

# 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](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.

00007FFBACDFB2AA	^ FFE0	jmp rax	00007FFBACF1D000:"MH_UNKNOWN"
00007FFBACDFB2AC	48:8D05 4D1D1200	lea rax,qword ptr ds:[7FFBACF1D000]	
00007FFBACDFB2B3	✓ EB 7C	jmp si-ac.7FFBACDFB331	00007FFBACF1D00C:"MH_OK"
00007FFBACDFB2B5	48:8D05 501D1200	lea rax,qword ptr ds:[7FFBACF1D00C]	
00007FFBACDFB2BC	✓ EB 73	jmp si-ac.7FFBACDFB331	
00007FFBACDFB2BE	48:8D05 531D1200	lea rax,qword ptr ds:[7FFBACF1D018]	00007FFBACF1D018:"MH_ERROR_ALREADY_IN
00007FFBACDFB2C5	✓ EB 6A	jmp si-ac.7FFBACDFB331	
00007FFBACDFB2C7	48:8D05 6A1D1200	lea rax,qword ptr ds:[7FFBACF1D038]	00007FFBACF1D038:"MH_ERROR_NOT_INITIA
00007FFBACDFB2CE	✓ EB 61	jmp si-ac.7FFBACDFB331	
00007FFBACDFB2D0	48:8D05 811D1200	lea rax,qword ptr ds:[7FFBACF1D058]	00007FFBACF1D058:"MH_ERROR_ALREADY_CR
00007FFBACDFB2D7	✓ EB 58	jmp si-ac.7FFBACDFB331	
00007FFBACDFB2D9	48:8D05 981D1200	lea rax,qword ptr ds:[7FFBACF1D078]	00007FFBACF1D078:"MH_ERROR_NOT_CREATE
00007FFBACDFB2E0	✓ EB 4F	jmp si-ac.7FFBACDFB331	
00007FFBACDFB2E2	48:8D05 A71D1200	lea rax,qword ptr ds:[7FFBACF1D090]	00007FFBACF1D090:"MH_ERROR_ENABLED"
00007FFBACDFB2E9	✓ EB 46	jmp si-ac.7FFBACDFB331	
00007FFBACDFB2EB	48:8D05 B61D1200	lea rax,qword ptr ds:[7FFBACF1D0A8]	00007FFBACF1D0A8:"MH_ERROR_DISABLED"
00007FFBACDFB2F2	✓ EB 3D	jmp si-ac.7FFBACDFB331	
00007FFBACDFB2F4	48:8D05 C51D1200	lea rax,qword ptr ds:[7FFBACF1D0C0]	00007FFBACF1D0C0:"MH_ERROR_NOT_EXECUT
00007FFBACDFB2FB	✓ EB 34	jmp si-ac.7FFBACDFB331	
00007FFBACDFB2FD	48:8D05 D41D1200	lea rax,qword ptr ds:[7FFBACF1D0D8]	00007FFBACF1D0D8:"MH_ERROR_UNSUPPORTE
00007FFBACDFB304	✓ EB 2B	jmp si-ac.7FFBACDFB331	
00007FFBACDFB306	48:8D05 EB1D1200	lea rax,qword ptr ds:[7FFBACF1D0F8]	00007FFBACF1D0F8:"MH_ERROR_MEMORY_ALL
00007FFBACDFB30D	✓ EB 22	jmp si-ac.7FFBACDFB331	
00007FFBACDFB30F	48:8D05 FA1D1200	lea rax,qword ptr ds:[7FFBACF1D110]	00007FFBACF1D110:"MH_ERROR_MEMORY_PRO
00007FFBACDFB316	✓ EB 19	jmp si-ac.7FFBACDFB331	
00007FFBACDFB318	48:8D05 091E1200	lea rax,qword ptr ds:[7FFBACF1D128]	00007FFBACF1D128:"MH_ERROR_MODULE_NOT
00007FFBACDFB31F	✓ EB 10	jmp si-ac.7FFBACDFB331	

