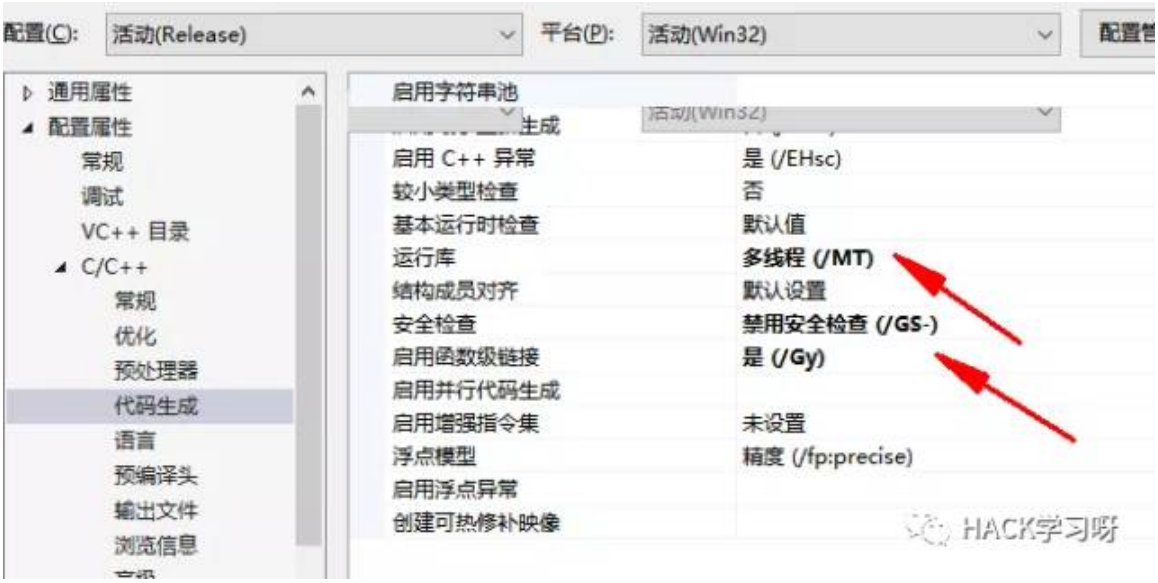
ShellCode生成框架

原创一寸一叶 HACK学习呀
2020-11-07原文

这里先写个简单的静态加载到exe文件中，明天再来写个动态的

因为vs编译后自己会生成很多东西，我们稍微配置下



先获取kernel32基址

```
__declspec(naked) DWORD getKernel32(){  __asm  {     mov eax, fs:[30h]
//PEB       mov eax, [eax + 0ch]  //PEB->Ldr       mov eax, [eax + 14h]
//PEB->Ldr.InMemOrder       mov eax, [eax]  //第二个模块       mov eax,
[eax]  //第三个模块       mov eax, [eax + 10h]  //base address       ret
}}
```

获取GetProcAddress函数地址

```
FARPROC     getProcAddress(HMODULE  hModuleBase){     PIMAGE_DOS_HEADER
lpDosHeader  =  (PIMAGE_DOS_HEADER)hModuleBase;     PIMAGE_NT_HEADERS32
lpNtHeader  =  (PIMAGE_NT_HEADERS32)((DWORD)hModuleBase  +  lpDosHeader-
>e_lfanew);                                  if               (!lpNtHeader-
>OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].Size)     {
return       NULL;               }              if       (!lpNtHeader-
>OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAdd
ress)   {      return  NULL;   }   PIMAGE_EXPORT_DIRECTORY  lpExports  =
(PIMAGE_EXPORT_DIRECTORY)((DWORD)hModuleBase    +    (DWORD)lpNtHeader-
>OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAdd
ress);      PDWORD   lpdwFunName   =   (PDWORD)((DWORD)hModuleBase   +
(DWORD)lpExports->AddressOfNames);            PWORD      lpwOrd      =
(PWORD)((DWORD)hModuleBase + (DWORD)lpExports->AddressOfNameOrdinals);
PDWORD  lpdwFunAddr  =  (PDWORD)((DWORD)hModuleBase  +  (DWORD)lpExports-
>AddressOfFunctions);  DWORD dwLoop = 0;  FARPROC pRet = NULL;  for (;
dwLoop <= lpExports->NumberOfNames - 1; dwLoop++)  {     char* pFunName
=   (char*)(lpdwFunName[dwLoop]   +   (DWORD)hModuleBase);           if
(pFunName[0] == 'G' &&       pFunName[1] == 'e' &&       pFunName[2] ==
't'  &&         pFunName[3]  ==  'P'  &&         pFunName[4]  ==  'r'  &&
pFunName[5] == 'o' &&       pFunName[6] == 'c' &&       pFunName[7] ==
'A'  &&         pFunName[8]  ==  'd'  &&         pFunName[9]  ==  'd'  &&
pFunName[10] == 'r' &&       pFunName[11] == 'e' &&       pFunName[12]
==  's'  &&           pFunName[13]  ==  's')       {           pRet  =
(FARPROC)(lpdwFunAddr[lpwOrd[dwLoop]]      +      (DWORD)hModuleBase);
break;    }  }  return pRet;}
```

