

2021/02/26_PE_第9课_导出表的应用

笔记本： PE

创建时间： 2021/2/26 星期五 10:11

作者： ileemi

- [导出表的应用](#)
- [模拟GetProcAddress](#)
- [通过IAT还原导入表](#)
- [分析加壳程序](#)
- [破坏导出表](#)

通过地址反查：

有名称：地址表 --> 序号表 --> [有 --> 名称表]

没名称：地址表 --> 序号表 --> [没有 --> 地址表下标 + 起止序号 --> 序号]

导出表的应用

通过地址表IAT去查询API的名称不是万能的，动态调用API就不行。就需要查询 "被查询的地址" 所处模块中的导出表（万能做法）。

调试器中显示API的地址就是去查询所处模块中的导出表。

导出函数不导出函数名时显示其导出序号。

防止内存dump：

抹掉可以行文件中所有的导入表项。内存dump后，dump后的程序不能运行。可以防止静态分析程序。

为了保证自己可以正常运行自己的程序，抽出导入表信息，在入口代码处自己填充IAT表（不能还原导入表）。

在OEP入口处等待软件填充IAT表，之后重新建立PE文件导入表，要知道库、函数名称。

反调试、病毒、加密、解密等防止API Hook可以手动实现所需的API函数。

模拟GetProcAddress

示例代码：

```
// 模拟GetProcAddress
void* MyGetProcAddress(void* pImageBase, const char* pszName)
{
    if (pszName == NULL || pImageBase == NULL)
```

```

{
    return NULL;
}

// 获取各个头的RVA
PIMAGE_DOS_HEADER pHeader = (PIMAGE_DOS_HEADER)pImageBase;
PIMAGE_NT_HEADERS32 PNTHeader = (PIMAGE_NT_HEADERS32)
((char*)pImageBase + pHeader->e_lfanew);
PIMAGE_EXPORT_DIRECTORY pExport = (PIMAGE_EXPORT_DIRECTORY)
((char*)pImageBase +
    PNTHeader->
    >OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress);

// 获取函数地址表、序号表、名称表
DWORD Ordinal = (DWORD)pszName;
DWORD* pAddressOfFunctions = (DWORD*)((char*)pImageBase + pExport->
    >AddressOfFunctions);
DWORD* pAddressOfNames = (DWORD*)((char*)pImageBase + pExport->
    >AddressOfNames);
WORD* pAddressNameOrdinals = (WORD*)((char*)pImageBase + pExport->
    >AddressOfNameOrdinals);
if (Ordinal >> 16 == 0)
{
    // 序号导出
    DWORD index = Ordinal - pExport->Base;
    if (index >= pExport->NumberOfFunctions)
    {
        return NULL;
    }
    return (char*)pImageBase + pAddressOfFunctions[index];
}
else
{
    //名称导出
    for (int i = 0; i < pExport->NumberOfNames; i++)
    {
        if (strcmp(((char*)pImageBase + pAddressOfNames[i]), pszName) ==
0)
        {
            //找到
            return (char*)pImageBase +
pAddressOfFunctions[pAddressNameOrdinals[i]];
        }
    }
}
}

```

```
return NULL;
}
```

通过IAT还原导入表

文件中的IAT表不具备参考性，内存中的IAT表才具有参考意义（修复后）。

内存IAT --> 通过导出表 --> 重建PE文件导入表（还原的是文件的）

分析加壳程序

通过工具分析，通过 "CFF" 查看其导入表以及节的名称是否正常，通过调试器查看软件的入口代码是否正常。

对于加壳程序，需要先修正其OEP，调试程序，等到解密代码走完到OEP位置后就dump内存代码到文件。之后在分析IAT表是否被抹。

通过调试器去定位目标软件的IAT表的首地址（访问程序中任意API在内存中的地址后，往上翻即可定位IAT表的首地址），并计算出表的字节大小。

使用IAT表重建工具（ImportREC）进行修复，在工具中输入（可点击 "AutoSearch" 在目标程序中自动搜索）OEP、RVA（IAT表在我见呢中的偏移）、Size（表的大小），之后点击 "Get Imports" --> "Fix Dump" 选择目标文件进行修复即可。

