Reverse-Engineering of the SCP: Secret Laboratory anti-cheat

Author: colby57 | Member of Team Enterial

Summer hello! For 4 whole months I haven't released content related to sample analysis, and now it's time to fix it. People subscribed to me know the reason for the long absence, if anyone is interested, you can read here: https://t.me/colby5engineering/233

INTRODUCTION

I decided to try a new format for myself, all my articles before that were completely in Russian, because why not?

Based on the opinions of my colleagues, I was decided to analyze the anti-cheat from the game SCP: Secret Laboratory

This article will present an analysis of the mechanisms of anti-cheat, which are designed to prevent manipulation of the game's memory.

According to the tradition that I borrowed from Arting, at the beginning of the article I will briefly describe what utilities I used during the analysis of the anti-cheat:

My main tool for dynamic analysis is **x64dbg**

Occasionally, but still, if I do static analysis, I do it through **IDA Pro 7.7.220118**

Also a list of debugger plugins that I used:

- **SharpOD** — anti-anti-debug plugin, which, as for me, in some moments is more effective than ScyllaHide against protectors (VMProtect, Themida).
- **x64dbgpy** — for executing scripts written in Python using the debugger SDK.
- **Scylla (x64)** — utility for reconstructing imports.
- **Visual Studio 2019** — to write your own DLL that performs all the necessary hooks and patches.

We proceed directly to the analysis itself :)

LAUNCHER AND MODULE IN GENERAL TERMS

Both targets have a x64 architecture and are covered with a Themida protector, according to all the canons of modern protection, they have a full protection preset + developers during validation they will use a function from the Themida SDK to check the integrity of the module - **SECheckCodeIntegrity**.

The game is based on the Unity engine, but almost all the protection is in the SL-AC module, which is completely written in native C++ without any hints of CLR assembly.

ANTI-CHEAT MODULE

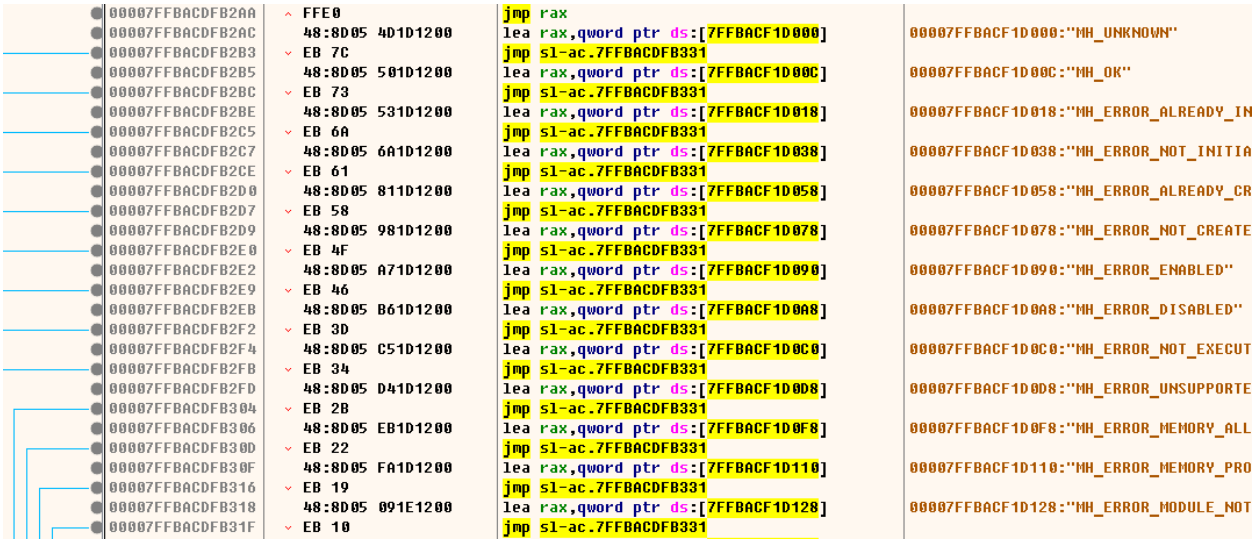
The debugger detection mechanism from Themida

Themida detected my debugger even after I intercepted the entire system call table, of course, nobody cancelled manual system calls under the virtual machine, but I was struck by the debugging detection method itself, I renamed my debugger and the protector stopped throwing up detection messages.

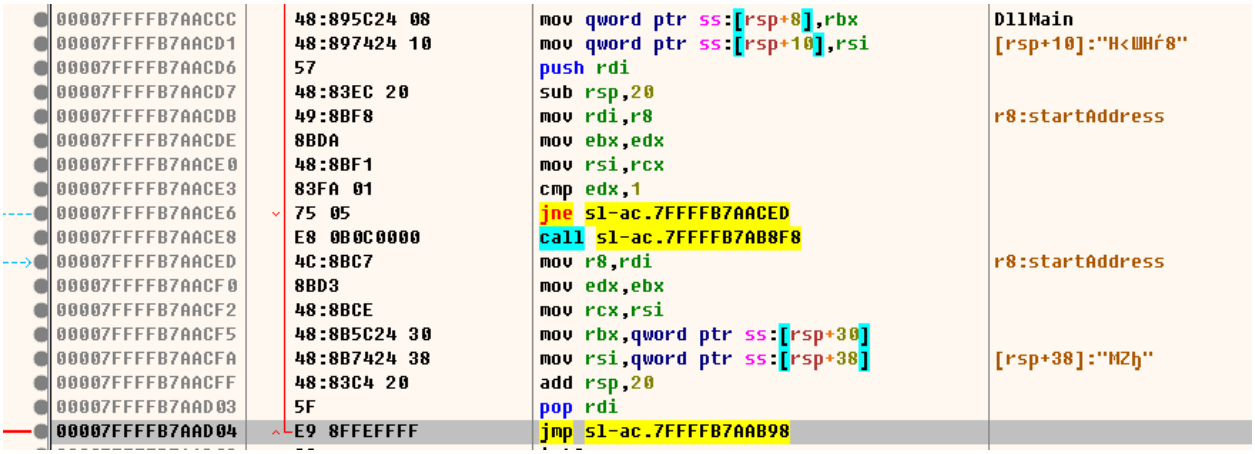
You can view the proof video here: https://youtu.be/JyV6_KNvJ70

INITIALIZATION

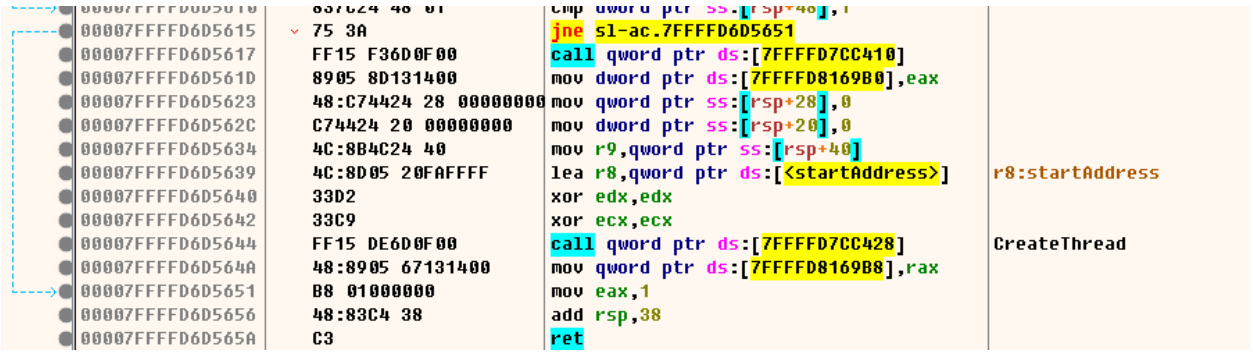
I noticed that the anticheat drags the MinHook library behind it, I realized this by the Enum the library leaves behind.



