

电子科技大学信息与软件工程学院

# 实 验 报 告

学 号 2018091618008

姓 名 袁昊男

(实验) 课程名称 软件安全

理论教师 王 静

实验教师 杨 珊

# 电子科技大学

## 实验报告

学生姓名：袁昊男      学号：2018091618008      指导教师：王静

实验地点：信软楼 306      实验时间：2020.12.18

一、实验室名称：信息与软件工程学院实验中心

二、实验名称：搜索 API 函数地址

三、实验学时：4 学时

四、实验原理：

编写 shellcode 时，一般需使用一些 API 函数，例如 `CreateProcess()`, `socket()` 等，这些函数的入口地址位于系统的动态链接库中，由于不同操作系统的动态链接库的加载地址不同，shellcode 中需增加 API 函数自搜索功能。

五、实验目的：

- 1、理解 API 函数搜索原理；
- 2、掌握搜索 `kernel32.dll` 地址的方法；
- 3、掌握搜索 `LoadLibrary()` 及 `GetProcAddress()` 地址的方法。

六、实验内容：

- 1、编程实现求动态链接库 `kernel32.dll` 的虚拟地址；
- 2、编程实现求 `LoadLibrary()` 及 `GetProcAddress()` 虚拟地址。

七、实验器材（设备、元器件）：

PC 机一台。

八、实验步骤：

- 1、定位 `kernel32.dll` 地址
  - (1) 通过段选择字 `FS` 在内存中找到当前的线程控制模块 `TEB`
  - (2) 线程控制块中偏移位置为 `0x30` 的地方存放着指向进程控制块 `PEB` 的指针

- (3) 进程控制块中偏移地址 0x0c 的地址存放着指向 PEB\_LDA\_DATA 结构体的指针，其中存放着已经被装载的动态链接库信息。
- (4) PEB\_LDA\_DATA 结构体偏移地址为 0x1c 的地方存放着指向模块初始化链表的头指针 InInitializationOrderModulelist。
- (5) 模块初始化链表 InInitializationOrderModulelist 中按顺序存放着 PE 装入运行时初始化模块信息，第一个链表节点是 ntdll.dll，第二个链表节点就是 kernel32.dll。
- (6) 找到属于 kernel32.dll 的节点后，在此基础上再偏移 0x08 就是 kernel32.dll 在内存中的基地址。

## 2、定位 LoadLibrary()及 GetProcAddress()地址

- (1) 从 kernel32.dll 的加载基地址开始偏移 0x3c 的地方就是其 PE 头
- (2) PE 头偏移 0x78 的地方存放着指向函数导出表的指针。
- (3) 按以下方式在导出表中算出所需函数的入口地址
  - 1) 导出表偏移 0x1c 处的指针指向存储导出函数偏移地址（RVA）的列表
  - 2) 导出表偏移 0x20 处的指针指向存储导出函数函数名的列表
  - 3) 函数的 RVA 地址和名称按顺序放在 RVA 列表及函数名列表中，根据函数名在函数名称列表中搜索函数序号，再根据函数序号在 RVA 列表中搜索函数对应的 RVA
- (4) 函数对应的 RVA 加上动态链接库的加载地址得到该函数的虚拟地址。

