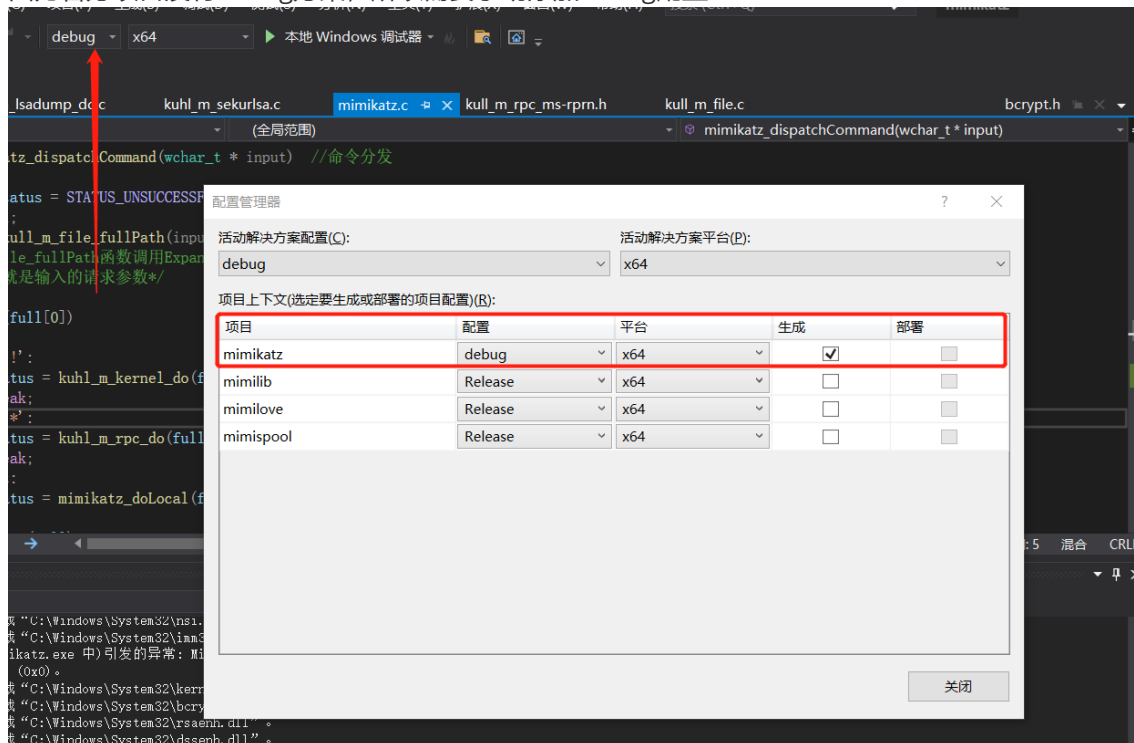


# S-Mimikatz源码调试