The **DllMain** function is not virtualized by Themida, this gives us the opportunity to find it by pattern and put a break on it.



During its initialization, the anti-cheat calls the virtualized CreateThread, this thread is considered the main one, and the first thing it will do is install its hooks.



Consider the list of hooks placed by anti-cheat (and a little bit by Themida)

The list of hooks:

- Imported library: GameAssembly.dll, exported function: il2cpp_resolve_icall
- Imported library: kernel32.dll, exported function: GetModuleHandleW
- Imported library: kernel32.dll, exported function: GetProcAddress
- Imported library: kernel32.dll, exported function: LoadLibraryA
- Imported library: kernel32.dll, exported function: LoadLibraryW
- Imported library: user32.dll, exported function: GetAsyncKetState
- Imported library: ntdll.dll, exported function: DbgUiRemoteBreakin
- (Supplied by Themida)
- Imported library: ntdll.dll, exported function: NtOpenFile

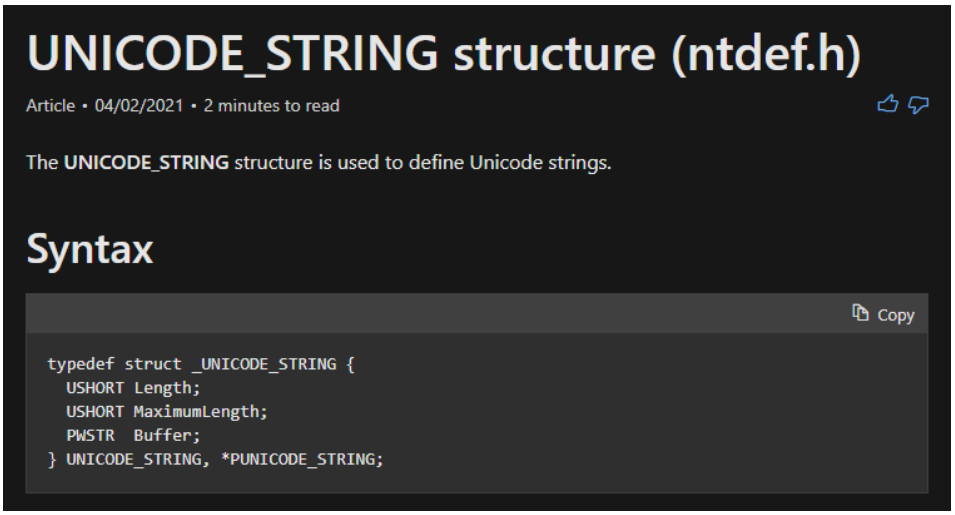
Blocking injection (NtOpenFile hook)

It is installed immediately after installing the hook on GetProcAddress.

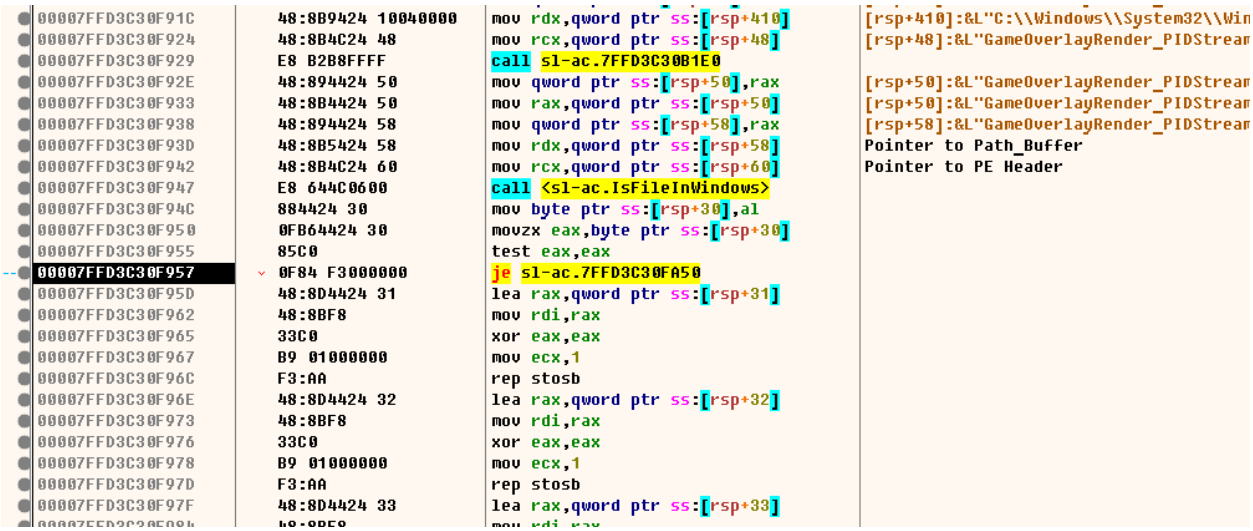
The decompiled **hkNtOpenFile** traces calls to the **std::wstring** constructor, which may indicate that the constructor will most likely be used to work with the **POBJECT_ATTRIBUTES** structure in the future.

To get the name of an object, the hook reads two pointers to structures at once: **POBJECT_ATTRIBUTES** and **PUNICODE_STRING**

The reading is done this way: **pObjectAttributes->pObjectName->Buffer**.

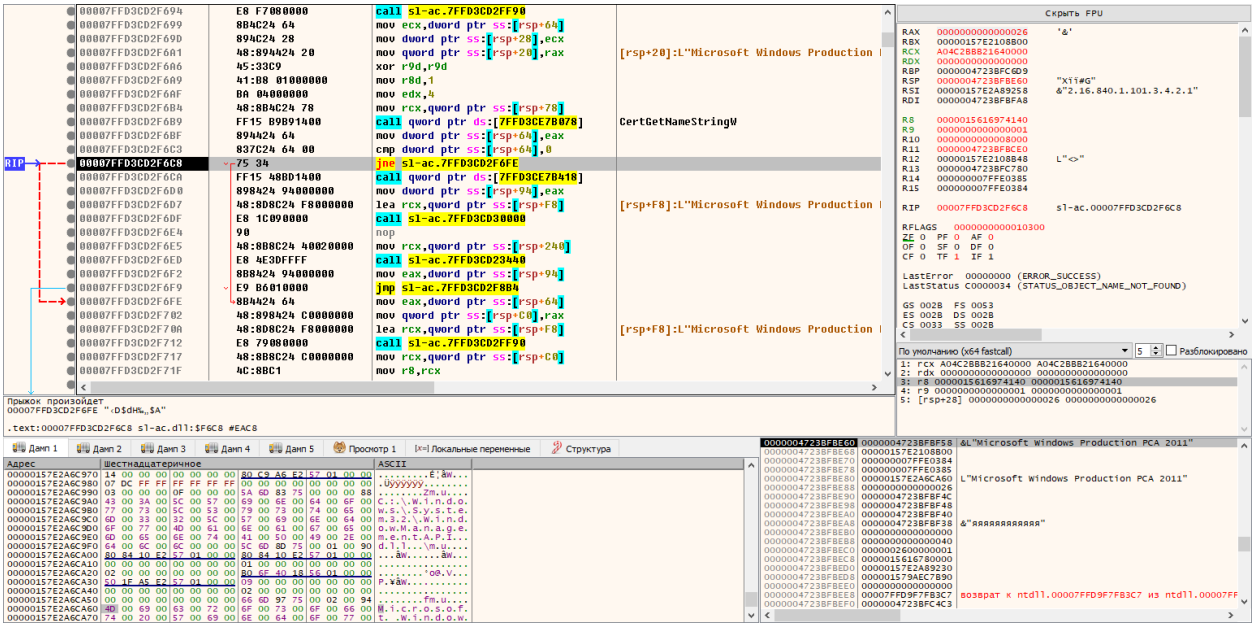


In the future, the anti-cheat will check whether the binary is located in the Windows folder and has various checks associated with the certificate.