首先定义ms-dos头

```
//some code……PIMAGE_DOS_HEADER lpDosHeader =
(PIMAGE_DOS_HEADER)hModuleBase;
```

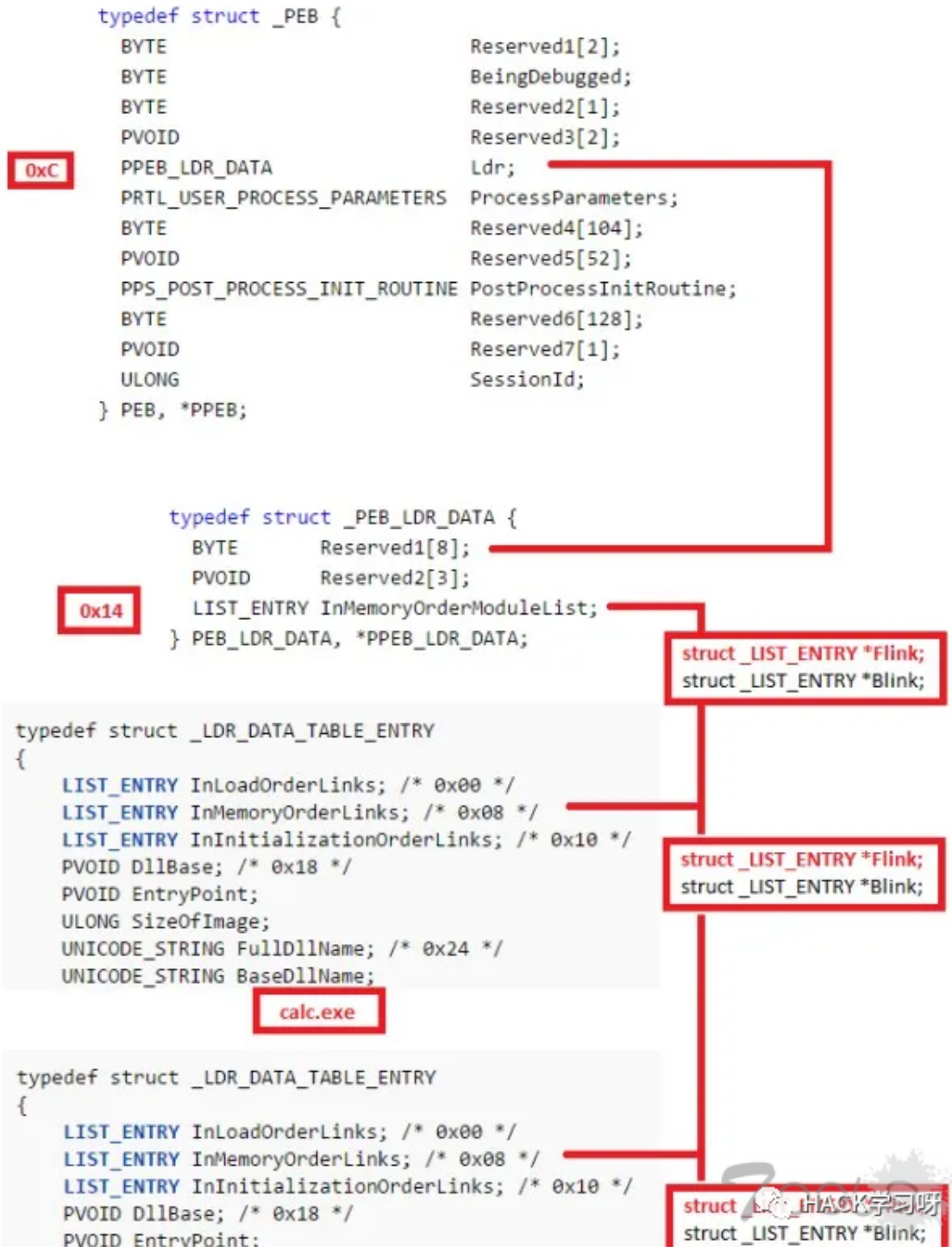
然后得到pe头image-nt-header

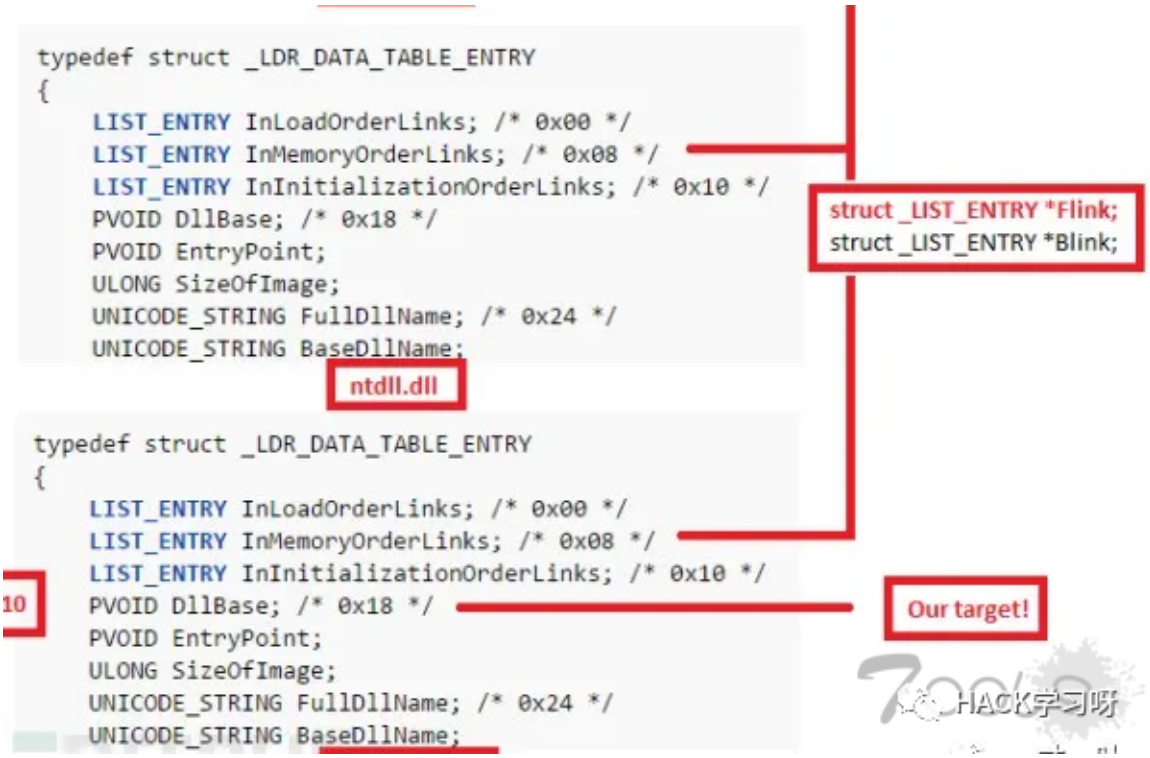
```
//some code……PIMAGE_NT_HEADERS32 lpNtHeader =
(PIMAGE_NT_HEADERS32)((DWORD)hModuleBase + lpDosHeader->e_lfanew);
```

直接dos头加e_lfanew，这里因为是c++代码就不用汇编写入偏移地址3c等等，后面也要贴上汇编代码，结合一起看其实也不难理解

```
//some code……if (!lpNtHeader-
>OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].Size) {
return NULL; } if (!lpNtHeader-
>OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAdd
ress) { return NULL; }
```

这里还是贴上这个图(转载的图)


typedef struct _PEB {
BYTE Reserved1[2];
BYTE BeingDebugged;
BYTE Reserved2[1];
PVOID Reserved3[2];
0xC
PPEB_LDR_DATA Ldr;
PRTL_USER_PROCESS_PARAMETERS ProcessParameters;
BYTE Reserved4[104];
PVOID Reserved5[52];
PPS_POST_PROCESS_INIT_ROUTINE PostProcessInitRoutine;
