分析程序伪代码

IDA View-A | Stack of _main_0 | Pseudocode-A

```
1  int __cdecl main_0(int argc, const char **argv, const char **envp)
2  {
3    size_t nNumberOfBytesToWrite; // eax
4    DWORD Buffer[4]; // [esp+4Ch] [ebp-30h] BYREF
5    int Src; // [esp+5Ch] [ebp-20h] BYREF
6    DWORD flOldProtect; // [esp+60h] [ebp-1Ch] BYREF
7    _BYTE v8[6]; // [esp+64h] [ebp-18h] BYREF
8    LPVOID lpAddress; // [esp+6Ch] [ebp-10h]
9    BOOL (__stdcall *WriteFile_1)(HANDLE, LPCVOID, DWORD, LPDWORD, LPOVERLAPPED); // [esp+70h] [ebp-Ch]
10   HMODULE hModule; // [esp+74h] [ebp-8h]
11   HANDLE hFile; // [esp+78h] [ebp-4h]
12
13   v8[0] = -23;
14   v8[1] = 0;
15   *(_DWORD *)&v8[2] = 0;
16   strcpy((char *)Buffer, "WriteFile");
17   hModule = GetModuleHandleA("kernel32.dll");
18   WriteFile_1 = GetProcAddress(hModule, "WriteFile");
19   lpAddress = WriteFile_1;
20   if ( VirtualProtect(WriteFile_1, 5u, PAGE_EXECUTE_READWRITE, &flOldProtect) )
21   {
22     memcpy(&unk_427C50, lpAddress, 5u);
23     Src = (char *)sub_401005 - (char *)WriteFile_1 - 5;
24     memcpy(&v8[1], &Src, 4u);
25     memcpy(WriteFile_1, v8, 5u);
26     VirtualProtect(WriteFile_1, 5u, flOldProtect, &flOldProtect);
27   }
28   hFile = CreateFileA("hook.txt", GENERIC_WRITE | 0x80000000, 0, 0, CREATE_ALWAYS, FILE_READ_ATTRIBUTES, 0);
29   nNumberOfBytesToWrite = strlen((const char *)Buffer);
30   WriteFile(hFile, Buffer, nNumberOfBytesToWrite, &Buffer[3], 0);
31   return 0;
32 }
```

可知程序将在执行路径新建一个文件 `hook.txt`

内容为变量 `Buffer` 中的 `WriteFile`

然而实际运行时 `hook.txt` 中的内容却为 `The magic of API Hook!`

程序调用了两次 `VirtualProtect` 函数修改了 `kernel32.dll` 中的 `WriteFile` 函数的前 5 个字节

分析两次 `VirtualProtect` 函数中的代码

```
if ( VirtualProtect(WriteFile_1, 5u, PAGE_EXECUTE_READWRITE, &flOldProtect) )
{
  memcpy(&unk_427C50, lpAddress, 5u);
  Src = (char *)sub_401005 - (char *)WriteFile_1 - 5;
  memcpy(&v8[1], &Src, 4u);
  memcpy(WriteFile_1, v8, 5u);
  VirtualProtect(WriteFile_1, 5u, flOldProtect, &flOldProtect);
}
```

`memcpy(&unk_427C50, lpAddress, 5u)` 将 `WriteFile` 函数前 5 个字节保存至 `unk_427C50`

`Src = (char *)sub_401005 - (char *)WriteFile_1 - 5` 计算 hook 函数与被 hook 函数的相对偏移

`memcpy(&v8[1], &Src, 4u)` 将相对偏移地址写入 v8

`memcpy(WriteFile_1, v8, 5u)` 将 v8 写入 `WriteFile` 函数的前 5 个字节

可得 v8[0] = -23 为汇编码中的 E9(JMP)

分析函数 sub_401005

```
int __stdcall sub_401100(int a1, int a2, int a3, int a4, int a5)
{
  HMODULE ModuleHandleA; // eax
  size_t v6; // eax
  char The_magic_of_API_Hook_[24]; // [esp+4Ch] [ebp-1Ch] BYREF
  BOOL (__stdcall *WriteFile_1)(HANDLE, LPCVOID, DWORD, LPDWORD, LPOVERLAPPED); // [esp+64h] [ebp-4h]

  strcpy(The_magic_of_API_Hook_, "The magic of API Hook!");
  sub_40100A();
  ModuleHandleA = GetModuleHandleA("kernel32.dll");
  WriteFile_1 = GetProcAddress(ModuleHandleA, "WriteFile");
  v6 = strlen(The_magic_of_API_Hook_);
  ((void (__stdcall *)(int, char *, size_t, int, int))WriteFile_1)(a1, The_magic_of_API_Hook_, v6, a4, a5);
  return 1;
}
```

可知该函数调用了 WriteFile 函数

且该函数接收了 5 个参数，与 WriteFile 函数的参数数量相同

可知该函数接管了 WriteFile 函数的调用

分析函数 sub_40100A

```
1  BOOL sub_401030()
2  {
3    HMODULE ModuleHandleA; // eax
4    FARPROC lpAddress; // [esp+50h] [ebp-8h]
5    DWORD flOldProtect; // [esp+54h] [ebp-4h] BYREF
6
7    ModuleHandleA = GetModuleHandleA("kernel32.dll");
8    lpAddress = GetProcAddress(ModuleHandleA, "WriteFile");
9    VirtualProtect(lpAddress, 5u, PAGE_EXECUTE_READWRITE, &flOldProtect);
0    memcpy(lpAddress, &unk_427C50, 5u);
1    return VirtualProtect(lpAddress, 5u, flOldProtect, &flOldProtect);
2  }
```

该函数调用了两次 `VirtualProtect` 函数修改了 `kernel32.dll` 中的 `WriteFile` 函数的前 5 个字节

且发现变量 `unk_427C50` 被写回 `WriteFile` 函数的前 5 个字节

可知该函数功能为将 `WriteFile` 函数恢复正常