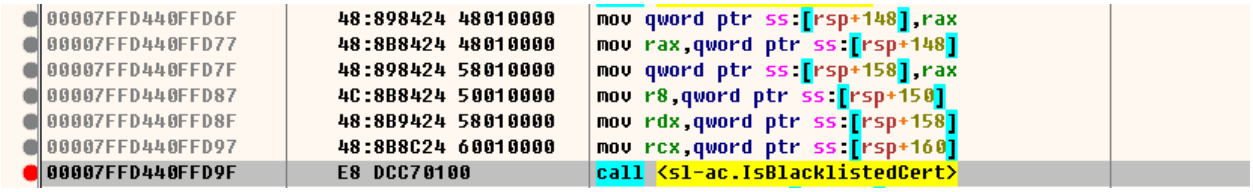
If the file was not in Windows, then in conjunction with the calls CryptQueryObject, CertGetNameStringW, CertFindCertificateInStore, it begins to check the publisher of the digital signature and get the name of the issuer.

```
00007FFD3C30F5A2   48:8D8424 30010000   lea rax,qword ptr ss:[rsp+130]
00007FFD3C30F5AA   48:894424 20          mov qword ptr ss:[rsp+20],rax          [rsp+20]:&L"Microsoft Windows"
00007FFD3C30F5AF   41:B9 00000B00        mov r9d,B0000
00007FFD3C30F5B5   45:33C0               xor r8d,r8d
00007FFD3C30F5B8   BA 01000100           mov edx,10001
00007FFD3C30F5BD   48:8B8C24 E0000000    mov rcx,qword ptr ss:[rsp+E0]
00007FFD3C30F5C5   FF15 0DBB1400         call qword ptr ds:[7FFD3C45B0D8]       CertFindCertificateInStore
00007FFD3C30F5CB   48:894424 78          mov qword ptr ss:[rsp+78],rax
00007FFD3C30F5D0   48:837C24 78 00       cmp qword ptr ss:[rsp+78],0
00007FFD3C30F5D6   75 26                 jne sl-ac.7FFD3C30F5FE
00007FFD3C30F5D8   FF15 3ABE1400         call qword ptr ds:[7FFD3C45B418]
00007FFD3C30F5DE   898424 8C000000       mov dword ptr ss:[rsp+8C],eax
00007FFD3C30F5E5   48:8B8C24 40020000    mov rcx,qword ptr ss:[rsp+240]
00007FFD3C30F5ED   E8 4E3EFFFF           call sl-ac.7FFD3C303440
00007FFD3C30F5F2   8B8424 8C000000       mov eax,dword ptr ss:[rsp+8C]
00007FFD3C30F5F9   E9 B6020000           jmp sl-ac.7FFD3C30F8B4
00007FFD3C30F5FE   C74424 28 00000000    mov dword ptr ss:[rsp+28],0
00007FFD3C30F606   48:C74424 20 00000000 mov qword ptr ss:[rsp+20],0            [rsp+20]:&L"Microsoft Windows"
00007FFD3C30F60F   45:33C9               xor r9d,r9d
00007FFD3C30F612   41:B8 01000000        mov r8d,1
00007FFD3C30F618   BA 04000000           mov edx,4
00007FFD3C30F61D   48:8B4C24 78          mov rcx,qword ptr ss:[rsp+78]
00007FFD3C30F622   FF15 50BA1400         call qword ptr ds:[7FFD3C45B078]       CertGetNameStringW
00007FFD3C30F628   894424 64             mov dword ptr ss:[rsp+64],eax
00007FFD3C30F62C   837C24 64 00          cmp dword ptr ss:[rsp+64],0
00007FFD3C30F631   75 26                 jne sl-ac.7FFD3C30F659
00007FFD3C30F633   FF15 DFBD1400         call qword ptr ds:[7FFD3C45B418]
```



Going further, I got to a virtualized function, which I called IsBlacklistedCert. The function inside the ThemidaVM compares the received certificate with the prohibited certificates, which is also stored under virtualization.

Certificates such as: ReShade, MIDKNIGHT LLC and Cheat Engine were checked, if he found any of them in the file, the function returned TRUE.



At the end of the entire check, he makes a log for himself: **[Windows Folder - %s]; [Signing: %s - %s]**

BYPASS BLOCKING INJECTION

There are many ways to deceive the anti-cheat and let your dll be injected, starting from the injection before the anti-cheat installs its hooks and ending with the fact that you can block the anti-cheat hook in a brazen way. My method will be related to the interception of the function for the placement of anti-cheat hooks.

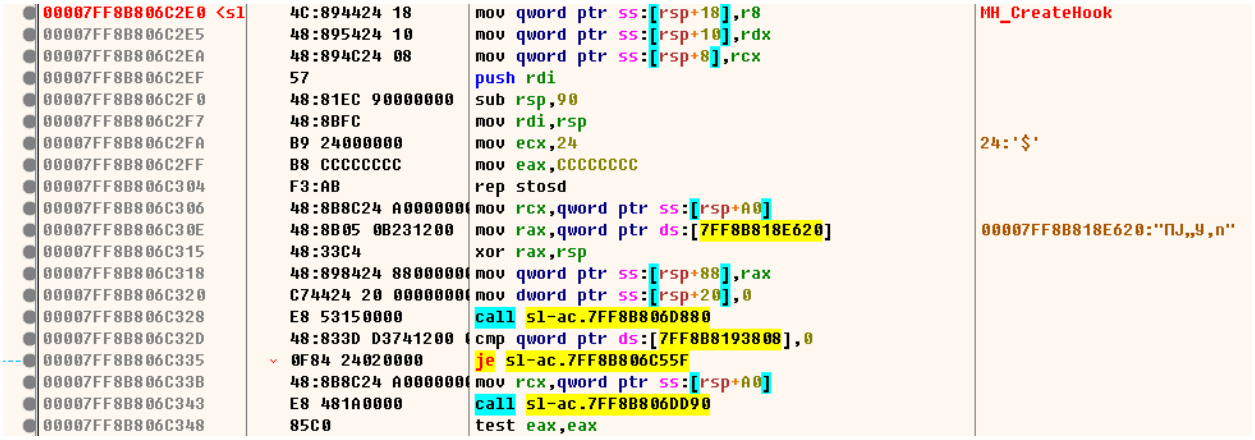
Since the lib function with hooks is not under the virtualization, nothing prevents us from finding MH_EnableHook and MH_CreateHook by patterns.