BYTE Reserved6[128];
PVOID Reserved7[1];
ULONG SessionId;
} PEB, *PPEB;
typedef struct _PEB_LDR_DATA {
BYTE Reserved1[8];
PVOID Reserved2[3];
0x14
LIST_ENTRY InMemoryOrderModuleList;
} PEB_LDR_DATA, *PPEB_LDR_DATA;
struct _LIST_ENTRY *Flink;
struct _LIST_ENTRY *Blink;
typedef struct _LDR_DATA_TABLE_ENTRY
{
LIST_ENTRY InLoadOrderLinks; /* 0x00 */
LIST_ENTRY InMemoryOrderLinks; /* 0x08 */
LIST_ENTRY InInitializationOrderLinks; /* 0x10 */
PVOID DllBase; /* 0x18 */
PVOID EntryPoint;
ULONG SizeOfImage;
UNICODE_STRING FullDllName; /* 0x24 */
UNICODE_STRING BaseDllName;
struct _LIST_ENTRY *Flink;
struct _LIST_ENTRY *Blink;
calc.exe
typedef struct _LDR_DATA_TABLE_ENTRY
{
LIST_ENTRY InLoadOrderLinks; /* 0x00 */
LIST_ENTRY InMemoryOrderLinks; /* 0x08 */
LIST_ENTRY InInitializationOrderLinks; /* 0x10 */
PVOID DllBase; /* 0x18 */
PVOID EntryPoint;
struct
struct _LIST_ENTRY *Blink;



