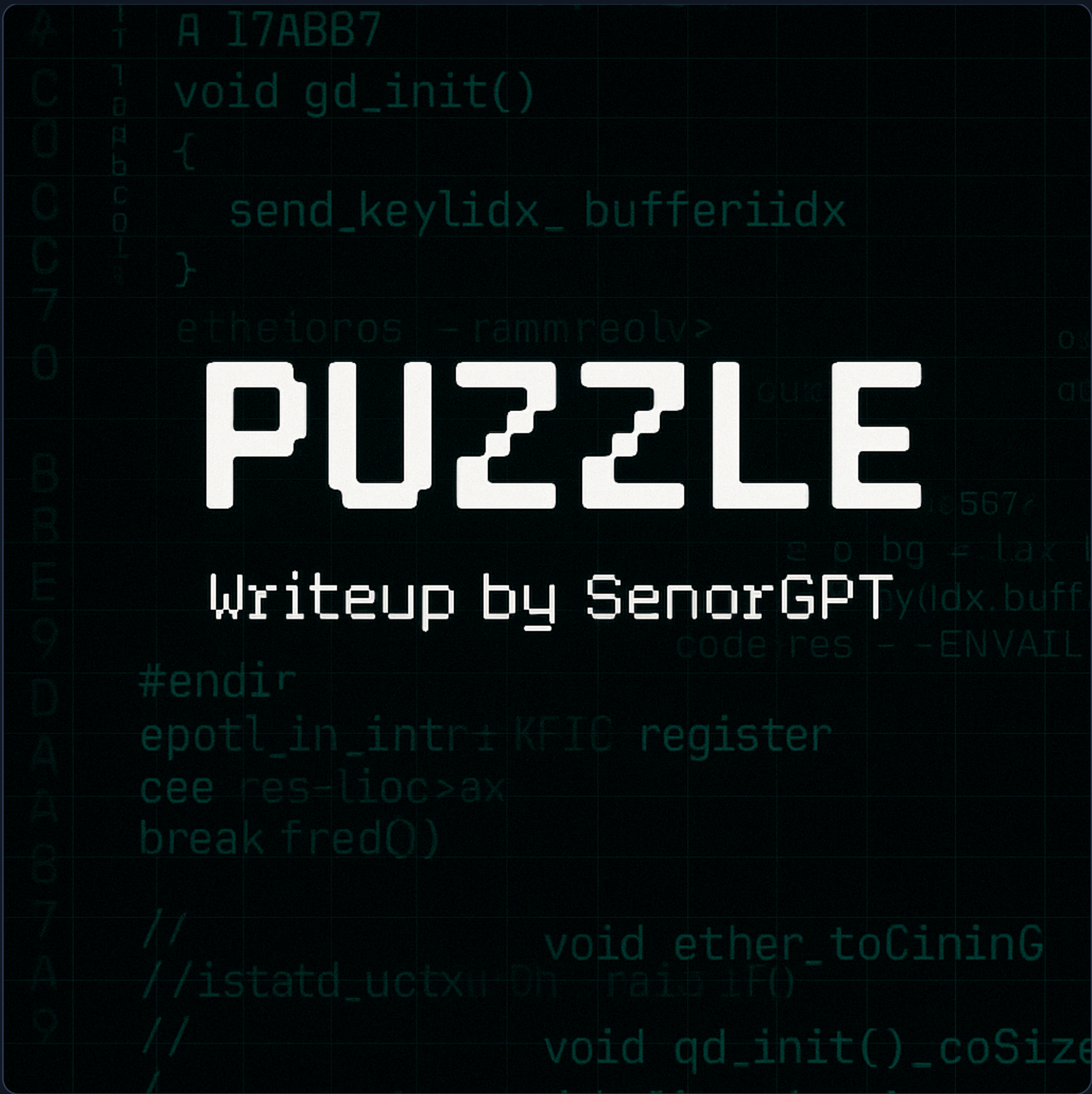
BigIsim04's puzzle — Reverse Engineering Write-up

Challenge link: https://crackmes.one/crackme/691de1d12d267f28f69b7f16
Author: *BigIsim04*
Write-up by: *SenorGPT*
Tools used: *CFF Explorer, Detect It Easy (DIE), x64dbg*

Platform	Difficulty	Quality	Arch	Language
Windows	2.5	3.5	x86-64	C/C++



Status: WIP
Goal: Document a clean path from initial recon → locating key-check logic → validation/reversal strategy

---

---

1. Executive Summary

This document captures my reverse-engineering process for the crackme `puzzle` by `Biglsim04`. The target appears to be a simple command line process that prompts the user for a password.

I successfully:

- Performed basic static reconnaissance.
- Surveyed imports. Confirmed there appears to be anti-debugging measures.

- Tried to locate strings associated with success & failure dialogs.
- Added breakpoints on functions that may be used for anti-debugging and begun to trace logic.

---

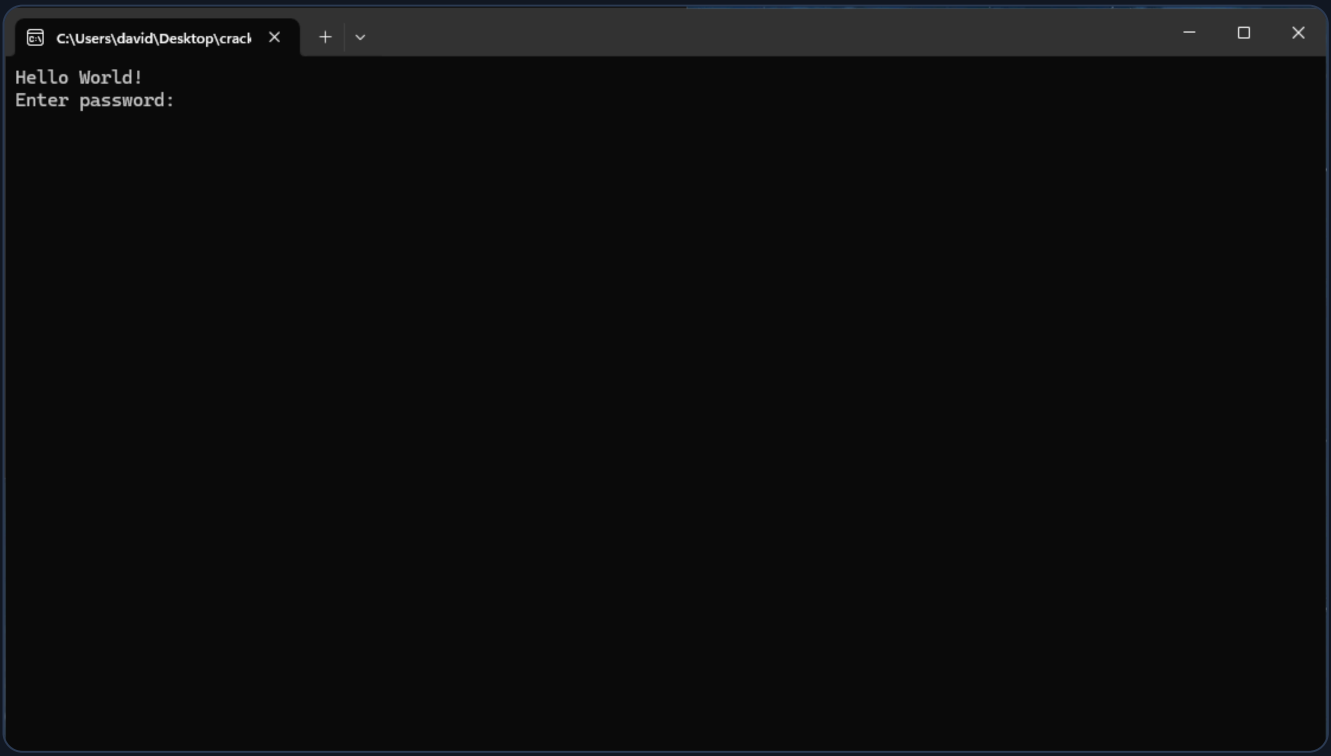
2. Target Overview

2.1 UI / Behaviour

- Inputs: *Enter password:*
- Outputs: *Access Denied*, *Access Accepted* (assumption based on wrong answer string).

