第3章

PE 文件结构分析

3.1 _______ 实验概述

本章实验旨在让学生深入了解 Windows 操作系统下的可执行文件格式(PE/PE+),并 通过学习各种 PE 编辑查看工具,详细了解 PE 文件的结构和组成部分。通过重点分析 PE 文件头、引入表、引出表以及资源表等关键部分,学生将深入了解 PE 文件的内部结构和 功能。

在本章中,学生将熟悉各种 PE 编辑查看工具,从而能够对 PE 文件进行查看和分析。 然后,本章重点分析了 PE 文件的各部分,包括文件头、节表、导入表、导出表以及资源表等, 通过分析这些部分,学生将了解 PE 文件的组织结构和功能。此外,学生将通过自己动手打 造一个尽可能小的 PE 文件的实践,加深对 PE 文件格式的理解,并学会如何通过编辑工具 来操作和修改 PE 文件。通过本章实验,学生将能够全面掌握 PE 文件的结构和格式,为进 一步学习逆向工程和漏洞分析奠定坚实的基础。

3.2 实验预备知识与基础

3.2.1 PE 查看、编辑与调试工具介绍

本次实验将使用相关工具对 PE 文件进行查看、编辑与调试。

1. PEview

PEview能够快速简便地查看可移植可执行文件(PE),以及组件对象文件格式(COFF)文件。该工具支持的文件类型包括EXE、DLL、OBJ、LIB、DBG等,可以以树状目录的方式显示文件的头部、节、引入表、引出表和资源等信息,以及更具体的各字段的含义。

2. StudyPE+

StudyPE+是一款国产的 PE 查看/分析集成工具,支持 PE32 与 PE32+,能够显示 PE 文件的重要字段(但不像 PEview 一样显示所有字段)。StudyPE+提供了许多实用的功能,例如丰富的 PE 编辑功能、RVA FOA 互相转换功能、PE 反汇编及反汇编编辑功能、PE 内 多种数据搜索功能、有限的查壳功能等。

3. 010Editor

010Editor 是一款专业的文本和十六进制编辑器,能够快速地编辑计算机上任何文件的 内容。该软件可以编辑文本文件,包括 Unicode 文件、批处理文件、C/C++、XML 等;而在 编辑二进制文件时,010Editor 不仅可以查看和编辑二进制文件的单个字节,还可以基于官 网提供的模板对各种类型的文件格式化显示,例如 PE 文件;此外,010Editor 还能对内存中 的数据进行编辑。

4. OllyDbg

OllyDbg(www.ollydbg.de)是 Windows 系统下的可视化的用户模式调试器,能够调试 32 位程序。OllyDbg 结合了动态调试与静态分析,它的反汇编能力很强,能够自动分析函 数、循环语句、字符串等,可以识别数千个 API,并注释其参数。它具有用户友好的界面,其 功能可以由第三方插件扩展。其版本 1.10 为 1.x 系列的最终发布版本。2.0 版本于 2010 年 6 月发布。

3.2.2 函数引入机制

1. 引入函数节

代码复用是程序的重要特性,PE文件也是如此,PE文件中使用的函数可能来自其他

库,例如 ExitProcess。这种被某模块调用但又不在调用者 模块中的函数称为引入函数。PE 文件通过引入函数机制 从其他(系统或第三方自定义的)DLL 中引入函数,例如 user32.dll、kernel32.dll 等,存储这种引入函数机制的节称 为引入函数节,节名一般为.rdata。图 3-1 展示了引入函数 节的结构。

● SECTION .rdata IMPORT Name Table IMPORT Address Table IMPORT Directory Table IMPORT Hints/Names & DLL Names 图 3-1 引入函数节

2. 引入机制

函数引入机制主要由 3 个重要的数据结构完成,如图 3-1 所示,分别是 IMPORT Directory Table(IDT)、IMPORT Name Table(INT)、IMPORT Address Table(IAT)。

IDT 是一个 IMAGE_IMPORT_DESCRIPORTs 数组,如图 3-2 所示,每个数组元素对 应一个 DLL,以全零的数组元素结束数组。每个数组元素有 5 个 DWORD 大小的项,第 1

pFile	Data	Description	Value
00000614	00002000	Import Name Table RVA	
00000618	00000000	Time Date Stamp	
0000061C	00000000	Forwarder Chain	
00000620	00002072	Name RVA	kernel32.dll
00000624	00002000	Import Address Table RVA	
00000628	00002058	Import Name Table RVA	
0000062C	00000000	Time Date Stamp	
00000630	00000000	Forwarder Chain	
00000634	0000209A	Name RVA	user32.dll
00000638	00002008	Import Address Table RVA	
0000063C	00000000		
00000640	00000000		
00000644	00000000		
00000648	00000000		
0000064C	00000000		

图 3-2 引入函数表

项为 INT 的 RVA(虚拟地址相对偏移),第5项为 IAT 的 RVA,第4项对应 DLL 名称字符 串,告诉加载器这个结构对应的 DLL。

INT 与 IAT 在文件上是相同的,都是一系列的元素大小为 DWORD 的数组,如图 3-3 所示,每个 DWORD 通常是指向引入函数的 Hint 及名称字符串的 RVA(如最高位为 1,即 第一字节为 80,则指引入函数的序号),以全零结束。

pFile	Data	Description	Value
00000600	00002064	Hint/Name RVA	0000 ExitProcess
00000604	00000000	End of Imports	kernel32.dll
00000658	0000208C	Hint/Name RVA	019D MessageBoxA
0000065C	00002080	Hint/Name RVA	0262 wsprintfA
00000660	00000000	End of Imports	user32.dll



IAT 与 INT 的区别在于,如图 3-4 所示,当 PE 文件加载到内存中时,IAT 中原本存放 引入函数名称的 DWORD 会替换为该函数的内存地址,这样 PE 文件运行时可以通过 IAT 在内存中找到相应函数。



图 3-4 文件与内存中的引入表

3.2.3 函数引出机制

1. 引出函数节

引出函数节的节名一般为.edata,用来描述本文件引出函数的列表等信息及各函数具体代码位置。引出函数节的具体结构如图 3-5 所示。

2. 引出函数机制

引出函数机制主要由 3 个数据结构完成。如图 3-6 所示, AddressOfFunctions(对应 PEview 中的 EXPORT Address Table)存放函数地址, AddressOfNames(对应 PEview 中

File View Go Help					
		3			
- MS-DOS Stub Program	^	pFile	Data	Description	Value
		00011E00	00000000	Characteristics	
IMAGE_SECTION_HEADER .text		00011E04	00000000	Time Date Stamp	
IMAGE_SECTION_HEADER .data		00011E08	0000	Major Version	
IMAGE_SECTION_HEADER .tls		00011E0A	0000	Minor Version	
— IMAGE_SECTION_HEADER .idata		00011E0C	00015064	Name RVA	Labeler.dll
— IMAGE_SECTION_HEADER .edata		00011E10	00000001	Ordinal Base	
IMAGE_SECTION_HEADER .rsrc		00011E14	00000006	Number of Functions	
IMAGE_SECTION_HEADER .reloc		00011E18	00000006	Number of Names	-
- SECTION .text		00011E1C	00015028	Address Table RVA	
- SECTION .data		00011E20	00015040	Name Pointer Table RVA	
- SECTION .tls		00011E24	00015058	Ordinal Table RVA	
ia-SECTION .idata					-
SECTION .edata					
IMAGE EXPORT DIRECTORY					
- EXPORT Address Table					
EXPORT Name Pointer Table					
- EXPORT Ordinal Table					
- EXPORT Names	~	<			
Viewing IMAGE_EXPORT_DIRECTORY					

图 3-5 引出函数节具体结构

的 EXPORT Name Pointer Table)存放函数名所在地址, AddressOfNameOrdinals(对应 PEview 中的 EXPORT Ordinal Table)存放每个函数地址在函数地址表中对应的序号。



如果希望通过函数名获取引出函数的地址,需要经过以下流程:

(1) 在 AddressOfNames 找到目标函数的函数名地址,并记下该数组序号 X;

(2) 定位 AddressOfNameOrdinals 的第 X 项,得到序号 Y;

(3) 定位 AddressOfFunctions 的第 Y 项,获得该函数的 RVA 函数地址。

3.2.4 资源节机制

1. 资源节

资源节的节名一般为.rsrc,放有图标、对话框等程序需要的资源。资源节以树状结构组织,它有一个主目录,主目录下又有子目录,子目录下可以是子目录或数据。通常有3层目录(资源类型、资源标识符、资源语言 ID),第4 层是具体的资源。图 3-7 展示了资源节的树状结构,即资源树。

2. 资源定位机制

资源一般使用树来保存,通常包含3层,最高层是类型,其次是名字,最后是语言。资源 的定位遵循以下步骤。

(1) 定位资源节开始的位置,首先是一个 IMAGE_RESOURCE_DIRECTORY 结构,

软件安全实践



后面紧跟着 IMAGE_RESOURCE_DIRECTORY_ENTRY 数组,这个数组的每个元素代表的资源类型不同。

(2)通过每个元素,可以找到第二层 IMAGE_RESOURCE_DIRECTORY,后面紧跟着 IMAGE_RESOURCE_DIRECTORY_ENTRY 数组,这个数组的每个元素代表的资源名字 不同。