在pe-option-header里面存在一个size和virualaddress，我们还是主要看VirtualAddress（相对虚拟地址）字段，我们得到这个结构体

```
//some code……PIMAGE_EXPORT_DIRECTORY lpExports = (PIMAGE_EXPORT_DIRECTORY)((DWORD)hModuleBase + (DWORD)lpNtHeader->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress);
```

我们将会使用这个结构的如下字段：

AddressOfFunctions：指向一个DWORD类型的数组，每个数组元素指向一个函数地址。AddressOfNames：指向一个DWORD类型的数组，每个数组元素指向一个函数名称的字符串。AddressOfNameOrdinals：指向一个WORD类型的数组，每个数组元素表示相应函数的排列序号（16位整数）

```
//some code……PDWORD lpdwFunName = (PDWORD)((DWORD)hModuleBase + (DWORD)lpExports->AddressOfNames); PWORD lpwOrd = (PWORD)((DWORD)hModuleBase + (DWORD)lpExports->AddressOfNameOrdinals);
```

```
PDWORD lpdwFunAddr = (PDWORD)((DWORD)hModuleBase + (DWORD)lpExports->AddressOfFunctions);
```

然后判断是否为GetProcAddress函数是就返回

```
//some code…… DWORD dwLoop = 0;  FARPROC pRet = NULL;  for (; dwLoop <= lpExports->NumberOfNames - 1; dwLoop++)  {  char* pFunName = (char*)(lpdwFunName[dwLoop] + (DWORD)hModuleBase);  if (pFunName[0] == 'G' &&  pFunName[1] == 'e' &&  pFunName[2] == 't' && pFunName[3] == 'P' &&  pFunName[4] == 'r' &&  pFunName[5] == 'o' &&  pFunName[6] == 'c' &&  pFunName[7] == 'A' && pFunName[8] == 'd' &&  pFunName[9] == 'd' &&  pFunName[10] == 'r' &&  pFunName[11] == 'e' &&  pFunName[12] == 's' && pFunName[13]  ==  's')  {  pRet  = (FARPROC)(lpdwFunAddr[lpwOrd[dwLoop]]  +  (DWORD)hModuleBase); break;  }  }  return pRet;
```

这里用到了导出表里面得一个single每次查找一次就+1这里返回回去就是-1然后逐一进行判断

头部再定义一下

```
//some code……DWORD getKernel32();FARPROC  getProcAddress(HMODULE hModuleBase);
```

这里kernel32.dll和GetProcess函数地址都得到了后面就好说了

这里我们举CreateFile和Messagebox例子

这里是原来应该得写法

```
//some code……typedef HANDLE(WINAPI *FN_CreateFileA)(  _In_ LPCSTR lpFileName,  _In_ DWORD dwDesiredAccess,  _In_ DWORD dwShareMode, _In_opt_ LPSECURITY_ATTRIBUTES lpSecurityAttributes,  _In_ DWORD dwCreationDisposition,  _In_ DWORD dwFlagsAndAttributes,  _In_opt_ HANDLE hTemplateFile  );  FN_CreateFileA fn_CreateFileA =
```

```
(FN_CreateFileA)GetProcAddress(LoadLibraryA("kernel32.dll"),
"CreateFileA");
```

我们先处理LoadLibraryA("kernel32.dll")

先得到GetProcAddress

```
typedef FARPROC(WINAPI * FN_GetProcAddress)(     _In_ HMODULE hModule,
_In_ LPCSTR lpProcName      );   FN_GetProcAddress fn_GetProcAddress =
(FN_GetProcAddress)getProcAddress((HMODULE)getKernel32());
```

然后把"CreateFileA"字符串替换了

```
char szCreateFile[] = { 'C', 'r', 'e', 'a', 't', 'e', 'F', 'i', 'l',
'e', 'A',0 };
```

这里完整为

```
typedef FARPROC(WINAPI * FN_GetProcAddress)(     _In_ HMODULE hModule,
_In_ LPCSTR lpProcName      );   FN_GetProcAddress fn_GetProcAddress =
(FN_GetProcAddress)getProcAddress((HMODULE)getKernel32());      typedef
HANDLE(WINAPI *FN_CreateFileA)(      _In_ LPCSTR lpFileName,      _In_
DWORD dwDesiredAccess,       _In_ DWORD dwShareMode,       _In_opt_
LPSECURITY_ATTRIBUTES   lpSecurityAttributes,            _In_   DWORD
dwCreationDisposition,    _In_ DWORD dwFlagsAndAttributes,    _In_opt_
HANDLE hTemplateFile      );   char szCreateFile[] = { 'C', 'r', 'e',
'a',  't',  'e',  'F',  'i',  'l',  'e',  'A',0  };    FN_CreateFileA
fn_CreateFileA                                                    =
(FN_CreateFileA)fn_GetProcAddress((HMODULE)getKernel32(),
szCreateFile);  char szNewFile[] = { '1', '.', 't', 'x', 't', '\0' };
fn_CreateFileA(szNewFile, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS, 0,
NULL);
```

