

**汇编语言与逆向技术**

**课程实验报告**

**实验六：import\_export**

****

学 院 网络空间安全

专 业 信息安全

学 号 2212998

姓 名 胡博浩

班 级­­­­ 2022级信息安全

1. **描述PE文件的输入表的作用和数据结构：**

（1）作用：

导入表用于描述程序运行时需要使用的外部函数，即程序引用的其他模块的函数。

1. 记录了一个模块引用其他模块的函数信息。

2. 允许程序在运行时动态链接到所需的外部模块。

3. 支持延迟加载，只在需要时加载模块。

（2）结构：

1. IMAGE\_IMPORT\_DESCRIPTOR（导入描述符）:

OriginalFirstThunk（指向原始未绑定IAT的RVA）。

TimeDateStamp（文件创建时的时间戳）。

ForwarderChain（第一个转发引用的索引）。

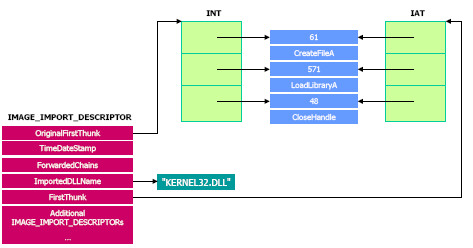
2. IMAGE\_THUNK\_DATA（导入函数的RVA或转发引用）:

Function（对于非转发的导入，是导入函数的RVA或转发引用）。

3. IMAGE\_IMPORT\_BY\_NAME（按名称导入的函数）:

Hint（对导出名称指针表的索引）。

Name（导入函数的名称）。



1. **描述PE文件的输出表的作用和数据结构**

（1）作用：

导出表用于描述一个模块中可以被其他模块引用的函数和数据。

1. 记录了一个模块中对外部可见的函数和数据。

2. 提供了标准接口，允许其他模块调用当前模块的功能。

3. 支持模块间通信，促进了模块化设计和代码重用。

4. 为插件系统提供基础，使得程序可以更容易地实现插件功能。

（2）结构：

1. IMAGE\_EXPORT\_DIRECTORY（导出目录）:

Characteristics（导出的特性标志）。

TimeDateStamp（文件创建时的时间戳）。

MajorVersion, MinorVersion（主版本号和次版本号）。

2. AddressOfFunctions（导出函数或转发器的RVA数组）:

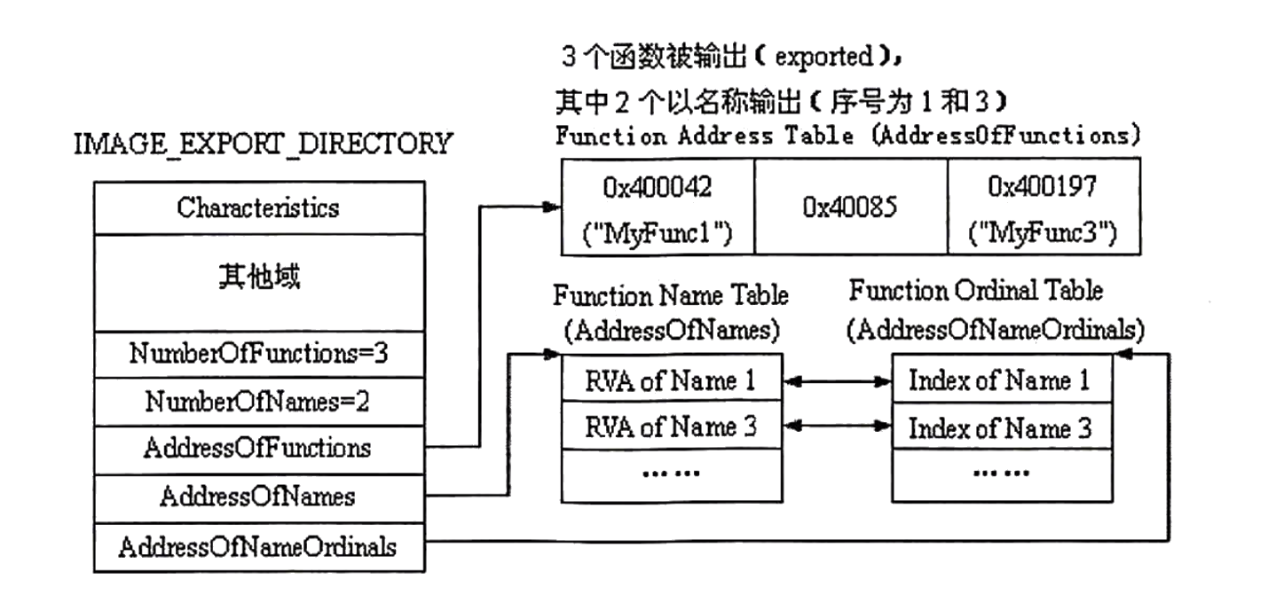
包含导出函数或转发器的RVA的数组。

3. AddressOfNames（导出函数名称的ASCII字符串的RVA数组）:

包含导出函数名称的ASCII字符串的RVA的数组。

4. AddressOfNameOrdinals（导出函数的序数值的数组）:

包含导出函数序数值的数组。



1. **实验步骤**

1. import\_export.asm的源代码和注释:

.386

.model flat, stdcall

option casemap :none

include \masm32\include\windows.inc

include \masm32\include\kernel32.inc

include \masm32\include\masm32.inc

includelib \masm32\lib\kernel32.lib

includelib \masm32\lib\masm32.lib

.data

Rva2Raw PROTO :DWORD ; 将RVA转换为文件偏移RAW的过程，参数为RVA

noExport BYTE "none",0

str0 BYTE "Please input a PE file:",0

str1 BYTE "Import table:",0

str2 BYTE "Export table:",0

strn BYTE 0ah,0h ; 换行符

path BYTE 20 DUP(0) ; 文件路径

filehandle DWORD 0 ; 文件句柄

buf BYTE 9999 DUP(0) ; 临时缓冲区

var1 DWORD 0 ; 中间变量1

var2 DWORD 0 ; 中间变量2

NThandle DWORD 0 ; NT头

OPhandle DWORD 0 ; 可选头首地址

SEhandle DWORD 0 ; 节区头首地址

EXaddr DWORD 0 ; 导出表地址

IMaddr DWORD 0 ; 导入表地址

SEadds DWORD 30h DUP(0) ; 节结构体

INTaddr DWORD 0 ; 导入地址表(IAT)地址

SEnum WORD 0 ; 节的数量

EXnamebase DWORD 0 ; 导出表名字数组首地址

node DWORD 0 ; 指向导入表的某个导入信息目录(IID)地址

Lastnode DWORD 0 ; 上一个IID地址

.code

; 将RVA转换为RAW的过程

Rva2Raw PROC rva:DWORD

mov eax,rva ; 将需要转换的地址存储在eax中

mov ecx,0 ; 初始化计数器ecx

L3:

shr ecx,2

cmp cx,SEnum ; 将ecx与节头的数量比较

JE LEnd3 ; 如果相等，搜索完成，跳转到LEnd3

shl ecx,2 ; ecx = ecx \* 4（IMAGE\_SECTION\_HEADER的大小）

mov edx,DWORD PTR SEadds[ecx\*4] ; 在edx中获取节的大小，地址在edi中

mov edi,DWORD PTR SEadds[ecx\*4+4h]

add edx,edi ; edx为上限，edi为下限

cmp rva,edi

JL LNext3

cmp rva,edx

JG LNext3

JMP LEnd3

LNext3:

add ecx,4 ; 如果没找到，则移到下一个节的信息结构体

JMP L3

LEnd3:

sub eax,DWORD PTR SEadds[ecx\*4+4h] ; 如果找到了，则减去RVA，加上RAW节点

add eax,DWORD PTR SEadds[ecx\*4+0ch]

ret

Rva2Raw ENDP

; 读取导入表中INT的函数

readFunc PROC

invoke Rva2Raw,eax ; eax是INTname的地址RVA

add eax,OFFSET buf ; 读取PE文件的基地址

mov var2,eax ; 临时保存

LNext:

mov eax,var2 ; 恢复eax

mov esi,DWORD PTR [eax] ; 读取名字地址

cmp esi,0 ; 与0比较

JE Ret1

mov edi,esi

and edi,0ff000000h ; 如果相等，则返回

cmp edi,80000000h

JNE L1

add eax,4h

mov esi,DWORD PTR [eax]

L1:

invoke Rva2Raw,esi ; 将读取的地址转换为RAW

add eax,OFFSET buf ; 读取PE文件的基地址

add eax,2 ; 前两个字节是序数，直接跳过

mov var1,eax

invoke StdOut,var1

invoke StdOut,addr strn

add var2,4 ; 更新eax的指向

JMP LNext

Ret1:

ret

readFunc ENDP

; 读取导入表中的IED

readIED PROC

add esi,20h

mov eax,DWORD PTR [esi]

invoke Rva2Raw,eax ; 将name的RVA地址转换为RAW

mov EXnamebase,eax ; 保存名字数组的基地址

mov Lastnode,eax ; 保存前一个名字指针的地址

LNext2:

add eax,OFFSET buf ; 读取PE文件的基地址

mov ebx,DWORD PTR [eax] ; 读取

mov edi,ebx ; 将edx放入edi中，与lastpointer比较

sub edi,Lastnode

cmp edi,0 ; 如果单调增，则还有一个要读取，否则说明结束

JL Ret2

invoke Rva2Raw,ebx ; 读取的字符串地址也是RVA，需要转换为RAW

add eax,OFFSET buf

invoke StdOut,eax ; 输出

invoke StdOut,addr strn

add EXnamebase,4 ; 下一个地址

mov eax,EXnamebase ; 恢复eax

JMP LNext2

Ret2:

ret

readIED ENDP

start:

invoke StdOut,addr str0

invoke StdIn,addr path,20

invoke StdOut,addr str1

invoke StdOut,addr strn

invoke CreateFile,addr path,GENERIC\_READ,FILE\_SHARE\_READ,0,

OPEN\_EXISTING,FILE\_ATTRIBUTE\_ARCHIVE,0

mov filehandle,eax

invoke SetFilePointer, filehandle,0,0,FILE\_BEGIN

invoke ReadFile, filehandle,addr buf,9999,0,0

mov esi,OFFSET buf

add esi,3ch

mov eax,DWORD PTR [esi]

mov esi,OFFSET buf

add esi,eax

mov ax,WORD PTR [esi+6h]

mov SEnum,ax

mov NThandle,esi

mov ebx,NThandle

add ebx,18h

mov OPhandle,ebx

add esi,14h

mov eax,0

mov ax,WORD PTR [esi]

mov ebx,OPhandle

add ebx,eax

mov SEhandle,ebx

mov esi,OPhandle

add esi,60h

mov eax,DWORD PTR [esi]

mov EXaddr,eax

add esi,8h

mov eax,DWORD PTR [esi]

mov IMaddr,eax

; 遍历节头表

mov esi,SEhandle

mov ecx,0

Lsec:

mov edx,DWORD PTR [esi]

cmp edx,0 ; 比较是否有节头内容

JE Lfind

add esi,8h

mov eax,DWORD PTR [esi] ; 虚拟内存大小

mov SEadds[ecx],eax

add esi,4h

mov eax,DWORD PTR [esi] ; 虚拟内存基址

mov SEadds[ecx+4h],eax

add esi,4h

mov eax,DWORD PTR [esi] ; 文件节区大小

mov SEadds[ecx+8h],eax

add esi,4h

mov eax,DWORD PTR [esi] ; 文件节区基址偏移

mov SEadds[ecx+0ch],eax

add ecx,10h

add esi,14h

JMP Lsec

Lfind:

; 提取导入表导出表的偏移和地址，准备转换

; 遍历节头，找到具体的节。转化成文件偏移，取值，读取

mov ebx,IMaddr

invoke Rva2Raw,ebx ; ebx为导入表的文件偏移（返回值为eax）

mov esi,OFFSET buf

add esi,eax

mov node,esi ; node存储IID每个元素的首地址

NextCh:

mov esi,node ; 将esi设为每个IID首地址（RAW）读取

mov edi,DWORD PTR [esi+0ch] ; 检测是否读完

cmp edi,0

JE ReadEnd

mov eax,DWORD PTR [esi]

mov INTaddr,eax

add esi,0ch

mov eax,DWORD PTR [esi]

invoke Rva2Raw,eax

add eax,OFFSET buf

invoke StdOut,eax

invoke StdOut,addr strn

add esi,4

mov eax,INTaddr

cmp INTaddr,0

JNE LEmpty

mov eax,DWORD PTR [esi]

LEmpty:

call readFunc ; 读取当前IID元素的函数

add node,14h

JMP NextCh

ReadEnd:

invoke StdOut,addr str2

invoke StdOut,addr strn

; 读取导出表

mov ebx,EXaddr

cmp ebx,0

JE NO ; 检查导出表是否为0

invoke Rva2Raw,ebx ; eax为返回值

mov esi,OFFSET buf

add esi,eax

call readIED ; 将esi置为IED的首地址（RAW）读取

JMP Lendc

NO:

invoke StdOut,addr noExport

Lendc:

invoke CloseHandle,filehandle

invoke ExitProcess,0

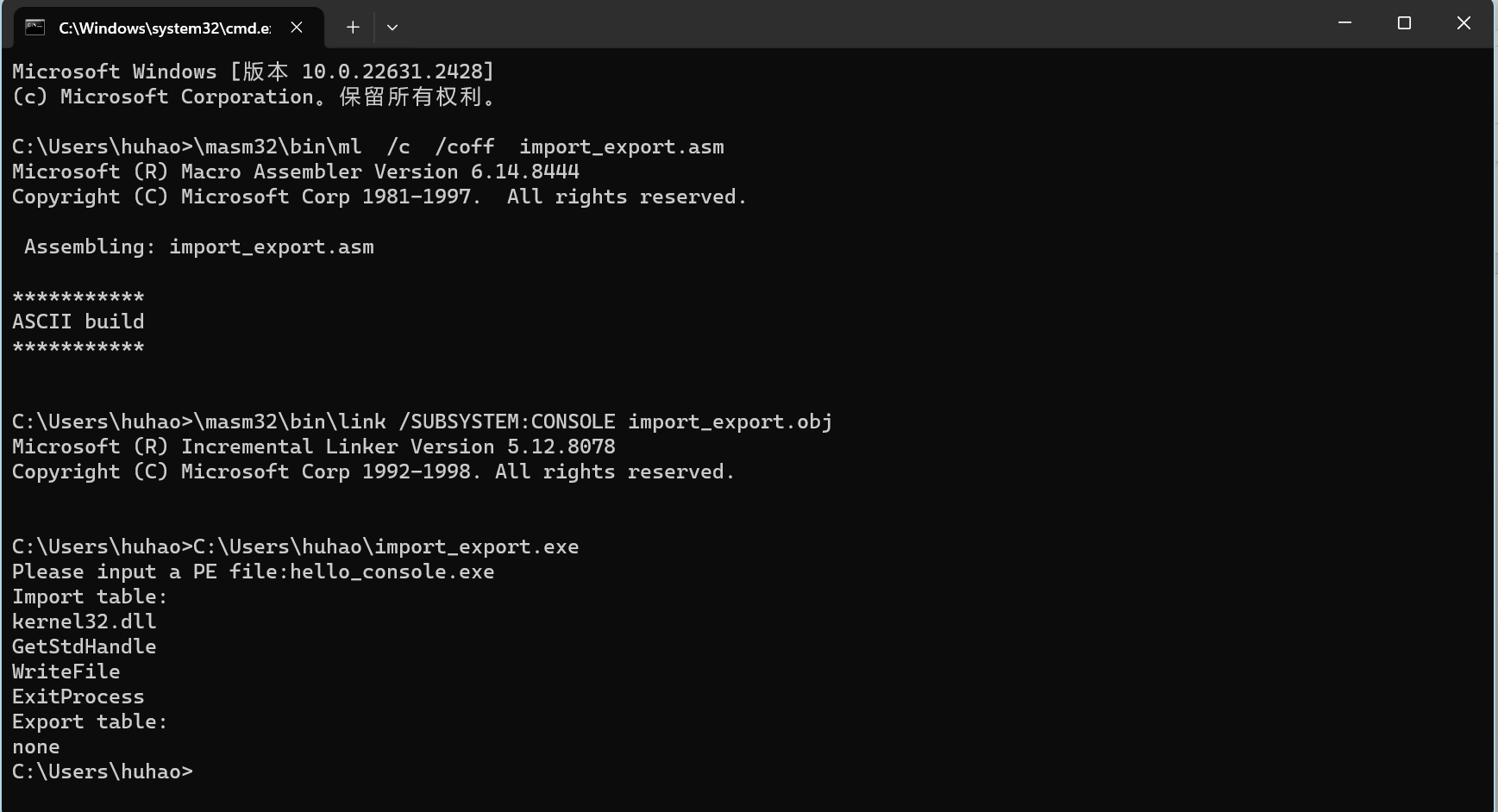
END start

2. 源代码的编译和链接过程说明

1）使用ml将bubble\_sort.asm文件汇编到dec2hex.obj目标文件，编译命令：“\masm32\bin\ml /c /coff import\_export.asm”

2）使用link将目标文件dec2hex.obj链接成dec2hex.exe可执行文件，链接命令：“\masm32\bin\link /SUBSYSTEM:CONSOLE import\_export.obj”

3. import\_export.exe运行截图

****

**五、讨论输入表的安全问题**

输入表在正常情况下是用于程序的正常执行的，但在某些情况下，输入表可能会成为恶意软件的攻击目标。以下是一些与输入表相关的安全问题。

1. 缓冲区溢出： 如果输入表中的数据未被正确验证，可能导致缓冲区溢出，使得恶意输入可以覆盖程序的关键数据结构或代码段，从而执行恶意代码。

2. 拒绝服务攻击： 恶意输入可能包含设计用于导致程序崩溃或变得不稳定的数据。这可能导致拒绝服务攻击，使得程序无法正常运行。

3. 注入攻击： 如果输入表中的数据用于构建查询或命令，未经适当验证的输入可能导致注入攻击。这种攻击可能在数据库查询中引入恶意代码，从而访问敏感信息。

4. 类型错误： 用户提供的输入可能与预期的不同，导致类型错误。这可能导致程序行为异常或执行不受控制的操作。

5. 敏感信息泄露： 如果输入表包含与敏感信息相关的字段，不正确处理这些字段可能导致敏感信息泄露。

为了防范这些安全问题，程序应该实施严格的输入验证和过滤，确保用户提供的数据符合预期格式和范围。此外，使用参数化查询和避免直接拼接用户输入以构建查询语句，以防止注入攻击。在处理输入时，要遵循最小特权原则，只授予程序执行所需任务的最低权限。