(3) 然后可以找到第三层 IMAGE_RESOURCE_DIRECTORY, 后面同样紧跟着 IMAGE_RESOURCE_DIRECTORY_ENTRY 数组,这个数组的每个元素代表的资源语言 不同,且直接指向最后的资源(IMAGE_RESOURCE_DATA_ENTRY)。

以上三类 IMAGE_RESOURCE_DIRECTORY 在 PEview 中分别带有 Type、 NameID、Language 的后缀,如图 3-8 左侧边栏所示。

(4) 最后通过每个 IMAGE_RESOURCE_DIRECTORY_ENTRY 找到每个 IMAGE_ RESOURCE_DATA_ENTRY,从而找到每个真正的资源。

3.2.5 重定位机制

重定位节存放了一个重定位表,定位了代码中使用了绝对地址的地方。若装载器在程 序默认的基地址加载映像文件,就不需要重定位,否则需要通过重定位表做一些调整,步骤 如下。

(1) 计算地址差异 delta。操作系统加载程序会计算默认的基地址(PE 头的 ImageBase 字段)与实际加载的映像文件的基地址的差异(delta)。

(2) 根据重定位的类型,将这个 delta 应用到重定位表指向的需要修改的地方。

重定位节是一个 IMAGE_BASE_RELOCATION 结构,该结构的每一项如表 3-1

File View Go Help						
IMAGE_SECTION_HEADER .edata	^	pFilo	Data	Description	- Value	
IMAGE_SECTION_HEADER .rsrc		000CF600	00000000	Characteristics		
IMAGE_SECTION_HEADER . reloc		000CF604	00000000	Time Date Stamp		
SECTION .text		000CF608	0004	Major Version	IMAGE_RESO	DURCE_DIRECTORY
SECTION .data		000CF60A	0000	Minor Version		
SECTION .tls		000CF60C	0001	Number of Named Entries		
SECTION .rdata		000CF60E	0009	Number of ID Entries		
SECTION .idata		000CF610	80001900	Name		
SECTION .edata		000CF614	80000060	Offset to DIRECTORY	KNOWNRESTYPE	
SECTION .rore		000CF618	00000001	ID		
IMAGE_RESOURCE_DIRECTORY Type		000CF61C	80000098	Offset to DIRECTORY	CURSOR	
- IMAGE_RESOURCE_DIRECTORY NameID		000CF620	00000002	ID		
 IMAGE_RESOURCE_DIRECTORY Language 		000CF624	800000B0	Offset to DIRECTORY	BITMAP	
- IMAGE_RESOURCE_DATA_ENTRY		000CF628	00000003	ID		
- IMAGE_RESOURCE_DIRECTORY_STRING		000CF62C	800001A0	Offset to DIRECTORY	ICON	
-KNOWNRESTYPE RES_ERRCODE 0804		000CF630	00000004	ID		IMAGE RESOURCE DIRECTORY ENTRY
KNOWNRESTYPE RES_EXCEPTNS 0804		000CF634	80000270	Offset to DIRECTORY	MENU	IMAGE_RESOURCE_DIRECTORT_ENTRY
-KNOWNRESTYPE RES_KNOWN 0804		000CF638	00000005	ID		
KNOWNRESTYPE RES_LOADDLL 0804		000CF63C	80000288	Offset to DIRECTORY	DIALOG	
-KNOWNRESTYPE RES_VXD 0804		000CF640	A0000000	ID		
		000CF644	800003F0	Offset to DIRECTORY	RCDATA	
-BITMAP ABOUTLOGO 0804		000CF648	0000000C	ID		
- BITMAP APPEAR 0804		000CF64C	80000408	Offset to DIRECTORY	GROUP_CURSOR	
BITMAP BREAKS 0804		000CF650	0000000E	ID		
- BITMAP CLOSE 0804		000CF654	80000420	Offset to DIRECTORY	GROUP_ICON	
-BITMAP CPU 0804		000CF658	00000010	ID		
-BITMAP GOTO 0804		000CF65C	800004E8	Offset to DIRECTORY	VERSION	
-BITMAP HANDLES 0804						1
BITMAP HELP 0804	~	<	_)
Viewing IMAGE RESOLIRCE DIRECTORY Type						

图 3-8 资源节示例

所示。

表 3-1 重定位节的数据结构

顺序	名字	大小(字节)	描述	
1	VirtualAddress	4	重定位数据开始的 RVA 地址	
2	SizeOfBlock	4	本结构大小	
3	TypeOffset[]	不定	重定项数组,每个元素占2字节	

IMAGE_BASE_RELOCATION 的每项都代表了一个 4K(一页)大小的内存区域中需 要重定位的地址。图 3-9 是 kernel32.dll 的重定位表,其中定位项的个数的计算方式为, SizeOfBlock 的大小减去前两项的字节数 8 得到 TypeOffset 数组的大小,再除以 2 得到定 位项的个数;定位项每项为 16 位(4 字节),高 4 位代表重定位的类型,剩下的 12 位代表页 内的偏移量,加上页地址 VirtualAddress 就得到了具体的内存地址。

🖏 PEview - C:\WINDOWS\system32\kernel32.dll						
File View Go Help						
⊡-kernel32.dll	pFile	Data	Description	Value		
IMAGE_DOS_HEADER	00092000	00010000	RVA of Block			
- MS-DOS Stub Program	00092004	0000008C	Size of Block			
- IMAGE_NT_HEADERS	00092008	3018	Type RVA	00010018 IMAGE_REL_BASED_HIGHLOW		
IMAGE_SECTION_HEADER .text	0009200A	301 D	Type RVA	0001001D IMAGE_REL_BASED_HIGHLOW		
- IMAGE_SECTION_HEADER .rdata	0009200C	302F	Type RVA	0001002F IMAGE_REL_BASED_HIGHLOW		
- IMAGE_SECTION_HEADER .data	0009200E	3047	Type RVA	00010047 IMAGE_REL_BASED_HIGHLOW		
- IMAGE_SECTION_HEADER .didat	00092010	3108	Type RVA	00010108 IMAGE_REL_BASED_HIGHLOW		
- IMAGE_SECTION_HEADER .rsrc	00092012	310D	Type RVA	0001010D IMAGE_REL_BASED_HIGHLOW		
- IMAGE_SECTION_HEADER .reloc	00092014	311F	Type RVA	0001011F IMAGE_REL_BASED_HIGHLOW		
- SECTION .text	00092016	3137	Type RVA	00010137 IMAGE_REL_BASED_HIGHLOW		
	00092018	32B0	Type RVA	00010280 IMAGE_REL_BASED_HIGHLOW		
- SECTION .data	0009201A	32C6	Type RVA	000102C6 IMAGE_REL_BASED_HIGHLOW		
	0009201C	3356	Type RVA	00010356 IMAGE_REL_BASED_HIGHLOW		
- SECTION .rsrc	0009201E	34B2	Type RVA	000104B2 IMAGE_REL_BASED_HIGHLOW		
- SECTION .reloc	00092020	352A	Type RVA	0001052A IMAGE_REL_BASED_HIGHLOW		
MAGE_BASE_RELOCATION	00092022	354B	Type RVA	0001054B IMAGE_REL_BASED_HIGHLOW		
	00092024	3657	Type RVA	00010657 IMAGE_REL_BASED_HIGHLOW		
	00092026	3678	Type RVA	00010678 IMAGE_REL_BASED_HIGHLOW		
	00000000	2007	T			

图 3-9 重定位表示例

3.3 PE 查看、编辑与调试工具的用法

3.3.1 实验目的

了解 PE 编辑查看与调试工具的用法, 了解 PE 文件在磁盘上的结构与在内存上的结构。

通过本实验,学生能够查看 PE 文件的各部分,包括文件头、节表、导入表、导出表以及资源表等,并可以进行调试和分析,这有助于深入理解 Windows 可执行文件的工作原理和运行机制。

3.3.2 实验内容及实验环境

1. 实验内容

(1)使用二进制查看工具 PEview 观察 PE 文件例子程序 hello25.exe 的十六进制数据, 并定位其中重要数据结构。

(2) 使用 StudyPE+观察 PE32+格式的目标程序 pe32+.exe, 了解 32 位 PE 程序与 64 位 PE 程序的差异。

(3) 使用 OllyDbg 对 hello25.exe 进行初步调试,初步了解 OllyDbg 的用法,理解该程序功能结构,在内存中观察该程序的完整结构。

(4) 使用 PE 编辑工具 010Editor 修改 hello25.exe,使得该程序仅弹出第二个对话框。

2. 实验环境

(1) 系统: Windows XP 版本及以上的操作系统,实机、虚拟机均可。

(2) 工具: OllyDbg1.10、010Editor、PEview、StudyPE+。

3.3.3 实验步骤

1. 观察 PE 文件示例程序 hello25.exe 的十六进制数据

(1) 用 PEview 打开示例程序。

使用 PEview 打开示例程序 hello25.exe(通过菜单栏的 File→Open 打开示例程序或者 直接将示例程序拖入打开的 OllyDbg 中),如图 3-10 所示,左侧的目录中显示了 hello25.exe 的结构,单击即可查看;右侧是对应的十六进制数据。

