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

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

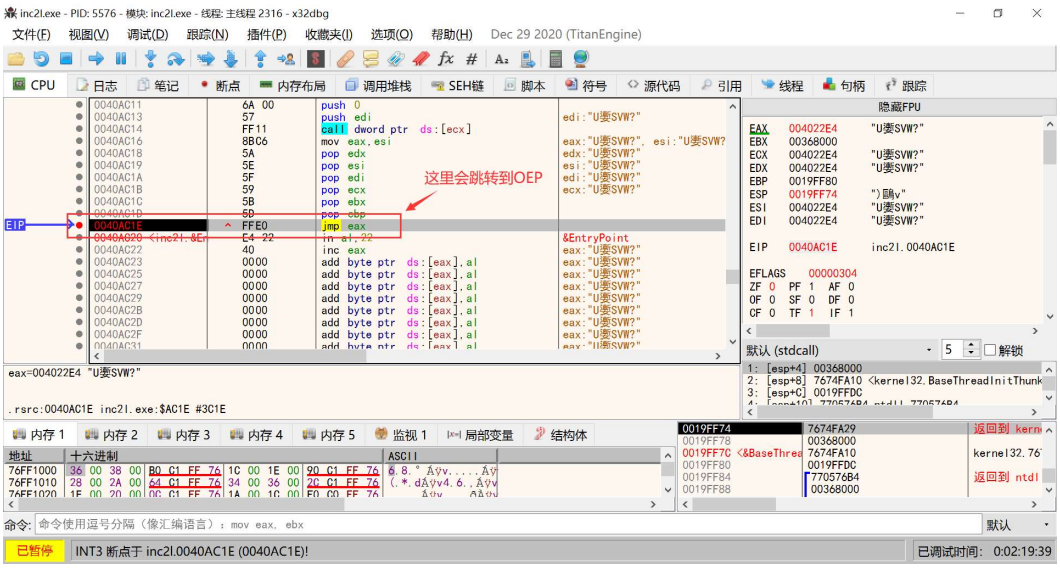
通过工具分析，通过 "CFF" 查看其导入表以及节的名称是否正常，通过调试器查看软件的入口代码是否正常。

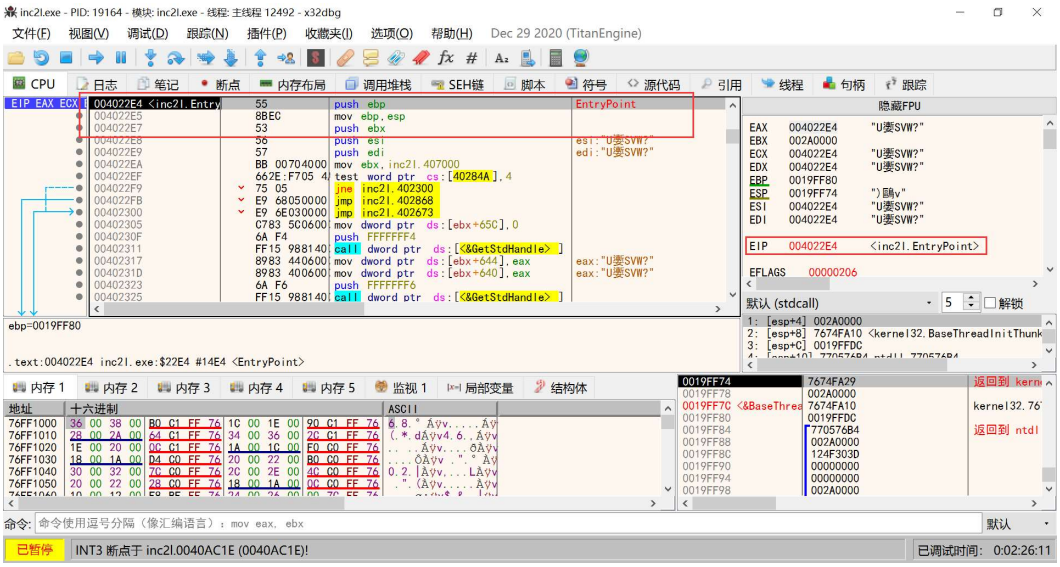
对于加壳程序，需要先修正其OEP，调试程序，等到解密代码走完到OEP位置后就dump内存代码到文件。之后在分析IAT表是否被抹。