定位OEP：

inc2l.exe - PID: 5576 - 模块: inc2l.exe - 线程: 主线程 2316 - x32dbg

文件(F) 视图(V) 调试(D) 跟踪(N) 插件(P) 收藏夹(I) 选项(O) 帮助(H) Dec 29 2020 (TitanEngine)

CPU 日志 笔记 断点 内存布局 调用堆栈 SEH链 脚本 符号 源代码 引用 线程 句柄 跟踪

0040AC11 6A 00 push 0
0040AC13 57 FF11 push edi
0040AC14 8B06 call dword ptr ds:[ecx]
0040AC16 5A mov eax, esi
0040AC18 5A mov edx, esi
0040AC19 5E pop esi
0040AC1A 5F pop edi
0040AC1B 59 pop ecx
0040AC1C 5B pop ebx
0040AC1D 5D pop ebp
0040AC1E FFE0 jmp eax
0040AC1F 40 inc eax
0040AC20 40 inc eax
0040AC21 40 inc eax
0040AC22 40 inc eax
0040AC23 0000 add byte ptr ds:[eax], al
0040AC24 0000 add byte ptr ds:[eax], al
0040AC25 0000 add byte ptr ds:[eax], al
0040AC26 0000 add byte ptr ds:[eax], al
0040AC27 0000 add byte ptr ds:[eax], al
0040AC28 0000 add byte ptr ds:[eax], al
0040AC29 0000 add byte ptr ds:[eax], al
0040AC2A 0000 add byte ptr ds:[eax], al
0040AC2B 0000 add byte ptr ds:[eax], al
0040AC2C 0000 add byte ptr ds:[eax], al
0040AC2D 0000 add byte ptr ds:[eax], al
0040AC2E 0000 add byte ptr ds:[eax], al
0040AC2F 0000 add byte ptr ds:[eax], al
0040AC30 0000 add byte ptr ds:[eax], al
0040AC31 0000 add byte ptr ds:[eax], al

EAX 004022E4 "U委SW?"
EBX 00368000 "U委SW?"
ECX 004022E4 "U委SW?"
EDX 004022E4 "U委SW?"
EBP 0019FF80 "U委SW?"
ESP 0019FF74 "U委SW?"
ESI 004022E4 "U委SW?"
EDI 004022E4 "U委SW?"

EIP 0040AC1E inc2l.0040AC1E

EFLAGS 00000304
ZF 0 PF 1 AF 0
OF 0 SF 0 DF 0
CF 0 TF 1 IF 1

默认 (stdcall) 5 解锁

1: [esp+4] 00368000
2: [esp+8] 7674FA10 <kernel32.BaseThreadInitThunk>
3: [esp+C] 0019FFDC
4: [esp+10] 77057684 <ntdll.RtlpWaitForCriticalSection>

内存 十六进制 内存 2 内存 3 内存 4 内存 5 监视 1 局部变量 结构体

76FF1000 36 00 38 00 B0 C1 FF 7A 1C 00 1E 00 90 C1 FF 7A 6 8 " A9v... A9
76FF1010 28 00 2A 00 64 C1 FF 7A 34 00 36 00 2C C1 FF 7A (. * dA9v4 6 . A9v
76FF1020 1F 00 20 00 0C C1 FF 7A 1A 00 1C 00 F0 C0 FF 7A A9v A9v

命令: 命令使用逗号分隔 (像汇编语言): mov eax, ebx

已暂停 INT3 断点于 inc2l.0040AC1E (0040AC1E)! 已调试时间: 0:02:19:39

inc21.exe - PID: 19164 - 模块: inc21.exe - 线程: 主线程 12492 - x32dbg

文件(F) 视图(V) 调试(D) 跟踪(N) 插件(P) 收藏夹(I) 选项(O) 帮助(H) Dec 29 2020 (TitanEngine)