可以看到,该 PE 文件由 MZ 头部(DOS_HEADER)、DOS Stub、PE 文件头(NT_HEADER)、可选文件头、节表、节组成,其中节分为代码节(.text),引入函数节(.rdata)与数据节(.data)。

(2) 观察 PE 文件头。

PEview 左侧目录显示,PE 文件头由签名(0x4550,即 PE)、文件头与可选文件头组成。 在目录中单击各组成部分可查看详细信息,例如单击可选文件头,如图 3-11 所示,右侧显示 了可选文件头的各个字段,包括入口点、映像基址、对齐粒度等重要的文件信息。

🔍 PEview - Falabatello25 ave				- 0	×
File View Go Help					
30000					
⊟-hello25.exe	pFile	Raw	Data	Value	^
-IMAGE_DOS_HEADER	00000000	4D 5A 90 00 03 00 00 00	04 00 00 00 FF FF 00 0	MZ	
- MS-DOS Stub Program	00000010	B8 00 00 00 00 00 00 00	40 00 00 00 00 00 00 0]	
- IMAGE_NT_HEADERS	00000020	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 0)	
- Signature	00000030	00 00 00 00 00 00 00 00	00 00 00 00 80 00 00 0		
IMAGE_FILE_HEADER	00000040	0E 1F BA 0E 00 B4 09 CD	21 B8 01 4C CD 21 54 6	3IL.!Th	
- IMAGE_OPTIONAL_HEADER	00000050	69 73 20 70 72 6F 67 72	61 6D 20 63 61 6E 6E 6	= is program canno	
IMAGE_SECTION_HEADER .text	00000060	74 20 62 65 20 72 75 6E	20 69 6E 20 44 4F 53 2) t be run in DOS	
IMAGE_SECTION_HEADER .rdata	00000070	6D 6F 64 65 2E 0D 0D 0A	24 00 00 00 00 00 00 0) mode\$	
IMAGE_SECTION_HEADER .data	00000080	5D 65 FD C8 19 04 93 9B	19 04 93 9B 19 04 93 9	3]e	
- SECTION .text	00000090	97 1B 80 9B 11 04 93 9B	E5 24 81 9B 18 04 93 9	3 \$	
SECTION .rdata	0000000A0	52 69 63 68 19 04 93 9B	00 00 00 00 00 00 00 0) Rich	
- SECTION .data	000000B0	50 45 00 00 4C 01 03 00	9B 4D 8F 42 00 00 00 0) PELM.B	
	000000000	00 00 00 00 E0 00 OF 01	OB 01 05 0C 00 02 00 0)	
	000000000	00 04 00 00 00 00 00 00	00 10 00 00 00 10 00 0		
	000000E0	00 20 00 00 00 00 40 00	00 10 00 00 00 02 00 0]@	
	000000F0	04 00 00 00 00 00 00 00	04 00 00 00 00 00 00 0)	~
Viewing hello25.exe					

图 3-10 PEview 查看示例程序 hello25.exe

Q, PEview - F:\lab\helio25.cxe					-	\times
File View Go Help						
30000						
⊟-hello25.exe	pFile	Data	Description	Value		^
IMAGE_DOS_HEADER	00000008	010B	Magic	→ 10B 对应32位		
MS-DOS Stub Program	000000CA	05	Major Linker Version			- 8
IMAGE_NT_HEADERS	000000CB	00	Minor Linker Version			
Signature	000000CC	00000200	Size of Code			
IMAGE_FILE_HEADER	000000D0	00000400	Size of Initialized Data			
IMAGE_OPTIONAL_HEADER	000000D4	00000000	Size of Uninitialized Data			
IMAGE_SECTION_HEADER .text	000000D8	00001000	Address of Entry Point	▶ 入口点		
IMAGE_SECTION_HEADER .rdata	000000DC	00001000	Base of Code			
IMAGE_SECTION_HEADER .data	000000E0	00002000	Base of Data			
SECTION .text	000000E4	00400000	Image Base	▶ 映像基址		
	000000E8	00001000	Section Alignment	计文学中		
SECTION . data	000000EC	00000200	File Alignment	▶ 刈介松皮		
	000000F0	0004	Major O/S Version			
	000000F2	0000	Minor O/S Version			
	000000F4	0000	Major Image Version			 ~
	<					>
Viewing IMAGE OPTIONAL HEADER						

图 3-11 hello25.exe 的 PE 文件头

(3) 观察引入函数节。

引入函数节(.rdata)是 PE 文件的重要数据结构。展开查看该节,如图 3-12 所示,引入 函数节包含引入地址表、引入目录表、引入名字表等内容。

🔍 PEview -		\bigcirc			- 0	×
File View Go Help						
30000						
- MS-DOS Stub Program	pFile	Data	Description	Value		^
- IMAGE_NT_HEADERS	00000614	00002050	Import Name Table RVA			
- Signature	00000618	00000000	Time Date Stamp			
IMAGE_FILE_HEADER	0000061C	00000000	Forwarder Chain			- 1
IMAGE_OPTIONAL_HEADER	00000620	00002072	Name RVA	kernel32.dll		
IMAGE_SECTION_HEADER .text	00000624	00002000	Import Address Table RVA			
IMAGE_SECTION_HEADER .rdata	00000628	00002058	Import Name Table RVA			
IMAGE_SECTION_HEADER .data	0000062C	00000000	Time Date Stamp			- 1
SECTION .text	00000630	00000000	Forwarder Chain			
⊨-SECTION .rdata	00000634	0000209A	Name RVA	user32.dll		- 1
IMPORT Address Table	00000638	00002008	Import Address Table RVA			_
IMPORT Directory Table	0000063C	00000000				
IMPORT Name Table	00000640	00000000				- 1
IMPORT Hints/Names & DLL N	00000644	00000000				
SECTION .data	00000648	00000000				
×	0000064C	00000000				~
< >	<					>
Viewing IMPORT Directory Table						

图 3-12 hello25.exe 的引入函数节

2. 查看 PE32+文件结构与 PE32 的差异

PE32+是 64 位 Windows 所使用的文件格式,在 PE 格式的基础上做了一些简单的修

改。虽然 PEview 能够很直观地展示各字段的意义,但这个工具不支持 PE32+格式,所以 接下来使用 StudyPE+观察 PE32+格式。将程序 pe32+.exe 拖到打开的 StudyPE+中,即 可观察该程序的重要信息,如图 3-13 所示,单击"PE 头"标签,可以查看 PE 头的重要字段。

😭 StudyPE+ (x64) 1.11 build 112> pe32+.exe	- 🗆 X
文件 选项 工具 杂项 帮助	
	· ?
[概况] [PE头] [数据表] [区段] [导入] [资源] [重定位] [异常]	é
Image DOS Header	с Э
头尺寸 00000040 节表数 00000006 时间戳 611908	FF 时间计算器 🕺
Stub 000000B0 可选头大小 000000F0 文件属性 000000	22 > >>> 💔
Image Optional Header	
链接器版本 LinkerVersion DE.1C 00.00 自定义版	동 ImageVersion
代码段大小 SizeOfCode 00000E00 00000003 > 子系统版	5号 SubsystemVersion
初始化数据大小 SizeOfInitData 00001E00 00000000 Win32Vers	ionValue
未初始化数据 SizeOfUninitData 00000000 00007000 内存中装	划总尺寸 SizeOfImage
程序入口 AddressOfEntryPoint 000012E4 00000400 头部总尺	SizeOfHead
代码段入口 BaseOfCode 00001000 00000000 映像校验	修正 CheckSum
删除 数据段入口 BaseOfData 00000003 文件子系统	ĉ Subsystem
8字节 程序默认基址 ImageBase 000140000000 00008160 > DLL特性 I	llCharacteristics
区段对齐 SectionAlignment 00001000 00000000 LoaderFla	gs
文件对齐 FileAlignment 00000200 00000010 NumberOff	vaAndSizes
操作系统版本号 OSVersion 06.00	>>>
ne32+.exe. 10 KB.	19:58:12
P	ISISTIE

图 3-13 StudyPE+打开 pe32+.exe

PE32+文件结构与 PE32 的差异主要有 3 方面。

(1) Magic。PE32+中的 Magic 值为 020B,PE32 中的 Magic 值为 010B,PE 装载器通 过检查该字段值来判断文件是 64 位还是 32 位。

(2) BaseOfData。图 3-13 中可以看到, PE32+中删除了该字段,在 32 位 PE 文件格式 中该字段表示指向数据段开头位置(RVA)。

(3)字段大小变化。PE32+中共5个字段由4字节拓展为8字节,来表达更大的内存范围。例如图3-13中的ImageBase字段,除此之外还有堆栈相关的4个字段:SizeOfStackReverse、SizeOfStackCommit、SizeOfHeapReverse、SizeOfHeapCommit。