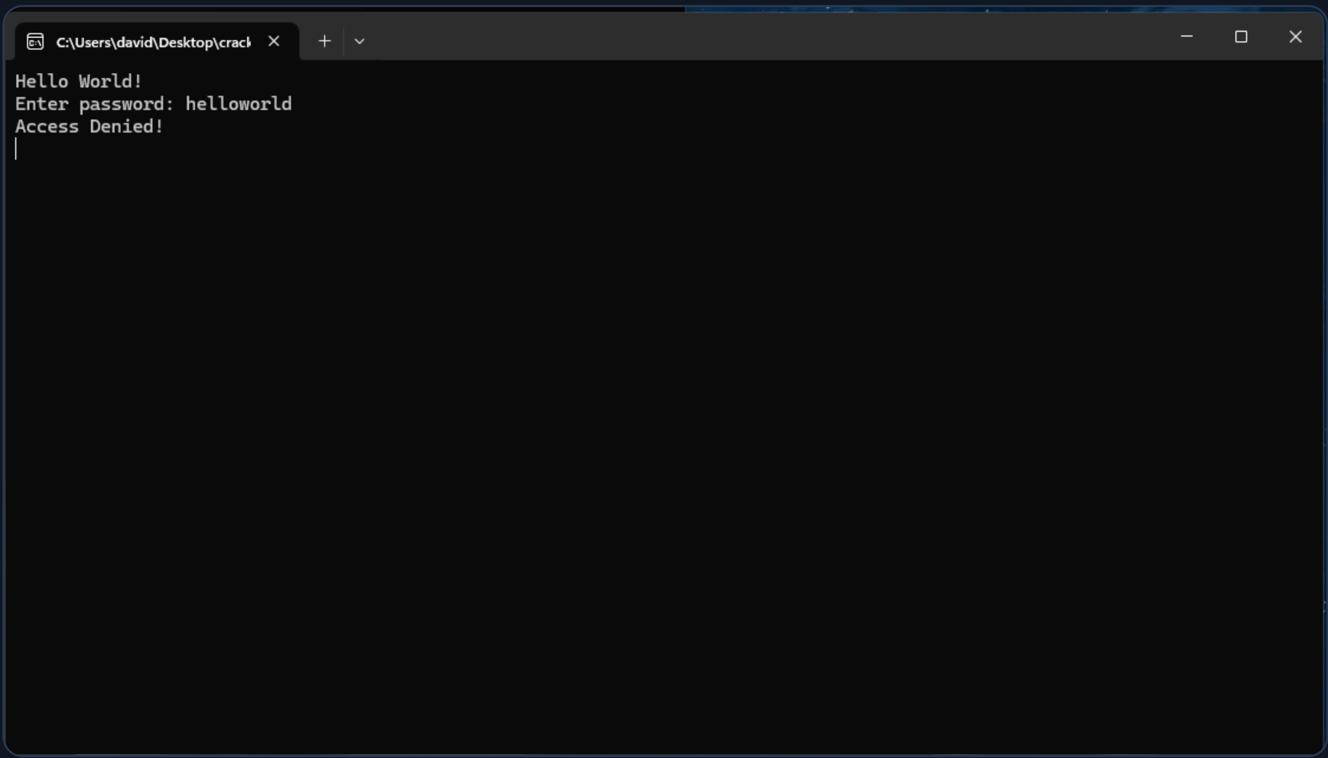
2.2 Screens

2.2.1 Start-up



2.2.2 Failure case

Followed by exit on next key input.



---

3. Tooling & Environment

- OS: *Windows 11*
- Debugger: *x64dbg*
- Static tools: *CFF Explorer, Detect It Easy (DIE)*

---

4. Static Recon

4.1 File & Headers

There appears to be no obvious signs of packing or obfuscation. The classic boring set of sections `.text`, `.rdata`, `.data`, `.reloc` represent a very typical layout for an unprotected Visual Studio type Portable Executable (*PE*).

The sizes also seem reasonable for a small console application.

puzzle.exe

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N...	Linenumbers ...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	00020000	00001000	0001FE00	00000400	00000000	00000000	0000	0000	60000020
.rdata	0000F000	00021000	0000EC00	00020200	00000000	00000000	0000	0000	40000040
.data	00008000	00030000	00001400	0002EE00	00000000	00000000	0000	0000	C0000040
.reloc	00000934	00038000	00000A00	00030200	00000000	00000000	0000	0000	42000040

Packed binaries often show one or more of these red flags:

- **Weird section names:**
  `.UPX0, .UPX1, .aspack, .petite`, or just random gibberish.
- **Very few sections:**
  Sometimes just one or two suspicious ones.
- **Abnormal size balance:**
  A tiny `.text` with a huge other section holding compressed payload.

It is *IMPORTANT* to note that headers alone can not confirm if the *PE* has been packed or obfuscated as the packer/obfuscator used might utilize normal looking section names, keep a standard layout, and/or hide the real tell in entropy or runtime behaviour.

4.2 Entropy

Entropy is a measure of how *random-looking* the bytes are in a section.

- Low entropy = looks like normal code/data (more patterns, more repetition).
- High entropy = looks compressed or encrypted (more random).

Why this matters:

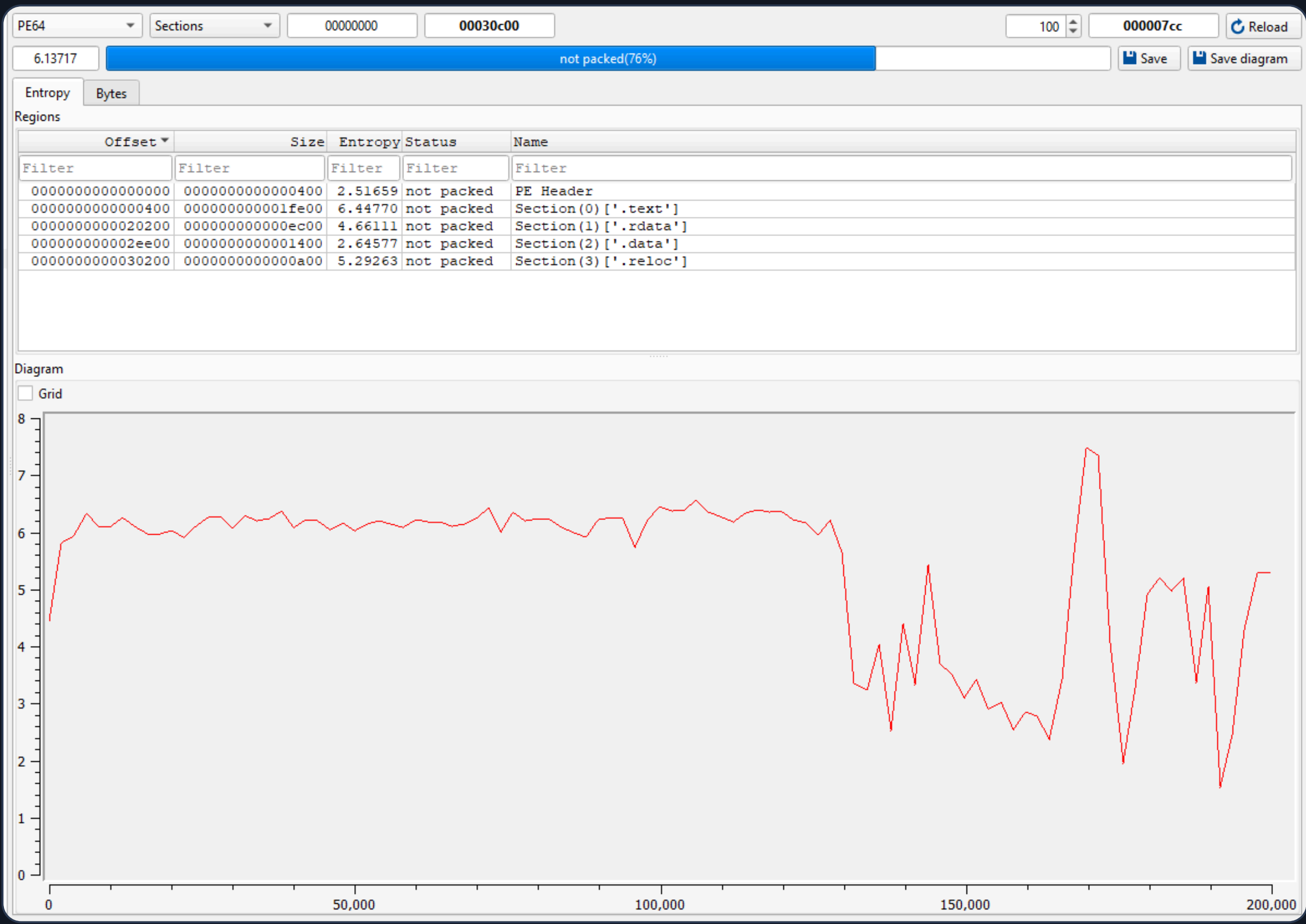
- Packed or encrypted payloads often have high entropy.
- Normal `.text` code usually has moderate entropy.

