IDA反汇编经验总结

一、IDA Pro操作指南

1. 启动界面：
    - 可以选择以`Binary file`的形式启动，这在分析shellcode，或者文件中包含额外信息时很有用
    - 可以手动指定加载基地址。如果用调试器分析程序时不是以首选基地址加载，为了与调试器保持一致，可以在这里进行手动设置。
    - 可以选择加载PE header和资源段
2. Graph模式：
    1. 使用空格在text和graph模式之间进行转换
    2. 在`Option->General`中，设置显示`Line prefixes`和`Number of Opcode Bytes(6)`
    3. 跳转时的箭头：红色：不跳转；绿色：跳转；蓝色：无条件跳转
3. IDA中存在imports, exports、strings、structures等子窗口。
4. 不同的链接类型
    1. `sub_*`表示函数的开始：`printf`、`sub_4010A0`
    2. `loc_*`表示跳转指令的目标：`loc_40107E`、`loc_401097`
    3. `offset`链接表示内存的偏移
5. 搜索：
    - 指令：`Search->test`
    - 操作码：`Search->Sequence of Bytes`
6. 跳转
    1. 在链接或者子窗口上的条目上双击，可以跳转到反汇编窗口中对应的代码位置
    2. 按键`g`可以跳转到指定位置
    3. 发生跳转后，左上角的←和→可以在跳转之间进行转移
7. 交叉引用
    1. 在函数或数据上按键`x`可以查看交叉引用
    2. Graphs功能
        - `View->Graphs`或者`工具栏上右键->Graphs`
        - `View->Graphs->Flow Chart`会在新窗口显示函数的流程图
        - 选择了一个函数之后，`User xrefs chart`，可以设置该函数的交叉引用深度，选择引用或被引用，最后显示一张引用图表
8. 协助分析
    1. 重命名：在函数或变量上右键(快捷键`n`)对其进行重命名
    2. 注释：按键`:`添加注释，按键`;`添加的注释可以在每次交叉引用该地址时进行显示。
    3. 修改操作数显示类型：右键选择不同进制或者显示为字符
    4. 将操作数设置为常量：右键，选择`Use Standard Symbolic Constant option`，可以为其设置已命名的常量名（根据MSDN中的定义）。

       使用`View->Open Subviews->Type Libraries`导入其他库文件：`mssdk`和`vs6win`是自动导入的，例如恶意软件经常使用的`ntapi`(windows nt family api)或`gnuunx`(gnu c++ unix)

5. 添加结构体信息：在`Structures`子窗口中按`Insert`键添加MSDN中定义的结构体。之后在操作数上右键(或者按键`t`)，可以选择`Structure offset`
6. 重新定义字节类型：快捷键`u`转换为纯字节数据，`c`转换为代码，`d`转换为数据，`a`转换为ascii字符串
7. ida分析错误：
    - 函数没有识别出来：按键`p`手动标注函数
    - BP based frame没有识别出来：按键`alt+p`进行设置，选择`Saved registers`为`0x4`
    - 错误的交叉引用识别：按键`o`将链接地址转换为数字

9. 常见脚本函数
    - `PatchByte(位置，目标字节)`

二、常见指令

1. 看见多个`xor，or，and，shl，ror，shr，rol`指令，大概率是遇到了加密或者编码函数，直接进行标注，无必要不需要分析具体算法。
2. `pusha`：把AX, CX, DX, BX, SP, BP, SI, DI压栈

   `pushad`：把EAX, ECX, EDX, EBX, ESP, ESI, EDI压栈

   编译器很少使用这两个指令，所以看见它们通常表示这是手动编写的汇编代码或shellcode
3. `fld`、`fstp`：浮点数操作。把操作数推入FPU栈顶寄存器；从栈顶寄存器ST(0)弹出数据到目标操作数
4. `test`指令≈`and`指令，主要关注`ZF`标志

   `cmp` 指令≈`sub`指令，主要关注`ZF`和`CF`标志
5. `rep`+`movsx，cmpsx，stosx，scasx(x = b，w，d)`：“字符串指令”

   `edi`为目的寄存器，`esi`为源寄存器，`DF`指示了移动方向(0是正向)

   `ecx`表示重复次数

重复前缀	停止条件1	停止条件2
`rep`	ECX = 0	--
`repe，repz`	ECX = 0	ZF = 0
`repne，repnz`	ECX = 0	ZF = 1

指令	等价函数	说明
`rep movsb`	`memcpy`	Move RCX bytes from [RSI] to [RDI].
`repe cmpsb`	`memcmp`	Find non-matching bytes in [RDI] and [RSI].
`rep stosb`	`memset`	Fill RCX bytes at [RDI] with AL. 经常用来初始化一块内存
`repne scasb`		Find AL, starting at [RDI].

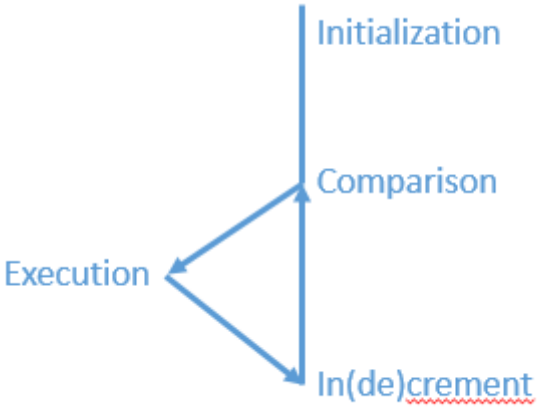
`repne scasb`还可用于计算字符串长度，等价于`strlen`。

```
mov     edi, offset buf     ; "hello"
xor     eax, eax            ; 字符串结尾
or      ecx, 0FFFFFFFFh     ; 设置ECX为-1
repne scasb                 ; 搜索字符串结尾，ECX会不断减1
not     ecx                 ; ECX取反
dec     ecx                 ; ECX减1，为字符串长度
```

三、代码分析

1. 在graph模式下可以更容易识别出代码中的条件跳转和循环结构。

   可以选择`View->Graphs->Flow Chart`，显示地更清晰。



2. 函数调用规范

	cdel	stdcall	fastcall
参数压入顺序	从右至左	从右至左	#1: ECX #2: EDX 其他: 从右至左压入
栈清理	调用者	被调用者	调用者
返回值	EAX	EAX	EAX
其他		Windows API标准	各编译器实现情况不同 效率更高

3. 链接表结构

> To recognize a linked list, you must first recognize that some object contains a pointer that points to another object of the same type.

```
004010CC        mov     eax, [ebp+var_4]
004010CF        mov     eax, [eax+4]
004010D2        mov     [ebp+var_4], eax
```

4.

四、经验与教训

1. 在之前的基础静态、动态分析过程中，已经对程序功能有了猜测，在此基础上进行代码分析
2. 面对一个函数，要先查看它的参数，交叉引用，流程图，返回值，建立初步认知

3. 及时对已知功能（哪怕是猜测）的函数及变量进行重命名，添加注释
4. 如果一个函数特别大，可以先查看所有的 `call` 指令
5. 根据实验7-3，如果有某个变量找不到数据来源，可以查看函数开头的变量表，是不是由于IDA的自动分析，数据传入了与该变量相邻的变量，而我们没有发现