## 九、实验数据及结果分析

### 1、汇编程序代码

```
1. .386
2. .model flat,stdcall
3.
4. include msvcrt.inc
5. includelib msvcrt.lib
6. include user32.inc
7. includelib user32.lib
8. include kernel32.inc
9. includelib kernel32.lib
10.
11. ASSUME FS:NOTHING
12.
13. .data
```

```

14. s1 db "The address of Kernel32.dll base is %x",0
15. s2 db "The address of LoadLibraryA is %x",0
16. s3 db "The address of GetProcAddress is %x",0
17. s4 db "LoadLibraryA",0
18.
19. .code
20. start:
21.     push ebp    ;寄存器保护
22.     mov ebp,esp ;新栈
23.     push ebx    ;寄存器保护
24.     push esi    ;寄存器保护
25.     push edi    ;寄存器保护
26.
27.     mov eax,dword ptr fs:[30h] ;eax = PEB
28.     mov eax,dword ptr [eax+0ch] ;eax = LDR
29.     mov esi,dword ptr [eax+1ch] ;eax = ntdll.dll
30.     lodsd      ;eax = InInitOrder.flink
31.     mov ebp,[eax+8] ;ebp = kernel32.dll
32.     mov edi, ebp ;edi = kernel32.dll
33.     invoke crt_printf,offset s1,ebp
34.
35.     mov eax,[ebp+3Ch] ;eax = PE Header 相对偏移地址
36.     mov edx,[ebp+eax+78h]
37.     add edx,ebp ;edx = 导出表地址
38.     mov ecx,[edx+18h] ;ecx = 输出函数的个数,确定循环次数
39.     mov ebx,[edx+20h]
40.     add ebx, ebp ;ebx = 函数名地址
41.
42. searchGet:
43.     dec ecx ;从后往前找
44.     mov esi,[ebx+ecx*4]
45.     add esi,ebp ;依次找每个函数名称
46.     ;GetProcAddress
47.     mov eax,50746547h
48.     cmp [esi], eax ;'PteG'
49.     jne searchGet
50.     mov eax,41636f72h
51.     cmp [esi+4],eax ;'Acor'
52.     jne searchGet
53.     ;如果是 GetProcAddress, 表示找到了
54.
55.     mov ebx,[edx+24h]
56.     add ebx,ebp ;ebx = 序号数组地址
57.     mov ecx,[ebx+ecx*2] ;ecx = 计算出的序号值

```

```

58.    mov ebx,[edx+1Ch]
59.    add ebx,ebp ;ebx = 函数地址的起始位置
60.    mov ebx,[ebx+ecx*4]
61.    add ebx,ebp ;利用序号值,得到出 GetProcAddress 的地址
62.    invoke crt_printf,offset s2,ebx
63.
64.    ;为局部变量分配空间
65.    push ebp
66.    sub esp, 50h
67.    mov ebp, esp
68.    ;查找 LoadLibrary 的地址
69.    mov [ebp + 40h], ebx    ;把 GetProcAddress 的地址保存到
    ebp + 40 中
70.    ;压入"LoadLibrary/0"的地址
71.    mov eax,offset s4
72.    push eax
73.    push edi    ;edi:kernel32 的基址
74.    call DWORD PTR[ebp + 40h] ;返回值(即 LoadLibrary 的地址)保存
    在 eax 中
75.    invoke crt_printf,offset s3,eax
76.
77.    pop        edi
78.    pop        esi
79.    pop        ebx
80.    mov        esp,ebp
81.    pop        ebp
82.    invoke ExitProcess,0h
83. end start

```

## 2、实验分析

```

1. mov eax,dword ptr fs:[30h] ;eax = PEB
2. mov eax,dword ptr [eax+0ch] ;eax = LDR
3. mov esi,dword ptr [eax+1ch] ;eax = ntdll.dll
4. lodsd        ;eax = InInitOrder.flink
5. mov ebp,[eax+8] ;ebp = kernel32.dll
6. mov edi, ebp   ;edi = kernel32.dll

```

首先通过段选择字 FS 在内存中找到当前的线程控制模块 TEB，线程控制块中偏移位置为 0x30 的地方存放着指向进程控制块 PEB 的指针，此时 EAX 寄存器中保存 PEB 的 RVA 地址；进程控制块中偏移地址 0x0c 的地址存放着指向 PEB\_LDA\_DATA 结构体的指针，其中存放着已经被装载的动态链接库信息，此时 EAX 寄存器中保存 PEB\_LDA\_DATA 的 RVA 地址；PEB\_LDA\_DATA 结构体偏移地址为 0x1c 的地方存放着指向模块初始