To find these functions, I have compiled two patterns:

4C 24 08 57 48 81 EC 90 00 00 00 48 8B FC B9 24 00 00 00 B8 CC CC CC CC F3 AB 48 8B 8C 24 A0 00 00 00 - **MH_CreateHook**

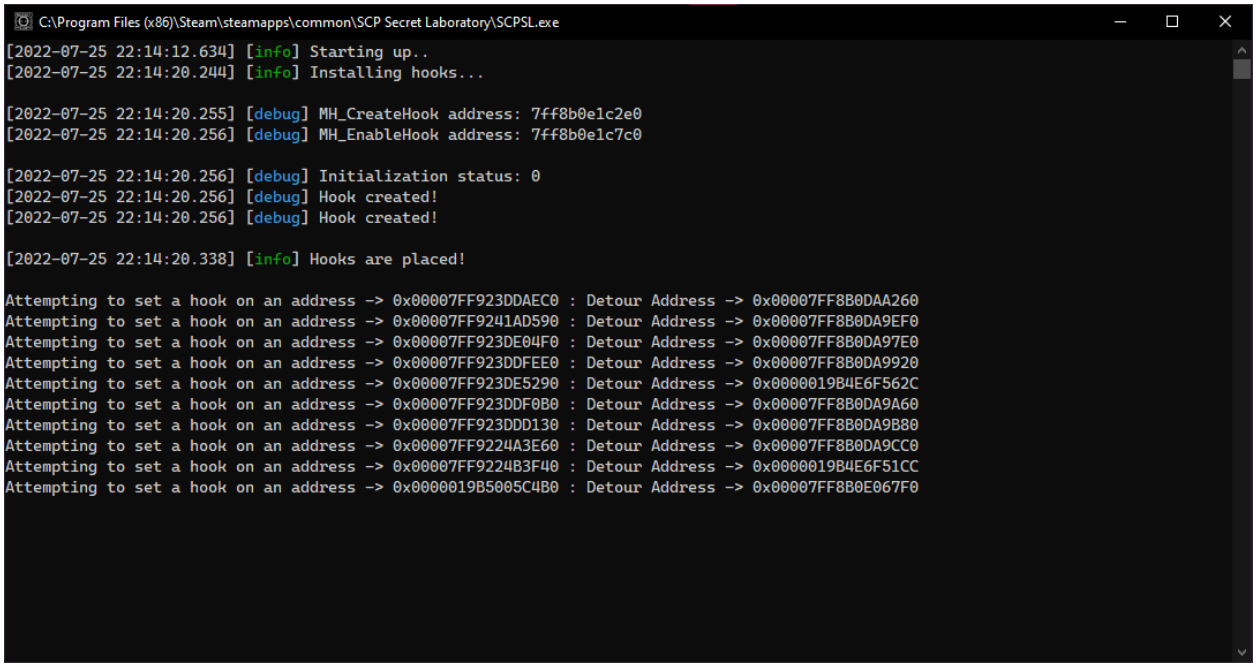
48 89 4C 24 08 57 48 83 EC 20 48 8B FC B9 08 00 00 - **MH_EnableHook**

Our interception functions have been found



```
00007FF8B806C7C0 <sl    48:894C24 08        mov qword ptr ss:[rsp+8],rcx        MH_EnableHook
00007FF8B806C7C5        57                  push rdi
00007FF8B806C7C6        48:83EC 20          sub rsp,20
00007FF8B806C7CA        48:8BFC             mov rdi,rsp
00007FF8B806C7CD        B9 08000000         mov ecx,8
00007FF8B806C7D2        B8 CCCCCCCC         mov eax,CCCCCCCC
00007FF8B806C7D7        F3:AB               rep stosd
00007FF8B806C7D9        48:8B4C24 30        mov rcx,qword ptr ss:[rsp+30]
00007FF8B806C7DE        BA 01000000         mov edx,1
00007FF8B806C7E3        48:8B4C24 30        mov rcx,qword ptr ss:[rsp+30]
00007FF8B806C7E8        E8 33110000         call sl-ac.7FF8B806D920
00007FF8B806C7ED        48:83C4 20          add rsp,20
00007FF8B806C7F1        5F                  pop rdi
00007FF8B806C7F2        C3                  ret
```

In addition to the fact that by intercepting **MH_EnableHook**, I will block the activation of **hkNtOpenFile**, returning **MH_OK** in parallel, I monitored which hooks he still placed, and left the addresses of anti-cheat detours in my console.



Of course, these checks for the authenticity of the file are not effective, you can skip these checks with one patch of the *JNE* instruction, or put a breakpoint on CloseHandle, because it is he who is responsible for locking the handle. No NtClose syscall, just an ordinary CloseHandle.

And here is the meme bypass of the injection lock: https://youtu.be/pxym5ha0Mkk

Recently, the developers updated the anti-cheat, and all my comments and tags on the function has been broken, and they virtualized a couple of functions and some more user code, CloseHandle was among them.

GetAsyncKeyState Hook

This hook transmits information about the return address to the anti-cheat. And also in the hook, **RtlPcToFileHeader** is called to read the PE header from the file from which this function was called, if the file did not have the first two bytes - ***0x5A4D***, then for the anti-cheat this is the first notice to report the user.

It all looks something like this:

```
PVOID imageBase;

RtlPcToFileHeader( _ReturnAddress( ), &imageBase );

if ( *( WORD * ) imageBase != 0x5A4D )
    // Hidden module
else
    // OK
```

Window/process detection is our everything!

Half of 2022 has already passed in the yard, and unfortunately if you were waiting for ingenious debugger detection mechanisms here, then forget about it, because both Themida and the developers of Northwood Studios decided to use the same detection techniques. To detect unwanted processes such as: **OllyDbg**, **Cheat Engine**, **Process Hacker**, **Process Monitor**, Themida uses **FindWindowA**, and anti-cheat, in turn, uses two mechanisms: **Window Detection and Process detection**.

Virtual Machine detection

Again, if you thought that the anti-cheat would use time-based sandbox/vm traversal methods, then you were very mistaken. In conjunction with the detection of unwanted processes, the second mechanism uses **VMware system processes in its list**. All this is implemented in a **Boolean function**, if the function returned **-1**, then it detected something from the VMware category and knocked out an error. This is patched with two instructions *"xor eax, eax; ret;"*

```
00007FF72965DAD0 <   33C0                 xor eax,eax                                  IsVirtualMachine
00007FF72965DAD2     C3                   ret
00007FF72965DAD3     90                   nop
00007FF72965DAD4     90                   nop
00007FF72965DAD5     48:894C24 08         mov qword ptr ss:[rsp+8],rcx
00007FF72965DADA     48:81EC 88020000     sub rsp,288
00007FF72965DAE1     48:8B05 B0280B00     mov rax,qword ptr ds:[7FF729710398]
00007FF72965DAE8     48:33C4              xor rax,rsp
00007FF72965DAEB     48:898424 70020000   mov qword ptr ss:[rsp+270],rax
00007FF72965DAF3     33D2                 xor edx,edx
00007FF72965DAF5     B9 02000000          mov ecx,2
00007FF72965DAFA     FF15 E8560800        call qword ptr ds:[7FF7296E31E8]             <= CreateToolhelp32Snapshot
00007FF72965DB00     48:894424 28         mov qword ptr ss:[rsp+28],rax
00007FF72965DB05     C74424 20 00000000   mov dword ptr ss:[rsp+20],0
00007FF72965DB0D     48:837C24 28 FF      cmp qword ptr ss:[rsp+28],FFFFFFFFFFFFFF
00007FF72965DB13     74 3D                je scpsl.7FF72965DB52
00007FF72965DB15     C74424 30 38020000   mov dword ptr ss:[rsp+30],238
00007FF72965DB1D     48:8D5424 30         lea rdx,qword ptr ss:[rsp+30]
00007FF72965DB22     48:8B4C24 28         mov rcx,qword ptr ss:[rsp+28]
00007FF72965DB27     FF15 C3560800        call qword ptr ds:[7FF7296E31F0]             <= ProcessNext32
00007FF72965DB2D     83F8 01              cmp eax,1
00007FF72965DB30     75 20                jne scpsl.7FF72965DB52
00007FF72965DB32     48:8D5424 5C         lea rdx,qword ptr ss:[rsp+5C]
00007FF72965DB37     48:8B8C24 98020000   mov rcx,qword ptr ss:[rsp+298]
00007FF72965DB3F     E8 CC210000          call <scpsl.compareString>
00007FF72965DB44     85C0                 test eax,eax
00007FF72965DB46     75 08                jne scpsl.7FF72965DB50
00007FF72965DB48     8B4424 38            mov eax,dword ptr ss:[rsp+38]
00007FF72965DB4C     894424 20            mov dword ptr ss:[rsp+20],eax
00007FF72965DB50     EB CB                jmp scpsl.7FF72965DB1D
00007FF72965DB52     8B4424 20            mov eax,dword ptr ss:[rsp+20]
00007FF72965DB56     894424 24            mov dword ptr ss:[rsp+24],eax
```