Rule of thumb (quick reference, not 100%):

- ~6.0–7.2 = often normalish

- ~7.4–8.0 = suspicious for compression/encryption

Unfortunately, *CFF Explorer* does not have an entropy viewer so I switch to *DIE*.



The top blue bar shows *DIE*'s overall heuristic guess based mostly on entropy patterns and layout. This is not necessarily proof, but a strong hint that this is not classically packed.

The table shows each row as a region - header + each *PE* section - with an entropy score.

Section Name	Entropy Score	Note
PE Header	2.51659	Low entropy is normal for headers.
.text	6.44770	normal looking code entropy. If this was packed or encrypted the value would be closer to ~7.5–8.0.
.rdata	4.66111	Normal for constants/strings/tables.

Section Name	Entropy Score	Note
.data	2.64577	Very normal (initialized globals).
.reloc	5.29263	Also not unusual.

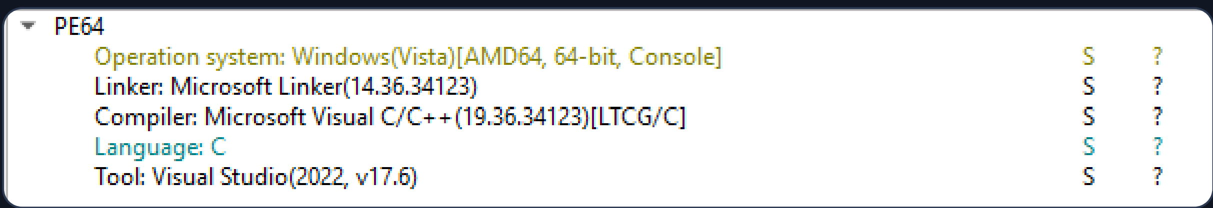
Nothing here also seems to scream that this *PE* is packed.

Finally, the graph represents a rolling entropy line across the file from start to end.
The long flatish area around *~6* matches the `.text` region.
The later dips and spikes reflect transitions into `.rdata`, `.data`, `.reloc`.

Again, if this *PE* was packed the graph would have a big chunk of the line hovering around *~7.4-8*.

4.3 Build & Toolchain Information

Screenshot summary provided by *DIE*



Operation system: Windows(Vista)AMD64, 64-bit, Console
The binary is a *64-bit Windows console app*. The *Vista* part usually reflects the *minimum subsystem version* or tool heuristics and *NOT* that it only runs on Vista.

Linker: Microsoft Linker (14.36.34123)
The exact *MSVC linker version* used to produce the EXE.

Compiler: Microsoft Visual C/C++ (19.36.34123) [LTCG/C]
Identifies the *Visual C++ compiler version*.
LTCG = *Link-Time Code Generation* (whole-program optimization). The `/C` part is just the tool's way of labelling the language/compile family.

Language: C
DIE's best guess for source language. In practice, this likely means *C or C++* with a C-like signature.

Tool: Visual Studio(2022, v17.6)
Maps those version numbers to the likely *IDE/toolchain family* that was used to build the *EXE*.

4.4 Imports / Exports

Since it is a simple console application, the only import *SEEMS* to be `KERNEL32.dll`.

OFTs	FTs (IAT)	Hint	Name
Qword	Qword	Word	szAnsi
000000000002F3E0	000000000002F3E0	0337	GetTickCount64
000000000002F3F2	000000000002F3F2	0336	GetTickCount
000000000002FA0E	000000000002FA0E	0657	WriteConsoleW
000000000002F410	000000000002F410	047B	QueryPerformanceCounter
000000000002F42A	000000000002F42A	023C	GetCurrentProcessId
000000000002F440	000000000002F440	0240	GetCurrentThreadId
000000000002F456	000000000002F456	0314	GetSystemTimeAsFileTime
000000000002F470	000000000002F470	0394	InitializeSListHead
000000000002F486	000000000002F486	0501	RtlCaptureContext
000000000002F49A	000000000002F49A	0509	RtlLookupFunctionEntry
000000000002F4B4	000000000002F4B4	0510	RtlVirtualUnwind
000000000002F4C8	000000000002F4C8	03AA	IsDebuggerPresent
000000000002F4DC	000000000002F4DC	05F3	UnhandledExceptionFilter
000000000002F4F8	000000000002F4F8	05B0	SetUnhandledExceptionFilter
000000000002F516	000000000002F516	02FB	GetStartupInfoW
000000000002F528	000000000002F528	03B2	IsProcessorFeaturePresent
000000000002F544	000000000002F544	029F	GetModuleHandleW
000000000002F558	000000000002F558	0151	EnterCriticalSection
000000000002F570	000000000002F570	03EA	LeaveCriticalSection
000000000002F588	000000000002F588	0391	InitializeCriticalSectionEx
000000000002F5A6	000000000002F5A6	012B	DeleteCriticalSection
000000000002F5BE	000000000002F5BE	014D	EncodePointer
000000000002F5CE	000000000002F5CE	0124	DecodePointer
000000000002F5DE	000000000002F5DE	041D	MultiByteToWideChar
000000000002F5F4	000000000002F5F4	0644	WideCharToMultiByte
000000000002F60A	000000000002F60A	03DD	LCMapStringEx
000000000002F61A	000000000002F61A	0302	GetStringTypeW
000000000002F62C	000000000002F62C	01E4	GetCPInfo
000000000002F638	000000000002F638	050F	RtlUnwindEx
000000000002F646	000000000002F646	050B	RtlPcToFileHeader

OFTs	FTs (IAT)	Hint	Name
0002E408	000202E8	0002E846	0002E848
Qword	Qword	Word	szAnsi
000000000002F65A	000000000002F65A	0492	RaiseException
000000000002F66C	000000000002F66C	0287	GetLastError
000000000002F67C	000000000002F67C	0570	SetLastError
000000000002F68C	000000000002F68C	0390	InitializeCriticalSectionAndSpinCou...
000000000002F6B4	000000000002F6B4	05E2	TlsAlloc
000000000002F6C0	000000000002F6C0	05E4	TlsGetValue
000000000002F6CE	000000000002F6CE	05E6	TlsSetValue
000000000002F6DC	000000000002F6DC	05E3	TlsFree
000000000002F6E6	000000000002F6E6	01CE	FreeLibrary
000000000002F6F4	000000000002F6F4	02D7	GetProcAddress
000000000002F706	000000000002F706	03F0	LoadLibraryExW
000000000002F718	000000000002F718	023B	GetCurrentProcess
000000000002F72C	000000000002F72C	05D0	TerminateProcess
000000000002F740	000000000002F740	02FD	GetStdHandle
000000000002F750	000000000002F750	0658	WriteFile
000000000002F75C	000000000002F75C	029B	GetModuleFileNameW
000000000002F772	000000000002F772	0180	ExitProcess
000000000002F780	000000000002F780	029E	GetModuleHandleExW
000000000002F796	000000000002F796	01F9	GetCommandLineA
000000000002F7A8	000000000002F7A8	01FA	GetCommandLineW
000000000002F7BA	000000000002F7BA	037A	HeapFree
000000000002F7C6	000000000002F7C6	0274	GetFileType
000000000002F7D4	000000000002F7D4	0376	HeapAlloc
000000000002F7E0	000000000002F7E0	0197	FindClose
000000000002F7EC	000000000002F7EC	019D	FindFirstFileExW
000000000002F800	000000000002F800	01AE	FindNextFileW
000000000002F810	000000000002F810	03B8	IsValidCodePage
000000000002F822	000000000002F822	01D5	GetACP
000000000002F82C	000000000002F82C	02C0	GetOEMCP
000000000002F838	000000000002F838	025C	GetEnvironmentStringsW

000000000002F852	000000000002F852	01CD	FreeEnvironmentStringsW
000000000002F86C	000000000002F86C	0552	SetEnvironmentVariableW
000000000002F886	000000000002F886	058B	SetStdHandle
000000000002F896	000000000002F896	028B	GetLocaleInfoW
000000000002F8A8	000000000002F8A8	03BA	IsValidLocale
000000000002F8B8	000000000002F8B8	0343	GetUserDefaultLCID
000000000002F8CE	000000000002F8CE	0175	EnumSystemLocalesW
000000000002F8E4	000000000002F8E4	01BC	FlsAlloc
000000000002F8F0	000000000002F8F0	01BE	FlsGetValue
000000000002F8FE	000000000002F8FE	01C0	FlsSetValue
000000000002F90C	000000000002F90C	01BD	FlsFree
000000000002F916	000000000002F916	0612	VirtualProtect
000000000002F928	000000000002F928	00B2	CompareStringW
000000000002F93A	000000000002F93A	03DE	LCMapStringW
000000000002F94A	000000000002F94A	02DE	GetProcessHeap
000000000002F95C	000000000002F95C	009C	CloseHandle
000000000002F96A	000000000002F96A	01C2	FlushFileBuffers
000000000002F97E	000000000002F97E	0223	GetConsoleOutputCP
000000000002F994	000000000002F994	021F	GetConsoleMode
000000000002F9A6	000000000002F9A6	04A3	ReadFile
000000000002F9B2	000000000002F9B2	0272	GetFileSizeEx
000000000002F9C2	000000000002F9C2	0561	SetFilePointerEx
000000000002F9D6	000000000002F9D6	04A0	ReadConsoleW
000000000002F9E6	000000000002F9E6	037D	HeapReAlloc
000000000002F9F4	000000000002F9F4	037F	HeapSize
000000000002FA00	000000000002FA00	00E2	CreateFileW
000000000002FA1E	000000000002FA1E	050E	RtlUnwind

4.4.1 KERNEL32.dll

Off the bat I notice at least one *VERY* interesting function that is commonly used as a direct check for anti-debugging, `IsDebuggerPresent`.