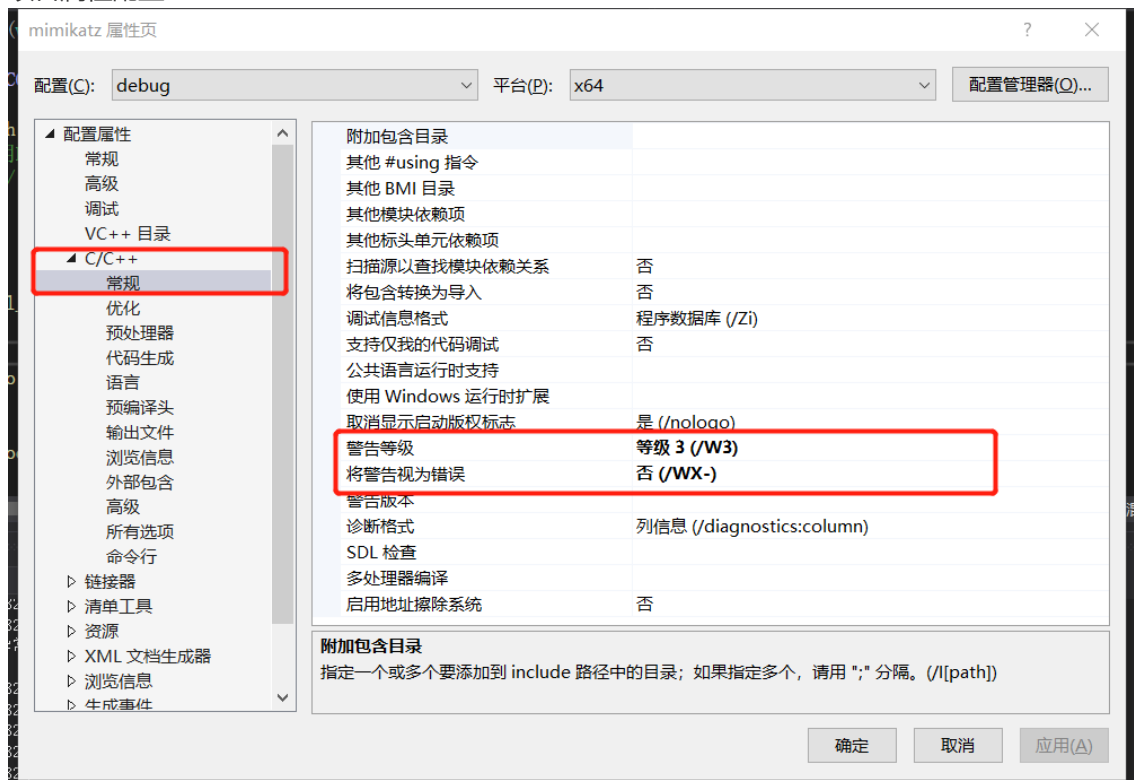
## 前期准备

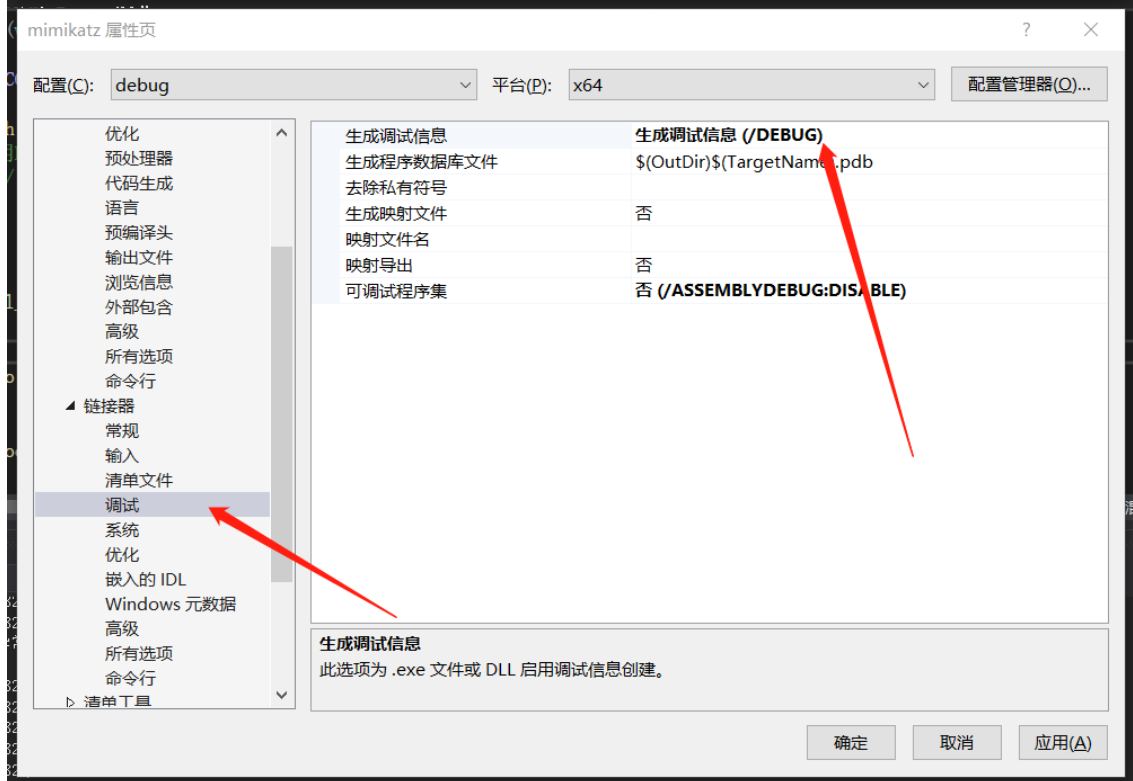