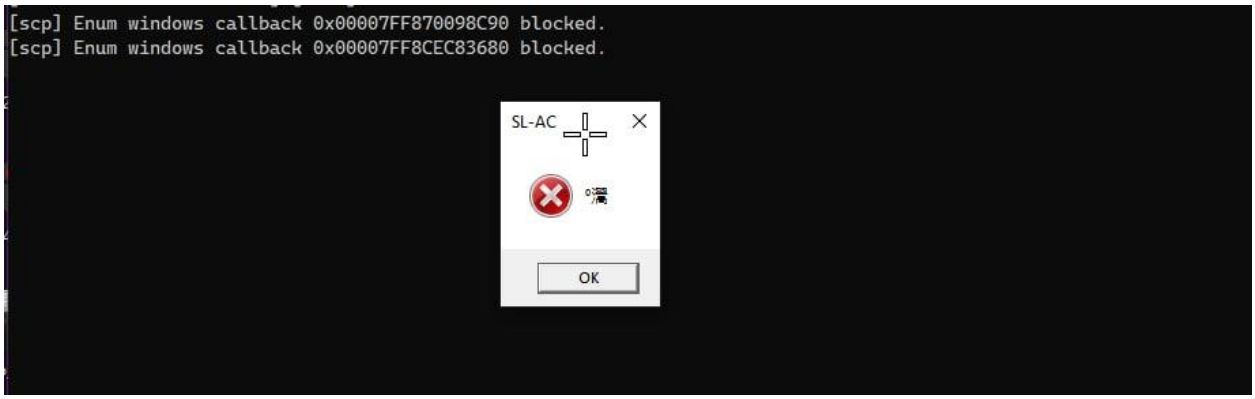
Well, the meme itself

Second detection mechanism

The second method consists in abusing the EnumWindows function, to be more precise, its callback, which acts as the first parameter. (https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-enumwindows)

This function will be called frequently, so we intercept it and return FALSE, because the anti-cheat does not check the return value of the function.

We receive the latest Chinese warning from the anti-cheat and skip it with a calm soul, because the game will continue loading into the main menu.



It is reckless to use this technique in your detection mechanisms only if reverse engineers who have no experience in dynamic analysis do not encroach on your product.

Heaven's Gate technique

Yes, yes, an anti-cheat under x64, but nevertheless use this technique, and apparently to prevent tracing, because as we all know xdbg does not know how to pass through the ret far instruction, like the same Cheat Engine or WinDbg.

These functions check for the presence of debug flags and debug port for the current process

```
00007FF907136CB1  | 6A 33                 | PUSH 33                                  | 0x33 - Access to the x64 code segment, 0x23 - к x32
00007FF907136CB3  | E8 00000000           | CALL sl-ac.7FF907136CB8                  | call $0
00007FF907136CB8  | 830424 05             | ADD DWORD PTR SS:[RSP],5                 |
00007FF907136CBC  | CB                    | RET FAR                                  |
00007FF907136CBD  | 49:89D8               | MOV R8,RBX                               |
00007FF907136CC0  | 48:89E3               | MOV RBX,RSP                              |
00007FF907136CC3  | FF3424                | PUSH QWORD PTR SS:[RSP]                  |
00007FF907136CC6  | FF3424                | PUSH QWORD PTR SS:[RSP]                  |
00007FF907136CC9  | 40:80E4 F0            | AND SPL,F0                               | Stack alignment 16 bytes
00007FF907136CCD  | 48:83C4 10            | ADD RSP,10                               |
00007FF907136CD1  | 48:83EC 30            | SUB RSP,30                               |
00007FF907136CD5  | 48:C7C1 FFFFFFFF      | MOV RCX,FFFFFFFFFFFFFFFF                 | Current process
00007FF907136CDC  | 48:C7C2 1E000000      | MOV RDX,1E                               | Process Debug Port
00007FF907136CE3  | 49:C7C1 08000000      | MOV R9,8                                 |
00007FF907136CEA  | 48:C74424 20 00000000 | MOV QWORD PTR SS:[RSP+20],0              |
00007FF907136CF3  | FFD6                  | CALL RSI                                 | Calling the system call
00007FF907136CF5  | 48:83C4 30            | ADD RSP,30                               |
00007FF907136CF9  | 48:89DC               | MOV RSP,RBX                              |
00007FF907136CFC  | E8 00000000           | CALL sl-ac.7FF907136D01                  | call $0
00007FF907136D01  | C74424 04 23000000    | MOV DWORD PTR SS:[RSP+4],23              |
00007FF907136D09  | 830424 0D             | ADD DWORD PTR SS:[RSP],D                 |
00007FF907136D0D  | CB                    | RET FAR                                  |
```

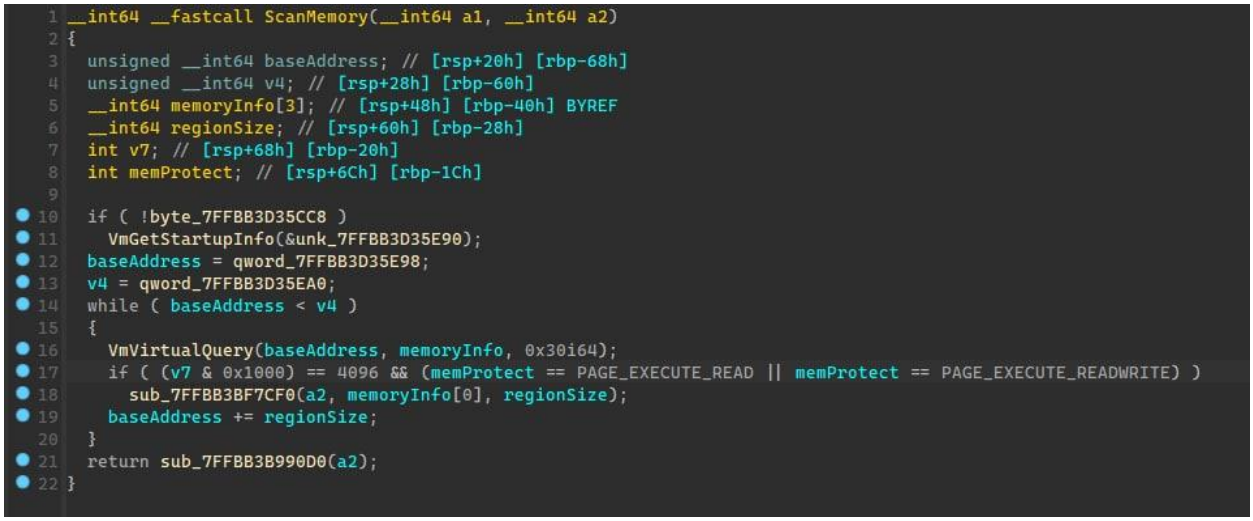
```
00007FF90721501C  | E8 00000000           | CALL sl-ac.7FF907215021                  | call $0
00007FF907215021  | 830424 05             | ADD DWORD PTR SS:[RSP],5                 |
00007FF907215025  | CB                    | RET FAR                                  |
00007FF907215026  | 49:89D8               | MOV R8,RBX                               |
00007FF907215029  | 48:89E3               | MOV RBX,RSP                              |
00007FF90721502C  | FF3424                | PUSH QWORD PTR SS:[RSP]                  |
00007FF90721502F  | FF3424                | PUSH QWORD PTR SS:[RSP]                  |
00007FF907215032  | 40:80E4 F0            | AND SPL,F0                               | Stack alignment 16 bytes
00007FF907215036  | 48:83C4 10            | ADD RSP,10                               |
00007FF90721503A  | 48:83EC 30            | SUB RSP,30                               |
00007FF90721503E  | 48:C7C1 FFFFFFFF      | MOV RCX,FFFFFFFFFFFFFFFF                 |
00007FF907215045  | 48:C7C2 07000000      | MOV RDX,7                                | Process Debug Flags
00007FF90721504C  | 49:C7C1 08000000      | MOV R9,8                                 |
00007FF907215053  | 48:C74424 20 00000000 | MOV QWORD PTR SS:[RSP+20],0              |
00007FF90721505C  | FFD6                  | CALL RSI                                 |
00007FF90721505E  | 48:83C4 30            | ADD RSP,30                               |
00007FF907215062  | 48:89DC               | MOV RSP,RBX                              |
00007FF907215065  | E8 00000000           | CALL sl-ac.7FF90721506A                  | call $0
00007FF90721506A  | C74424 04 23000000    | MOV DWORD PTR SS:[RSP+4],23              |
00007FF907215072  | 830424 0D             | ADD DWORD PTR SS:[RSP],D                 |
00007FF907215076  | CB                    | RET FAR                                  |
```