Other functions that caught my eye are the timing functions; `QueryPerformanceCounter`, `GetTickCount`, `GetTickCount64`, `GetSystemTimeAsFileTime`. These aren't necessarily indicative of anything, but *could* be used to support debugger detection logic by performing timing checks.

`GetCurrentProcessId`, `GetStartupInfoW`, and `GetCurrentThreadId` *could* also be used as anti-debug logic to check for certain flags or conditions on the program itself.

`LoadLibraryExW`, `GetProcAddress` and `FreeLibrary` *could* be used to hide libraries/modules by dynamic resolution.

`GetLastError`, `SetLastError`, `RaiseException`, `UnhandledExceptionFilter`, and `SetUnhandledExceptionFilter` *could* all be used in exception based anti-debugging measures.

`IsProcessorFeaturePresent` is also interesting as it *could* be used for certain anti-debug exception tricks.

`VirtualProtect` is often used for unpacking, self-modifying code, patching stubs, and flipping page protections around anti-debug regions.

Some additional functions that are not in the import table from `KERNEL32.dll` that might be worth adding breakpoints to are; `CheckRemoteDebuggerPresent`, `OutputDebugStringA/W`, `NtQueryInformationProcess`

Furthermore, adding breakpoints on `NTDLL.DLL` functions that are used for anti-debug logic just in case; `NtQueryInformationProcess`, `NtSetInformationThread`, and `RtlAddVectoredExceptionHandler` / `RtlRemoveVectoredExceptionHandler`.

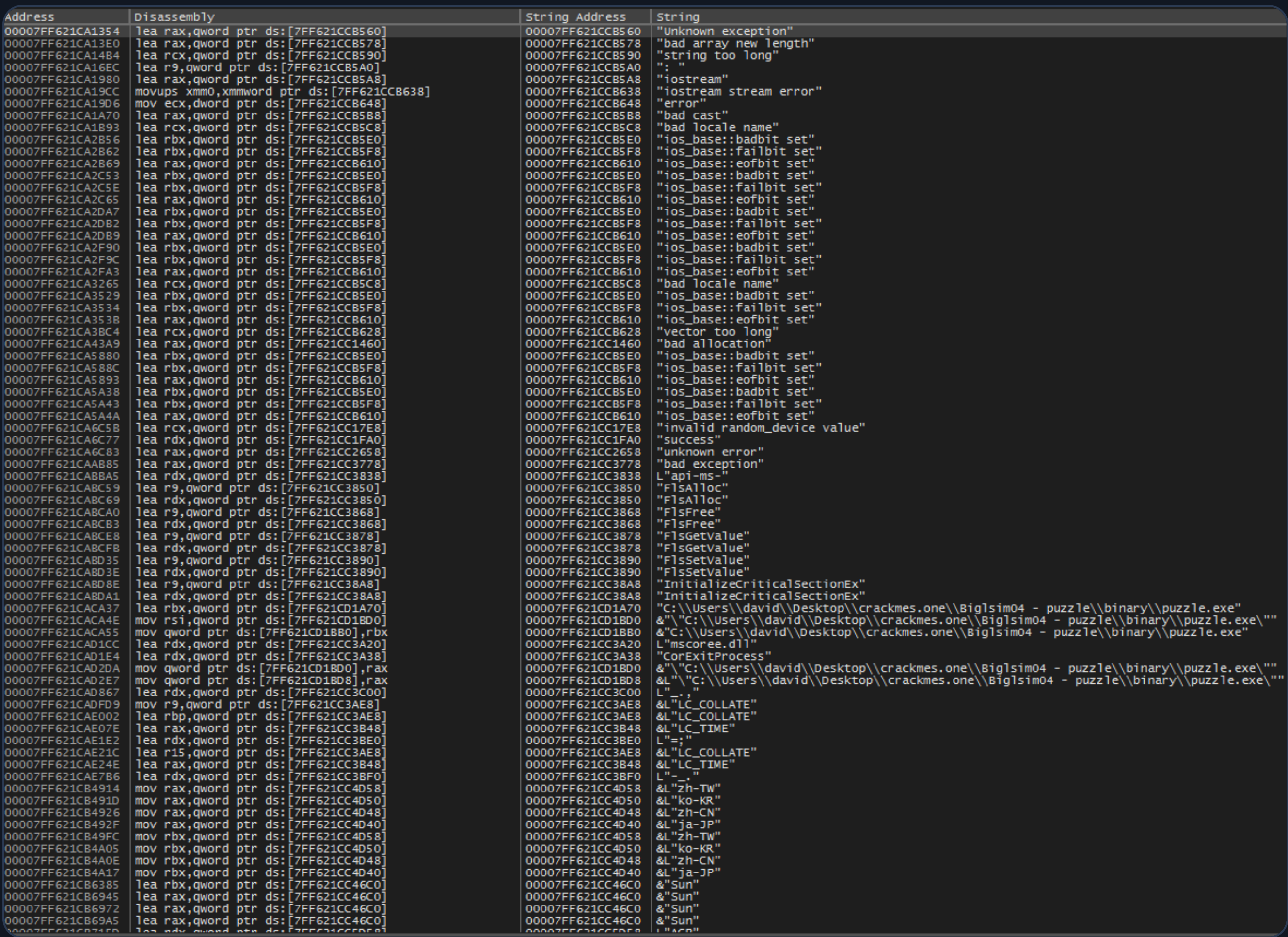
---

5. Dynamic Analysis

5.1 String-Driven Entry

Starting the program in *x64dbg* to see if any immediate anti-debug code triggers yields nothing, yet...

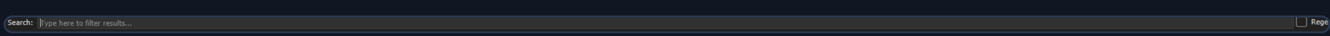
As always, my first point of attack is a string-driven entry. Searching for string references in *x64dbg* yields the following results:
(Specifically looking for strings that I observed during start-up and failure case; `Hello World!`, `Enter password:`, and `Access Denied!`)



