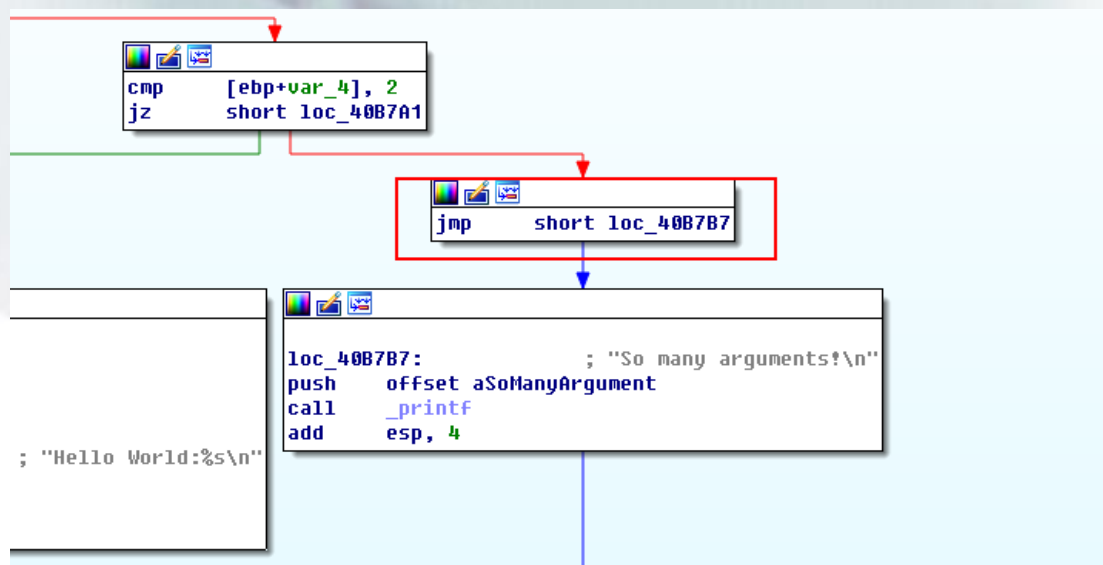


④ 和if else if选择控制块流程相比较，二者十分相似的



二. 流程控制语句

- ❑ 除了在最后switch case选择控制块使用了default，使得程序有了一个单入单出的jmp指令
- ❑ if else if选择控制块，则没有这个单入单出的jmp指令基础块



二. 流程控制语句

- 前面的**case**值都是简单而且单增，线性的
- 如果改变其中一个**case**值为**255**，整个选择控制块结构又会发生什么变化呢？



二. 流程控制语句

④ 非线性case值 (vc++ 6.0 debug版本)



二. 流程控制语句

```
Int main(int argc, char *argv[])
{
    switch(argc)
    {
        case1:
            printf("Usage: %s [arbitrary string]\n", argv[0]);
            break;
        case2:
            printf("Hello World:%s\n", argv[1]);
            break;
        case3:
            printf("Hello boy:%s\n", argv[2]);
            break;
        case4:
            printf("Hello guys:%s\n", argv[3]);
            break;
        case255:
            printf("Hello girls:%s\n", argv[4]);
            break;
        default:
            printf("So many arguments!\n");
    };
    return0;
}
```



一 法控生制五句

IDA - C:\Documents and Settings\Administrator\桌面\3-2-6 case255\Debug\3-2-6.exe

File Edit Jump Search View Debugger Options Windows Help

No debugger

Library function Data Regular function Unexplored Instruction External symbol

Functions window

Function name	Segment	S
_main	.text	0
_main_0	.text	0
_printf	.text	0
_chkesp	.text	0
_start	.text	0
_msg_exit	.text	0
_fast_error_exit	.text	0
_stbuf	.text	0
_ftbuf	.text	0
sub_4016E0	.text	0
_write_char	.text	0
_write_multi_char	.text	0
_write_string	.text	0
_get_int_arg	.text	0
_get_int64_arg	.text	0
_get_short_arg	.text	0
_initstdio	.text	0
_endstdio	.text	0
sub_402760	.text	0
_CrtSetReportMode	.text	0
_CrtSetReportFile	.text	0
_CrtDbgReport	.text	0
_CrtMessageBoxWindow	.text	0
_cinit	.text	0
_exit	.text	0
_exit	.text	0
_c_exit	.text	0
_c_exit	.text	0
_doexit	.text	0
_initterm	.text	0
_XcptFilter	.text	0
_xcplookup	.text	0
_setenvp	.text	0
_setargv	.text	0
_parse_cmdline	.text	0
_crtGetEnvironmentStringsA	.text	0
_iointit	.text	0