化链表的头指针 `InInitializationOrderModulelist`，模块初始化链表 `InInitializationOrderModulelist` 中按顺序存放着 PE 装入运行时初始化模块信息，第一个链表节点是 `ntdll.dll`，第二个链表节点就是 `kernel32.dll` 找到属于 `kernel32.dll` 的节点后，在此基础上再偏移 `0x08` 就是 `kernel32.dll` 在内存中的加载基地址。此时 `EBP` 寄存器保存 `kernel32.dll` 的加载基地址 `VA`。

```
1. mov eax,[ebp+3Ch] ;eax = PE Header 相对偏移地址
2. mov edx,[ebp+eax+78h]
3. add edx,ebp ;edx = 导出表地址
4. mov ecx,[edx+18h] ;ecx = 输出函数的个数,确定循环次数
5. mov ebx,[edx+20h]
6. add ebx, ebp ;ebx = 函数名地址
```

从 `kernel32.dll` 的加载基地址开始偏移 `0x3c` 的地方是 PE 头，PE 头偏移 `0x78` 的地方存放着指向函数导出表的指针，此时 `EDX` 寄存器存放函数导出表地址；函数导出表偏移 `0x18` 处存放函数个数用以确定循环次数，保存至 `ECX` 寄存器中；函数导出表偏移 `0x20` 处存放函数名表地址，存放至 `EBX` 寄存器中。

```
1. dec ecx ;从后往前找
2. mov esi,[ebx+ecx*4]
3. add esi,ebp ;依次找每个函数名称
4. ;GetProcAddress
5. mov eax,50746547h
6. cmp [esi], eax ;'PteG'
7. jne searchGet
8. mov eax,41636f72h
9. cmp [esi+4],eax ;'Acor'
10.jne searchGet
11.;如果是 GetProcAddress, 表示找到了
12.
13.mov ebx,[edx+24h]
14.add ebx,ebp ;ebx = 序号数组地址
15.mov ecx,[ebx+ecx*2] ;ecx = 计算出的序号值
16.mov ebx,[edx+1Ch]
17.add ebx,ebp ;ebx = 函数地址的起始位置
18.mov ebx,[ebx+ecx*4]
19.add ebx,ebp ;利用序号值, 得到出 GetProcAddress 的地址
```

从函数名表从后往前查找 `GetProcAddress()` 函数。依次比较每一个函数的函数名，如果和 `GetProcAddress()` 函数名一致，则找到。函数导出表偏移 `0x24` 处存放函数序号数组地址，计算出 `GetProcAddress()` 函数的序号值保存至 `ECX` 寄存器中。函数导出表偏移 `0x1c` 处存放函数偏移地址表。从而按序号找到 `GetProcAddress()` 函数地址。

```

1. ;查找 LoadLibrary 的地址
2. mov [ebp + 40h], ebx    ;把 GetProcAddress 的地址保存到 ebp + 40
   中
3. ;压入"LoadLibrary/0"的地址
4. mov eax,offset s4
5. push eax
6. push edi    ;edi:kernel32 的基址
7. call DWORD PTR[ebp + 40h] ;返回值(即 LoadLibrary 的地址)保存在 eax
   中

```

这里通过调用 GetProcAddress()函数来查找 LoadLibrary()函数地址。GetProcAddress()是一个计算机函数，功能是检索指定的动态链接库中的输出库函数地址。如果函数调用成功，返回值是 DLL 中的输出函数地址。首先将 GetProcAddress()函数地址保存到[ebp+40]中，并将要查找的函数名压栈作为参数传递给 GetProcAddress()，最后返回值即 LoadLibrary()函数地址，保存在 EAX 寄存器中。