3. 初步调试该程序

(1) 用 OD 打开程序。

使用 PE 调试工具 OllyDbg 打开示例程序 hello25.exe (通过菜单栏的 File→Open 打开 示例程序或者直接将示例程序拖入打开的 OllyDbg 中),图 3-14 展示了程序加载后的 OllyDbg 界面,主要分为4个区域,左上角是反汇编界面,显示了程序的反汇编代码;右上角 是寄存器界面,显示各寄存器的值;左下角显示程序内存的值;右下角显示栈的内容。

(2) 单步步过到第一个弹框。

接下来让程序单步运行到第一个弹框,快捷键 F8 代表单步步过(每次执行一条指令, 不跟踪到调用内部),这个过程中观察程序的行为:首先压栈 4 个参数,第一个参数代表弹 框类型,第二、三个参数分别为对话框的标题字符串地址与内容字符串地址;再调用 user32. dll 中的弹框函数 MessageBoxA。运行到这里就会出现图 3-15 所示的弹框。

KaraalMada halla 25 ava 186 D118 main thread madula halla 2.1					
w Antennoos - neno-saleer (dato - main mead indone neno-z.)					
00181000 -* 68 h0100000 lough 0x1000	SHIN - HR OKING TONOSTERISKING				
CDUSTION F 66 4010000 piss hello-2_r00403000 09461005 66 00340400 piss hello-2_r00403000 09461006 66 00340400 piss hello-2_r00403000 09461007 66 00340400 piss hello-2_r00403000 09461007 66 00340400 piss hello-2_r00403000 09461021 68 40130000 piss hello-2_r00403000 09461021 68 40130000 piss hello-2_r00403000 09461022 68 31304000 piss hello-2_r00403000 09461025 66 00 piss hello-2_r00403000 09461025 66 00 piss hello-2_r00403001 09461025 66 00 piss hello-2_r00403001 piss 09461025 66 00 piss hello-2_r00403001 piss 09461026 16 000 piss hello-2_r00403001 piss 09461046 16 <t< th=""><th><pre>style - HB_OK[HB_[COMMSTERISK]HB Title - "PE人口点测试1, 进入第一→ hoomer = NUL HessageBoxA Title - "R之间试试" Title - "R之间试试" HessageBoxA Lets - "R之间试试" Hoomer = NUL L'AttSoageBoxA ExitProcess kernel22.ExitProcess user32.NessageBoxA</pre></th><th>Registers (FPU) くくく EAX 709/0388 kernel32.BaseIhreadInithumk EAX 709/0388 kernel32.BaseIhreadInithumk EAX 709/0388 kernel32.BaseIhreadInithumk EAX 8000000 EAX 8000000 EAX 8000000 EAX 8000000 EAX 8000000 EAN 800000 EDI 8000000 EDI 80000000 EDI 80000000 EDI 8000000 EDI 80000000 EDI 80000000 EDI 80000000 EDI 80000000 EDI 800000000 EDI 800000000 EDI 8000000000000000000000000000000000000</th></t<>	<pre>style - HB_OK[HB_[COMMSTERISK]HB Title - "PE人口点测试1, 进入第一→ hoomer = NUL HessageBoxA Title - "R之间试试" Title - "R之间试试" HessageBoxA Lets - "R之间试试" Hoomer = NUL L'AttSoageBoxA ExitProcess kernel22.ExitProcess user32.NessageBoxA</pre>	Registers (FPU) くくく EAX 709/0388 kernel32.BaseIhreadInithumk EAX 709/0388 kernel32.BaseIhreadInithumk EAX 709/0388 kernel32.BaseIhreadInithumk EAX 8000000 EAX 8000000 EAX 8000000 EAX 8000000 EAX 8000000 EAN 800000 EDI 8000000 EDI 80000000 EDI 80000000 EDI 8000000 EDI 80000000 EDI 80000000 EDI 80000000 EDI 80000000 EDI 800000000 EDI 800000000 EDI 8000000000000000000000000000000000000			
88481849 88 db 88	反汇编	EFL 00000246 (NO.NB.E.BE.NS.PE.GE.LE)			
0040104H 00 01 00 0040104B 00 db 00 0040104C 00 db 00 0040104D 00 db 00		ST0 enpty 0.0 ST2 enpty 0.0 ST2 enpty 0.0			
		ST4 enpty 0.0			
		CIS onnell & M			
Contraction	u_u X内存 ?内存 32.dll tfA.?Ne A.user32	UD10FF00 76743364 HE UNK to Kernel32,70943 - 0010FF00 76FD600 0010FF01 7765062 RETURN to ntdll_12.77C89 0010FF00 7661068 0010FF01 9661068 0010FF01 96000000 0010FF04 00000000			
M1 M2 H3 H4 M5 Command:		ESP EBP NONE			
Memory Window i Statt: 0x402020 End: 0x402020 Size: 0x1 Value: 0x0					

图 3-14 OllyDbg 打开 hello25.exe

(3) 单步运行到程序结束。

继续单步运行,接下来的代码是第二个 MessageBoxA 弹框,如图 3-16 所示,与第一个 弹框标题相同但内容不同。反汇编代码中也体现了这一点,第一个弹框压入的文本参数为 0x403009,第二个弹框压入的文本参数则是 0x403031(如果在调用 MessageBoxA 函数的位 置按快捷键 F7 来单步步入,跟踪到函数内部,会发现程序停在了连续的 JMP 指令中,这些 JMP 指令指向了函数的 IAT 表,来获得函数的内存地址)。

 教学测试 	 軟学测试
● PE入口点测试1:进入第一入口位置401000H!	● PE入口点测试2:进入第二入口位置401016H!
[确定
图 3-15 hello25.exe 的第一个弹框	图 3-16 hello25.exe 的第二个弹框

继续单步,调用 ExitProcess 函数结束运行。按快捷键 Ctrl+F2 可以重新运行程序,帮助再次分析。

4. 修改该程序,使该程序仅弹出第二个对话框

使程序仅弹出第二个程序可以有许多方法,可以修改程序的入口点(Address of Entry Point),使程序加载后从第二个弹框处开始运行。

(1) 定位入口点。

首先需要知道入口点字段在什么位置。010Editor 官网提供了 PE 模板,可以辅助分析 PE 文件字段;也可以用 PEview 打开一份程序副本,查看入口点位置,如图 3-17 所示,可见 入口点在 D8 位置。

软件安全实践

pFile	Data	Description
000000C8	010B	Magic
000000CA	05	Major Linker Version
000000CB	0C	Minor Linker Version
000000CC	00000200	Size of Code
000000D0	00000400	Size of Initialized Data
000000D4	00000000	Size of Uninitialized Data
000000D8	00001000	Address of Entry Point
000000DC	00001000	Base of Code
000000E0	00002000	Base of Data

图 3-17 PEview 查看 hello25.exe 的入口点

(2) 修改入口点。

在前面的调试过程中(图 3-14),我们知道了第二个弹框的开始内存地址为0x00401016,所以使用010Editor打开程序,将入口点(D8 处)的值修改为第二个弹框的RVA,即0x00001016,如图 3-18 所示。

hello-2	2.5.	exe*															
∓ Edi	t As:	Hex		Ru	n Se	ript	*	Rur	. Tem	plat	e 🔻						
	0	1	2	3	4	5	6	7	8	9	A	B	Ç	D	E	F	0123456789ABCDEF
0000h:	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZÿÿ
0010h:	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	,@
0020h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0030h:	00	00	00	00	00	00	00	00	00	00	00	00	BO	00	00	00	°
0040h:	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	º´.Í!,.LÍ!Th
0050h:	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is program canno
0060h:	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t be run in DOS
0070h:	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode\$
0080h:	5D	65	FD	C8	19	04	93	9B	19	04	93	9B	19	04	93	9B]eýÈ">">">
0090h:	97	1B	80	9B	11	04	93	9B	E5	24	81	9B	18	04	93	9B	–.€>">å\$.>">
00A0h:	52	69	63	68	19	04	93	9B	00	00	00	00	00	00	00	00	Rich">
00B0h:	50	45	00	00	4C	01	03	00	9B	4D	8F	42	00	00	00	00	PEL>M.B
OOCOh:	00	00	00	00	EO	00	OF	01	0B	01	05	0C	00	02	00	00	à
OODOh:	00	04	00	00	00	00	00	00	16	10	00	00	00	10	00	00	
OOEOh:	00	20	00	00	00	00	40	00	00	10	00	00	00	02	00	00	
OOFOh:	04	00	00	00	00	00	00	00	04	00	00	00	00	00	00	00	

图 3-18 修改 hello25.exe 的入口点为 0x00001016(D8 处的 4 字节)