下面为MessageBoxA

```
typedef HMODULE (WINAPI* FN_LoadLibraryA)( _In_ LPCSTR lpLibFileName ); char szLoadLibraryA[] = { 'L', 'o', 'a', 'd', 'L', 'i', 'b', 'r', 'a', 'r', 'y', 'A', 0 }; FN_LoadLibraryA fn_LoadLibraryA = (FN_LoadLibraryA)fn_GetProcAddress((HMODULE)getKernel32(), szLoadLibraryA); typedef int (WINAPI* FN_MessageBoxA)( _In_opt_ HWND hWnd, _In_opt_ LPCSTR lpText, _In_opt_ LPCSTR lpCaption, _In_ UINT uType); // 正常写法 //FN_MessageBoxA fn_MessageBoxA = (FN_MessageBoxA)GetProcAddress(LoadLibraryA("user32.dll"), "MessageBoxA"); char szUser32[] = { 'U', 's', 'e', 'r', '3', '2', '.', 'd', 'l', 'l', 0 }; char szMessageboxA[] = { 'M', 'e', 's', 's', 'a', 'g', 'e', 'B', 'o', 'x', 'A', 0 }; FN_MessageBoxA fn_MessageBoxA = (FN_MessageBoxA)fn_GetProcAddress(fn_LoadLibraryA(szUser32), szMessageboxA); char szHello[] = { 'y', 'i', 'c', 'u', 'n', 'y', 'i', 'y', 'e', 0 }; char szTip[] = { 't', 'i', 'p', 0 }; fn_MessageBoxA(NULL, szHello, szTip, MB_OK);
```

看看正常写法

```
FN_MessageBoxA fn_MessageBoxA = (FN_MessageBoxA)GetProcAddress(LoadLibraryA("user32.dll"), "MessageBoxA");
```

因为获取得是user32.dll而不是直接一样得kernel32.dll所以我们要获取下LoadLibraryA得地址

```
typedef HMODULE (WINAPI* FN_LoadLibraryA)( _In_ LPCSTR lpLibFileName ); char szLoadLibraryA[] = { 'L', 'o', 'a', 'd', 'L', 'i', 'b', 'r', 'a', 'r', 'y', 'A', 0 }; FN_LoadLibraryA fn_LoadLibraryA = (FN_LoadLibraryA)fn_GetProcAddress((HMODULE)getKernel32(), szLoadLibraryA);
```

然后就是获取MessageBoxA得地址

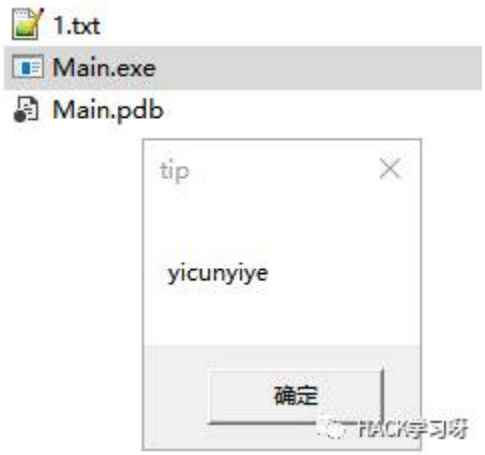
```
typedef int (WINAPI* FN_MessageBoxA)( _In_opt_ HWND hWnd, _In_opt_ LPCSTR lpText, _In_opt_ LPCSTR lpCaption, _In_ UINT
```

```
uType);     //   正 常 写 法      //FN_MessageBoxA  fn_MessageBoxA  =
(FN_MessageBoxA)GetProcAddress(LoadLibraryA("user32.dll"),
"MessageBoxA");   char szUser32[] = { 'U', 's', 'e', 'r', '3', '2',
'.', 'd', 'l', 'l', 0 };  char szMessageboxA[] = { 'M', 'e', 's', 's',
'a', 'g', 'e', 'B', 'o', 'x', 'A', 0 };  FN_MessageBoxA fn_MessageBoxA
=          (FN_MessageBoxA)fn_GetProcAddress(fn_LoadLibraryA(szUser32),
szMessageboxA);
```

最后再输出

```
char szHello[] = { 'y', 'i', 'c', 'u', 'n', 'y', 'i', 'y', 'e', 0 };
char szTip[] = { 't', 'i', 'p', 0 };  fn_MessageBoxA(NULL, szHello,
szTip, MB_OK);
```