CPU 日志 笔记 断点 内存布局 调用堆栈 SEH链 脚本 符号 源代码 引用 线程 句柄 跟踪

EIP: 004022E4 <inc21.EntryPoint>

地址	十六进制	汇编	注释
004022E4	55	push ebp	
004022E5	8BEC	mov ebp, esp	
004022E7	53	push ebx	
004022E8	57	push esi	
004022E9	BB 00704000	push edi	
004022EA	66 2E F705 4	test word ptr ds:[40284A], 4	
004022EB	75 05	jnz inc21.402300	
004022EC	E9 68050000	jmp inc21.402868	
004022ED	E9 6E030000	jmp inc21.402678	
004022EE	C783 500600	mov dword ptr ds:[ebx+65C], 0	
004022EF	6A F4	push 0	
004022F0	FF15 988140	call dword ptr ds:[&GetStdHandle]	
004022F1	8B83 440600	mov dword ptr ds:[ebx+644], eax	
004022F2	8B83 400600	mov dword ptr ds:[ebx+640], eax	
004022F3	6A F6	push 0	
004022F4	FF15 988140	call dword ptr ds:[&GetStdHandle]	

ebp=0019FF80

.text:004022E4 inc21.exe:\$22E4 #14E4 <EntryPoint>

命令: 命令使用逗号分隔 (像汇编语言): mov eax, ebx

已暂停 INT3 断点于 inc21.0040AC1E (0040AC1E)!

已调试时间: 0:02:26.11

inc21.exe - PID: 19164 - 模块: inc21.exe - 线程: 主线程 12492 - x32dbg

文件(F) 视图(V) 调试(D) 跟踪(N) 插件(P) 收藏夹(I) 选项(O) 帮助(H) Dec 29 2020 (TitanEngine)

CPU 日志 笔记 断点 内存布局 调用堆栈 SEH链 脚本 符号 源代码 引用 线程 句柄 跟踪

EIP: 004022E4 <inc21.EntryPoint>

地址	十六进制	汇编	注释
00402339	8B83 400600	mov dword ptr ds:[ebx+64C], eax	
0040233A	6A 00	push 0	
0040233B	6A 00	push 0	
0040233C	6A 00	push 0	
0040233D	6A 01	push 1	
0040233E	68 00000080	push 80000000	
0040233F	68 3F284000	push inc21.40283F	
00402340	FF15 408140	call dword ptr ds:[&CreateFileA]	
00402341	8B83 300600	mov dword ptr ds:[ebx+63C], eax	
00402342	C783 340600	mov dword ptr ds:[ebx+634], 0	
00402343	C783 380600	mov dword ptr ds:[ebx+638], 0	
00402344	50	push eax	
00402345	54	push esp	
00402346	FFB3 400600	push dword ptr ds:[ebx+64C]	
00402347	FF15 788140	call dword ptr ds:[&GetConsoleMode]	
00402348	85C0	test eax, eax	