(3) 确认修改效果。

保存文件之后重新运行程序,可见只弹出了第二个对话框。再次用 OllyDbg 打开程 序,如图 3-19 所示,程序代码从第二个弹框开始。

Se Kanada da dalla 2.5 ave 190 D.18. avez davez davez dalla dalla 2.1		
Kernelwode - nello-2.5.exe - ["G.P.O" - main thread, module nello-2_]		
C File View Debug Plugins Options Window Help Tools BreakPoint->		- 8 ×
	B R S ?	
88481816 . 68 40100000 8x1040	_Style = MB_OK[MB_ICONASTERISK[MB	Registers (FPU) < <
0040101B . 68 00304000 push hello-200403000	Title = "教学测试"	EAX 769433B8 kernel32.BaseThreadInitThunk
88481828 . 68 31384888 push nello-288483831	Text = "PE人口点测试2: 进入第二/	ECX 00000000
88481827 F8 148888888 rall (inn &user32 MessaneBox0)	MessaneRoya	EDX 88481816 hello-2 <moduleentrypoint></moduleentrypoint>
0040102C . 6A 00 Dush 0x0	rExitCode = 0x0	EBA 7EFDE000
0040102E L. E8 01000000 call <jmp.&kernel32.exitprocess></jmp.&kernel32.exitprocess>	-ExitProcess	FRP 8818FF94
88481833 CC int3		ESI 00000000
00401034 FF25 0020400 jmp dword ptr ds:[<&kernel32.ExitProces	kernel32.ExitProcess	EDI 00000000
0040103A - FF25 0C20400 jmp dword ptr ds:[<&user32.wsprintfA>]	user32.wsprintfA	EIP 88481816 belle=2 (ModuleEntruPoint)
00401040 5- FF25 0820400 jmp dword ptr ds:[<&user32.MessageBoxH)	user32.MessageBoxA	en soustons herrs zchoudrechergroune/
88481840 88 db 88		C 0 ES 002B 32DIC 0(FFFFFFF)
66461648 66 db 66		P = 63 0023 32010 0(FFFFFFF)
66461649 60 db 60		7 1 DS $882B$ $32bit$ $8(FFFFFFFF)$
8848184A 88 db 88		S 0 FS 0053 32bit 7EFDD000(FFF)
6646164B 66 db 66		T Ø GS ØØ2B 32bit Ø(FFFFFFFF)
6646164C 00 db 66		D 0
00101040 00 00 00 00101015 00 db 00		0 0 LastErr ERROR_SXS_KEY_NOT_FOUND (000036B7)
6040104E 00 00 00 00		EFL 888888246 (NO,NB,E,BE,NS,PE,GE,LE)
88481858 88 db 88		STR omotu R R
00401051 00 db 00		STO EMPLY 5.5
00401052 00 db 00		ST2 emptu 0.0
00/01053 00 db 00		ST3 empty 0.0
		ST4 empty 0.0
		ST5 empty 0.0
Address Hex dump ASCII		0018FF8C 769433CA RETURN to kernel32.76943 .
88482888 18 7A 94 76 88 88 88 88 1E FD B9 75 5F AE B5 75 ■z猩	∎ u_ u	
		8818FF98 77CR9FD2 RETURN to ptd11 42 77CR0
	x	BB18FF9C ZFFDFB88
		0018FFA0 76FBF16B
88482858 88 89 88 88 88 88 88 86 86 88 88 88 88 88 88	?∎	0018FFA4 00000000
00402060 00 00 00 00 00 00 45 78 69 74 50 72 6F 63 65 73Ex	itProces	0018FFA8 00000000
00402070 73 00 6B 65 72 6E 65 6C 33 32 2E 64 6C 6C 00 00 s.kernel	32.dll	0018FFAC 7EFDE000
00402080 62 02 77 73 70 72 69 6E 74 66 41 00 9D 01 4D 65 b wsprin	tfA.?Ne	0018FFB0 00000000
00402090 73 73 61 67 65 42 6F 78 41 00 75 73 65 72 33 32 ssageBox	A.user32	- 0018FFB4 00000000 -
M1 M2 M3 M4 M5 Connand:		ESP EBP NONE
Memory Window 1 Start: 0x402000 End: 0x401FFF Size: 0x0 Value: 0x76947A10		

图 3-19 OD 调试修改入口点的 hello25.exe

除了快捷键调试程序,OllyDbg还在菜单栏下方提供了图形按钮进行调试,如图 3-20 所示,标注的 4 个按钮的功能分别为重新运行程序、关闭程序、单步步入、单步步过。

3.4 函数的引入/引出机制分析与修改

3.4.1 实验目的

熟悉 PE 文件头部、引入表的结构,了解与使用引入表引入函数,了解与使用 user32 的 函数手动引入函数。本实验可以让学生深入了解 Windows 可执行文件的内部机制,并为软 件开发和逆向工程提供重要的基础,同时有利于学生理解恶意软件脱壳及进行 API 函数定 位的机制。

3.4.2 实验内容及实验环境

1. 实验内容

(1)使用 PE 查看工具 PEview 与调试工具 OllyDbg,结合预备知识与示例程序 hello25.exe,熟悉 PE 文件引入表结构,熟悉函数导入的基本原理。

(2)从 hello25.exe 的内存空间找到模块 user32,进一步查找函数 MessageBoxA 的地址,并验证该地址是否正确。通过实例了解 PE 文件如何引入外部模块的函数。

(3) 用二进制编辑工具修改 hello25.exe 程序的引入表,使该程序仅可以从 kernel32.dll 中引入 LoadLibrary 和 GetProcAddress 函数,而不从 user32.dll 导入任何函数,在代码节中 写入部分代码利用这两个函数获取 MessageBoxA 的函数地址,使 hello25.exe 程序原有功 能正常。

2. 实验环境

(1) 系统: Windows XP 以上的操作系统,实机、虚拟机均可。

(2) 工具: OllyDbg1.10、010Editor、PEview。

3.4.3 实验步骤

1. 观察示例程序 hello25.exe 的函数导入机制

(1) 文件中的 IAT。

程序通过 IAT 定位函数的地址, PE 文件的 IAT 在文件中与内存中含义有区别。 hello25.exe 示例程序在文件中的 IAT 如图 3-21 所示, 通过函数名称引入了函数 ExitProcess、MessageBoxA 与 wsprintfA(也可以通过序号引入函数)。

(2) 内存中的 IAT。

而 PE 文件运行过程中,如何在内存中找到函数地址呢? 我们通过 OllyDbg 来观察。 用 OD 打开 hello25.exe,如图 3-22 所示,在代码节尾部存放着通过 IAT 表间接跳转到引入

Q PEview - C:\Users\15427\Desktop\pe\hello-2.5.	exe			
File View Go Help				
	-			
⊡-hello-2.5.exe	pFile	Data	Description	Value
IMAGE_DOS_HEADER	00000600	00002064	Hint/Name RVA	0000 ExitProcess
MS-DOS Stub Program	00000604	00000000	End of Imports	kernel32.dll
IMAGE_NT_HEADERS	00000608	0800806D	Hint/Name RVA	
IMAGE_SECTION_HEADER .text	0000060C	00002080	Hint/Name RVA	0262 wsprintfA
IMAGE_SECTION_HEADER .rdata	00000610	00000000	End of Imports	user32.dll
IMAGE_SECTION_HEADER .data				
SECTION .text				
È-SECTION .rdata				
IMPORT Name Table				
 IMPORT Address Table 				
- IMPORT Directory Table				
- IMPORT Hints/Names & DLL Names				
- SECTION .data				
	•	III		۲.
Viewing IMPORT Address Table				

图 3-21 hello25.exe 的 IAT 表

函数的指令,这就是跳转表。图 3-22 中用方框圈出了跳转表跳转的地址,分别是 0x402000、0x40200c、0x402008,换算为文件偏移 FA 就是 0x600、0x60c、0x608,正好是图 3-21 显示的 IAT 的函数地址。

	FF25 00204000	jmp dword ptr ds:[<&kernel32.ExitProces	kernel32.ExitProcess
	FF25 0C204000	<pre>jmp dword ptr ds:[<&user32.wsprintfA>]</pre>	user32.wsprintfA
\$-	FF25 08204000	jmp dword ptr ds:[<&user32.MessageBoxA>	user32.MessageBoxA
	G G 🛛 🗖 🖓 🖓 🖓 🖓	db 00	

图 3-22 hello25.exe 的跳转表

(3) IAT 的函数地址变化。

下面转到 IAT 的内存处查看,如图 3-23 所示,此时内存中 IAT 的元素与文件 IAT 的元素不同,内存中 IAT 的元素是函数的内存地址,而不是指向函数名的 RVA。这样跳转表可以根据 IAT 找到函数在内存中的真实地址。

图 3-23 内存中的 IAT 表

2. 从该程序的内存空间定位查找 user32.dll 的函数 MessageBoxA 的地址

(1) 定位模块 user32.dll。

要从程序的内存空间找到 user32 再找到 MessageBoxA 地址,首先要找到 user32 模块的内存地址。如图 3-24 所示,单击 M(即 Memory)方块查看内存中的模块,然后找到 user32 模块,基址为 0x75b30000,双击即可进入该地址。