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

00007FFFFB7AACCC	48:895C24 08	mov qword ptr ss:[rsp+8],rbx	DllMain
00007FFFFB7AACD1	48:897424 10	mov qword ptr ss:[rsp+10],rsi	[rsp+10]:"H<WH#8"
00007FFFFB7AACD6	57	push rdi	
00007FFFFB7AACD7	48:83EC 20	sub rsp,20	
00007FFFFB7AACDB	49:8BF8	mov rdi,r8	r8:startAddress
00007FFFFB7AACDE	8BDA	mov ebx,edx	
00007FFFFB7AACE0	48:8BF1	mov rsi,rcx	
00007FFFFB7AACE3	83FA 01	cmp edx,1	
00007FFFFB7AACE6	75 05	jne sl-ac.7FFFFB7AACED	
00007FFFFB7AACE8	E8 0B0C0000	call sl-ac.7FFFFB7AB8F8	
00007FFFFB7AAEC7	4C:8BC7	mov r8,rdi	r8:startAddress
00007FFFFB7AAEF0	8BD3	mov edx,ebx	
00007FFFFB7AAEF2	48:8BCE	mov rcx,rsi	
00007FFFFB7AAEF5	48:8B5C24 30	mov rbx,qword ptr ss:[rsp+30]	
00007FFFFB7AAEFA	48:8B7424 38	mov rsi,qword ptr ss:[rsp+38]	[rsp+38]:"MZh"
00007FFFFB7AAEFF	48:83C4 20	add rsp,20	
00007FFFFB7AAD03	5F	pop rdi	
00007FFFFB7AAD04	E9 8FFEFFFF	jmp sl-ac.7FFFFB7AB998	

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.

00007FFFFD6D5610	67:0C4 40 01	cmp qword ptr ss:[rsp+40],1	
00007FFFFD6D5615	75 3A	jne sl-ac.7FFFFD6D5651	
00007FFFFD6D5617	FF15 F36D0F00	call qword ptr ds:[7FFFFD7CC410]	
00007FFFFD6D561D	8905 8D131400	mov dword ptr ds:[7FFFFD8169B0],eax	
00007FFFFD6D5623	48:C74424 28 00000000	mov qword ptr ss:[rsp+28],0	
00007FFFFD6D562C	C74424 20 00000000	mov dword ptr ss:[rsp+20],0	
00007FFFFD6D5634	4C:8B4C24 40	mov r9,qword ptr ss:[rsp+40]	
00007FFFFD6D5639	4C:8D05 20FAFFFF	lea r8,qword ptr ds:[<startAddress>]	r8:startAddress
00007FFFFD6D5640	33D2	xor edx,edx	
00007FFFFD6D5642	33C9	xor ecx,ecx	
00007FFFFD6D5644	FF15 DE6D0F00	call qword ptr ds:[7FFFFD7CC428]	CreateThread
00007FFFFD6D564A	48:8905 67131400	mov qword ptr ds:[7FFFFD8169B8],rax	
00007FFFFD6D5651	B8 01000000	mov eax,1	
00007FFFFD6D5656	48:83C4 38	add rsp,38	
00007FFFFD6D565A	C3	ret	

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: GetAsyncKerState
- Imported library: ntdll.dll, exported function: DbgUiRemoteBreakin
- (Supplied by Themida)
- Imported library: ntdll.dll, exported function: NtOpenFile

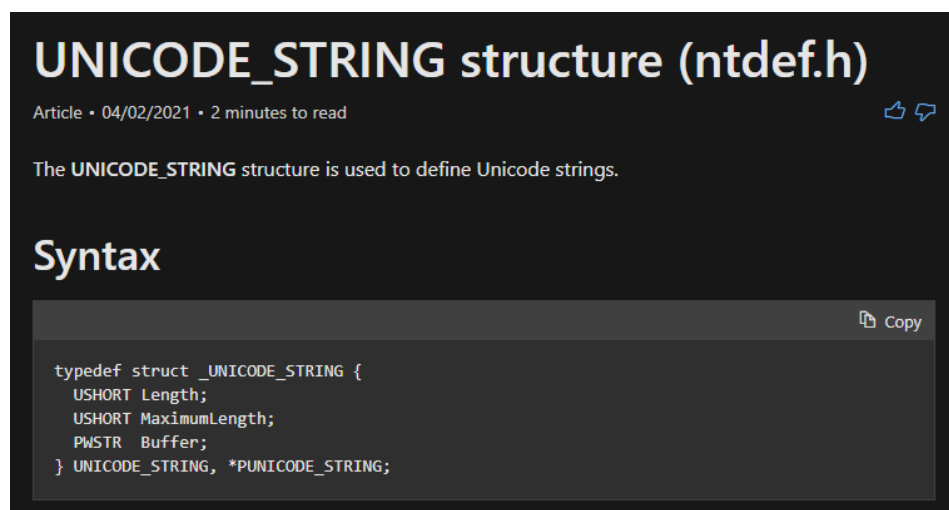
## Blocking injection (NtOpenFile hook)

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.