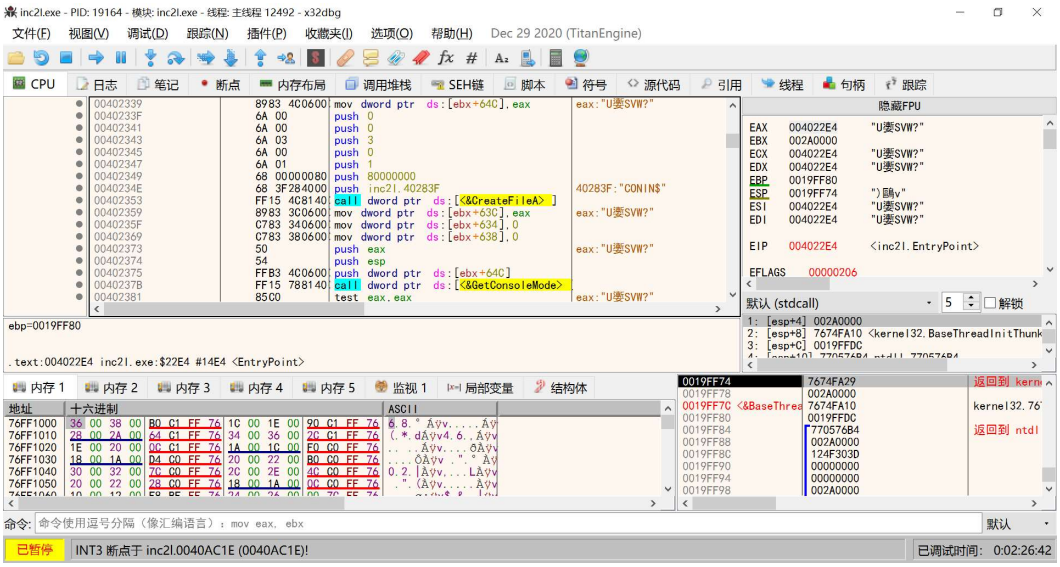
通过调试器去定位目标软件的IAT表的首地址（访问程序中任意API在内存中的地址后，往上翻即可定位IAT表的首地址），并计算出表的字节大小。

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表在我呢见中的偏移）、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。