运行结果可以看到没什么问题



然后我们peid打开

看下偏移是400然后我们ue打开然后找到对应得偏移地址复制这个16进制就是我们需要的shellcode，然后把shellcode插入到进程中执行就可以了，这里我们可以静态得插入到未执行得exe文件中，或者动态的插入到正在执行得进程的内存中，这里我们试试插入到未执行的exe文件中

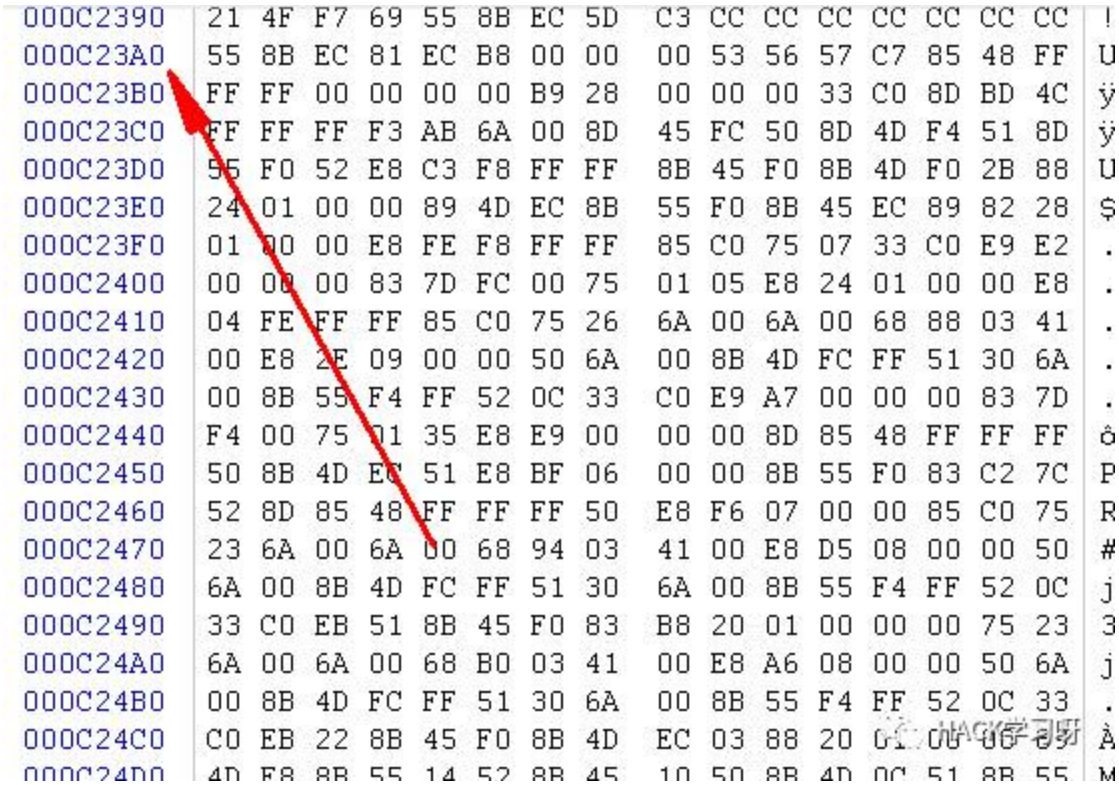
```
558BEC83EC4C56E8E40000008BC8E8FD0000008BF0C745D0437265618D45D0C745D474
65466950C745D86C654100E8BD00000050FFD66A006A006A026A006A0068000000408D
4DF4C745F4312E74785166C745F87400FFD08D45B4C745B44C6F616450C745B84C6962
72C745BC61727941C645C000E87600000050FFD68D4DC4C745DC55736572518D4DDCC7
45E033322E645166C745E46C6CC645E600C745C44D657373C745C861676542C745CC6F
784100FFD050FFD66A008D4DFCC745E879696375518D4DE8C745EC6E796979516A0066
C745F06500C745FC74697000FFD033C05E8BE55DC3CCCCCCCCCCCCCCCCCCCC64A1300000
008B400C8B40148B008B008B4010C3CCCCCCCCCCCCCCCCCCCCCCCC558BEC8BD183EC08
8B423C837C107C00750633C08BE55DC38B44107885C074F28B4C102403CA538B5C1020
894DFC03DA8B4C101C568B74101803CA57894DF833FF33C94E8B048B03C2803847754E
807801657548807802747542807803507 53C807804727536807805 6F75308078066375
2A807807417524807808647 51E807809647 51880780A72751280780B65750C80780C73
750680780D73740E413BCE76A38BC75F5E5B8BE55DC38B45FC8B7DF80FB704488B3C87
03FA8BC75F5E5B8BE55DC30000
```

