WriteFile-HOOK1逆向分析

基础内容

- `VirtualProtect()`：一个关键的 Windows API 调用，用于改变调用进程虚拟地址空间中一页区域的保护属性。

```
BOOL VirtualProtect(
LPVOID lpAddress,          // [in]  要改变保护属性的内存区域的起始地址。
SIZE_T dwSize,             // [in]  内存区域的大小（字节）。
DWORD  flNewProtect,       // [in]  新的内存保护选项。
PDWORD lpflOldProtect      // [out] 指向一个变量的指针，该变量将接收之前的访问保护属性。
);
```

- `CreateFileW`：

```
HANDLE CreateFileW(
LPCWSTR                lpFileName,            // [in] 文件或设备的名称
DWORD                  dwDesiredAccess,       // [in] 请求的访问模式
DWORD                  dwShareMode,           // [in] 文件或设备的共享模式
LPSECURITY_ATTRIBUTES lpSecurityAttributes, // [in, optional] 安全属性
DWORD                  dwCreationDisposition,// [in] 创建或打开文件的方式
DWORD                  dwFlagsAndAttributes, // [in] 文件或设备的属性和标志
HANDLE                 hTemplateFile         // [in, optional] 模板文件的句柄
);
```

主流程

1. 先定义 `pJmpCode`，第一个元素为 `E9` 即该汇编指令码，后面四个字节定义0，用于存储跳转到 `MyWriteFile` 的地址，然后把 `ProcName("WriteFile")` 写入 `buf`，再分别调 `GetModuleHandleW` 和 `GetProcAddress` 获取到 `WriteFile` 的函数地址，存到 `pWriteFile` 和 `pEditFunc` 里，然后调 `VirtualProtect`，第一个参数为 `pEditFunc` 也就是 `WriteFile` 的首地址，第二个是5即要修改权限的内存区域大小（5个字节），第三个 `0x40` 代表 `windows.h` 中预定义的一个常量 `PAGEPAGE_EXECUTE_READWRITE`（可执行可读可写），用于指定

新的内存保护选项，第四个是原来的内存保护选项；

```
mov     [rbp+200h+pJmpCode], 0E9h
mov     [rbp+200h+pJmpCode+1], 0
lea     rax, [rbp+200h+pJmpCode+2]
mov     rdi, rax
xor     eax, eax
mov     ecx, 4
rep stosb
lea     rax, [rbp+200h+buf]
lea     rcx, ProcName   ; "WriteFile"
mov     rdi, rax
mov     rsi, rcx
mov     ecx, 0Ah
rep movsb
lea     rcx, aKernel32Dll_1 ; "kernel32.dll"
call    cs:__imp_GetModuleHandleW
mov     [rbp+200h+hKernel32], rax
lea     rdx, ProcName   ; "WriteFile"
mov     rcx, [rbp+200h+hKernel32] ; hModule
call    cs:__imp_GetProcAddress
mov     [rbp+200h+pWriteFile], rax
mov     rax, [rbp+200h+pWriteFile]
mov     [rbp+200h+pEditFunc], rax
lea     r9, [rbp+200h+dwOldProtect] ; lpflOldProtect
mov     r8d, 40h ; '@'  ; flNewProtect
mov     edx, 5          ; dwSize
mov     rcx, [rbp+200h+pEditFunc] ; lpAddress
call    cs:__imp_VirtualProtect
test    eax, eax
jz      loc_140011B92
```

2. 先把原函数的前5个字节保存到 `pOrgByte` 里（`memcpy`），然后算出 `MyWriteFile` 的首地址地址相对于原函数的首地址的偏移量（作差），再减掉5预留出 `JMP` 指令的位置，将这个偏移量写到 `pJmpCode` 的对应位置，然后将写好的跳转指令（5个字节）写到原函数的前5个字节，即修改了原函数的第一条指令为跳转到我们自

己写出的函数，然后再次确认这条指令的保护选项（可执行可读可写）；

```
mov     r8d, 5          ; Size
mov     rdx, [rbp+200h+pEditFunc] ; Src
lea     rcx, ?pOrgByte@@3PAEA ; void *
call    j_memcpy_0
nop
lea     rax, j_?MyWriteFile@@YAHPEAXPEBXKPEAKPEAU_OVERLAPPED@@@Z
sub     rax, [rbp+200h+pWriteFile]
sub     rax, 5
mov     [rbp+200h+pOffset], eax
mov     eax, 1
imul    rax, 1
lea     rax, [rbp+rax+200h+pJmpCode]
mov     r8d, 4          ; Size
lea     rdx, [rbp+200h+pOffset] ; Src
mov     rcx, rax        ; void *
call    j_memcpy_0
nop
mov     eax, 1
imul    rax, 0
lea     rax, [rbp+rax+200h+pJmpCode]
mov     r8d, 5          ; Size
mov     rdx, rax        ; Src
mov     rcx, [rbp+200h+pWriteFile] ; void *
call    j_memcpy_0
nop
lea     r9, [rbp+200h+dwOldProtect] ; lpflOldProtect
mov     r8d, [rbp+200h+dwOldProtect] ; flNewProtect
mov     edx, 5          ; dwSize
mov     rcx, [rbp+200h+pWriteFile] ; lpAddress
call    cs:__imp_VirtualProtect
nop
```