Address	Disassembly	String Address	String
00007FF621CA1354	lea rax,qword ptr ds:[7FF621CCB560]	00007FF621CCB560	"Unknown exception"
00007FF621CA13E0	lea rax,qword ptr ds:[7FF621CCB578]	00007FF621CCB578	"bad array new length"
00007FF621CA14B4	lea rcx,qword ptr ds:[7FF621CCB590]	00007FF621CCB590	"string too long"
00007FF621CA16EC	lea r9,qword ptr ds:[7FF621CCB5A0]	00007FF621CCB5A0	": "
00007FF621CA1980	lea rax,qword ptr ds:[7FF621CCB5A8]	00007FF621CCB5A8	"iostream"
00007FF621CA19CC	movups xmm0,xmmword ptr ds:[7FF621CCB638]	00007FF621CCB638	"iostream stream error"
00007FF621CA19D6	mov ecx,dword ptr ds:[7FF621CCB648]	00007FF621CCB648	"error"
00007FF621CA1A70	lea rax,qword ptr ds:[7FF621CCB5B8]	00007FF621CCB5B8	"bad cast"
00007FF621CA1B93	lea rcx,qword ptr ds:[7FF621CCB5C8]	00007FF621CCB5C8	"bad locale name"
00007FF621CA2B56	lea rbx,qword ptr ds:[7FF621CCB5E0]	00007FF621CCB5E0	"ios_base::badbit set"
00007FF621CA2B62	lea rbx,qword ptr ds:[7FF621CCB5F8]	00007FF621CCB5F8	"ios_base::failbit set"
00007FF621CA2B69	lea rax,qword ptr ds:[7FF621CCB610]	00007FF621CCB610	"ios_base::eofbit set"
00007FF621CA2C53	lea rbx,qword ptr ds:[7FF621CCB5E0]	00007FF621CCB5E0	"ios_base::badbit set"
00007FF621CA2C5E	lea rbx,qword ptr ds:[7FF621CCB5F8]	00007FF621CCB5F8	"ios_base::failbit set"
00007FF621CA2C65	lea rax,qword ptr ds:[7FF621CCB610]	00007FF621CCB610	"ios_base::eofbit set"
00007FF621CA2DA7	lea rbx,qword ptr ds:[7FF621CCB5E0]	00007FF621CCB5E0	"ios_base::badbit set"
00007FF621CA2DB2	lea rbx,qword ptr ds:[7FF621CCB5F8]	00007FF621CCB5F8	"ios_base::failbit set"
00007FF621CA2DB9	lea rax,qword ptr ds:[7FF621CCB610]	00007FF621CCB610	"ios_base::eofbit set"
00007FF621CA2F90	lea rbx,qword ptr ds:[7FF621CCB5E0]	00007FF621CCB5E0	"ios_base::badbit set"
00007FF621CA2F9C	lea rbx,qword ptr ds:[7FF621CCB5F8]	00007FF621CCB5F8	"ios_base::failbit set"
00007FF621CA2FA3	lea rax,qword ptr ds:[7FF621CCB610]	00007FF621CCB610	"ios_base::eofbit set"
00007FF621CA3265	lea rcx,qword ptr ds:[7FF621CCB5C8]	00007FF621CCB5C8	"bad locale name"
00007FF621CA3529	lea rbx,qword ptr ds:[7FF621CCB5E0]	00007FF621CCB5E0	"ios_base::badbit set"
00007FF621CA3534	lea rbx,qword ptr ds:[7FF621CCB5F8]	00007FF621CCB5F8	"ios_base::failbit set"
00007FF621CA353B	lea rax,qword ptr ds:[7FF621CCB610]	00007FF621CCB610	"ios_base::eofbit set"
00007FF621CA3BC4	lea rcx,qword ptr ds:[7FF621CCB628]	00007FF621CCB628	"vector too long"
00007FF621CA43A9	lea rax,qword ptr ds:[7FF621CC1460]	00007FF621CC1460	"bad allocation"
00007FF621CA5880	lea rbx,qword ptr ds:[7FF621CCB5E0]	00007FF621CCB5E0	"ios_base::badbit set"
00007FF621CA588C	lea rbx,qword ptr ds:[7FF621CCB5F8]	00007FF621CCB5F8	"ios_base::failbit set"
00007FF621CA5893	lea rax,qword ptr ds:[7FF621CCB610]	00007FF621CCB610	"ios_base::eofbit set"
00007FF621CA5A38	lea rbx,qword ptr ds:[7FF621CCB5E0]	00007FF621CCB5E0	"ios_base::badbit set"
00007FF621CA5A43	lea rbx,qword ptr ds:[7FF621CCB5F8]	00007FF621CCB5F8	"ios_base::failbit set"
00007FF621CA5A4A	lea rax,qword ptr ds:[7FF621CCB610]	00007FF621CCB610	"ios_base::eofbit set"
00007FF621CA6C5B	lea rcx,qword ptr ds:[7FF621CC17E8]	00007FF621CC17E8	"invalid random_device value"
00007FF621CA6C77	lea rdx,qword ptr ds:[7FF621CC1FA0]	00007FF621CC1FA0	"success"
00007FF621CA6C83	lea rax,qword ptr ds:[7FF621CC2658]	00007FF621CC2658	"unknown error"
00007FF621CAAB85	lea rax,qword ptr ds:[7FF621CC3778]	00007FF621CC3778	"bad exception"
00007FF621CABBA5	lea rdx,qword ptr ds:[7FF621CC3838]	00007FF621CC3838	L"api-ms-"
00007FF621CABC59	lea r9,qword ptr ds:[7FF621CC3850]	00007FF621CC3850	"FlsAlloc"
00007FF621CABC69	lea rdx,qword ptr ds:[7FF621CC3850]	00007FF621CC3850	"FlsAlloc"
00007FF621CABCA0	lea r9,qword ptr ds:[7FF621CC3868]	00007FF621CC3868	"FlsFree"
00007FF621CABCB3	lea rdx,qword ptr ds:[7FF621CC3868]	00007FF621CC3868	"FlsFree"
00007FF621CABCE8	lea r9,qword ptr ds:[7FF621CC3878]	00007FF621CC3878	"FlsGetValue"
00007FF621CABCFB	lea rdx,qword ptr ds:[7FF621CC3878]	00007FF621CC3878	"FlsGetValue"
00007FF621CABD35	lea r9,qword ptr ds:[7FF621CC3890]	00007FF621CC3890	"FlsSetValue"
00007FF621CABD3E	lea rdx,qword ptr ds:[7FF621CC3890]	00007FF621CC3890	"FlsSetValue"
00007FF621CABD8E	lea r9,qword ptr ds:[7FF621CC38A8]	00007FF621CC38A8	"InitializeCriticalSectionEx"
00007FF621CABDA1	lea rdx,qword ptr ds:[7FF621CC38A8]	00007FF621CC38A8	"InitializeCriticalSectionEx"
00007FF621CACA37	lea rbx,qword ptr ds:[7FF621CD1A70]	00007FF621CD1A70	"C:\\Users\\david\\Desktop\\crackmes.one\\Biglsim04 - puzzle\\binary\\puzzle.exe"
00007FF621CACA4E	mov rsi,qword ptr ds:[7FF621CD1BD0]	00007FF621CD1BD0	&"\"C:\\Users\\david\\Desktop\\crackmes.one\\Biglsim04 - puzzle\\binary\\puzzle.exe\""
00007FF621CACA55	mov qword ptr ds:[7FF621CD1BB0],rbx	00007FF621CD1BB0	&"C:\\Users\\david\\Desktop\\crackmes.one\\Biglsim04 - puzzle\\binary\\puzzle.exe"
00007FF621CAD1CC	lea rdx,qword ptr ds:[7FF621CC3A20]	00007FF621CC3A20	L"mscoree.dll"
00007FF621CAD1E4	lea rdx,qword ptr ds:[7FF621CC3A38]	00007FF621CC3A38	"CorExitProcess"
00007FF621CAD2DA	mov qword ptr ds:[7FF621CD1BD0],rax	00007FF621CD1BD0	&"\"C:\\Users\\david\\Desktop\\crackmes.one\\Biglsim04 - puzzle\\binary\\puzzle.exe\""
00007FF621CAD2E7	mov qword ptr ds:[7FF621CD1BD8],rax	00007FF621CD1BD8	&L"\"C:\\Users\\david\\Desktop\\crackmes.one\\Biglsim04 - puzzle\\binary\\puzzle.exe\""
00007FF621CAD867	lea rdx,qword ptr ds:[7FF621CC3C00]	00007FF621CC3C00	L"_.,"
00007FF621CADFD9	mov r9,qword ptr ds:[7FF621CC3AE8]	00007FF621CC3AE8	&L"LC_COLLATE"
00007FF621CAE002	lea rbp,qword ptr ds:[7FF621CC3AE8]	00007FF621CC3AE8	&L"LC_COLLATE"
00007FF621CAE07E	lea rax,qword ptr ds:[7FF621CC3B48]	00007FF621CC3B48	&L"LC_TIME"
00007FF621CAE1E2	lea rdx,qword ptr ds:[7FF621CC3BE0]	00007FF621CC3BE0	L"=;"
00007FF621CAE21C	lea r15,qword ptr ds:[7FF621CC3AE8]	00007FF621CC3AE8	&L"LC_COLLATE"
00007FF621CAE24E	lea rax,qword ptr ds:[7FF621CC3B48]	00007FF621CC3B48	&L"LC_TIME"
00007FF621CAE7B6	lea rdx,qword ptr ds:[7FF621CC3BF0]	00007FF621CC3BF0	L"-_."
00007FF621CB4914	mov rax,qword ptr ds:[7FF621CC4D58]	00007FF621CC4D58	&L"zh-TW"
00007FF621CB491D	mov rax,qword ptr ds:[7FF621CC4D50]	00007FF621CC4D50	&L"ko-KR"
00007FF621CB4926	mov rax,qword ptr ds:[7FF621CC4D48]	00007FF621CC4D48	&L"zh-CN"
00007FF621CB492F	mov rax,qword ptr ds:[7FF621CC4D40]	00007FF621CC4D40	&L"ja-JP"
00007FF621CB49FC	mov rbx,qword ptr ds:[7FF621CC4D58]	00007FF621CC4D58	&L"zh-TW"
00007FF621CB4A05	mov rbx,qword ptr ds:[7FF621CC4D50]	00007FF621CC4D50	&L"ko-KR"
00007FF621CB4A0E	mov rbx,qword ptr ds:[7FF621CC4D48]	00007FF621CC4D48	&L"zh-CN"
00007FF621CB4A17	mov rbx,qword ptr ds:[7FF621CC4D40]	00007FF621CC4D40	&L"ja-JP"
00007FF621CB6385	lea rbx,qword ptr ds:[7FF621CC46C0]	00007FF621CC46C0	&"Sun"
00007FF621CB6945	lea rax,qword ptr ds:[7FF621CC46C0]	00007FF621CC46C0	&"Sun"
00007FF621CB6972	lea rax,qword ptr ds:[7FF621CC46C0]	00007FF621CC46C0	&"Sun"
00007FF621CB69A5	lea rax,qword ptr ds:[7FF621CC46C0]	00007FF621CC46C0	&"Sun"