IDA View-A Hex View-1 Structures Enums Imports Exports

Attributes: bp-based frame

_main_0 proc near

var_4h: byte ptr -4h

var_4h: dword ptr -4

arg_0: dword ptr 0

arg_4: dword ptr 0ch

push ebp

mov ebp, esp

sub esp, 4h

push ebx

push esi

push edi

lea edi, [ebp+var_4h]

mov ecx, 11h

mov ecx, 0CCCCCCCch

rep stsd

mov eax, [ebp+arg_0]

mov [ebp+var_4], eax

mov ecx, [ebp+var_4]

sub ecx, 1

mov [ebp+var_4], ecx

cmp [ebp+var_4], 0Fh ; switch 255 cases

ja short loc_40100F ; jumtable 00401000 default case

loc_401007: ; jumtable 00401000 case 0

mov eax, [ebp+var_4]

xor ecx, ecx

mov dl, ds:byte_401007[eax]

jmp ds:off_401000[ecx*4] ; switch jump

loc_401007: ; jumtable 00401000 case 0

mov eax, [ebp+arg_4]

mov ecx, [eax*4]

push ecx

push offset aHelloWorldS ; "Hello World!\n"

call _printf

add esp, 8

jmp short loc_40100C

loc_401007: ; jumtable 00401000 case 2

mov edx, [ebp+arg_4]

mov eax, [edx*8]

push eax

push offset aHelloBugs ; "Hello bugs!\n"

call _printf

add esp, 8

jmp short loc_40100C

loc_401009: ; jumtable 00401000 case 3

mov ecx, [ebp+arg_4]

mov edx, [ecx*0Ch]

push edx

push offset aHelloBugsS ; "Hello bugs!\n"

call _printf

add esp, 8

jmp short loc_40100C

loc_401009: ; jumtable 00401000 case 254

mov eax, [ebp+arg_4]

mov ecx, [eax*10h]

push ecx

push offset aHelloGirlsS ; "Hello girls!\n"

call _printf

add esp, 8

jmp short loc_40100C

loc_40100F: ; jumtable 00401000 default case

push offset aNoArgument

call _printf

add esp, 4

loc_40100C: ;

xor eax, eax

pop edi

pop esi

pop ebx

add esp, 4h

cmp ebp, esp

call _chkesp

mov esp, ebp

pop ebp

ret

_main_0 endp

64.00% (223,-76) (943,632) 00001010 00401010: _main_0 (Synchronized with Hex View-1)

Output window

Python v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>

Python

AU: idle Up Disk: 350B

开始 IDA_Pro_v6.8 test 3-2 IDA - C:\Documen... IDA - C:\Documen...