(2) user32 的引出表。

弹出的 Dump 窗口显示了 user32 的头部信息。向下翻可以获取数据目录项中的引出 表的地址,如图 3-25 所示,引出表 RVA 为 0x10548,加上基址,引出表的 VA 就 是 0x75B40548。

图 3-25 的 Dump 不适合查看内存,因此在 OllyDbg 的内存窗口中继续分析。单击上方的 C(即 CPU)方块切换到之前的窗口,在内存窗口中按快捷键 Ctrl+G 跳转到 user32 的引

💥 Kerne	🕻 KernelMode - hello-2.5.exe - [Memory map]														
M File	View I	Debug P	Plug	ins Opti	ons Windo	w H	lelp To	ools Br	eakPoint->						
Paused	aused														
Address	Size	Owner		Section	Contains	Type	Access	Initil	Mapped as						
75980000	00002000	KernelBa	759	. data	data	Imag	R	RWE							
75982000	00001000	KernelBa	759	.rsrc	resources	Imag	R	RWE							
75983000	00003000	KernelBa	759	.reloc		Imag	R	RWE							
75B30000	00001000	user32	75E		PE header	Imag	R	RWE							
75B40000	00060000	user32	75E	. text	SFX, code, im	Imag	RE	RWE							
75BB0000	00001000	user32	75E	. data	data	Imag	RW	RWE							
75BC0000	00058000	user32	75E	.rsrc	resources	Imag	R	RWE	(
75C20000	00004000	user32	75E	.reloc		Imag	Ĩ.	RWE							
75050000	00001000	sechost	750		PE header	Imag	R	KUE							
75C51000	00013000	sechost	750	. text	SFX. code. im	Imag	R	RWE							
75C64000	00003000	sechost	750	. data	data	Imag	R	RWE	- usersz						
75C67000	00001000	sechost	750	rsrc	resources	Imag	R	RWE							
75C68000	00001000	sechost	750	.reloc		Imag	R	RWE							
76930000	00001000	kernel32	769		PE header	Imag	R	RWE							
70040000	00001000	1 100	700		0721 1		D 72	7010172							

图 3-24 OD 查看程序的模块

D Dump	- user32 75	B3000075B30FFF		- • ×
75B30154	00100000	DD 00001000	SizeOfHeapCommit = 1000 (4096.)	
75B30158	00000000	DD 0000000	LoaderFlags = 0x0	
75B3015C	10000000	DD 00000010	NumberOfRvaAndSizes = 10 (16.)	
75B30160	48050100	DD 00010548	Export Table address = 0x10548	
75B30164	7E5A0000	DD 00005A7E	Export Table size = 5A7E (23166.)	
75B30168	EOAB0700	DD 0007ABE0	Import Table address = 0x7ABE0	
75B3016C	64000000	DD 00000064	Import Table size = 64 (100.)	
75B30170	00000900	DD 00090000	Resource Table address = 0x90000	
75B30174	80420500	DD 00054280	$R_{0} = 200 r_{0} = 100 r_{0} = 500 r_{0} = 100 r_{0$	

图 3-25 user32 的 Dump 界面

出表,地址为0x75B40548,如图 3-26 所示。

引出表的结构见图 3-26。图 3-26 中地址 0x75B40548 处是 user32 的引出表目录,图中数据从 0 开始计数,偏移 0C 处的 2 个 DWORD 分别为函数名的起始 RVA(所有函数的名称字符串),起始函数序号 0x5dc;偏移 1C 处的 3 个 DWORD 分别是函数地址表、函数名指针表、函数序号表的 RVA。

Address	Hex	c di	IMD														ASCII
75840548	00	66	80	66	89	8F	Ε7	4C	88	80	66	88	60	28	01	66	撥鐺`(£
75B40558	DC	05	88	66	EB	03	66	00	36	03	66	88	70	65	01	66	??6p ¥£.
75B40568	10	15	01	66	F4	21	01	00	CB	89	07	88	91	80	07	00	■■£.?£.藟■.憖■.

图 3-26 内存中查看 user32 的引出表

(3) 定位 MessageBoxA 的字符串。

首先找到 MessageBoxA 字符串的地址。根据所有名称字符串的 RVA(0x12860),跳转 到相应的 VA(0x75b42860),按快捷键 Ctrl+B 搜索字符串 MessageBoxA,得知 MessageBoxA 字符串的地址为 0x75B44AF8,如图 3-27 所示。

Address	Hex	(di	JWD														ASCII
75B44AF8	4D	65	73	73	61	67	65	42	6F	78	41	00	4D	65	73	73	MessageBoxA.Mess
75B44B08	61	67	65	42	бF	78	45	78	41	66	4D	65	73	73	61	67	ageBoxExA.Messag
75B44B18	65	42	бF	78	45	78	57	66	4D	65	73	73	61	67	65	42	eBoxExW.MessageB
75B44B28	бF	78	49	бE	64	69	72	65	63	74	41	86	4D	65	73	73	oxIndirectA.Mess

图 3-27 内存中查找 user32 的 MessageBoxA 字符串

(4) 计算 MessageBoxA 的项数。

然后计算 MessageBoxA 在函数名指针表中的表项位置。根据函数名指针表 RVA (0x1151c),跳转到 VA(0x75b4151c)。此时需要搜索的是 MessageBoxA 字符串的 RVA,即 0x14af8,因此搜索 HEX 值"F8 4A 01 00",得到相应地址 0x75b41d54,如图 3-28 所示。因为(0x75b41d54-0x75b4151c)/4 为 526,所以 MessageBoxA 是第 526 项。

Address	Hex d	ump														ASCII
75B41D54	F8 4A	61	00	04	4B	01	00	12	4B	01	00	20	4B	61	00	鳭£. ₭£.∎₭£. ₭£.
75B41D64	34 4B	01	00	48	4B	01	66	5B	4B	01	00	бE	4B	61	66	<u>4K£.</u> HK <u>₹</u> .[K <u>₹</u> .nK <u>₹</u> .
75B41D74	7A 4B	01	00	86	4B	01	00	92	4B	01	00	A3	4B	01	00	ヹ゚゚ゟヹゟ゙゙ヹゟ゙゚゚゚゚ヹゟ゚゚゚゚゚゚゚゚゚゚ヹ゚゚゚゚゚゚゚゚ヹ゚゚゚゚゚゚゚゚

图 3-28 函数名指针表中查找对应 MessageBoxA 的项

(5) 计算 MessageBoxA 的函数序号。

接着,找到 MessageBoxA 的函数序号。根据函数序号表 RVA(0x121f4),跳转到序号 表的第 526 项(0x75b42610 = 0x75b421f4 + 2 × 526,序号表元素大小为 2 字节),得到 MessageBoxA 的函数序号为 0x21b,如图 3-29 所示。

Address	Hex	(du	JMD														ASCI	I		
75B42610	1B	02	1C	02	1D	02	1E	02	1F	02	20	02	21	02	22	02			1	-n-
75B42620	23	02	24	02	25	02	26	02	27	62	28	02	29	02	2A	02	#_\$_	ያ ይ	(_)	*

图 3-29 函数序号表中查找 MessageBoxA 对应序号

(6) 获取 MessageBoxA 的 RVA。

最后,函数地址表第 0x21b 项就是 MessageBoxA 的 RVA。根据函数地址表 RVA (0x10570),跳转到目的 VA(0x75b40ddc=0x75b40570+4×0x21b),得到 MessageBoxA 函数的 RVA 为 0x6fd1e,如图 3-30 所示。

Address	Hea	k di	итр														ASCII
75B40DDC	1E	FD	06	88	Dó	FC	06	88	FA	FC	06	66	D1	FB	06	00	■?.贮■. ■.激■.
75B40DEC	9D	FC	06	00	28	FB	06	88	CD	FA	06	66	3F	FD	06	90	濣∎.(?.旺∎.??.
75B40DFC	38	68	07	00	Dő	C 0	05	88	81	52	02	66	A0	E7	03	99	8h∎ 掷ギッニ犵

图 3-30 MessageBoxA 函数的 RVA

(7) 验证函数地址。

现在验证寻找的函数地址是否正确。手动寻找的 MessageBoxA 的 RVA 为 0x6fd1e,则 VA 为 0x75b9fd1e;在反汇编窗口中单击跳转表中的 MessageBoxA 函数,图 3-31 中的方框提示了函数地址也是 0x75b9fd1e,两者一致,说明地址正确。

00401033	CC	int3										
00401034	FF25 00204000	jmp dword ptr ds:[<&kernel32.ExitProces	kernel32.ExitProcess									
0040103A	FF25 0C204000	jmp dword ptr ds:[<&user32.wsprintfA>]	user32.wsprintfA									
00401040	\$- FF25 08204000	<pre>jmp dword ptr ds:[<&user32.MessageBoxA>]</pre>	user32.MessageBoxA									
00401046	88	db 00										
00401047	00	db 00										
00401048	00	db 00										
00401049	00	db 00										
0040104A	00	db 00										
0040104B	00	db 00										
00 481846		db 88										
ds:[00402	ds[[00402008]=75B9FD1E (user32.MessageBoxA)											
Local cal												