ebp=0019FF80

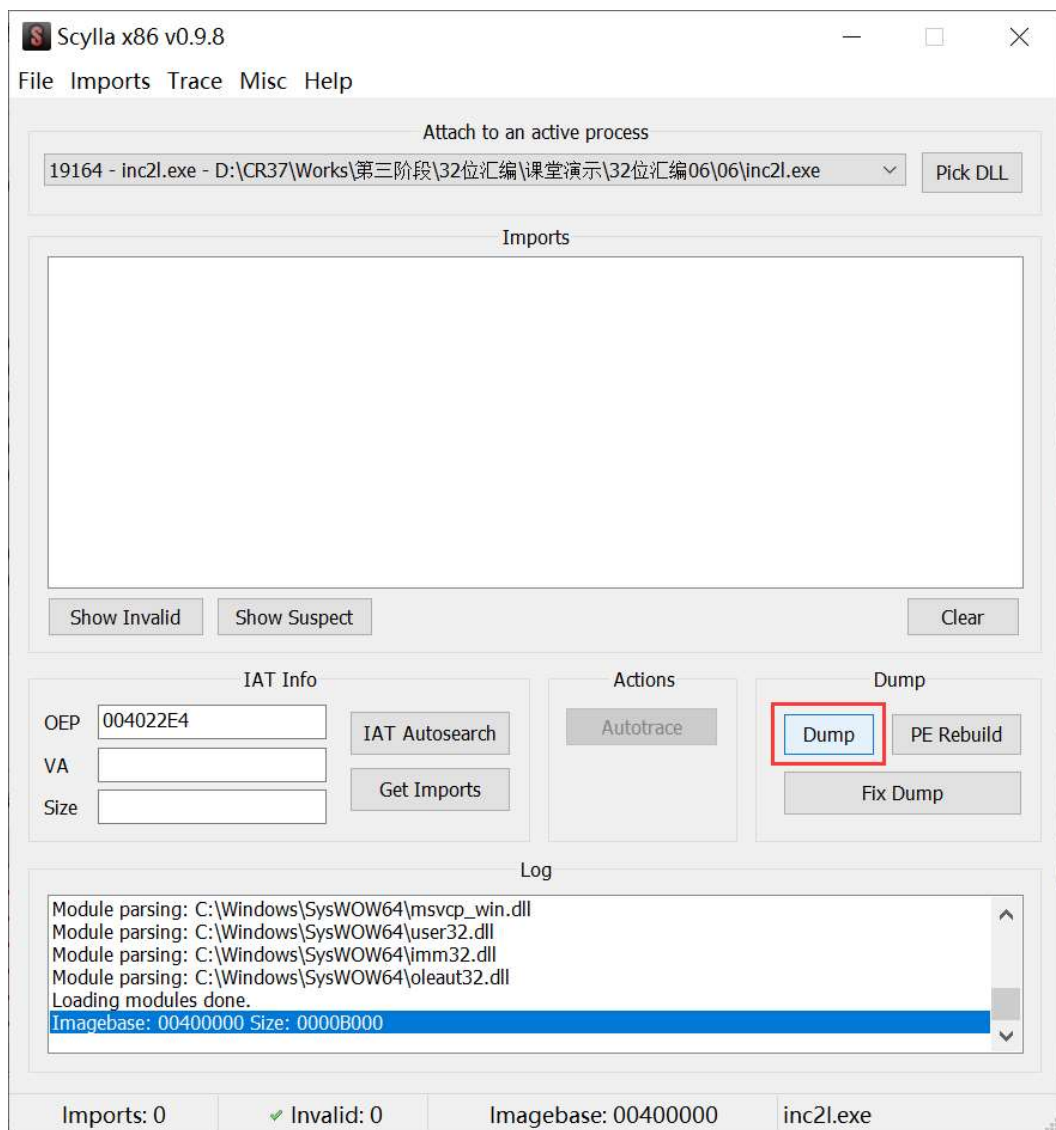
.text:004022E4 inc21.exe:\$22E4 #14E4 <EntryPoint>