00007FFD3C30F91C	48:8B9424 10040000	mov rdx,qword ptr ss:[rsp+410]	[rsp+410]:&L"C:\\Windows\\System32\\Win
00007FFD3C30F924	48:8B4C24 48	mov rcx,qword ptr ss:[rsp+48]	[rsp+48]:&L"GameOverOverlayRender_PIDStrear
00007FFD3C30F929	E8 B2B8FFFF	call s1-ac.7FFD3C30B1E0	
00007FFD3C30F92E	48:894424 50	mov qword ptr ss:[rsp+50],rax	[rsp+50]:&L"GameOverOverlayRender_PIDStrear
00007FFD3C30F933	48:8B4424 50	mov rax,qword ptr ss:[rsp+50]	[rsp+50]:&L"GameOverOverlayRender_PIDStrear
00007FFD3C30F938	48:894424 58	mov qword ptr ss:[rsp+58],rax	[rsp+58]:&L"GameOverOverlayRender_PIDStrear
00007FFD3C30F93D	48:8B5424 58	mov rdx,qword ptr ss:[rsp+58]	Pointer to Path_Buffer
00007FFD3C30F942	48:8B4C24 60	mov rcx,qword ptr ss:[rsp+60]	Pointer to PE Header
00007FFD3C30F947	E8 644C0600	call <s1-ac.IsFileInWindows>	
00007FFD3C30F94C	884424 30	mov byte ptr ss:[rsp+30],al	
00007FFD3C30F950	0FB64424 30	movzx eax,byte ptr ss:[rsp+30]	
00007FFD3C30F955	85C0	test eax,eax	
00007FFD3C30F957	0F84 F3000000	jg s1-ac.7FFD3C30FA50	
00007FFD3C30F95D	48:8D4424 31	lea rax,qword ptr ss:[rsp+31]	
00007FFD3C30F962	48:8BF8	mov rdi,rax	
00007FFD3C30F965	33C0	xor eax,eax	
00007FFD3C30F967	B9 01000000	mov ecx,1	
00007FFD3C30F96C	F3:AA	rep stosb	
00007FFD3C30F96E	48:8D4424 32	lea rax,qword ptr ss:[rsp+32]	
00007FFD3C30F973	48:8BF8	mov rdi,rax	
00007FFD3C30F976	33C0	xor eax,eax	
00007FFD3C30F978	B9 01000000	mov ecx,1	
00007FFD3C30F97D	F3:AA	rep stosb	
00007FFD3C30F97F	48:8D4424 33	lea rax,qword ptr ss:[rsp+33]	
00007FFD3C30F984	48:8BF8	mov rdi,rax	

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]	[rsp+20]:&L"Microsoft Windows"
00007FFD3C30F5A8	48:894424 20	mov qword ptr ss:[rsp+20],rax	
00007FFD3C30F5AF	41:B9 00000B00	mov r9d,B0000	
00007FFD3C30F5B5	45:33C0	xor r8d,r8d	
00007FFD3C30F5B8	BA 01000100	mov edx,10001	
00007FFD3C30F5C5	48:8B8C24 E0000000	mov rcx,qword ptr ss:[rsp+E0]	
00007FFD3C30F5C5	FF15 0DBB1400	call qword ptr ds:[7FFD3C45B008]	CertFindCertificateInStore
00007FFD3C30F5C8	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]	
00007FFD3C30F5E5	898424 8C000000	mov dword ptr ss:[rsp+8C],eax	
00007FFD3C30F5ED	48:8B8C24 40020000	mov rcx,qword ptr ss:[rsp+240]	
00007FFD3C30F5F2	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]	