图 3-31 跳转表中的 MessageBoxA 的地址

3. 修改该程序,使该程序使用 LoadLibrary 和 GetProcAddress 导入函数 MessageboxA

(1) 修改引入表。

程序的引入表中没有 LoadLibrary、GetProcAddress 这两个函数,因此先使用 010Editor 修改 PE 文件引入表,才能在代码中使用这两个函数。

① 修改字符串。添加两个函数的字符串,如图 3-32(注意函数 LoadLibrary 的字符串 是 LoadLibraryA)。

② 修改 IDT。函数 LoadLibrary 和 GetProcAddress 都在 kernel32.dll 中,程序需要的 另一个函数 ExitProcess 也在 kernel32 中,所以程序只需要从 kernel32 中引入所需函数,引 入表原来的 user32 部分可以去掉,改为全零表示 IDT 结束。

③ 修改 IAT 与 INT。IAT 与 INT 在文件中完全一致,这里以 IAT 举例。IAT 在 0x600 的位置开始,0x604 的 DWORD 引入 LoadLibrary。LoadLibrary 字符串的地址是 0x682,但字符串前两字节表示 hit(hit 的值不影响引入),所以引入的起始地址应为 0x680, 对应 RVA2080。对 GetProcAddress 以同样的方式引入,然后以全零 DWORD 结束 IAT。

0600h:	64	20	00	00	A3	20	00	00	в0	20	00	00	o ¢A	\T 0	00	00	d£°
0610h:	00	00	00	00	50	20	00	00	00	00	00	00	00	00	00	00	P
0620h:	72	20	00	00	00	20[Dato	00	00	00	00	00	00	00	00	00	r
0630h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0640h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
0650h:	64	20	00	00	A3	20	00	00	в0	20	00	00	٥bN	100	00	00	d£°
0660h:	00	00	00	00	80	00	45	78	69	74	50	72	6F	63	65	73	€.ExitProces
0670h:	73	00	6B	65	72	6E	65	6C	33	32	2E	64	6C	6C	00	00	s.kernel32.dll
0680h:	62	02	77	73	70	72	69	6E	74	66	41	00	9D	01	4D	65	b.wsprintfAMe
0690h:	73	73	61	67	65	42	6F	78	41	00	75	73	65	72	33	32	ssageBoxA.user32
06A0h:	2E	64	6C	6C	00	4 <u>e</u>	婚日	_ 61	64	4C	69	62	72	61	72	79	.dll.LoadLibrary
06B0h:	41	00	47	65	74	50	172	6F	63	41	64	64	72	65	73	73	A.GetProcAddress
06C0b .	L00	-00	-00	-00	00	-00	-00	-00	00	-00	-00	-00	00	00	-00	-00	1

图 3-32 修改 hello25.exe 的引入表

(2) 修改程序,手动加载函数 MessageBoxA。

成功引入函数之后,接下来使用 OD 修改代码,使程序通过 LoadLibrary 与 GetProcAddress 引入函数 MessageBoxA。这里相当灵活,同学们可以用各种方法实现实验 目的,例如修改入口点在原代码尾部,引入 MessageBoxA 后再跳转到原代码开头。本书采 用另一种方法,完全重写代码,先引入函数 MessageBoxA,再调用该函数弹框,最后结束程 序。使用 OD 打开修改好引入表的程序,如图 3-33 所示。

00401000	г\$	68 40100000		
00401005	-	68 00304000	push hello-200403000	ASCII "教学测试"
0040100A	- II	68 09304000	push hello-200403009	┏ProcNameOrOrdinal = "PE入口点测词
0040100F	- I	6A 00	push 0x0	hModule = NULL
00401011	- I	E8 2A000000	call <jmp.&kernel32.getprocaddress></jmp.&kernel32.getprocaddress>	GetProcAddress
00401016	- I	68 40100000	push 0x1040	
0040101B	- I	68 00304000	push hello-200403000	ASCII "教学测试"
00401020	- I	68 31304000	push hello-200403031	┏ProcNameOrOrdinal = "PE入口点测词
00401025	- I	6A 00	push 0x0	hModule = NULL
00401027	- II	E8 14000000	call <jmp.&kernel32.getprocaddress></jmp.&kernel32.getprocaddress>	GetProcAddress
0040102C	- II	6A 00	push 0x0	rExitCode = 0x0
0040102E	L.	E8 01000000	call <jmp.&kernel32.exitprocess></jmp.&kernel32.exitprocess>	ExitProcess
00401033		CC	int3	
00401034		FF25 0020400	jmp dword ptr ds:[<&kernel32.ExitProces	kernel32.ExitProcess
0040103A		FF25 0C20400	jmp dword ptr ds:[0x40200C]	
00401040	\$-	FF25 0820400	jmp dword ptr ds:[<&kernel32.GetProcAdd	kerne132.GetProcAddress
00401046		00	db 00	
001.04.01.7		00	-16 - 0.0	

图 3-33 OD 打开修改了引入表的 hello25.exe

双击汇编代码可以对代码进行修改,首先压栈字符串 user32.dll,然后调用 LoadLibrary (这里直接 call IAT 中的函数地址);接着调用 GetProcAddress 加载 MessageBoxA 的地址, 用寄存器 ebx 保存;然后是两次弹框代码与结束进程。可以反复单步调试与重新运行来检查代码的正确性,如图 3-34 所示。

(3)运行程序。

反汇编窗口右击,选择"Copy to Executable"(复制到可执行文件)→"All Modifications"(所有修改),可以在修改后保存为可执行程序。保存后运行,确认功能正常。

软件安全实践

68 9A204000	push hello250040209A	ASCII "user32.dll"
FF15 0420400	call dword ptr ds:[<&kernel32.LoadLibra	kerne132.LoadLibraryA eax=LoadLibrary('user32.dll')
68 8E204000	push hello250040208E	ASCII "MessageBoxA"
50	push eax	ebx=GetProcAdress(eax, 'MessageBoxA')
FF15 0820400	call dword ptr ds:[<&kernel32.GetProcAd	kerne132.GetProcAddress
8BC3	mov eax.ebx	
68 40100000	push 0x1040	
68 00304000	push hello2500403000	ASCII "教学测试"
68 09304000	push hello2500403009	ASCII "PE入口点测试1. 进入第一入口位置401000H!"
6A 00	push 0x0	
FFD3	call ebx	
68 40100000	push 0x1040	
68 00304000	push hello2500403000	ASCII "教学测试"
68 31304000	push hello2500403031	ASCII "PE人口点测试1: 进入第二入口位置401016H!"
6A 88	push 0x0	
FFD3	call ebx	
FFD3	call ebx	
6A 00	push 0x0	
FF15 0020400	call dword ptr ds:[<&kernel32.ExitProce	kernel32.ExitProcess

图 3-34 在 OD 中修改代码,手动加载 MessageBoxA

3.5 资源节资源操作

3.5.1 实验目的

了解 PE 文件资源节结构,能够修改 PE 文件图标,汉化 PE 文件。

通过本实验,学生将深入了解 PE 文件的资源节结构,包括资源的组织方式和存储位置。这不仅有助于学生理解 PE 文件中资源的存储和调用方式,同时也能帮助学生理解病毒经常采用的资源寄生与图标替换等机制。

3.5.2 实验内容及实验环境

1. 实验内容

(1) 用二进制编辑工具修改 PEview.exe,使得该文件的图标变成 csWhu.ico(图标见光盘,也可以是任意 ICO 图标文件)。

(2) 熟悉 eXeScope 工具的使用,并利用该工具汉化程序 PEview.exe。

2. 实验环境

(1) 系统: Windows XP 及以上的操作系统,实机、虚拟机均可。

(2) 工具: OllyDbg1.10、010Editor、PEview、eXeScope。

3.5.3 实验步骤

1. 修改程序 PEview.exe 的图标

修改程序的图标就是将程序资源节中的图标数据进行修改,因此需要了解资源节的结构和图标的结构,然后进行修改。

(1) 查看图标。

用 010Editor 打开要替换的图标 csWhu.ico,如图 3-35 所示。前面 6 字节是图标的头部,3 个 WORD 分别是保留部分、类型、数量。接着是图标的目录项,可知图标大小为 32×32,由于只有一个图标,0x16 开始到文件末尾是图标数据。

(2) 查看 PEview 的图标资源。

图标开始位置 图标宽度与高度

0000h: 00 00 01 00 01 00 <u>20 20</u> 10 00 00 00 00 00 E8 02 0010h: 00 00 16 00 00 02 28 00 00 00 20 00 00 00 40 00 0020h: 00 00 01 00 04 00 00 00 00 00 80 02 00 00 00 €. 0050b. 00 00 80 00 80 00 80 80 00 00 C0 C0 C0 00 80 80 e e 图 3-35 图标 csWhu, ico 的十六进制数据