What also struck me was their attempt to wrap this technique under the Themida virtual machine, because this function was in the Themida segment where the VM is running. Apparently, Northwood Studios is not aware that the protectors cannot throw the Heaven's Gate technique just because of the **RET FAR** instructions. Therefore, such clean code is lying in a segment among virtualized code.

Memory scanning

At this stage of protection, the anti-cheat using GetStartUpInfo and VirtualQuery is passed through memory regions.

In decompiled form:



```
__int64 __fastcall ScanMemory(__int64 a1, __int64 a2)
{
  unsigned __int64 baseAddress; // [rsp+20h] [rbp-68h]
  unsigned __int64 v4; // [rsp+28h] [rbp-60h]
  __int64 memoryInfo[3]; // [rsp+48h] [rbp-40h] BYREF
  __int64 regionSize; // [rsp+60h] [rbp-28h]
  int v7; // [rsp+68h] [rbp-20h]
  int memProtect; // [rsp+6Ch] [rbp-1Ch]

  if ( !byte_7FFBB3D35CC8 )
    VmGetStartupInfo(&unk_7FFBB3D35E90);
  baseAddress = qword_7FFBB3D35E98;
  v4 = qword_7FFBB3D35EA0;
  while ( baseAddress < v4 )
  {
    VmVirtualQuery(baseAddress, memoryInfo, 0x30i64);
    if ( (v7 & 0x1000) == 4096 && (memProtect == PAGE_EXECUTE_READ || memProtect == PAGE_EXECUTE_READWRITE) )
      sub_7FFBB3BF7CF0(a2, memoryInfo[0], regionSize);
    baseAddress += regionSize;
  }
  return sub_7FFBB3B990D0(a2);
}
```

Scanning cheat patterns in memory

In addition to the fact that the anti-cheat scans the entire memory, it also manages to check the list of patterns that it collected from other cheats in the received memory regions.

- *"\x41\x69\x6D\x62\x6F\x74\x20\x54\x61\x72\x67\x65\x74"* - **Aimbot Target**
- *"\x53\x63\x69\x65\x6E\x74\x69\x73\x74\x20\x4B\x65\x79\x63\x61\x72\x64"* - **Scientist Keycard**
- *"\x70\x65\x64\x6F\x70\x68\x69\x6C\x65\x67\x61\x6D\x69\x6E\x67\x2E\x63\x63"* - **pedophilegaming.cc**
- *"\x6B\x69\x74\x65\x68\x34"* - **kiteh4x**
- *"\x3C\x63\x6F\x6C\x6F\x72\x3D\x77\x68\x69\x74\x65\x3E\x70\x6C\x61\x79\x65\x72\x73\x3A\x20\x25\x69\x3C\x2F\x63\x6F\x6C\x6F\x72\x3E"* - **<color=white>players: %i</color>**
- *"\x3C\x63\x6F\x6C\x6F\x72\x3D\x25\x73\x3E\x25\x73\x20\x2D\x20\x25\x2E\x32\x66\x3C\x2F\x63\x6F\x6C\x6F\x72\x3E"*- **<color=%s>%s - %.2f</color>**
- *"\x23\x23\x4D\x61\x69\x6E\x4D\x65\x6E\x75\x42\x61\x72"* - **##MainMenuBar**

Communication with the protected thread

In order to maintain the protected thread, the anti-cheat creates 1 thread. Anti-cheat takes the ID of the thread and will check it in every possible way after a while using Sleep.

Checking whether the protected thread is working:

```
| E8 72BD0000             | CALL sl-ac.7FFE325A71A0                  | Check if the thread dead
| 0FB6C0                  | MOVZX EAX,AL                             |
| 85C0                    | TEST EAX,EAX                             |
| 75 56                   | JNE sl-ac.7FFE3259B48B                   |
| 48:8D4424 20            | LEA RAX,QWORD PTR SS:[RSP+20]            |
| 48:8BF8                 | MOV RDI,RAX                              |
| 33C0                    | XOR EAX,EAX                              |
| B9 01000000             | MOV ECX,1                                |
| F3:AA                   | REP STOSB                                |
| 48:8D4424 21            | LEA RAX,QWORD PTR SS:[RSP+21]            |
| 48:8BF8                 | MOV RDI,RAX                              |
| 33C0                    | XOR EAX,EAX                              |
| B9 01000000             | MOV ECX,1                                |
| F3:AA                   | REP STOSB                                |
| 48:8D4424 22            | LEA RAX,QWORD PTR SS:[RSP+22]            |
| 48:8BF8                 | MOV RDI,RAX                              |
| 33C0                    | XOR EAX,EAX                              |
| B9 01000000             | MOV ECX,1                                |
| F3:AA                   | REP STOSB                                |
| 44:0FB64C24 20          | MOVZX R9D,BYTE PTR SS:[RSP+20]           |
| 44:0FB64424 21          | MOVZX R8D,BYTE PTR SS:[RSP+21]           |
| 0FB65424 22             | MOVZX EDX,BYTE PTR SS:[RSP+22]           |
| 48:8D4C24 60            | LEA RCX,QWORD PTR SS:[RSP+60]            |
| E8 4D000000             | CALL sl-ac.7FFE3259B4D0                  |
| 48:8BC8                 | MOV RCX,RAX                              |
| E8 55040000             | CALL sl-ac.7FFE3259B8E0                  | Log "A SL-AC thread has stopped execution inadvertently!"
| B9 0F000000             | MOV ECX,F                                |
| FF15 8A1D0F00           | CALL QWORD PTR DS:[<&Sleep>]             |
| E9 3EFFFFFF             | JMP sl-ac.7FFE3259B3D9                   | Returning to JE check
```

- What will happen if the thread is frozen? **Nothing**.