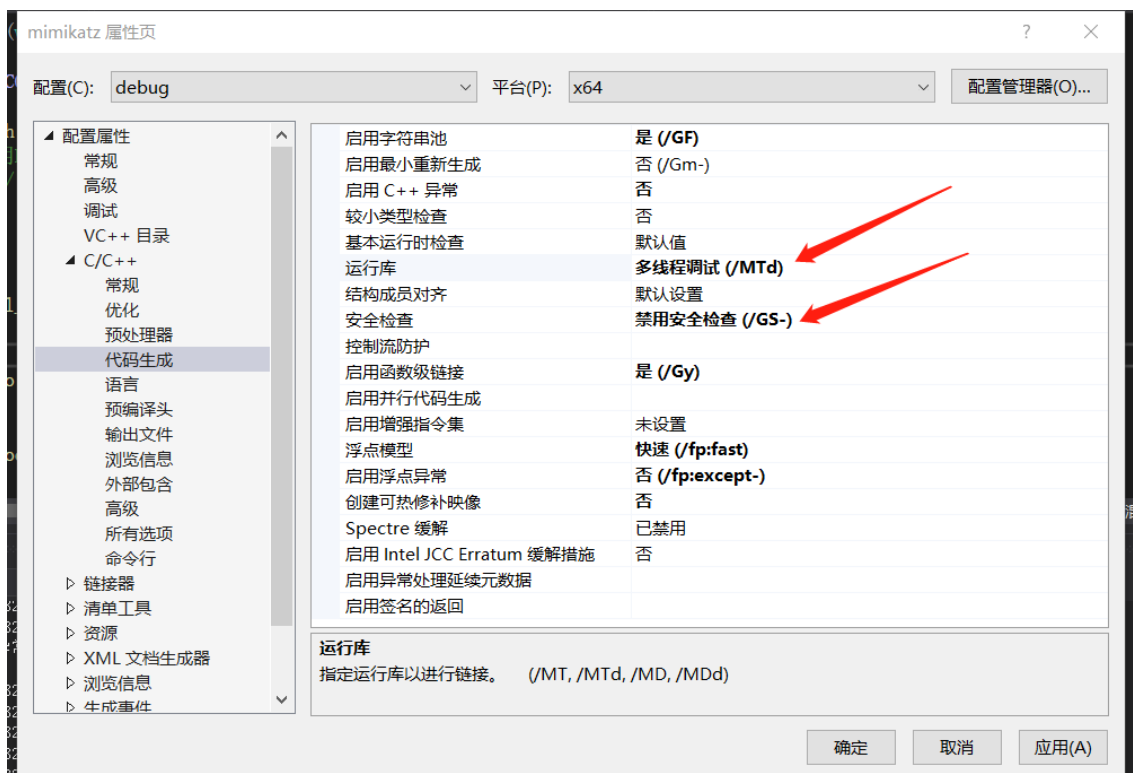
- mimikatz源码: [地址](#)
- 调试环境: vs2019
- 几点设置:

1. 因为官方项目没有debug方案, 所以需要手动添加debug配置



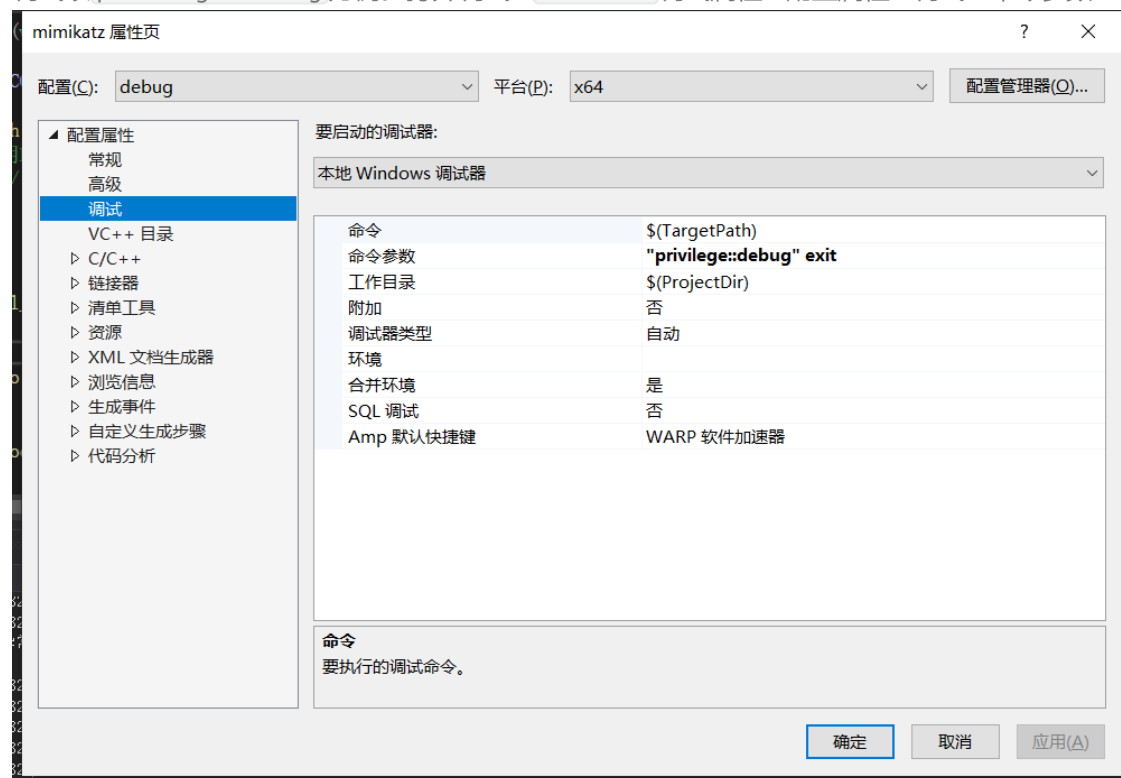
2. 项目属性配置





## 程序入口

调试以 `privilege::debug` 为例。打开调试->mimikatz 调试属性->配置属性->调试->命令参数



`wmain()` 是 mimikatz 的入口函数。

```
38 int wmain(int argc, wchar_t * argv[]) //入口文件
39 {
40     NTSTATUS status = STATUS_SUCCESS;
41     int i;
42     if !defined(_POWERKATZ)
43     {
44         size_t len;
45         wchar_t input[0xffff];
46     }
47     mimikatz_begin(); //调用开始函数
48     for(i = MIMIKATZ_AUTO_COMMAND_START ; i < argc) && (status != STATUS_PROCESS_IS_TERMINATING) && (status != STATUS_THREAD_IS_TERMINATING) ; i++)
49     {
50         kprintf(L"\n" MIMIKATZ L"(" MIMIKATZ_AUTO_COMMAND_STRING L") # %s\n", argv[i]);
51         status = mimikatz_dispatchCommand(argv[i]); //将输入参数传递进入命令分发请求
52     }
```

## 命令分发

从上面的循环中获取到请求参数之后就进入到命令分发的 `mimikatz_dispatchCommand()` 函数。

```
149 NTSTATUS mimikatz_dispatchCommand(wchar_t * input) //命令分发
150 {
151     NTSTATUS status = STATUS_UNSUCCESSFUL;
152     PWCHAR full;
153     if(full = kull_m_file_fullPath(input))
154     /*kull_m_file_fullPath函数调用ExpandEnvironmentStrings和localAlloc函数，具体功能未知
155     一般返回的就是输入的请求参数*/
156     {
157         switch(full[0])
158         {
159             case L'!':
160                 status = kuhl_m_kernel_do(full + 1);
161                 break;
162             case L'*':
163                 status = kuhl_m_rpc_do(full + 1);
164                 break;
165             default:
166                 status = mimikatz_doLocal(full); //参数提取
167         }
168         LocalFree(full);
169     }
170     return status;
171 }
```