1:10



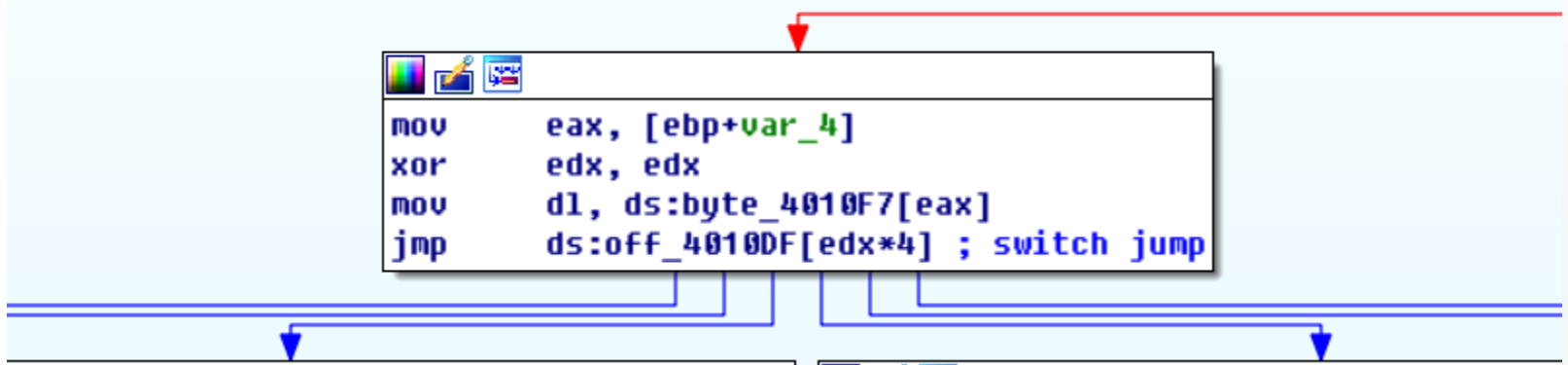
二. 流程控制语句

- 可以看到非线性的**switch**结构和之前线性的**switch**结构相比，从控制流结构上似乎并没有什么不同
- 都是由一个类似分发器的基础块根据我们的输入，**jmp**到相应的地址。

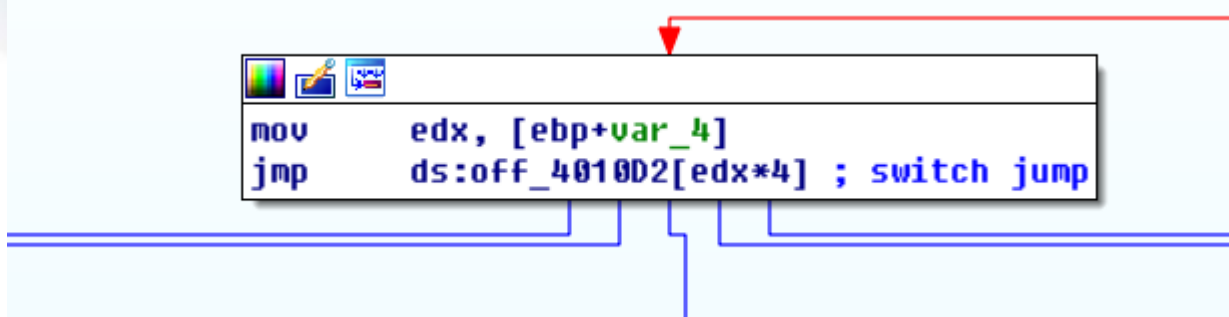


二. 流程控制语句

□但是负责分发的那个基础块则有很大的不同



```
mov     eax, [ebp+var_4]
xor     edx, edx
mov     dl, ds:byte_4010F7[eax]
jmp     ds:off_4010DF[edx*4] ; switch jump
```



```
mov     edx, [ebp+var_4]
jmp     ds:off_4010D2[edx*4] ; switch jump
```

[illegible]

二. 流程控制语句

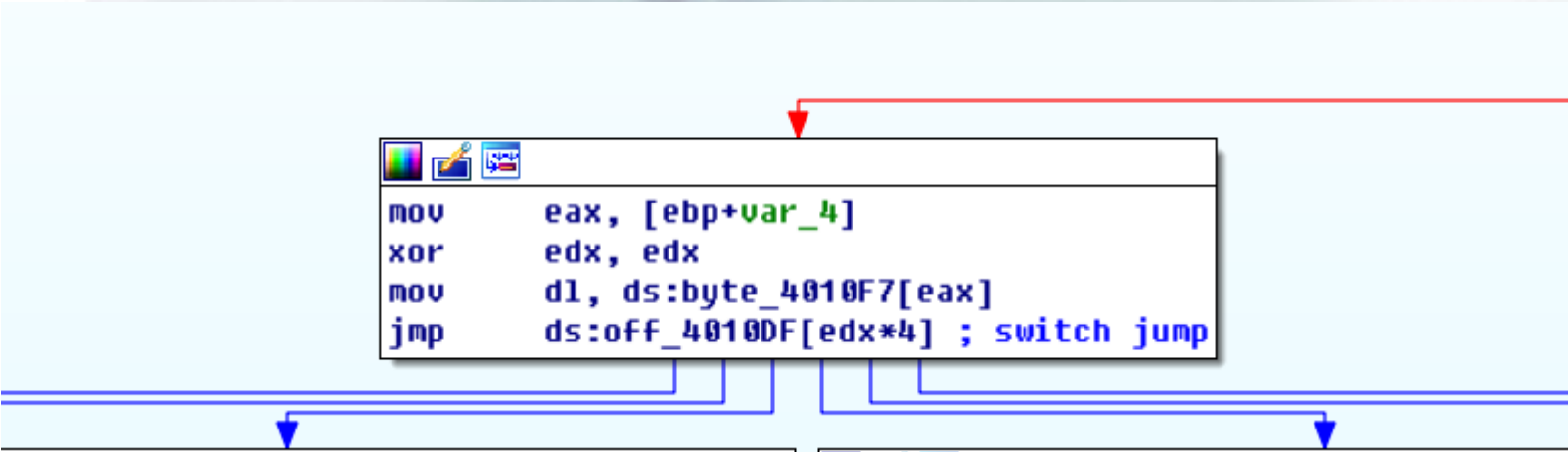
- 这张表由**255**个元素组成，数组的大小由我们**case**值最大的确定即**255**
- 程序根据输入的**case**值，在第一张表中取索引值，然后在第二个地址数组中，获取相应的地址进行跳转

```
off_4010DF      dd offset loc_401052, offset loc_401067, offset loc_40107D  
                ; DATA XREF: _main_0+3B↑r  
                dd offset loc_401093, offset loc_4010A9, offset loc_4010BF ; jump table for switch statement
```