- What will happen if the thread is closed? **Anti-cheat will start spamming logs that the thread is dead.**

Protected thread

There is nothing unusual in the protected thread, GetTickCount64 is called first, then two functions are immediately called several times, where GetTickCount64 also appears, the whole thing looks like this:

```
00007FFE2D3CD930  | 48:894C24 08            | MOV QWORD PTR SS:[RSP+8],RCX
00007FFE2D3CD935  | 48:83EC 28              | SUB RSP,28
00007FFE2D3CD939  | 48:8D0D D0A41300        | LEA RCX,QWORD PTR DS:[7FFE2D507E10]
00007FFE2D3CD940  | E8 CB9DF9FF             | CALL sl-ac.7FFE2D367710
00007FFE2D3CD945  | 48:8BC8                 | MOV RCX,RAX
00007FFE2D3CD948  | E8 330F0000             | CALL sl-ac.7FFE2D3CE880
00007FFE2D3CD94D  | 48:8D0D 5CA41300        | LEA RCX,QWORD PTR DS:[7FFE2D507DB0]
00007FFE2D3CD954  | E8 B79DF9FF             | CALL sl-ac.7FFE2D367710
00007FFE2D3CD959  | 48:8BC8                 | MOV RCX,RAX
00007FFE2D3CD95C  | E8 1F0F0000             | CALL sl-ac.7FFE2D3CE880
00007FFE2D3CD961  | 48:8D0D 98A41300        | LEA RCX,QWORD PTR DS:[7FFE2D507E00]
00007FFE2D3CD968  | E8 A39DF9FF             | CALL sl-ac.7FFE2D367710
00007FFE2D3CD96D  | 48:8BC8                 | MOV RCX,RAX
00007FFE2D3CD970  | E8 0B0F0000             | CALL sl-ac.7FFE2D3CE880
00007FFE2D3CD975  | 48:8D0D B4A41300        | LEA RCX,QWORD PTR DS:[7FFE2D507E30]
00007FFE2D3CD97C  | E8 8F9DF9FF             | CALL sl-ac.7FFE2D367710
00007FFE2D3CD981  | 48:8BC8                 | MOV RCX,RAX
00007FFE2D3CD984  | E8 F70E0000             | CALL sl-ac.7FFE2D3CE880
00007FFE2D3CD989  | 48:8D0D 90A41300        | LEA RCX,QWORD PTR DS:[7FFE2D507E20]
00007FFE2D3CD990  | E8 7B9DF9FF             | CALL sl-ac.7FFE2D367710
00007FFE2D3CD995  | 48:8BC8                 | MOV RCX,RAX
00007FFE2D3CD998  | E8 E30E0000             | CALL sl-ac.7FFE2D3CE880
00007FFE2D3CD99D  | 48:8D0D 3CA41300        | LEA RCX,QWORD PTR DS:[7FFE2D507DE0]
00007FFE2D3CD9A4  | E8 679DF9FF             | CALL sl-ac.7FFE2D367710
00007FFE2D3CD9A9  | 48:8BC8                 | MOV RCX,RAX
00007FFE2D3CD9AC  | E8 CF0E0000             | CALL sl-ac.7FFE2D3CE880
00007FFE2D3CD9B1  | EB 00                   | JMP sl-ac.7FFE2D3CD9B3
00007FFE2D3CD9B3  | 48:83C4 28              | ADD RSP,28
00007FFE2D3CD9B7  | C3                      | RET
```

In the function at **7FFE2D3CE880**, it will constantly compare the current result of the **GetTickCount64** call with the result that was in the last call.

A little bit about GetTickCount64

And since we have already decided to touch on the topic with **GetTickCount64**, now I will try to explain the method of operation of this function and the bypass method.

If you look at the contents of this function in the debugger, then it is quite simple to implement.

```
8B0C25 0400FE7F         | MOV ECX,DWORD PTR DS:[7FFE0004]
B8 2003FE7F             | MOV EAX,7FFE0320
48:C1E1 20              | SHL RCX,20
48:8B00                 | MOV RAX,QWORD PTR DS:[RAX]
48:C1E0 08              | SHL RAX,8
48:F7E1                 | MUL RCX
48:8BC2                 | MOV RAX,RDX
C3                      | RET
```

000000007FFE0000 - Static address of the **KUSER_SHARED_DATA** structure (https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/ntddk/ns-ntddk-kuser_shared_data), which is accessed by the **GetTickCount64** function with the first instruction, and gets the value **TickCount.HighPart.**

The second instruction shoves **TickCount.LowPart** into **EAX** An approximate decompile of the function from IDA:

```
ULONG GetTickCount64( )
{
    return ( ( TickCount.HighPart << 32 ) * ( TickCount.LowPart << 8 ) ) >> 64;
}
```

So, we know that nothing can be written to **KUSER_SHARED_DATA**, because the structure is only Read Only, but we can intercept **GetTickCount64**, I will not show the hook code, but I will describe its algorithm only briefly.

- We check the return address (**GetTickCount64** is also used in system libraries in addition to the anti-cheat)
- We write a couple of checks for a period of time that will be suspicious for the anti-cheat
- We return our constant to the function.

Collecting data of PC components

Anti-cheat collects information about PC hardware devices due to incoming WMI requests.

To begin with, the module calls the **SysAllocString** function with the argument *"ROOT\\CIMV2",* in order to later use the pointer to the *IWbemLocator* interface to call the **ConnectServer** function to connect to the local namespace *ROOT\CIMV2*, also in **ConnectServer**, the last parameter is *IWbemServices **ppNamespace*, this pointer to the interface is needed to make calls to *IWbemServices*, for those who don't know - through this interface, then it will be possible to make a request to WMI using **ExecQuery**, but before that, the anti-cheat will also call **CoSetProxyBlanket**.

List of anti-cheat requests:

- *SELECT * FROM Win32_BaseBoard* - Win32_BaseBoard class provides information about the computer's motherboard, the anti-cheat takes the serial number and the string value Manufacturer, which contains the name of the motherboard manufacturer. In my case, after performing the anti-cheat function, the result will be called Gigabyte Technology Co, Ltd.
- *SELECT * FROM Win32_ComputerSystem* - from Win32_ComputerSystem class, the anti-cheat takes only the string value Model.

- *SELECT * FROM Win32_DiskDrive* - Win32_DiskDrive class, which contains information about existing hard drives, the anti-cheat from this class reads only the serial number.
- *SELECT * FROM Win32_Processor* - Win32_Processor class, which contains information about the processor, the anti-cheat reads only the processor name.
- *SELECT * FROM Win32_NetworkAdapter* - Win32_NetworkAdapter class, which contains information about the network adapter, according to the MSDN documentation, this class is outdated, and it is recommended to use the MSFT_NetAdapter class instead. Anti-cheat reads the MAC Address from there.
- *SELECT * FROM Win32_VideoController* - Win32_VideoController class, which allows you to get information about the video card, from this class you can notice a lot of properties for further generation of the video, but the developers, unfortunately, read only the name of the video card here.

I will supplement the information provided by the developer of the cheat MIDNIGHT, for which he thanks a lot.

Also through *__cpuid* collect maximum information and recheck (already on the server) with smbios.

Logging anti-cheat as a work of art

After a general analysis, it is worth mentioning one notable thing that helped me identify further actions of the anti-cheat - logging. Before performing some important action, the anti-cheat logs it for itself, of course, using the xor lines for this. So when I found out about it, I started looking for decryption of the string, I found it quickly.