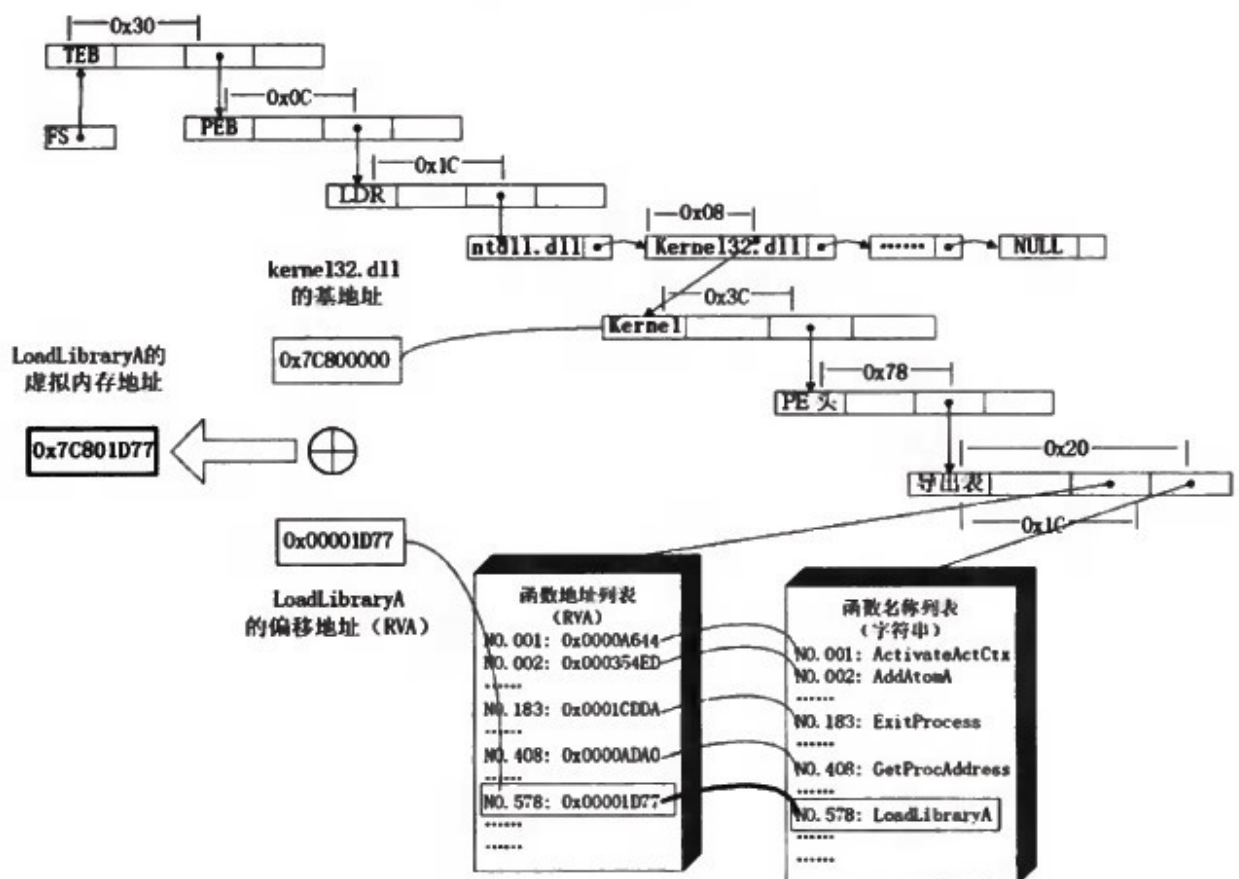
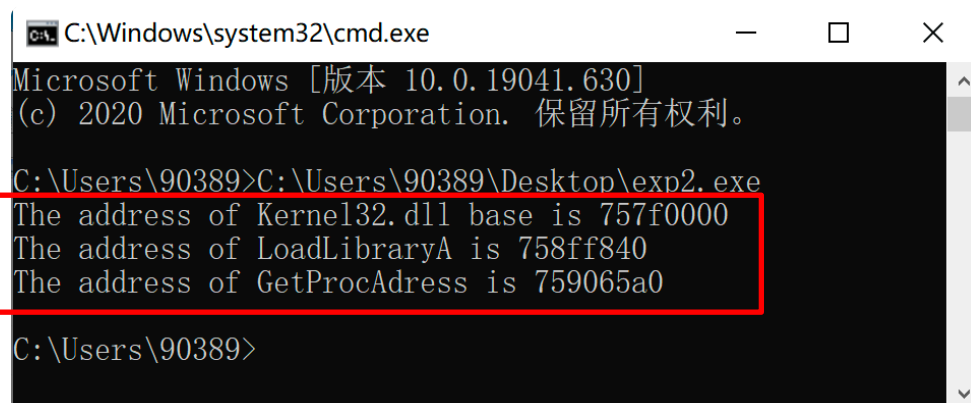