这里是29行+4个，我用以前写的端口扫描做测试

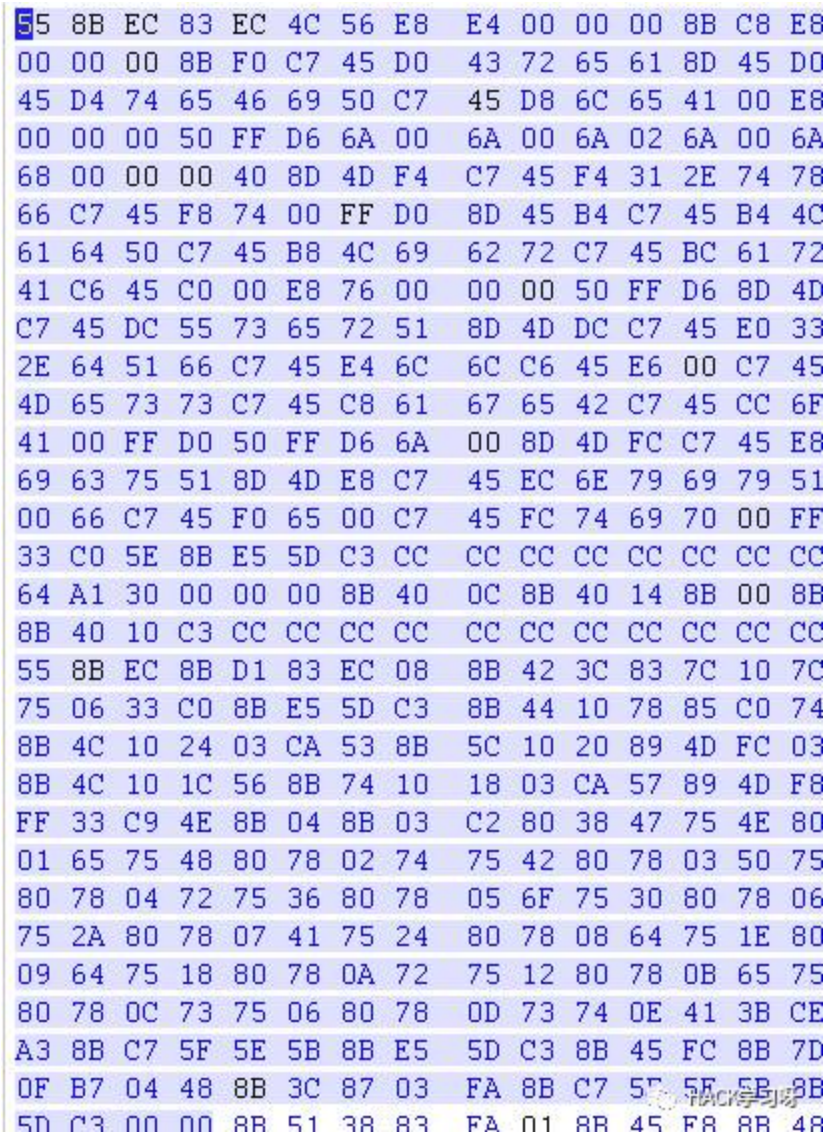
先看看入口的文件偏移

000C23A0然后用winhex打开

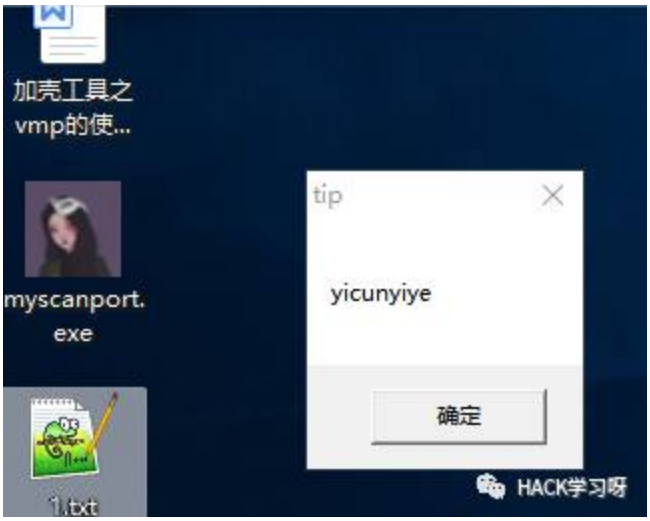
然后我们转到偏移地址



修改同样大小他shellcode替换了，所以只要运行这个exe就会运行我们的shellcode


55 8B EC 83 EC 4C 56 E8 E4 00 00 00 8B C8 E8
00 00 00 8B F0 C7 45 D0 43 72 65 61 8D 45 D0
45 D4 74 65 46 69 50 C7 45 D8 6C 65 41 00 E8
00 00 00 50 FF D6 6A 00 6A 00 6A 02 6A 00 6A
68 00 00 00 40 8D 4D F4 C7 45 F4 31 2E 74 78
66 C7 45 F8 74 00 FF D0 8D 45 B4 C7 45 B4 4C
61 64 50 C7 45 B8 4C 69 62 72 C7 45 BC 61 72
41 C6 45 C0 00 E8 76 00 00 00 50 FF D6 8D 4D
C7 45 DC 55 73 65 72 51 8D 4D DC C7 45 E0 33
2E 64 51 66 C7 45 E4 6C 6C C6 45 E6 00 C7 45
4D 65 73 73 C7 45 C8 61 67 65 42 C7 45 CC 6F
41 00 FF D0 50 FF D6 6A 00 8D 4D FC C7 45 E8
69 63 75 51 8D 4D E8 C7 45 EC 6E 79 69 79 51
00 66 C7 45 F0 65 00 C7 45 FC 74 69 70 00 FF
33 C0 5E 8B E5 5D C3 CC CC CC CC CC CC CC CC