00007FFD3C2F694	E8 F7080000	call sl-ac.7FFD3C2F6F9	
00007FFD3C2F699	8B4C24 64	mov ecx,dword ptr ss:[rsp+64]	
00007FFD3C2F6A1	894C24 28	mov dword ptr ss:[rsp+28],ecx	[rsp+20]:&L"Microsoft Windows Production I
00007FFD3C2F6A6	45:33C9	xor r9d,r9d	
00007FFD3C2F6A9	41:B8 01000000	mov r8d,1	
00007FFD3C2F6AF	BA 04000000	mov edx,4	
00007FFD3C2F6B4	48:8B4C24 78	mov rcx,qword ptr ss:[rsp+78]	
00007FFD3C2F6B9	FF15 09B91400	call qword ptr ds:[7FFD3C2F6078]	CertGetNameStringW
00007FFD3C2F6BF	894424 64	mov dword ptr ss:[rsp+64],eax	
00007FFD3C2F6C3	837C24 64 00	cmp dword ptr ss:[rsp+64],0	
00007FFD3C2F6C8	75 36	jne sl-ac.7FFD3C2F6FE	
00007FFD3C2F6C8	FF15 48BD1400	call qword ptr ds:[7FFD3C2F6418]	
00007FFD3C2F6D0	898424 94000000	mov dword ptr ss:[rsp+94],eax	
00007FFD3C2F6D7	48:8B8C24 F8000000	lea rcx,qword ptr ss:[rsp+F8]	[rsp+F8]:&L"Microsoft Windows Production I
00007FFD3C2F6D7	E8 1C090000	call sl-ac.7FFD3C303000	
00007FFD3C2F6E4	90	nop	
00007FFD3C2F6E5	48:8B8C24 40020000	mov rcx,qword ptr ss:[rsp+240]	
00007FFD3C2F6E6	E8 4E30FFFF	call sl-ac.7FFD3C2F6340	
00007FFD3C2F6F2	898424 94000000	mov eax,dword ptr ss:[rsp+94]	
00007FFD3C2F6F9	E9 B6010000	jmp sl-ac.7FFD3C2F6B4	
00007FFD3C2F6FE	8B4424 64	mov eax,dword ptr ss:[rsp+64]	
00007FFD3C2F702	48:898424 C0000000	mov qword ptr ss:[rsp+C0],rax	[rsp+F8]:&L"Microsoft Windows Production I
00007FFD3C2F708	48:8B8C24 F8000000	lea rcx,qword ptr ss:[rsp+F8]	
00007FFD3C2F712	E8 79080000	call sl-ac.7FFD3C2F6F9	
00007FFD3C2F717	48:8B8C24 C0000000	mov rcx,qword ptr ss:[rsp+C0]	
00007FFD3C2F71F	4C:8BC1	mov r8,rcx	

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.

00007FFD440FFD6F	48:898424 48010000	mov qword ptr ss:[rsp+148],rax	
00007FFD440FFD77	48:8B8424 48010000	mov rax,qword ptr ss:[rsp+148]	
00007FFD440FFD7F	48:898424 58010000	mov qword ptr ss:[rsp+158],rax	
00007FFD440FFD87	4C:8B8424 50010000	mov r8,qword ptr ss:[rsp+150]	
00007FFD440FFD8F	48:8B9424 58010000	mov rdx,qword ptr ss:[rsp+158]	
00007FFD440FFD97	48:8B8C24 60010000	mov rcx,qword ptr ss:[rsp+160]	
00007FFD440FFD9F	E8 DCC70100	call <sl-ac.IsBlacklistedCert>	