The very function of decrypting the string looks like this:

```
4C:894424 18            | MOV QWORD PTR SS:[RSP+18],R8          |
48:895424 10            | MOV QWORD PTR SS:[RSP+10],RDX         |
48:894C24 08            | MOV QWORD PTR SS:[RSP+8],RCX          |
48:83EC 38              | SUB RSP,38                            |
48:8B4424 48            | MOV RAX,QWORD PTR SS:[RSP+48]         |
F3:0F6F00               | MOVDQU XMM0,XMMWORD PTR DS:[RAX]      |
66:0F7F4424 10          | MOVDQA XMMWORD PTR SS:[RSP+10],XMM0   |
48:8B4424 50            | MOV RAX,QWORD PTR SS:[RSP+50]         |
F3:0F6F00               | MOVDQU XMM0,XMMWORD PTR DS:[RAX]      |
66:0F7F0424             | MOVDQA XMMWORD PTR SS:[RSP],XMM0      |
66:0F6F0424             | MOVDQA XMM0,XMMWORD PTR SS:[RSP]      |
66:0FEF4424 10          | PXOR XMM0,XMMWORD PTR SS:[RSP+10]     |
66:0F7F4424 20          | MOVDQA XMMWORD PTR SS:[RSP+20],XMM0   |
48:8B4424 50            | MOV RAX,QWORD PTR SS:[RSP+50]         |
66:0F6F4424 20          | MOVDQA XMM0,XMMWORD PTR SS:[RSP+20]   |
F3:0F7F00               | MOVDQU XMMWORD PTR DS:[RAX],XMM0      | There will be a decrypted string in RAX
48:83C4 38              | ADD RSP,38                            |
C3                      | RET                                   |
```

This place will be repeatedly called, so we will intercept it.

And so I put together a list of logs that the anti-cheat leaves behind:

- [decrypt] CRC32 checks activated for %s
- [decrypt] The CRC checks have failed!
- [decrypt] Hooking thread started, parameter: %p
- [decrypt] MinHook initialized! %d
- [decrypt] Increasing working memory size to %p!
- [decrypt] Waiting on %s...
- [decrypt] A command has been received from the client "%s"
- [decrypt] Thread Start Address
- [decrypt] Hook "%s" placed on %s!
- [decrypt] Hook "%s" couldn't be placed on %s
- [decrypt] An uninitialized hook was activated.
- [decrypt] Failed to activate the "%s" hook!
- [decrypt] Managed to gather the address to il2cpp::find_game_objects in %p ms (%p tries)
- [decrypt] Protected launcher thread by ID %d
- [decrypt] SL-AC heartbeat tick!
- [decrypt] Status nominal.
- [decrypt] SL-AC thread protection tick!
- [decrypt] Running integrity check on watched addresses...
- [decrypt] Return Address: Thread RIP Check
- [decrypt] The return address at %s was invalid.
- [decrypt] A SL-AC thread has stopped execution inadvertently!
- [decrypt] Window check executed!
- [decrypt] Process check executed!
- [decrypt] Blacklisted program detected!
- [decrypt] Pattern scan executed!
- [decrypt] Developer Mode Enabled!
- [decrypt] Scanning mapped memory region: %p - %p
- [decrypt] Scanning module memory region: %p - %p
- [decrypt] Increasing working memory size to %p!
- [decrypt] Watching address ID %d
- [decrypt] Cheat Engine
- [decrypt] MIDKNIGHT LLC
- [decrypt] ReShade

COMMUNICATION WITH THE SERVER

In client/server communication, developers prefer the open-source cURL and Crypto++ libraries. Link to the repositories: github.com/curl/curl and github.com/weidai11/cryptopp.

Absolutely all packets sent from the anti-cheat to the server and the responses received from the server come through the functions of these libraries.

I will not describe in detail their connection, I will describe what the anti-cheat collects for subsequent encryption and sending to the server.

First of all, the anti-cheat starts logging the same actions as before, but at the same time wraps it in json, encrypts it and sends it to the server.

The decrypted packet looks like this:

- "{\"detection_information\":\"Status nominal.\",\"detection_status\":0,\"game_files\":[{\"name\":\"appdata.bat\"},{\"name\":\"ConfigTemplates\"},{\"name\":\"CreditsCache.json\"},{\"name\":\"GameAssembly.dll\"},{\"name\":\"license.txt\"},{\"name\":\"log.bat\"},{\"name\":\"log.txt\"},{\"name\":\"mono.msi\"},{\"name\":\"monoinstall.vdf\"},{\"name\":\"readme.txt\"},{\"name\":\"scl-sl.log\"},{\"name\":\"SCPSL.exe\"},{\"name\":\"SCPSL_Data\"},{\"name\":\"SL-AC.dll\"},{\"name\":\"Translations\"},{\"name\":\"Un"
- "{\"detection_information\":\"Initial heartbeating.\",\"detection_status\":0,\"game_files\":[{\"name\":\"appdata.bat\"},{\"name\":\"ConfigTemplates\"},{\"name\":\"CreditsCache.json\"},{\"name\":\"GameAssembly.dll\"},{\"name\":\"license.txt\"},{\"name\":\"log.bat\"},{\"name\":\"log.txt\"},{\"name\":\"mono.msi\"},{\"name\":\"monoinstall.vdf\"},{\"name\":\"readme.txt\"},{\"name\":\"scl-sl.log\"},{\"name\":\"SCPSL.exe\"},{\"name\":\"SCPSL_Data\"},{\"name\":\"SL-AC.dll\"},{\"name\":\"Translations\"},{\"name"

Then, when connecting to some server, the anti-cheat sends a list of downloaded modules, their size and base address

- "\",\"loaded_modules\":[{\"module_base\":140696840306688,\"module_name\":\"C:\\\\Program Files (x86)\\\\Steam\\\\steamapps\\\\common\\\\SCP Secret Laboratory\\\\SCPSL.exe\",\"module_size\":9986048},{\"module_base\":140714089709568,\"module_name\":\""

There was also information about the threads, the module goes through all the threads started earlier, collecting their ID, start address, current rip address and the address of the point stack.

- ",\"thread_start_address\":140712428775056},{\"thread_id\":1036,\"thread_module_start_address\":140712420179968,\"thread_rip\":140712447511423,\"thread_rsp\":140712447511423,\"thread_start_address\":140712428775056},{\"thread_id\":2252,\"thread_module_start_address\":140712420179968,\"thread_rip\":140712447511423,\"thread_rsp\":140712447511423,\"thread_start_address\":140712428775056},{\"thread_id\":8996,\"thread_module_start_address\":140712420179968,\"thread_rip\":140712447511423,\"thread_rsp\":140712"

But the anti-cheat writes all these addresses in the form of decimal numbers, if translated into hex, we get for example

module_base: **140696840306688** -> **00007FF689300000**. (SCP:SL Base Address)

module_size: **9986048** -> **0000000000986000**

Also, the anti-cheat after initializing heartbeating will repeatedly access the URL https://slac.scpslgame.com/beat / and call **GetTickCount64**, you don't have to go, because you will fail the User-Agent check and get a 405 error.

Conclusion

First of all, I will ask you to subscribe to my blog: https://t.me/colby5engineering

Thank you for reading this article, I was always lazy to do it, I was constantly distracted by something insignificant and just lazy, but at one point the motivation came to me by itself, and so in a couple of days I collected enough material to tell you :D

I will not stop thanking my team Team Enterial: Arting, anarh1st47, Dark_Bull, Easton, nelfo

I hope you liked the article and you learned something new for yourself! My team's blog: https://t.me/team_enterial_blog