这里首先有一个 `kull_m_file_fullPath` 方法，然后进行匹配，暂时不知道具体作用是什么，之后进入 `mimikatz_doLocal()` 方法。

```

173 NTSTATUS mimikatz_dolocal(wchar_t * input)
174 {
175     NTSTATUS status = STATUS_SUCCESS;
176     int argc;
177     wchar_t ** argv = CommandLineToArgvW(input, &argc), *module = NULL, *command = NULL, *match;
178     /**
179     LPWSTR * CommandLineToArgvW(
180     LPCWSTR lpCmdLine,
181     int *pNumArgs);
182     lpCmdLine:指向包含完整命令行的以空结尾的Unicode字符串的指针, 如果此参数为空字符串, 则函数返回当前可执行文件的路径。
183     pNumArgs:指向一个int的指针, 该int接收返回的数组元素数, 类似于argc。
184     */
185     unsigned short indexModule, indexCommand;
186     BOOL moduleFound = FALSE, commandFound = FALSE;
187
188     if(argv && (argc > 0))
189     {
190         if(match = wcsstr(argv[0], L".:")) //wcsstr指向找到的子字符串的第一个字符的指针dest, 或者NULL如果找不到这样的子字符串。如果src指向一个空字符串, 则返回dest。
191         {
192             if(module = (wchar_t *) LocalAlloc(LPTR, (match - argv[0] + 1) * sizeof(wchar_t))) //分配内存保存module
193             {
194                 if((unsigned int) (match + 2 - argv[0]) < wcslen(argv[0])) //去掉两个.:比较长度, wcslen返回字节长度。
195                     command = match + 2; //得到command=debug
196                 RtlCopyMemory(module, argv[0], (match - argv[0]) * sizeof(wchar_t)); //module 保存模块参数privilege
197             }
198         }
199         else command = argv[0];

```

## 命令执行

在对命令进行请求分发之后获取到 module 和 command 两个参数, 之后就进入了命令执行的阶段, 这个地方涉及到结构体的知识。

```

201     /**
202     * 重要部分, 此处开始执行命令。
203     * wcsicmp: 宽字符串比较函数
204     * 返回值 说明
205     < 0    string1 小于 string2
206     0      string1 等于 string2
207     > 0    string1 大于 string2
208     */
209     int size_ = ARRAYSIZE(mimikatz_modules);
210     for(indexModule = 0; !moduleFound && (indexModule < ARRAYSIZE(mimikatz_modules)); indexModule++)
211         if(moduleFound = (!module || (_wcsicmp(module, mimikatz_modules[indexModule]->shortName) == 0)))
212             if(command)
213                 for(indexCommand = 0; !commandFound && (indexCommand < mimikatz_modules[indexModule]->nbCommands); indexCommand++)
214                     if(commandFound = _wcsicmp(command, mimikatz_modules[indexModule]->commands[indexCommand].command) == 0)
215                         status = mimikatz_modules[indexModule]->commands[indexCommand].pCommand(argc - 1, argv + 1);
216

```

首先 mimikatz\_modules[] 是一个数组, 数组里面存放的是每一个模块的结构体的指针。那么第 210 行就是将 module 的值和每个模块结构体中定义的 shortName 进行比较, 如果相同, 返回 0。

```

8     const KUHL_M * mimikatz_modules[] = { //定义mimikatz拥有的模块
9         &kuhl_m_standard,
10        &kuhl_m_crypto,
11        &kuhl_m_sekurlsa,
12        &kuhl_m_kerberos,
13        &kuhl_m_ngc,
14        &kuhl_m_privilege,
15        &kuhl_m_process,
16        &kuhl_m_service,
17        &kuhl_m_lsadump,
18        &kuhl_m_ts,
19        &kuhl_m_event,
20        &kuhl_m_misc,
21        &kuhl_m_token,
22        &kuhl_m_vault,
23        &kuhl_m_minesweeper,