命令: 命令使用逗号分隔 (像汇编语言): mov eax, ebx

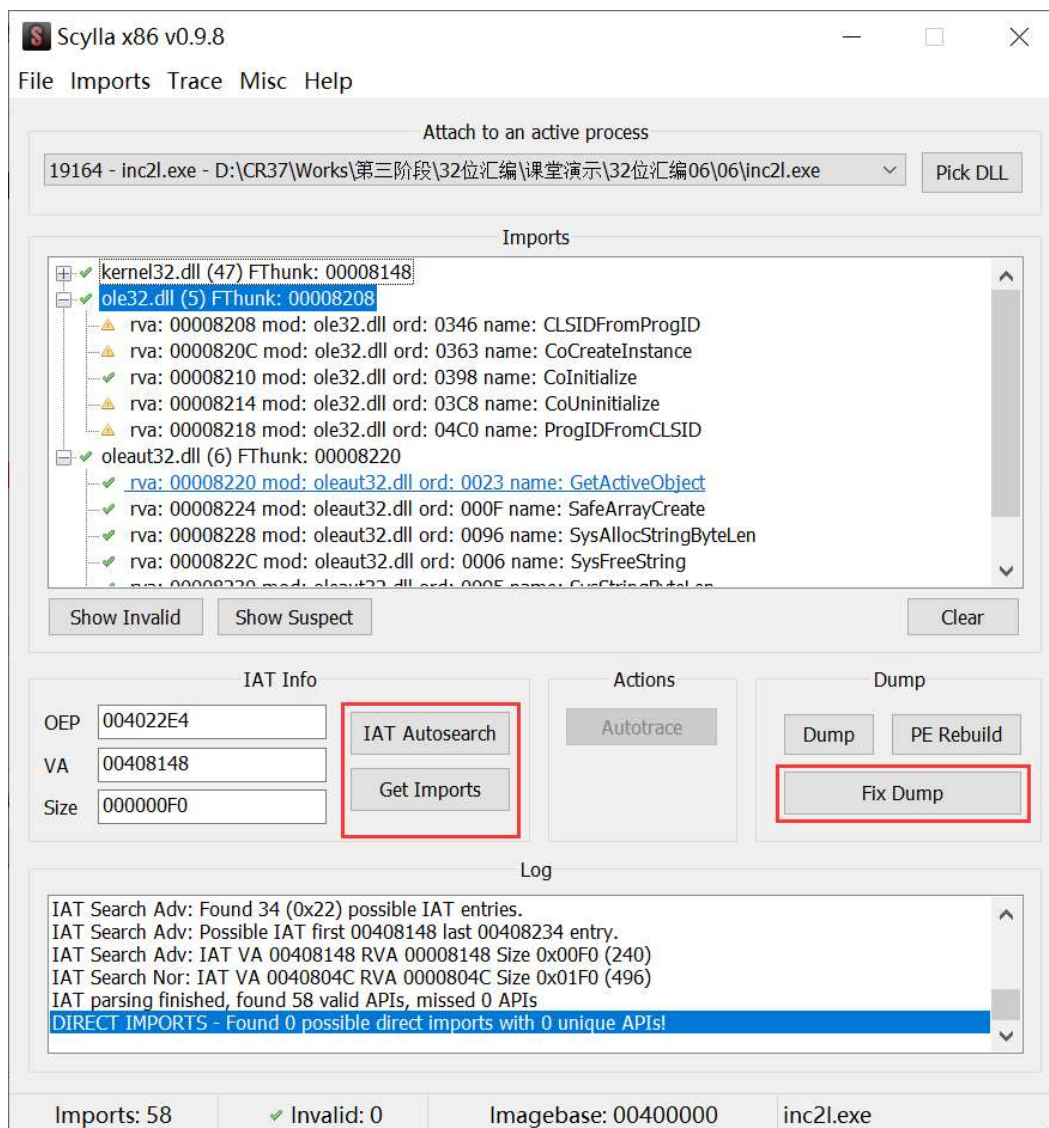
已暂停 INT3 断点于 inc21.0040AC1E (0040AC1E)!

已调试时间: 0:02:26.42

对目标进程进行内存dump:



找到OEP后，可通过通过 "x64dbg" 的 "Scylla" 插件对目标进程修复IAT表，示例如下：



破坏导出表

会导致系统的API出现问题（破坏的使内存，只限当前进程），也可以修改目标进程中API间的RVA（调整API于API的RVA），程序可以自己正常调用，通过调试器、IDA等进行静态分析时，API的调用是乱序的。