二. 流程控制语句

- 将两张表结合起来看，可以得到以下结论
 - 根据**case**值从索引表里获取索引，根据索引从地址数组里获取地址，并跳转
 - 索引数组中均为5的索引全部都指向的**default**分支，其余的分支均为**case**值对应的分支。



```
mov     eax, [ebp+var_4]
xor     edx, edx
mov     dl, ds:byte_4010F7[eax]
jmp     ds:off_4010DF[edx*4] ; switch jump
```



二. 流程控制语句

④ 这些并不是**switch**控制流结构的全部，讲解的这些东西仅仅是作为入门，抛砖引玉，希望能提高大家自我学习的兴趣，查阅资料进行深入学习



二. 流程控制语句

- ❑ (1) **if,elseif,else**选择控制块
- ❑ (2) **switch case**选择控制块
- ❑ (3) **while/for/do**循环控制块



二. 流程控制语句

□ (3) while/for/do循环控制块

- while循环控制块

- for循环控制块

- do循环控制块



二. 流程控制语句

@while循环 (vc++ 6.0 debug版本)

```
Int main()  
{  
    int i=100;  
    while(i--)  
    {  
  
    }  
    return 0;  
}
```



一 法控制语句

IDA - C:\Documents and Settings\Administrator\桌面\3-2-7 while循环\Debug\3-2-7.exe

File Edit Jump Search View Debugger Options Windows Help

Library function Data Regular function Unexplored Instruction External symbol

Functions window

Function name	Segment	S
_main	.text	0
_main_0	.text	0
start	.text	0
_msg_exit	.text	0
_fast_error_exit	.text	0
_cinit	.text	0
_exit	.text	0
_cexit	.text	0
_c_exit	.text	0
_doexit	.text	0
_initterm	.text	0
_XcptFilter	.text	0
_xcptlookup	.text	0
_setenvp	.text	0
_setargv	.text	0
_parse_cmdline	.text	0
_crGetEnvironmentStringsA	.text	0
_joininit	.text	0
_iosterm	.text	0
sub_4021C0	.text	0
sub_402220	.text	0
sub_402450	.text	0
_global_unwind2	.text	0
_unwind_handler	.text	0
_local_unwind2	.text	0
_abnormal_termination	.text	0
_NLG_Notify	.text	0
_except_handler3	.text	0
_seh_longjmp_unwind(x)	.text	0
_FF_MSGBANNER	.text	0
_NMSG_WRITE	.text	0
_GET_RTERMSG	.text	0
_malloc	.text	0
_malloc_dbg	.text	0
_nh_malloc	.text	0
_nh_malloc_dbg	.text	0

not found

Graph overview

Output window

IDAView-A Hex View-1 Structures Enums Imports Exports

```
var_44= byte ptr -44h
var_4= dword ptr -4

push ebp
mov ebp, esp
sub esp, 44h
push ebx
push esi
push edi
lea edi, [ebp+var_44]
mov ecx, 11h
mov eax, 0CCCCCCCCh
rep stosd
mov [ebp+var_4], 64h

loc_40102F:
mov eax, [ebp+var_4]
mov ecx, [ebp+var_4]
sub ecx, 1
mov [ebp+var_4], ecx
test eax, eax
jz short loc_401041

jmp short loc_40102F

loc_401041:
xor eax, eax
pop edi
pop esi
pop ebx
mov esp, ebp
pop ebp
retn
_main_0 endp
```

100.00% (-537,102) (1113,382) 00001010 00401010: _main_0 (Synchronized with Hex View-1)

Python v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>

Python

AU: idle Up Disk: 350B

开始 IDA_Pro_v6.8 test 3-2 IDA - C:\Documen... IDA - C:\Documen...