00007FF621CB715D	lea rdx,qword ptr ds:[7FF621CC5D58]	00007FF621CC5D58	L"ACP"
00007FF621CB716D	lea rdx,qword ptr ds:[7FF621CC5D48]	00007FF621CC5D48	L"utf8"
00007FF621CB7180	lea rdx,qword ptr ds:[7FF621CC5D60]	00007FF621CC5D60	L"utf-8"
00007FF621CB7193	lea rdx,qword ptr ds:[7FF621CC5D70]	00007FF621CC5D70	L"OCP"
00007FF621CB736B	lea rcx,qword ptr ds:[7FF621CC51C0]	00007FF621CC51C0	&L"america"
00007FF621CB73A4	lea rcx,qword ptr ds:[7FF621CC4DA0]	00007FF621CC4DA0	&L"american"
00007FF621CB7511	lea r8,qword ptr ds:[7FF621CC5D48]	00007FF621CC5D48	L"utf8"
00007FF621CB7B8D	lea rdx,qword ptr ds:[7FF621CC5D58]	00007FF621CC5D58	L"ACP"
00007FF621CB7B9D	lea rdx,qword ptr ds:[7FF621CC5D70]	00007FF621CC5D70	L"OCP"
00007FF621CB7DD7	lea rcx,qword ptr ds:[7FF621CC51C0]	00007FF621CC51C0	&L"america"
00007FF621CB7E3A	lea rcx,qword ptr ds:[7FF621CC4DA0]	00007FF621CC4DA0	&L"american"
00007FF621CB8069	lea r9,qword ptr ds:[7FF621CC6398]	00007FF621CC6398	"CompareStringEx"
00007FF621CB807C	lea rdx,qword ptr ds:[7FF621CC6398]	00007FF621CC6398	"CompareStringEx"
00007FF621CB811B	lea rdx,qword ptr ds:[7FF621CC3838]	00007FF621CC3838	L"api-ms-"
00007FF621CB812E	lea rdx,qword ptr ds:[7FF621CC6368]	00007FF621CC6368	L"ext-ms-"
00007FF621CB8257	lea r9,qword ptr ds:[7FF621CC64C8]	00007FF621CC64C8	"AppPolicyGetProcessTerminationMethod"
00007FF621CB8265	lea rdx,qword ptr ds:[7FF621CC64C8]	00007FF621CC64C8	"AppPolicyGetProcessTerminationMethod"
00007FF621CB82BE	lea rdx,qword ptr ds:[7FF621CC6380]	00007FF621CC6380	"AreFileApisANSI"
00007FF621CB83D6	lea r9,qword ptr ds:[7FF621CC63B0]	00007FF621CC63B0	"EnumSystemLocalesEx"
00007FF621CB83E4	lea rdx,qword ptr ds:[7FF621CC63B0]	00007FF621CC63B0	"EnumSystemLocalesEx"
00007FF621CB84A9	lea r9,qword ptr ds:[7FF621CC6400]	00007FF621CC6400	"GetLocaleInfoEx"
00007FF621CB84B7	lea rdx,qword ptr ds:[7FF621CC6400]	00007FF621CC6400	"GetLocaleInfoEx"
00007FF621CB852D	lea r9,qword ptr ds:[7FF621CC6430]	00007FF621CC6430	"GetUserDefaultLocaleName"
00007FF621CB853B	lea rdx,qword ptr ds:[7FF621CC6430]	00007FF621CC6430	"GetUserDefaultLocaleName"
00007FF621CB859F	lea r9,qword ptr ds:[7FF621CC6458]	00007FF621CC6458	"IsValidLocaleName"
00007FF621CB85AD	lea rdx,qword ptr ds:[7FF621CC6458]	00007FF621CC6458	"IsValidLocaleName"
00007FF621CB861D	lea r9,qword ptr ds:[7FF621CC6490]	00007FF621CC6490	"LCIDToLocaleName"
00007FF621CB862B	lea rdx,qword ptr ds:[7FF621CC6490]	00007FF621CC6490	"LCIDToLocaleName"
00007FF621CB86A5	lea r9,qword ptr ds:[7FF621CC6478]	00007FF621CC6478	"LCMapStringEx"
00007FF621CB86B3	lea rdx,qword ptr ds:[7FF621CC6478]	00007FF621CC6478	"LCMapStringEx"
00007FF621CB8789	lea r9,qword ptr ds:[7FF621CC64B0]	00007FF621CC64B0	"LocaleNameToLCID"
00007FF621CB8797	lea rdx,qword ptr ds:[7FF621CC64B0]	00007FF621CC64B0	"LocaleNameToLCID"
00007FF621CB87ED	lea r9,qword ptr ds:[7FF621CC64F8]	00007FF621CC64F8	"SystemFunction036"
00007FF621CB87FB	lea rdx,qword ptr ds:[7FF621CC64F8]	00007FF621CC64F8	"SystemFunction036"
00007FF621CB8860	lea rdx,qword ptr ds:[7FF621CC6380]	00007FF621CC6380	"AreFileApisANSI"
00007FF621CB887D	lea r9,qword ptr ds:[7FF621CC63B0]	00007FF621CC63B0	"EnumSystemLocalesEx"
00007FF621CB888B	lea rdx,qword ptr ds:[7FF621CC63B0]	00007FF621CC63B0	"EnumSystemLocalesEx"
00007FF621CB88A6	lea r9,qword ptr ds:[7FF621CC63E8]	00007FF621CC63E8	"GetDateFormatEx"
00007FF621CB88B4	lea rdx,qword ptr ds:[7FF621CC63E8]	00007FF621CC63E8	"GetDateFormatEx"
00007FF621CB88CF	lea r9,qword ptr ds:[7FF621CC6400]	00007FF621CC6400	"GetLocaleInfoEx"
00007FF621CB88DD	lea rdx,qword ptr ds:[7FF621CC6400]	00007FF621CC6400	"GetLocaleInfoEx"
00007FF621CB88F8	lea r9,qword ptr ds:[7FF621CC6418]	00007FF621CC6418	"GetTimeFormatEx"
00007FF621CB8906	lea rdx,qword ptr ds:[7FF621CC6418]	00007FF621CC6418	"GetTimeFormatEx"
00007FF621CB8921	lea r9,qword ptr ds:[7FF621CC6430]	00007FF621CC6430	"GetUserDefaultLocaleName"
00007FF621CB892F	lea rdx,qword ptr ds:[7FF621CC6430]	00007FF621CC6430	"GetUserDefaultLocaleName"
00007FF621CB894A	lea r9,qword ptr ds:[7FF621CC6458]	00007FF621CC6458	"IsValidLocaleName"
00007FF621CB8958	lea rdx,qword ptr ds:[7FF621CC6458]	00007FF621CC6458	"IsValidLocaleName"
00007FF621CB8973	lea r9,qword ptr ds:[7FF621CC6478]	00007FF621CC6478	"LCMapStringEx"
00007FF621CB8981	lea rdx,qword ptr ds:[7FF621CC6478]	00007FF621CC6478	"LCMapStringEx"
00007FF621CB899C	lea r9,qword ptr ds:[7FF621CC6490]	00007FF621CC6490	"LCIDToLocaleName"
00007FF621CB89AA	lea rdx,qword ptr ds:[7FF621CC6490]	00007FF621CC6490	"LCIDToLocaleName"
00007FF621CB89C5	lea r9,qword ptr ds:[7FF621CC64B0]	00007FF621CC64B0	"LocaleNameToLCID"
00007FF621CB89D3	lea rdx,qword ptr ds:[7FF621CC64B0]	00007FF621CC64B0	"LocaleNameToLCID"
00007FF621CB8A32	lea r9,qword ptr ds:[7FF621CC63D0]	00007FF621CC63D0	"FlsGetValue2"
00007FF621CB8A40	lea rdx,qword ptr ds:[7FF621CC63D0]	00007FF621CC63D0	"FlsGetValue2"
00007FF621CBD248	lea rax,qword ptr ds:[7FF621CD0C30]	00007FF621CD0C30	&"PST"
00007FF621CBD256	lea rax,qword ptr ds:[7FF621CD0C40]	00007FF621CD0C40	&L"PST"
00007FF621CBED42	lea rcx,qword ptr ds:[7FF621CC9848]	00007FF621CC9848	L"CONOUT$"
00007FF621CBEDF1	lea rcx,qword ptr ds:[7FF621CC9848]	00007FF621CC9848	L"CONOUT$"
00007FF621CC101A	or ch,byte ptr ds:[7FF621CC901C]	00007FF621CC901C	L"-gr"
00007FF621CC104A	or ebp,dword ptr ds:[7FF621CC904C]	00007FF621CC904C	L"-ca"
00007FF621CC107A	or ebp,dword ptr ds:[7FF621CC907C]	00007FF621CC907C	L"-ie"
00007FF621CC10BA	or ebp,dword ptr ds:[7FF621CC90BC]	00007FF621CC90BC	L"-tt"
00007FF621CC112A	or ebp,dword ptr ds:[7FF621CC912C]	00007FF621CC912C	L"-co"
00007FF621CC113A	or ebp,dword ptr ds:[7FF621CC913C]	00007FF621CC913C	L"-cr"
00007FF621CC114A	or ch,byte ptr ds:[7FF621CC914C]	00007FF621CC914C	L"-do"
00007FF621CC115A	or ebp,dword ptr ds:[7FF621CC915C]	00007FF621CC915C	L"-ec"
00007FF621CC11FA	or ebp,dword ptr ds:[7FF621CC91FC]	00007FF621CC91FC	L"-sv"
00007FF621CC121A	or ebp,dword ptr ds:[7FF621CC921C]	00007FF621CC921C	L"-ve"
00007FF621CC123A	or ebp,dword ptr ds:[7FF621CC923C]	00007FF621CC923C	L"-es"