64 A1 30 00 00 00 8B 40 0C 8B 40 14 8B 00 8B
8B 40 10 C3 CC CC CC CC CC CC CC CC CC CC CC
55 8B EC 8B D1 83 EC 08 8B 42 3C 83 7C 10 7C
75 06 33 C0 8B E5 5D C3 8B 44 10 78 85 C0 74
8B 4C 10 24 03 CA 53 8B 5C 10 20 89 4D FC 03
8B 4C 10 1C 56 8B 74 10 18 03 CA 57 89 4D F8
FF 33 C9 4E 8B 04 8B 03 C2 80 38 47 75 4E 80
01 65 75 48 80 78 02 74 75 42 80 78 03 50 75
80 78 04 72 75 36 80 78 05 6F 75 30 80 78 06
75 2A 80 78 07 41 75 24 80 78 08 64 75 1E 80
09 64 75 18 80 78 0A 72 75 12 80 78 0B 65 75
80 78 0C 73 75 06 80 78 0D 73 74 0E 41 3B CE
A3 8B C7 5F 5E 5B 8B E5 5D C3 8B 45 FC 8B 7D
0F B7 04 48 8B 3C 87 03 FA 8B C7 5F 5E 5B 8B
5D C3 00 00 8B 51 38 83 FA 01 8B 45 E8 8B 48

然后我们保存运行



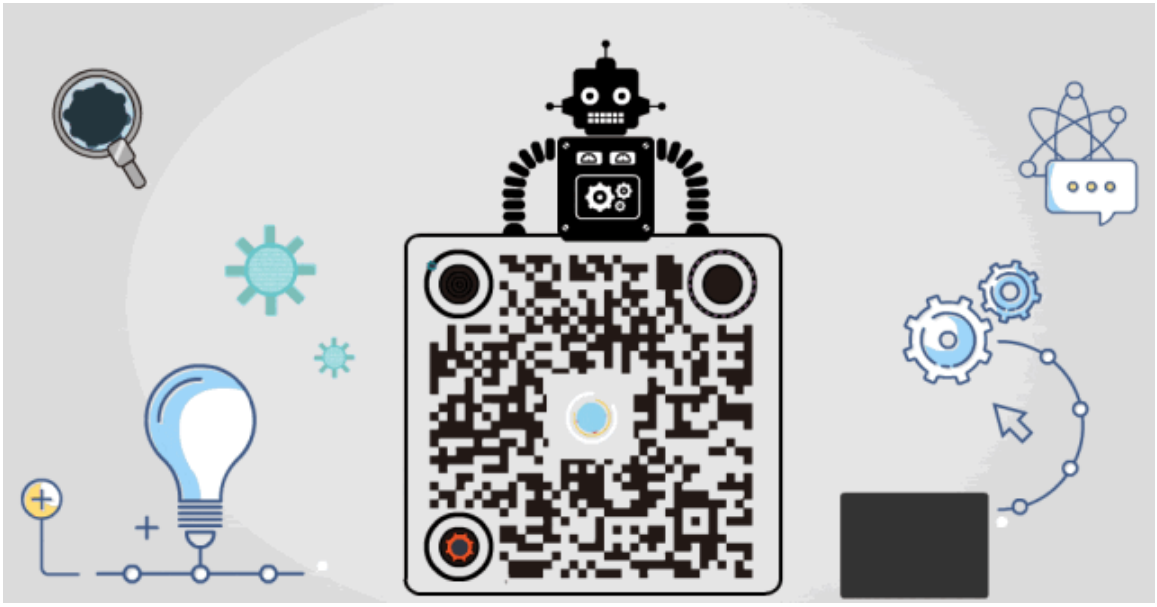
说明我们的shellcode插入了这个exe中，执行他就执行了我们的shellcode

我们也可以把他shellcode生成为一个bin文件再写个加载器运行

END

点赞 在看 转发

原创投稿作者：一寸一叶



精选留言

用户设置不下载评论