二. 流程控制语句

- ④ 我们可以看到，**while**循环有两次跳转，所在的循环控制块一定是由闭合的基本块组成，循环，顾名思义，一定是闭合的。




```
var_44= byte ptr -44h
var_4= dword ptr -4
```

```
push    ebp
mov     ebp, esp
sub     esp, 44h
push    ebx
push    esi
push    edi
lea     edi, [ebp+var_44]
mov     ecx, 11h
mov     eax, 0CCCCCCCCh
rep stosd
mov     [ebp+var_4], 64h
```

```
loc_40102F:
mov     eax, [ebp+var_4]
mov     ecx, [ebp+var_4]
sub     ecx, 1
mov     [ebp+var_4], ecx
test    eax, eax
jz      short loc_401041
```

```
jmp     short loc_40102F
```

```
loc_401041:
xor     eax, eax
pop     edi
pop     esi
pop     ebx
mov     esp, ebp
pop     ebp
retn
_main_0 endp
```

二. 流程控制语句

@for循环 (vc++ 6.0 debug版本)

```
Int main()  
{  
    inti=100;  
    for(;i;i--)  
    {  
  
    }  
    return0;  
}
```



一 法控制语句

IDA - C:\Documents and Settings\Administrator\桌面\3-2-8 for循环\Debug\3-2-8.exe

File Edit Jump Search View Debugger Options Windows Help

Library function Data Regular function Unexplored Instruction External symbol

Functions window

Function name	Segment	S
_main	.text	0
_main_0	.text	0
start	.text	0
_msg_exit	.text	0
_fast_error_exit	.text	0
_cinit	.text	0
_exit	.text	0
_exit	.text	0
_c_exit	.text	0
_c_exit	.text	0
_doexit	.text	0
_initterm	.text	0
_XcptFilter	.text	0
_xcptlookup	.text	0
_setenvp	.text	0
_setargv	.text	0
_parse_cmdline	.text	0
_crGetEnvironmentStringsA	.text	0
_ioint	.text	0
_iosterm	.text	0
sub_4021C0	.text	0
sub_402220	.text	0

Line 2 of 173

Graph overview

IDA View-A

Hex View-1

Structures

Enums

Imports

Exports

```
push    ebp
mov     ebp, esp
sub     esp, 44h
push    ebx
push    esi
push    edi
lea     edi, [ebp+var_44]
mov     ecx, 11h
mov     eax, 0CCCCCCCCh
rep stosd
mov     [ebp+var_4], 64h
jmp     short loc_40103A

loc_40103A:
cmp     [ebp+var_4], 0
jz      short loc_401042

jmp     short loc_401031

loc_401042:
xor     eax, eax
pop     edi
pop     esi
pop     ebx
mov     esp, ebp
pop     ebp
retn    _main_0_endp

loc_401031:
mov     eax, [ebp+var_4]
sub     eax, 1
mov     [ebp+var_4], eax
```

100.00% (-543,166) | (1019,304) 00001010 00401010: _main_0 (Synchronized with Hex View-1)

Output window

IDA Python v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>

Python

AU: idle Up Disk: 350B

开始 IDA_Pro_v6.8 test 3-2 IDA - C:\Documen... IDA - C:\Documen...



二. 流程控制语句

- 对于**for**循环同样有两次跳转，整个循环的判断逻辑也和**while**表示式的运算顺序一模一样，在汇编中，能清楚地看到代码逻辑的执行。



```

sub     esp, 44h
push    ebx
push    esi
push    edi
lea     edi, [ebp+var_44]
mov     ecx, 11h
mov     eax, 0CCCCCCCCh
rep stosd
mov     [ebp+var_4], 64h
jmp     short loc_40103A

```

```

loc_40103A:
cmp     [ebp+var_4], 0
jz      short loc_401042

```

```

jmp     short loc_401031

```

```

loc_401042:
xor     eax, eax
pop     edi
pop     esi
pop     ebx
mov     esp, ebp
pop     ebp
retn
_main_0 endp

```

```

loc_401031:
mov     eax, [ebp+var_4]
sub     eax, 1
mov     [ebp+var_4], eax