图 3.4.1 在 shellcode 中动态定位 API 的原理

上图清晰地展示出了动态定位 API 的原理。

### 3、运行程序



```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 10.0.19041.630]
(c) 2020 Microsoft Corporation. 保留所有权利。

C:\Users\90389>C:\Users\90389\Desktop\exp2.exe
The address of Kernel32.dll base is 757f0000
The address of LoadLibraryA is 758ff840
The address of GetProcAddress is 759065a0

C:\Users\90389>
```

结论: 输出 kernel32.dll 地址为 0x757f0000, LoadLibrary() 地址为 0x758ff840, GetProcAddress() 地址为 0x759065a0。

## 十、实验结论

编写 shellcode 时, 一般需使用一些 API 函数, 例如 CreateProcess(), socket() 等, 这些函数的入口地址位于系统的动态链接库中, 由于不同操作系统的动态链接库的加载地址不同, shellcode 中需增加 API 函数自搜索功能。通过学习动态定位 API 的原理并编写汇编程序, 正确输出了 kernel32.dll、LoadLibrary()、GetProcAddress() 在内存中的地址, 并对实验代码进行了进一步分析。

## 十一、总结及心得体会

1、为什么 PEB、TEB 中能找到 kernel32.dll 的地址?

答: 根据图 3.4.1 动态定位 API 的原理: 首先通过段选择字 FS 在内存中找到当前的线程控制模块 TEB, 线程控制块中偏移位置为 0x30 的地方存放着指向进程控制块 PEB 的指针; 进程控制块中偏移地址 0x0c 的地址存放着指向 PEB\_LDA\_DATA 结构体的指针, 其中存放着已经被装载的动态链接库信息; PEB\_LDA\_DATA 结构体偏移地址为 0x1c 的地方存放着指向模块初始化链表的头指针 InInitializationOrderModulelist, 模块初始化链表 InInitializationOrderModulelist 中按顺序存放着 PE 装入运行时初始化模块信息, 第一个链表节点是 ntdll.dll, 第二个链表节点就是 kernel32.dll 找到属于 kernel32.dll 的节点后, 在此基础上再偏移 0x08 就是 kernel32.dll 在内存中的基地址。

2、dll 文件的导出表的结构是怎样的?

答: 导出表数据结构定义如下:

DWORD     Characteristics;
----------------------------



DWORD	TimeStamp;	
WORD	MajorVersion;	
WORD	MinorVersion;	
DWORD	Name;	
DWORD	Base;	
DWORD	NumberOfFunctions;	
DWORD	NumberOfNames;	
DWORD	AddressOfFunctions;	// RVA from base of image
DWORD	AddressOfNames;	// RVA from base of image
DWORD	AddressOfNameOrdinals;	// RVA from base of image

3、搜索得到 LoadLibrary()及 GetProcAddress()虚拟地址后，怎样得到例如 printf()等函数的地址？

答：调用 GetProcAddress()函数，将待查找的函数名作为参数传入，返回值即是函数地址。

## 十二、对本实验过程及方法、手段的改进建议

本实验设计与教材结合紧密、具有一定难度。通过学习动态定位 API 的原理，进而编写汇编代码输出 kernel32.dll、LoadLibrary()、GetProcAddress()函数地址，运行后得到了验证。本实验强化了学生对 shellcode 编写过程中动态定位 API 的理解；此外，还使学生熟悉汇编语言编译环境，对软件安全的深入学习打下了坚实的基础。

**报告评分：**

**指导教师签字：**