3. 先创建要操作的文件，第一个参数是文件名，然后是一个固定值，代表读写权限（GENERIC_WRITE | GENERIC_READ获得），然后两个0分别代表文件的独占访问和文件获得默认的安全描述符并且返回的句柄不能被子进程继承，之后的2代表 `CREATE_ALWAYS`，然后 `80h` 为 `FILE_ATTRIBUTE_NORMAL` 代表文件没有设置其他特殊属性，最后一个0代表在不使用模板的情况下创建，创建完成后句柄放到 `hFile` 里，最后调用

`WriteFile` 写文件。

```
loc_140011B92:          ; hTemplateFile
mov     [rsp+240h+hTemplateFile], 0
mov     [rsp+240h+dwFlagsAndAttributes], 80h ; '€' ; dwFlagsAndAttributes
mov     [rsp+240h+dwCreationDisposition], 2 ; dwCreationDisposition
xor     r9d, r9d        ; lpSecurityAttributes
xor     r8d, r8d        ; dwShareMode
mov     edx, 0C0000000h ; dwDesiredAccess
lea     rcx, FileName   ; "hook.txt"
call    cs:__imp_CreateFileW
mov     [rbp+200h+hFile], rax
lea     rcx, [rbp+200h+buf] ; Str
call    j_strlen_0
mov     qword ptr [rsp+240h+dwCreationDisposition], 0 ; lpOverlapped
lea     r9, [rbp+200h+dwWritenSize] ; lpNumberOfBytesWritten
mov     r8d, eax        ; nNumberOfBytesToWrite
lea     rdx, [rbp+200h+buf] ; lpBuffer
mov     rcx, [rbp+200h+hFile] ; hFile
call    cs:__imp_WriteFile
```

`MyWriteFile()`

```
lea     rax, [rbp+120h+buf]
lea     rcx, aTheMagicOfApiH ; "The magic of API Hook!"
mov     rdi, rax
mov     rsi, rcx
mov     ecx, 17h
rep movsb
call    j_?unhook@@YAXXZ ; unhook(void)
nop
lea     rcx, ModuleName ; "kernel32.dll"
call    cs:__imp_GetModuleHandleA
lea     rdx, ProcName   ; "WriteFile"
mov     rcx, rax        ; hModule
call    cs:__imp_GetProcAddress
mov     [rbp+120h+pFunc], rax
lea     rcx, [rbp+120h+buf] ; Str
call    j_strlen_0
mov     rcx, [rbp+120h+lpOverlapped]
mov     [rsp+150h+var_130], rcx
mov     r9, [rbp+120h+lpNumberOfBytesWritten]
mov     r8d, eax
lea     rdx, [rbp+120h+buf]
mov     rcx, [rbp+120h+hFile]
call    [rbp+120h+pFunc]
nop
```

目的字符串写到 `buf` 里，调 `unhook()`，然后获取 `WriteFile` 地址调用。

unhook()

```
lea     rcx, ModuleName ; "kernel32.dll"
call    cs:__imp_GetModuleHandleA
lea     rdx, ProcName   ; "WriteFile"
mov     rcx, rax        ; hModule
call    cs:__imp_GetProcAddress
mov     [rbp+130h+pFunc], rax
mov     rax, [rbp+130h+pFunc]
mov     [rbp+130h+pWriteFile], rax
lea     r9, [rbp+130h+dwOldProtect] ; lpflOldProtect
mov     r8d, 40h ; '@'  ; flNewProtect
mov     edx, 5          ; dwSize
mov     rcx, [rbp+130h+pWriteFile] ; lpAddress
call    cs:__imp_VirtualProtect
nop
mov     r8d, 5          ; Size
lea     rdx, ?pOrgByte@@3PAEA ; Src
mov     rcx, [rbp+130h+pWriteFile] ; void *
call    j_memcpy_0
nop
lea     r9, [rbp+130h+dwOldProtect] ; lpflOldProtect
mov     r8d, [rbp+130h+dwOldProtect] ; flNewProtect
mov     edx, 5          ; dwSize
mov     rcx, [rbp+130h+pWriteFile] ; lpAddress
call    cs:__imp_VirtualProtect
nop
```

获取 `WriteFile` 地址，把前5个字节修改为原来的内容，指定内存保护选项为可执行可读可写。目的为防止无限调用`MyWriteFile()`。