PE 文件中的图标保存格式与 ICO 文件中图标的保存格式略有不同。PE 文件中,把 ICON 目录项和图标资源作为两种资源类型分别保存,前者是 GROUP ICON 类型,后者是 ICON 类型。用 PEview 打开要修改的 PEview, GROUP ICON 如图 3-36 所示, 可见该文 件包含3个图标数据(对应文件夹的不同大小的视图),其中20×20的图标刚好与 csWhu. ico 大小一致,所以可以对第二个图标进行替换。

pFile		<u>Raw</u> Data	Value
0000F3E4	00 00 01 00 03 00 30	30 10 00 01 00 04 00 68 06	h.
0000F3F4	00 00 01 00 20 20 10	00 01 00 04 00 E8 02 00 00	
0000F404	02 00 10 10 10 00 01	00 04 00 28 01 00 00 03 00	(

找到二号图标 ICON 0002,如图 3-37 所示,该图标从 0xDB28 开始,到 0xDE0F 结束 (图 3-37 中未显示)。

1	⁽³ 、PEview - F:\learning\20\课程-09-软安教辅\1-2次实验\例子\PEview-ico.exe																		
	<u>F</u> ile <u>V</u> iew <u>G</u> o <u>H</u> elp																		
IMAGE_SECTION_HEADER .rsrc pFile																			
	IMAGE_SECTION_HEADER .idata		0000DB28	28	00	00	00	20	00	00	00	40	00	00	00	01	00	04	00
	SECTION code		0000DB38	00	00	00	00	00	02	00	00	00	00	00	00	00	00	00	00
	SECTION data		0000DB48	00	00	00	00	00	00	00	00	00	00	00	00	00	00	80	00
	- SECTION const		0000DB58	00	80	00	00	00	80	80	00	80	00	00	00	80	00	80	00
	SECTION .rsrc		0000DB68	80	80	00	00	CO	CO	CO	00	80	80	80	00	00	00	FF	00
	IMAGE_RESOURCE_DIRECTC		0000DB78	00	FF	00	00	00	FF	FF	00	FF	00	00	00	FF	00	FF	00
	IMAGE_RESOURCE_DIRECTC		0000DB88	FF	FF	00	00	FF	FF	FF	00	00	00	00	00	00	00	00	00
	IMAGE_RESOURCE_DIRECTC		0000DB98	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	- IMAGE_RESOURCE_DATA_EI		0000DBA8	00	00	00	00	08	F7	00	00	00	00	00	00	00	00	00	00
	BITMAP 0001 1009		0000DBB8	00	00	00	00	00	8F	70	00	00	00	00	00	00	00	00	00
	BITMAP 0002 1009		0000DBC8	00	00	00	00	00	87	F7	00	00	00	00	00	00	00	00	00
	-ICON 0001 1009		0000DBD8	00	00	00	00	08	7F	7F	70	00	00	00	00	00	00	00	00
	ICON 0002 1009		0000DBE8	00	00	00	00	87	F7	08	FO	00	00	00	00	00	00	00	00
	-ICON 0003 1009		0000DBF8	00	00	00	08	7F	70	00	80	00	00	00	00	00	00	00	00

图 3-37 PEview 的图标资源

(3) 替换图标。

现在被替换的数据与替换的数据都已确认,用 010Editor 打开图标和程序,将图标数据 复制到程序中(使用快捷键 Ctrl+Shift+C、Ctrl+Shift+V) 🐘 PEview-ico.exe

并保存。如图 3-38 所示,切换到列表视图就可以看到被修改 的图标。

图 3-38 替换后的 PEview

2. 汉化该程序

汉化程序就是将对应的英文字符串改为中文。实验使用的 eXeScope 可以很方便地辅 助汉化。

用 eXeScope 打开 PEview,如图 3-39 所示,左边树状结构中,Resource→Menu→1 是程

序目录的字符串。

把"&"后的英文修改为中文(双击即可修改),如图 3-40 所示。

File Edit Query View	Help				
B 🖬 🗎 🏝 📕 🗚 🗎	🔝 🔜 🗟 No Lo	ogging			
Header Header Hanport Resource Bilmap Henu Hen	✓ Extended Menu Grayed Disabled □ 0.8文件 - 36.8打开 37.8关闭Su - 0, - 38.8選出 - 0,8View - 41,8Status - 42,8Tea8U - 43,8Heade - 0,8Address - 45,8Fill - 45,8Fill - 47,8Re - 48,8Vir	Checked Hillte Default SogCtrI+O J9CtrI+F4 Bar ew er s e Offset &Offset #&Offset lative Virtual Addre bual Address	MenuBarBreak Separator	☐ RadioCheck ☐ RightJustify ☐ RightOrder	Comparison Compa

图 3-39 exscope 打开 PEview

图 3-40 汉化后的 PEview

手工重定位 3.6

3.6.1 实验目的

通过手工对代码节进行重定位修复,理解绝对地址为什么需要重定位,明白重定位的机制与原理。

在这个实验中,学生将深入研究可执行文件的代码节,了解其中绝对地址的特点和作用。通过手工进行重定位修复,学生将亲身体验在程序加载和执行过程中,绝对地址需要进行重定位的原因和必要性。通过实践操作,学生将理解重定位的需求、机制和原理,包括重定位表的结构和使用方法。这将有助于学生在逆向工程、病毒与漏洞分析中更好地理解程序的运行机制,提高其实践能力和分析水平。

3.6.2 实验内容及实验环境

1. 实验内容

(1) 修改 hello25.exe 的加载基址为 0x600000,破坏程序的运行机制。

(2) 在代码节中手工修正数据,使程序能够正常运行,理解重定位必要性。

2. 实验环境

(1) 系统: Windows XP 以上的操作系统,实机、虚拟机均可。

(2) 工具: OllyDbg1.10、010Editor。

3.6.3 实验步骤

1. 修改 ImageBase 字段

通过 PEview(或下载 010Editor 的 PE 模板)获取 ImageBase 字段的偏移为 0xE4,使用 010Editor 打开 hello25.exe,修改 ImageBase 字段为 0x600000,保存本次修改,如图 3-41 所示。

hello2	hello25_reloc. exe* 📓 👘																
∓ Edit	As:	Hex	*	Run	Scr	ipt	- R	un 1	Cemp	late	*						
	Q	1	2	3	4	5	6	7	8	9	A	B	Ç	D	E	Ę	0123456789ABCDEF
00A0h:	52	69	63	68	19	04	93	9B	00	00	00	00	00	00	00	00	Rich
00B0h:	50	45	00	00	4C	01	03	00	9B	4D	8F	42	00	00	00	00	PEL>M.B
00C0h:	00	00	00	00	E0	00	0F	01	0B	01	05	0C	00	02	00	00	à
00D0h:	00	04	00	00	00	00	00	00	00	10	00	00	00	10	00	00	
00E0h:	00	20	00	00	00	00	60	00	00	10	00	00	00	02	00	00	`
00F0h:	04	00	00	00	00	00	00	00	04	00	00	00	00	00	00	00	
0100h:	00	40	00	00	00	04	00	00	00	00	00	00	02	00	00	00	.@

图 3-41 修改 ImageBase

2. 手工重定位

(1) 观察程序异常。

使用 OllyDbg 打开程序,观察代码节的异常与需要修改的地方。图 3-42 中用方框标明 了需要修改的地方。首先,弹框函数中压栈字符串的地方直接使用了绝对地址,需要修改 (例如 push 0x403000,字符串地址已变为了 0x603000);然后可以看到 call 指令的目标地址 没有解析为对应 API,但这不是 call 指令的问题,而是 call 指令的目标对象——跳转表需要 修改(例如 jmp dword ptr[0x402000])。这些需要修改的地方都有同样的特征:使用了绝 对地址,这正是重定位表的任务,即修正代码中的绝对地址。

00601000	\$ 68 40100000	push 0x1040	
00601005	68 00304000	push 0x403000	
0060100A	68 09304000	push 0x403009	
0060100F	6A 00	push 0x0	
00601011	E8 2A000000	call 00601040	hello2500601040
00601016	68 40100000	push 0x1040	
0060101B	68 00304000	push 0x403000	
00601020	68 31304000	push 0x403031	
00601025	6A 00	push 0x0	
00601027	E8 14000000	call 00601040	hello2500601040
0060102C	6A 00	push 0x0	
0060102E	E8 0100000	call 00601034	hello2500601034
00601033	CC	int3	
00601034	\$ FF25 0020400	jmp dword ptr ds:[0x402000]	
0060103A	FF	db FF	
0060103B	25	db 25	CHAR '%'
0060103C	9C	db 0C	
0060103D	20 40 00	ascii " @",0	
00601040	\$ FF25 0820400	jmp dword ptr ds:[0x402008]	
99691946	ពព	ар ни	

图 3-42 程序中的绝对地址

(2) 手工修复。

计算加载的基址差 delta(0x600000-0x400000=0x200000),然后对所有绝对地址加上 delta 即可,如图 3-43 所示(跳转表的第二项没有正确解析,但没有影响,直接双击修改 即可)。

修改完毕后保存为新文件(反汇编窗口右击→复制到可执行文件→所有修改→全部复制),运行查看是否功能正常。