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

00007FF8B806C2E0 <sl	4C:894424 18	mov qword ptr ss:[rsp+18],r8	MH_CreateHook
00007FF8B806C2E5	48:895424 10	mov qword ptr ss:[rsp+10],rdx	
00007FF8B806C2EA	48:894C24 08	mov qword ptr ss:[rsp+8],rcx	
00007FF8B806C2EF	57	push rdi	
00007FF8B806C2F0	48:81EC 90000000	sub rsp,90	
00007FF8B806C2F7	48:8BFC	mov rdi,rsi	
00007FF8B806C2FA	B9 24000000	mov ecx,24	24: '\$'
00007FF8B806C2FF	B8 CCCCCCCC	mov eax,CCCCCCCC	
00007FF8B806C304	F3:AB	rep stosd	
00007FF8B806C306	48:8B8C24 A0000000	mov rcx,qword ptr ss:[rsp+A0]	
00007FF8B806C30E	48:8B05 0B231200	mov rax,qword ptr ds:[7FF8B818E620]	00007FF8B818E620: "PJ,,Y,n"
00007FF8B806C315	48:33C4	xor rax,rsi	
00007FF8B806C318	48:898424 88000000	mov qword ptr ss:[rsp+88],rax	
00007FF8B806C320	C74424 20 00000000	mov dword ptr ss:[rsp+20],0	
00007FF8B806C328	E8 53150000	call sl-ac.7FF8B806D880	
00007FF8B806C32D	48:833D D3741200	cmp qword ptr ds:[7FF8B8193808],0	
00007FF8B806C335	0F84 24020000	je sl-ac.7FF8B806C55F	
00007FF8B806C33B	48:8B8C24 A0000000	mov rcx,qword ptr ss:[rsp+A0]	
00007FF8B806C343	E8 481A0000	call sl-ac.7FF8B806DD90	
00007FF8B806C348	85C0	test eax,ecx	

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,rsi	
00007FF8B806C7CD	B9 00000000	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.

```

C:\Program Files (x86)\Steam\steamapps\common\SCP Secret Laboratory\SCPSL.exe
[2022-07-25 22:14:12.634] [info] Starting up..
[2022-07-25 22:14:20.244] [info] Installing hooks...

[2022-07-25 22:14:20.255] [debug] MH_CreateHook address: 7ff8b0e1c2e0
[2022-07-25 22:14:20.256] [debug] MH_EnableHook address: 7ff8b0e1c7c0

[2022-07-25 22:14:20.256] [debug] Initialization status: 0
[2022-07-25 22:14:20.256] [debug] Hook created!
[2022-07-25 22:14:20.256] [debug] Hook created!

[2022-07-25 22:14:20.338] [info] Hooks are placed!

Attempting to set a hook on an address -> 0x00007FF923DDAEC0 : Detour Address -> 0x00007FF8B0DAA260
Attempting to set a hook on an address -> 0x00007FF9241AD590 : Detour Address -> 0x00007FF8B0DA9EF0
Attempting to set a hook on an address -> 0x00007FF923DE04F0 : Detour Address -> 0x00007FF8B0DA97E0
Attempting to set a hook on an address -> 0x00007FF923DDFEE0 : Detour Address -> 0x00007FF8B0DA9920
Attempting to set a hook on an address -> 0x00007FF923DE5290 : Detour Address -> 0x0000019B4E6F562C
Attempting to set a hook on an address -> 0x00007FF923DDF0B0 : Detour Address -> 0x00007FF8B0DA9A60
Attempting to set a hook on an address -> 0x00007FF923DD130 : Detour Address -> 0x00007FF8B0DA9B80
Attempting to set a hook on an address -> 0x00007FF9224A3E60 : Detour Address -> 0x00007FF8B0DA9CC0
Attempting to set a hook on an address -> 0x00007FF9224B3F40 : Detour Address -> 0x0000019B4E6F51CC
Attempting to set a hook on an address -> 0x0000019B5005C4B0 : Detour Address -> 0x00007FF8B0E067F0

```

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.