That's a lot of references! Utilizing the search functionality at the bottom of the `References` tab will help make searching for the desired string references a breeze.



Nada! Well that's a first for me, never before have I had it where there are zero string references found. Something new is always interesting!

Time to switch to a breakpoint approach.

5.2 Break-it down-point Time

5.2.1 Anti-Debugging Breakpoints

Before I start adding breakpoints trying to trace any flag related logic, I first want to see where and how some of the functions for anti-debugging measures are being used.

Function	Reason for Interest
`IsDebuggerPresent`	The simplest direct debugger check; breaking here often shows the exact branch that decides the *good* vs *bad* branch paths.
`SetUnhandledExceptionFilter`	Programs use this to install custom crash/exception handling; it's commonly part of exception-based anti-debug tricks.
`UnhandledExceptionFilter`	Often hit when the program deliberately triggers an exception; breaking here helps you see whether the exception flow is being used as a debugger test.
`RaiseException`	A strong indicator of intentional exception-based detection; it usually marks the start of an anti-debug probe.
`QueryPerformanceCounter`	Used for high-resolution timing checks; stepping/breakpoints can cause delays that the program detects.
`GetTickCount`, `GetTickCount64`	Lower-resolution timing checks; still commonly used to detect "debugger slowdowns" around sensitive code blocks.

Function	Reason for Interest
`VirtualProtect`	Frequently used for unpacking or self-modifying anti-debug stubs; breaking here can lead you to the real code being revealed or patched in memory.
`GetProcAddress`	Shows when the binary dynamically resolves hidden anti-debug APIs (often from `ntdll`); the requested function name is a big giveaway.
`LoadLibraryExW`	Often paired with `GetProcAddress` to pull in `ntdll` / `user32` at runtime; breaking here can expose the moment advanced anti-debug tooling gets loaded.

For those that are following along, here is an *x64dbg* command to add all these breakpoints:

```
bp kernel32.IsDebuggerPresent; bp kernel32.SetUnhandledExceptionFilter; bp
kernel32.UnhandledExceptionFilter; bp kernel32.RaiseException; bp
kernel32.QueryPerformanceCounter;  bp kernel32.GetTickCount;  bp
kernel32.GetTickCount64;  bp kernel32.VirtualProtect;  bp
kernel32.GetProcAddress;  bp kernel32.LoadLibraryExW
```

See Anti-Debugging Breakpoints for a more detailed breakpoint breakdown and logic tracing.

5.2.2 Input Breakpoints

After, I decide to start with breakpoints that might be used for obtaining the user input from the console;

Function	Reason for Interest
`ReadConsoleA/W`	Catches direct keyboard input from the console,can see exactly where the program reads the name/serial and what buffer it lands in.
`WriteConsoleA/W`	Hits when the program prints prompts or messages; stepping right after often leads straight into the input and validation flow.
`ReadFile`	Many console apps read STDIN via a handle as if it were a file, so this is a reliable fallback when `ReadConsoleA/W` isn't used.
`WriteFile`	Console output is sometimes routed through file-style writes, so it helps catch prompts and trace the execution path around user interaction.
`GetStdHandle`	Usually called right before `ReadConsoleA/W` / `ReadFile` or output calls, so it's a great "early warning" breakpoint for the I/O path.
`GetCommandLineA/W`	Useful when input is passed as command-line args; you can see raw input early before it gets parsed or transformed. Doesn't seem necessary for this CTF as it doesn't appear to use command line arguements, although it does not hurt to add it.
`GetProcAddress`	Reveals dynamically resolved APIs (often hidden checks or CRT - C Runtime - calls); the requested function name can instantly expose the program's real strategy.

For those that are following along, here is an *x64dbg* command to add all these breakpoints:

```
bp kernel32.ReadConsoleW; bp kernel32.ReadConsoleA; bp kernel32.WriteConsoleW; bp kernel32.WriteConsoleA; bp kernel32.ReadFile; bp kernel32.WriteFile; bp kernel32.GetStdHandle; bp kernel32.GetCommandLineA; bp kernel32.GetCommandLineW; bp kernel32.GetProcAddress
```

See Input Breakpoints for a more detailed breakpoint breakdown and logic tracing.

---

6. Dynamic Analysis - Tracing Breakpoints and Stepping Over Logic

See Windows x64 Calling Convention for a quick refresher on Windows x64 calling convention.

6.1 Anti-Debugging Breakpoints

With the new breakpoints added, I resume program execution from the entry breakpoint.

Address	Module/Label/Exception	State	Disassembly	Hits
00007FFDA1B72B00	<kernel32.dll.GetTickCount64>	Enabled	mov ecx,dword ptr ds:[7FFE0004]	1
00007FFDA1B78600	<kernel32.dll.GetTickCount>	Disabled	mov ecx,7FFE0320	27
00007FFDA1B83FB0	<kernel32.dll.QueryPerformanceCounter>	Enabled	jmp qword ptr ds:[<QueryPerformanceCounter>]	0
00007FFDA1B93C70	<kernel32.dll.GetProcAddress>	Enabled	mov r8,qword ptr ss:[rsp]	17
00007FFDA1B982B0	<kernel32.dll.VirtualProtect>	Enabled	jmp qword ptr ds:[<VirtualProtect>]	27
00007FFDA1B9B110	<kernel32.dll.RaiseException>	Enabled	jmp qword ptr ds:[<RaiseException>]	0
00007FFDA1B9C600	<kernel32.dll.LoadLibraryExW>	Enabled	jmp qword ptr ds:[<LoadLibraryExW>]	9
00007FFDA1B9D9A0	<kernel32.dll.IsDebuggerPresent>	Enabled	jmp qword ptr ds:[<IsDebuggerPresent>]	0
00007FFDA1BA3600	<kernel32.dll.SetUnhandledExceptionFilter>	Enabled	jmp qword ptr ds:[<SetUnhandledExceptionFilter>]	1
00007FFDA1BBBE60	<kernel32.dll.UnhandledExceptionFilter>	Enabled	jmp qword ptr ds:[<UnhandledExceptionFilter>]	0

I disabled the `GetTickCount` breakpoint as it was getting triggered on each frame, instead replacing it with a breakpoint in the caller that I hope will bring me closer to the flag comparison logic.

There seems to be more going on than a simple console checker. `GetProcAddress` (x17) + `LoadLibraryExW` (x9) on start-up shows that the binary is *OR* might-be keeping the static import table small/boring and resolving lots of APIs at runtime.

I also noticed that `IsDebuggerPresent` breakpoint never gets triggered, even upon input validation.