```

```

var_44= byte ptr -44h
var_4= dword ptr -4

```

```

push    ebp
mov     ebp, esp
sub     esp, 44h
push    ebx
push    esi
push    edi
lea     edi, [ebp+var_44]
mov     ecx, 11h
mov     eax, 0CCCCCCCCh
rep stosd
mov     [ebp+var_4], 64h

```

```

loc_40102F:
mov     eax, [ebp+var_4]
mov     ecx, [ebp+var_4]
sub     ecx, 1
mov     [ebp+var_4], ecx
test    eax, eax
jz      short loc_401041

```

```

jmp     short loc_40102F

```

```

loc_401041:
xor     eax, eax
pop     edi
pop     esi
pop     ebx
mov     esp, ebp
pop     ebp
retn
_main_0 endp

```

二. 流程控制语句

@do循环 (vc++ 6.0 debug版本)

```
Int main()  
{  
    inti=100;  
    do  
    {  
        i--;  
    }while(i);  
    return0;  
}
```



一 法胆控生五句

IDA - C:\Documents and Settings\Administrator\桌面\3-2-9 do循环\Debug\3-2-9.exe

File Edit Jump Search View Debugger Options Windows Help

Library function Data Regular function Unexplored Instruction External symbol

Functions window

Function name	Segment	S
_main	.text	0
_main_0	.text	0
start	.text	0
_msg_exit	.text	0
_fast_error_exit	.text	0
_cinit	.text	0
_exit	.text	0
_exit	.text	0
_cexit	.text	0
_c_exit	.text	0
_doexit	.text	0
_initterm	.text	0
_xcpFilter	.text	0
_xcpLookup	.text	0
_setenvp	.text	0
_setargv	.text	0
_parse_cmdline	.text	0
_crGetEnvironmentStringsA	.text	0
_ioint	.text	0
_ioter	.text	0
sub_4021C0	.text	0
sub_402220	.text	0
sub_402450	.text	0
_global_unwind2	.text	0
_unwind_handler	.text	0
_local_unwind2	.text	0
_abnormal_termination	.text	0
_NLG_Notify	.text	0
_except_handler3	.text	0
_seh_longjmp_unwind(x)	.text	0
_FF_MSGBANNER	.text	0
_NMSG_WRITE	.text	0
_GET_RTRMSG	.text	0
_malloc	.text	0
_malloc_dbg	.text	0
_nh_malloc	.text	0
_nh_malloc_dbg	.text	0

Line 2 of 173

Graph overview

Output window

Python v1.7.0 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>

Python

AU: idle Up Disk: 350B

开始 IDA_Pro_v6.8 test 3-2 IDA - C:\Documen... IDA - C:\Documen...

100.00% (-610,-27) (1054,463) 00001010 00401010: _main_0 (Synchronized with Hex View-1)

Attributes: bp-based frame

_main_0 proc near

```
var_44= byte ptr -44h
var_4= dword ptr -4

push ebp
mov ebp, esp
sub esp, 44h
push ebx
push esi
push edi
lea edi, [ebp+var_44]
mov ecx, 11h
mov eax, 0CCCCCCCCh
rep stosd
mov [ebp+var_4], 64h
```

loc_40102F:

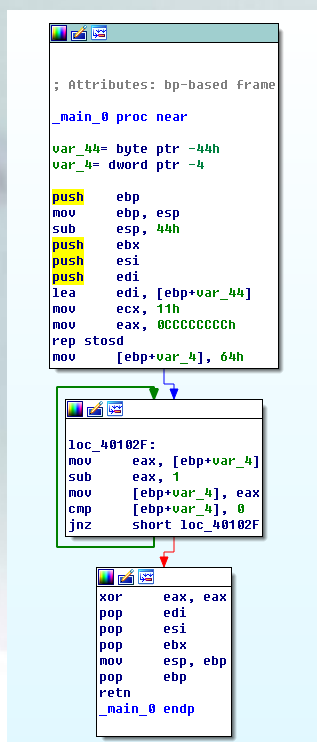
```
mov eax, [ebp+var_4]
sub eax, 1
mov [ebp+var_4], eax
cmp [ebp+var_4], 0
jnz short loc_40102F
```

xor eax, eax
pop edi
pop esi
pop ebx
mov esp, ebp
pop ebp
retn
_main_0 endp



二. 流程控制语句

② 可以看到do循环只有一次跳转，这也是为什么do循环的效率要更高一些的原因。



二. 流程控制语句

- ④ 以上就是三种循环的简单对比，在逆向分析过程中，循环代码都是很好辨认的代码