00007FF8B7AEA18E	4C:8D8C24 50010000	lea r9,qword ptr ss:[rsp+150]	
00007FF8B7AEA196	4C:8B8424 A0000000	mov r8,qword ptr ss:[rsp+A0]	
00007FF8B7AEA19E	48:8B9424 A0000000	mov rdx,qword ptr ss:[rsp+A8]	
00007FF8B7AEA1A6	48:8B8C24 B0000000	mov rcx,qword ptr ss:[rsp+B0]	
00007FF8B7AEA1AE	E8 4D6E0000	call si-ac.7FF8B7AF1000	
00007FF8B7AEA1B3	0FB6C0	movzx eax,al	
00007FF8B7AEA1B6	85C0	test eax,eax	
00007FF8B7AEA1B8	75 57	jne si-ac.7FF8B7AEA211	
00007FF8B7AEA1BA	48:8B8424 B0010000	mov rax,qword ptr ss:[rsp+1B0]	
00007FF8B7AEA1C2	48:8B08	mov rcx,qword ptr ds:[rax]	<= Handle
00007FF8B7AEA1C5	FF15 55221500	call qword ptr ds:[7FF8B7C3C420]	<= Virtualized CloseHandle Call
00007FF8B7AEA1CB	48:8B8424 B0010000	mov rax,qword ptr ss:[rsp+1B0]	
00007FF8B7AEA1D3	48:C700 FFFFFFFF	mov qword ptr ds:[rax],FFFFFFFFFFFFFFFF	
00007FF8B7AEA1DA	C74424 34 D8100000	mov dword ptr ss:[rsp+34],si-ac.00007FF8B7CA6AFE	
00007FF8B7AEA1E2	48:8D8C24 30010000	lea rcx,qword ptr ss:[rs	jmp si-ac.7FF8B81863CA
00007FF8B7AEA1EA	E8 419BFFFF	call si-ac.7FF8B7AE3D30	jmp si-ac.7FF8B7F702D5
00007FF8B7AEA1EF	90	nop	
00007FF8B7AEA1F0	48:8D8C24 50010000	lea rcx,qword ptr ss:[rs	mov ecx,E98A9458
00007FF8B7AEA1F8	E8 339BFFFF	call si-ac.7FF8B7AE3D30	mov ebp,E9001962
00007FF8B7AEA1FD	90	nop	adc edi,ecx
00007FF8B7AEA1FE	48:8D8C24 70010000	lea rcx,qword ptr ss:[rs	add byte ptr ds:[r8],r8b
00007FF8B7AEA206	E8 259BFFFF	call si-ac.7FF8B7AE3D30	add byte ptr ds:[rax],al
00007FF8B7AEA20B	8B4424 34	mov eax,dword ptr ss:[rs	add byte ptr ds:[rax],al
00007FF8B7AEA20F	EB 2D	jmp si-ac.7FF8B7AEA23E	add eax,0
00007FF8B7AEA211	48:8D8C24 30010000	lea rcx,qword ptr ss:[rs	add byte ptr ds:[rcx],al
00007FF8B7AEA219	E8 129BFFFF	call si-ac.7FF8B7AE3D30	std
00007FF8B7AEF021F	00	nop	cmp al,0
			add esp,ebp
			cni

## 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;"`

00007FF72965D0D0	33C0	xor eax,eax	IsVirtualMachine
00007FF72965D0D2	C3	ret	
00007FF72965D0D3	90	nop	
00007FF72965D0D4	90	nop	
00007FF72965D0D5	48:894C24 08	mov qword ptr ss:[rsp+8],rcx	
00007FF72965D0DA	48:81EC 88020000	sub rsp,288	
00007FF72965DAE1	48:8B05 B0280B00	mov rax,qword ptr ds:[7FF729718398]	
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+28],0	
00007FF72965DB0B	48:837C24 28 FF	cmp qword ptr ss:[rsp+28],FFFFFFFFFFFFFFFF	
00007FF72965DB13	74 3D	je scps1.7FF72965DB52	
00007FF72965DB15	C74424 30 88020000	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]	<= ProcessNext32
00007FF72965DB27	FF15 C3560800	call qword ptr ds:[7FF7296E31F0]	
00007FF72965DB2D	83F8 01	cmp eax,1	
00007FF72965DB30	75 20	jne scps1.7FF72965DB52	
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 <scps1.compareString>	
00007FF72965DB44	85C0	test eax,eax	
00007FF72965DB46	75 08	jne scps1.7FF72965DB50	
00007FF72965DB48	8B4424 38	mov eax,dword ptr ss:[rsp+38]	
00007FF72965DB4C	894424 20	mov dword ptr ss:[rsp+20],eax	
00007FF72965DB50	EB CB	jmp scps1.7FF72965DB10	
00007FF72965DB52	8B4424 20	mov eax,dword ptr ss:[rsp+20]	
00007FF72965DB56	894424 24	mov dword ptr ss:[rsp+24],eax	