6.1.1 <kernel32.dll.LoadLibraryExW>

See [LoadLibraryExW Function Definition] for function definition details.

Switching over to the *Call Stack* tab I can see that it is being directly called by the *PE*. This means that the target binary is the one actually making the call to `LoadLibraryExW` and not some other module/code.

Address	To	From	Size	Party	Comment
000000B6B374F908	00007FF621CABB8B	00007FFC2D0BC600	8	User	kernel32.LoadLibraryExW
000000B6B374F910	0000000000000000	00007FF621CABB8B		User	puzzle.00007FF621CABB8B

Continuing the execution into `KERNEL32.DLL.LoadLibraryExW` I see that the following registers have the values:

- **Call #1** - Most likely *normal OS dependency resolution* with a *safe flag* restricting search to *System32*.

Register	Value	Note
RCX	00007FF685B937E0	L"api-ms-win-core-synch-l1-2-0"
RDX	0000000000000000	
R8	0000000000000800	`LOAD_LIBRARY_SEARCH_SYSTEM32` = 0x00000800

- **Call #2** - Most likely *normal OS dependency resolution* with a *safe flag* restricting search to *System32*.

Register	Value	Note
RCX	00007FF685B937A0	L"api-ms-win-core-fibers-l1-1-1"
RDX	0000000000000000	
R8	0000000000000800	`LOAD_LIBRARY_SEARCH_SYSTEM32` = 0x00000800

- **Call #3** - Most likely *normal OS dependency resolution* with a *safe flag* restricting search to *System32*.

Register	Value	Note
RCX	00007FF685B95E90	L"api-ms-win-core-fibers-l1-1-2"
RDX	0000000000000000	
R8	0000000000000800	`LOAD_LIBRARY_SEARCH_SYSTEM32` = 0x00000800

- **Call #4** - Most likely *normal OS dependency resolution* with a *safe flag* restricting search to *System32*.

Register	Value	Note
RCX	00007FF685B95F80	L"api-ms-win-core-localization-l1-2-1"
RDX	0000000000000000	
R8	0000000000000800	`LOAD_LIBRARY_SEARCH_SYSTEM32` = 0x00000800

- **Call #5** - Likely normal runtime/loader behaviour.

Register	Value	Note
RCX	00007FF685B93820	L"kernel32"
RDX	0000000000000000	
R8	0000000000000800	`LOAD_LIBRARY_SEARCH_SYSTEM32` = 0x00000800

- **Call #6** - Most likely *normal OS dependency resolution* with a *safe flag* restricting search to *System32*.

Register	Value	Note
RCX	00007FF685B96080	L"api-ms-win-core-string-l1-1-0"

Register	Value	Note
RDX	0000000000000000	
R8	0000000000000800	`LOAD_LIBRARY_SEARCH_SYSTEM32` = 0x00000800

- **Call #7** - Most likely *normal OS dependency resolution* with a *safe flag* restricting search to *System32*.

Register	Value	Note
RCX	00007FF685B95E50	L"api-ms-win-core-datetime-l1-1-1"
RDX	0000000000000000	
R8	0000000000000800	`LOAD_LIBRARY_SEARCH_SYSTEM32` = 0x00000800

- **Call #8** - Most likely *normal OS dependency resolution* with a *safe flag* restricting search to *System32*.

Register	Value	Note
RCX	00007FF685B95FD0	L"api-ms-win-core-localization-obsolete-l1-2-0"
RDX	0000000000000000	
R8	0000000000000800	`LOAD_LIBRARY_SEARCH_SYSTEM32` = 0x00000800

- **Call #9** - Most likely *normal OS dependency resolution* with a *safe flag* restricting search to *System32*.

Register	Value	Note
RCX	00007FF685B961D0	L"api-ms-win-security-systemfunctions-l1-1-0"

Register	Value	Note
RDX	0000000000000000	
R8	0000000000000800	`LOAD_LIBRARY_SEARCH_SYSTEM32` = 0x00000800

The binary consistently restricts DLL search to System32 during early initialization which aligns with modern safe-loading practices and reduces the likelihood of DLL search-order hijacking. The `api-ms-win-*` entries reflect Windows API-set indirection. Their presence here is typical for modern MSVC builds and does not by itself indicate obfuscation. None of the observed `LoadLibraryExW` function calls directly load `ntdll.dll` or other modules - `user32.dll`, `dbghelp.dll` - commonly associated with advanced anti-debug checks. The initial loads appear consistent with baseline OS/runtime dependencies.

6.1.2 <kernel32.dll.SetUnhandledExceptionFilter>

See SetUnhandledExceptionFilter Function Definition for function definition details.

Non-Interesting Breaks

Following the first uninteresting break on `LoadLibraryExW`, followed more uninteresting breaks, starting with; `GetProcAddress`, another `LoadLibraryExW`, `GetProcAddress`, `GetProcAddress`, `VirtualProtect`, `LoadLibraryExW`, `GetProcAddress`, `VirtualProtect`, `VirtualProtect`, `LoadLibraryExW`, `GetProcAddress`, `VirtualProtect`

6.2 Input Breakpoints

---

7. Validation Path

---

8. Useful Notes and Reminders

8.1 Windows x64 Calling Convention

On Windows x64 calling convention:

- RCX = 1st parameter
- RDX = 2nd
- R8 = 3rd
- R9 = 4th
- RAX = return value
- If there are *more than four arguements*, the rest go on the *stack*.

8.1.1 Volatile & Non-Volatile registers

Volatile (caller-saved): `RAX, RCX, RDX, R8-R11`

If you're tracking values across calls, expect volatile regs to get clobbered.

Non-volatile (callee-saved): `RBX, RBP, RSI, RDI, R12-R15`

8.1.2 Shadow Space

The caller reserves 32 bytes of *shadow space* on the stack before the call. So even if a function has fewer than 4 parameters, you'll still see that stack layout pattern.

8.2 Function Definitions

8.2.1 KERNEL32.dll.LoadLibraryExW

`LoadLibraryExW` has three parameters and returns an *HMODULE* (or *NULL* on failure).

```
HMODULE LoadLibraryExW(
  LPCWSTR lpLibFileName,
  HANDLE  hFile,
  DWORD   dwFlags
);
```

1. **lpLibFileName** (`LPCWSTR`)
    - Path or name of the DLL to load.
    - Often something like:
        - `L"kernel32.dll"`
        - `L"C:\\Windows\\System32\\something.dll"`
        - Or an app-local DLL name.
2. **hFile** (`HANDLE`)
    - Usually **NULL**.
    - Historically used for loading from an already-open file handle.
3. **dwFlags** (`DWORD`)
    - Controls *how* the module is loaded / searched.
    - Common ones include:

Flag	Value	Note
	`0x00000000`	Default load behaviour (normal DLL search order).

Flag	Value	Note
`DONT_RESOLVE_DLL_REFERENCES`	`0x00000001`	Maps the DLL but *doesn't call* `DllMain` or resolve imports - useful for inspection.
`LOAD_LIBRARY_AS_DATAFILE`	`0x00000002`	Loads the module *as a data file* (resources), not for code execution.
`LOAD_LIBRARY_AS_IMAGE_RESOURCE`	`0x00000020`	Loads *only as an image resource*, mostly for resource access.
`LOAD_LIBRARY_AS_DATAFILE_EXCLUSIVE`	`0x00000040`	Like `AS_DATAFILE` but tries to keep it exclusive so others can't modify it.
`LOAD_WITH_ALTERED_SEARCH_PATH`	`0x00000008`	Changes search order to prioritize the DLL's directory - older/legacy pattern.
`LOAD_LIBRARY_SEARCH_DLL_LOAD_DIR`	`0x00000100`	Search the directory of the DLL being loaded
`LOAD_LIBRARY_SEARCH_APPLICATION_DIR`	`0x00000200`	Search the executable's directory

Flag	Value	Note
`LOAD_LIBRARY_SEARCH_USER_DIRS`	`0x00000400`	Search directories added via `AddDllDirectory`
`LOAD_LIBRARY_SEARCH_SYSTEM32`	`0x00000800`	Search `System32` only
`LOAD_LIBRARY_SEARCH_DEFAULT_DIRS`	`0x00001000`	A safe default set: app directory + system32 + user-added directories (recommended modern choice).

If a weird flag value is present, it may be a *bitwise OR* of multiple flags.

- **Return Value**
  - Success: `HMODULE` for the loaded module.
  - Failure: **NULL** (`RAX = 0`).
    - Then `GetLastError()` can inform you as to why.

8.2.2 KERNEL32.dll.SetUnhandledExceptionFilter

9.

---

10. Conclusion

- Summary of final understanding.
- What you’d improve next time.
- Optional lessons learned.