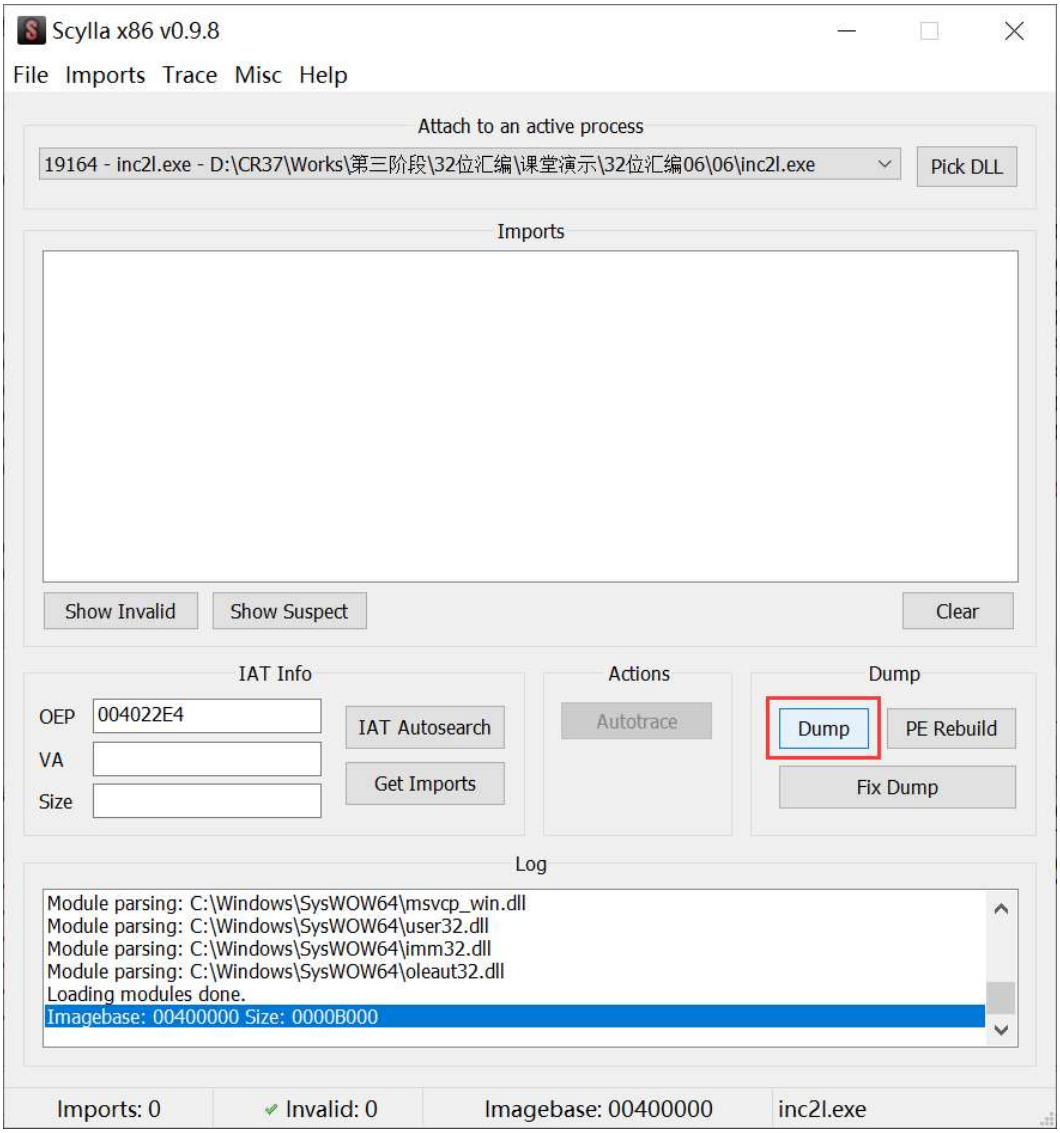
定位OEP：



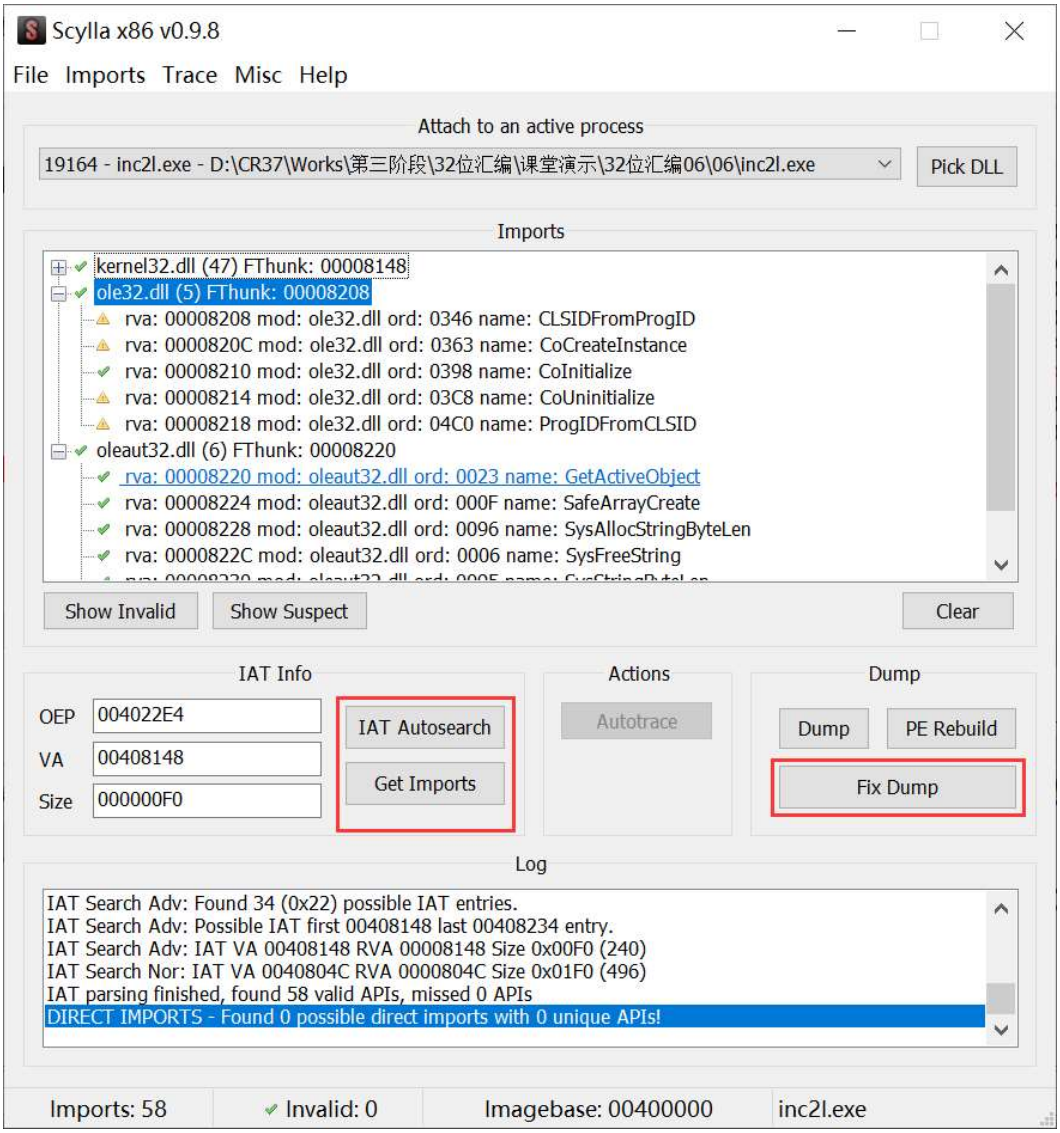




对目标进程进行内存dump：



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。