```

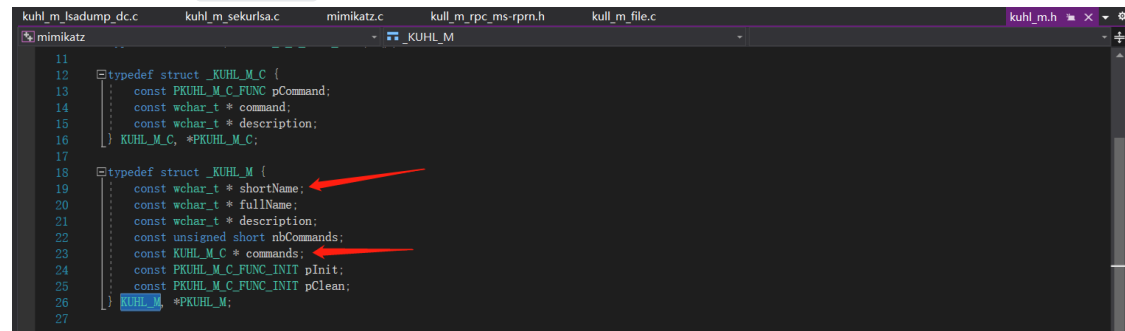
kuhl\_m\_lsadump.dc.c   kuhl\_m\_sekurlsa.c   mimikatz.c   kull\_m\_rpc\_ms-rprn.h   kull\_m\_file.c   kuhl\_m\_privilege.c

```

6     #include "kuhl_m_privilege.h"
7
8     const KUHL_M_C kuhl_m_c_privilege[] = {
9         {kuhl_m_privilege_debug, L"debug", L"Ask debug privilege"},
10        {kuhl_m_privilege_driver, L"driver", L"Ask load driver privilege"},
11        {kuhl_m_privilege_security, L"security", L"Ask security privilege"},
12        {kuhl_m_privilege_tcb, L"tcb", L"Ask tcb privilege"},
13        {kuhl_m_privilege_backup, L"backup", L"Ask backup privilege"},
14        {kuhl_m_privilege_restore, L"restore", L"Ask restore privilege"},
15        {kuhl_m_privilege_sysenv, L"sysenv", L"Ask system environment privilege"},
16
17        {kuhl_m_privilege_id, L"id", L"Ask a privilege by its id"},
18        {kuhl_m_privilege_name, L"name", L"Ask a privilege by its name"},
19    };
20
21     const KUHL_M kuhl_m_privilege = {
22         L"privilege", L"Privilege module", NULL,
23         ARRAYSIZE(kuhl_m_c_privilege), kuhl_m_c_privilege, NULL, NULL
24     };
25

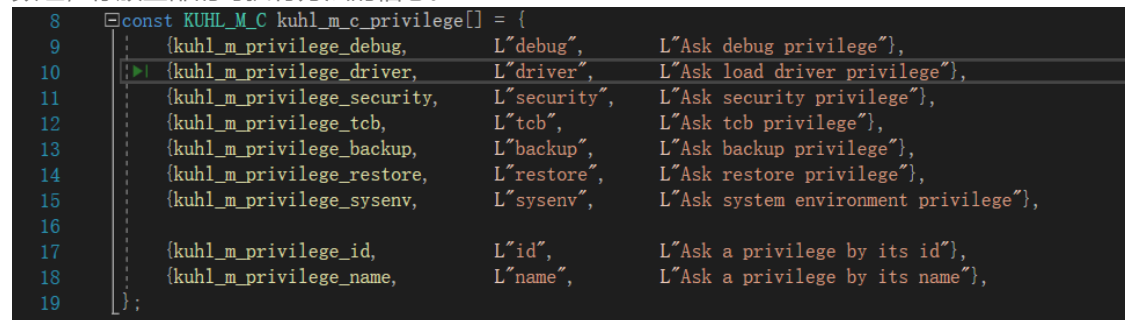
```

结构体的结构在 `kuhl_m.h` 这个头文件中进行定义。



```
11
12 typedef struct _KUHL_M_C {
13     const PKUHL_M_C_FUNC pCommand;
14     const wchar_t * command;
15     const wchar_t * description;
16 } KUHL_M_C, *PKUHL_M_C;
17
18 typedef struct _KUHL_M {
19     const wchar_t * shortName;
20     const wchar_t * fullName;
21     const wchar_t * description;
22     const unsigned short nbCommands;
23     const KUHL_M_C * commands;
24     const PKUHL_M_C_FUNC_INIT pInit;
25     const PKUHL_M_C_FUNC_INIT pClean;
26 } KUHL_M;
27
```

之后第213和214两行相同的方式去寻找同一个模块下存在的 `command`，每个模块都预先定义一个数组，存放全部的可执行方法的信息。



```
8  const KUHL_M_C kuhl_m_c_privilege[] = {
9      {kuhl_m_privilege_debug, L"debug", L"Ask debug privilege"},
10     {kuhl_m_privilege_driver, L"driver", L"Ask load driver privilege"},
11     {kuhl_m_privilege_security, L"security", L"Ask security privilege"},
12     {kuhl_m_privilege_tcb, L"tcb", L"Ask tcb privilege"},
13     {kuhl_m_privilege_backup, L"backup", L"Ask backup privilege"},
14     {kuhl_m_privilege_restore, L"restore", L"Ask restore privilege"},
15     {kuhl_m_privilege_sysenv, L"sysenv", L"Ask system environment privilege"},
16
17     {kuhl_m_privilege_id, L"id", L"Ask a privilege by its id"},
18     {kuhl_m_privilege_name, L"name", L"Ask a privilege by its name"},
19 };