Well, the meme itself

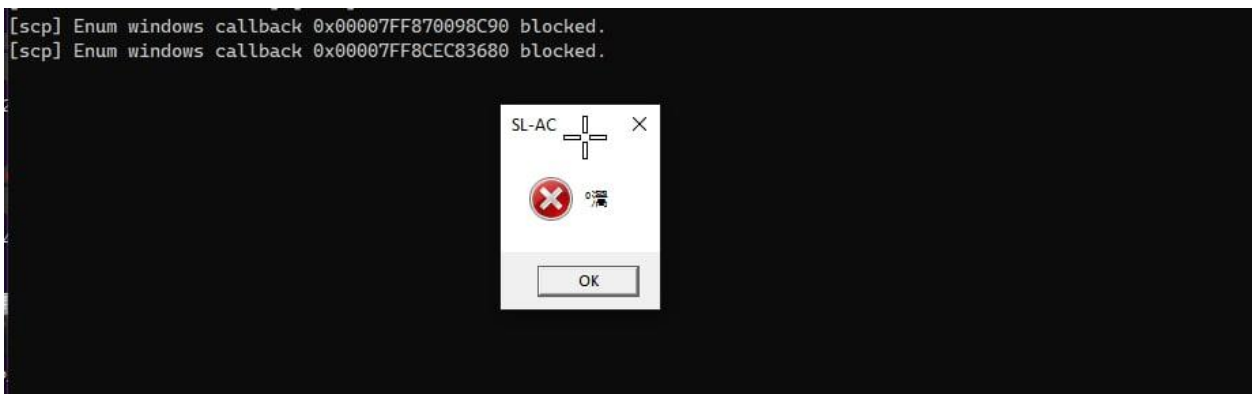


## Second detection mechanism

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 - x32
00007FF907136CB3	E8 00000000	CALL s1-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 s1-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 s1-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 s1-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:

```
1 __int64 __fastcall ScanMemory(__int64 a1, __int64 a2)
2 {
3     unsigned __int64 baseAddress; // [rsp+20h] [rbp-68h]
4     unsigned __int64 v4; // [rsp+28h] [rbp-60h]
5     __int64 memoryInfo[3]; // [rsp+48h] [rbp-40h] BYREF
6     __int64 regionSize; // [rsp+60h] [rbp-28h]
7     int v7; // [rsp+68h] [rbp-20h]
8     int memProtect; // [rsp+6Ch] [rbp-1Ch]
9
10    if ( !byte_7FFBB3D35CC8 )
11        VmGetStartupInfo(&unk_7FFBB3D35E90);
12    baseAddress = qword_7FFBB3D35E98;
13    v4 = qword_7FFBB3D35EA0;
14    while ( baseAddress < v4 )
15    {
16        VmVirtualQuery(baseAddress, memoryInfo, 0x30i64);
17        if ( (v7 & 0x1000) == 4096 && (memProtect == PAGE_EXECUTE_READ || memProtect == PAGE_EXECUTE_READWRITE) )
18            sub_7FFBB3BF7CF0(a2, memoryInfo[0], regionSize);
19        baseAddress += regionSize;
20    }
21    return sub_7FFBB3B990D0(a2);
22 }
```

## 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:[<6Sleep>]	
E9 3EFFFFFFF	JMP sl-ac.7FFE3259B3D9	Returning to JE check
48:8BF8	MOV RDI,RAX	

- 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](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 \*\*ppNamespaces*, 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	
48:83C4 38	ADD RSP,38	There will be a decrypted string in RAX
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](https://github.com/curl/curl) and [github.com/weidai11/cryptopp](https://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\":\"ConfigTemp  
lates\"},{\"name\":\"CreditsCache.json\"},{\"name\":\"GameAssembly.dll\"},{\"name\":\"license.txt\"},  
{\"name\":\"log.bat\"},{\"name\":\"log.txt\"},{\"name\":\"mono.msil\"},{\"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\":\"ConfigT  
emplates\"},{\"name\":\"CreditsCache.json\"},{\"name\":\"GameAssembly.dll\"},{\"name\":\"license.t  
xt\"},{\"name\":\"log.bat\"},{\"name\":\"log.txt\"},{\"name\":\"mono.msil\"},{\"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":140712447511423}

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.scpssgame.com/beat /](https://slac.scpssgame.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](https://t.me/team_enterial_blog)