```

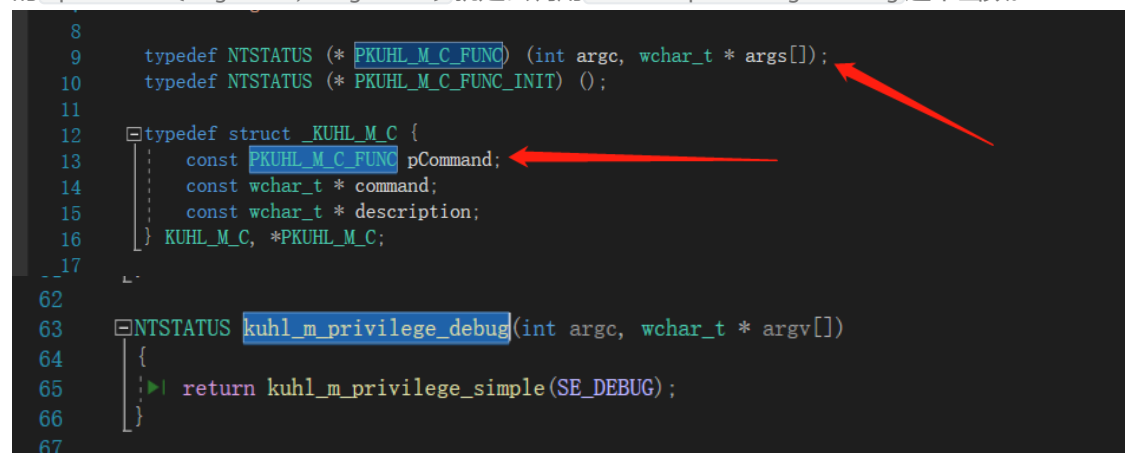
最重要的就是第215行，`status = mimikatz_modules[indexModule]-`

`>commands[indexCommand].pCommand(argc - 1, argv + 1);`，执行这个模块和命令。

`mimikatz_modules[indexModule]->commands[1]` 这一步相当于找到了

`kuhl_m_c_privilege[]` 这个数组的第一个元素，然后这个 `const KUHL_M_C`

`kuhl_m_c_privilege[]` 数组，是一个结构体数组，这个第一项表示的是一个 `指针函数`，那后面的 `.pCommand(argc - 1, argv + 1)` 就是去调用 `kuhl_m_privilege_debug` 这个函数。



```
8
9  typedef NTSTATUS (* PKUHL_M_C_FUNC) (int argc, wchar_t * args[]);
10 typedef NTSTATUS (* PKUHL_M_C_FUNC_INIT) ();
11
12 typedef struct _KUHL_M_C {
13     const PKUHL_M_C_FUNC pCommand;
14     const wchar_t * command;
15     const wchar_t * description;
16 } KUHL_M_C, *PKUHL_M_C;
17
62
63 NTSTATUS kuhl_m_privilege_debug(int argc, wchar_t * argv[])
64 {
65     return kuhl_m_privilege_simple(SE_DEBUG);
66 }
67
```

可以看到的是对于 `privilege::debug` 这个功能，执行的函数是

`kuhl_m_privilege_simple()`，而最后调用的系统API是 `RtlAdjustPrivilege()`。

```
NTSTATUS kuhl_m_privilege_simple(ULONG privId)
{
    ULONG previousState;
    NTSTATUS status = RtlAdjustPrivilege(privId, TRUE, FALSE, &previousState);
    if(NT_SUCCESS(status))
        kprintf(L"Privilege \'%u\' OK\n", privId);
    else PRINT_ERROR(L"RtlAdjustPrivilege (%u) %08x\n", privId, status);
    return status;
}
```

至此，整个简单的流程分析已经结束了，关于 `mimikatz` 的请求流程，和命令分发已经了解清楚了。

