Steam and freeware Japanese shit [Part 1].

September 21, 2016

"Romance of the Three Kingdoms Maker" .

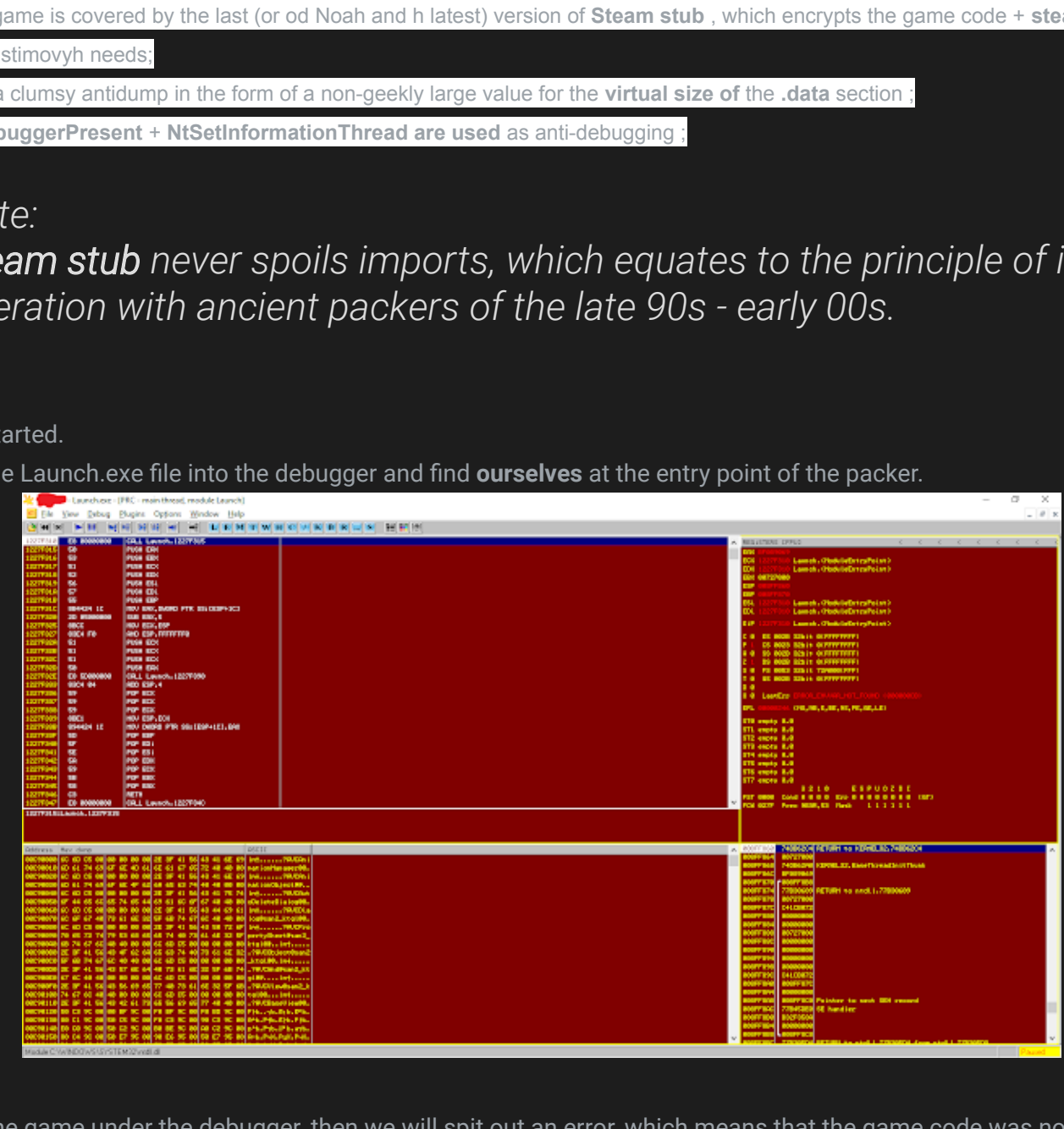
This game is added in the form of a free license to the account, so if you don't fucking get it, then don't fucking add it.

I will immediately clarify a few important points:

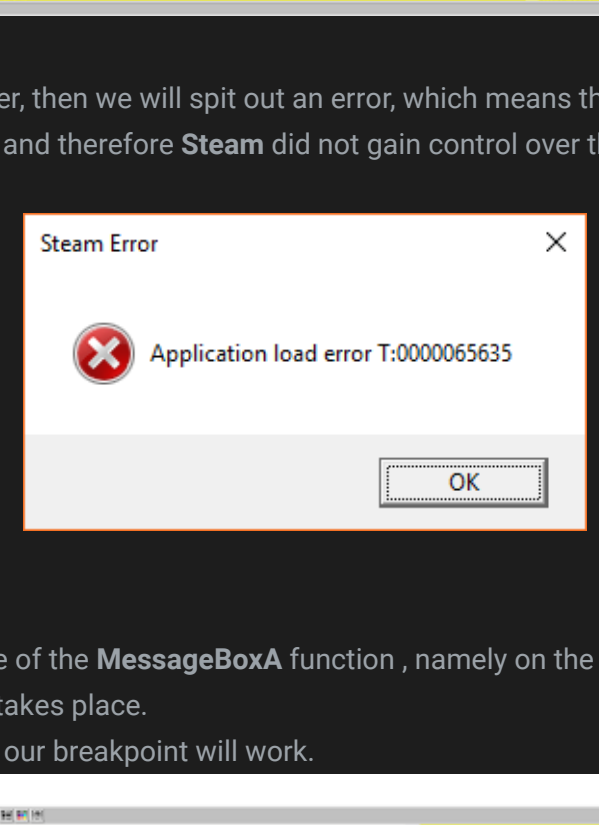
1. The game is covered by the last (or od Noah and h latest) version of **Steam stub** , which encrypts the game code + **steam api** for vnutristimovyh needs;
2. Has a clumsy antidump in the form of a non-geekly large value for the **virtual size of** the **.data** section ;
3. **IsDebuggerPresent** + **NtSetInformationThread are used** as anti-debugging ;

> *Note:*
> **Steam stub** *never spoils imports, which equates to the principle of its operation with ancient packers of the late 90s - early 00s.*

Let's get started.
We drop the Launch.exe file into the debugger and find **ourselves** at the entry point of the packer.



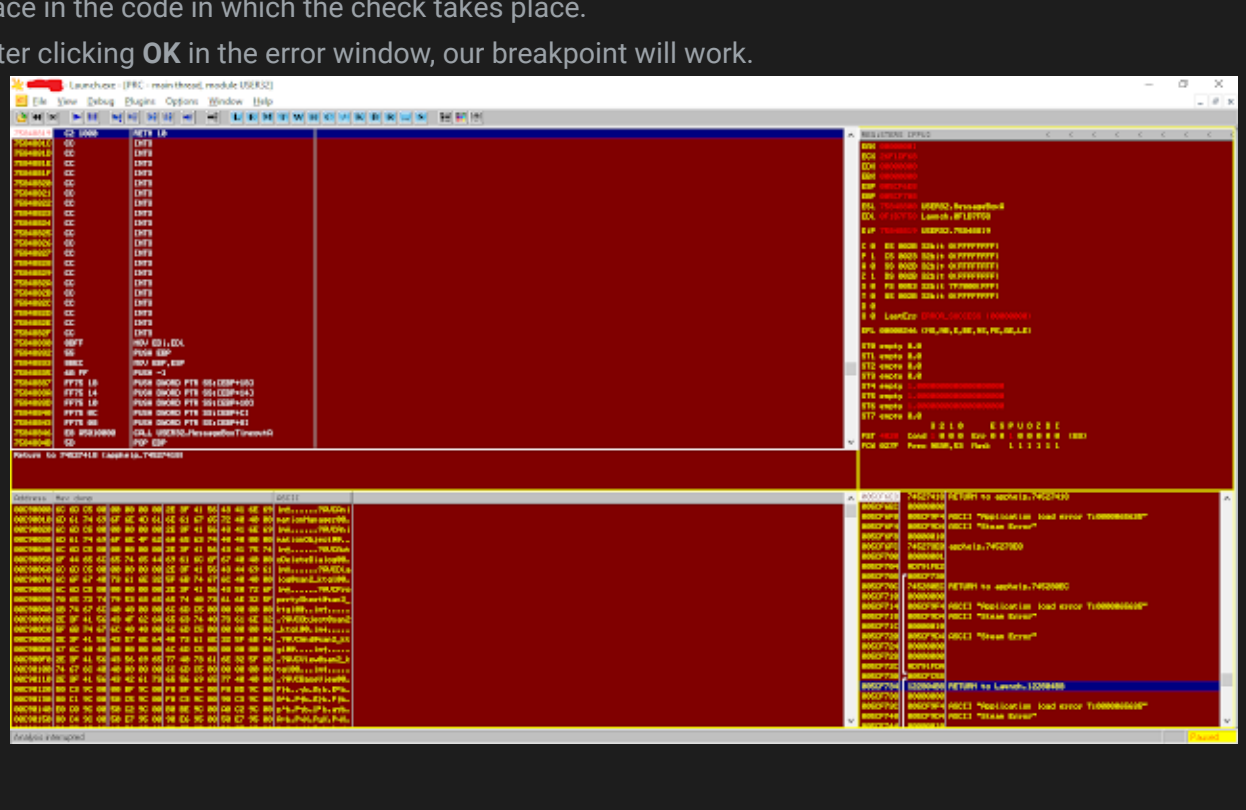
If we run the game under the debugger, then we will spit out an error, which means that the game code was not unpacked due to the detection of the debugger, and therefore **Steam** did not gain control over the process.



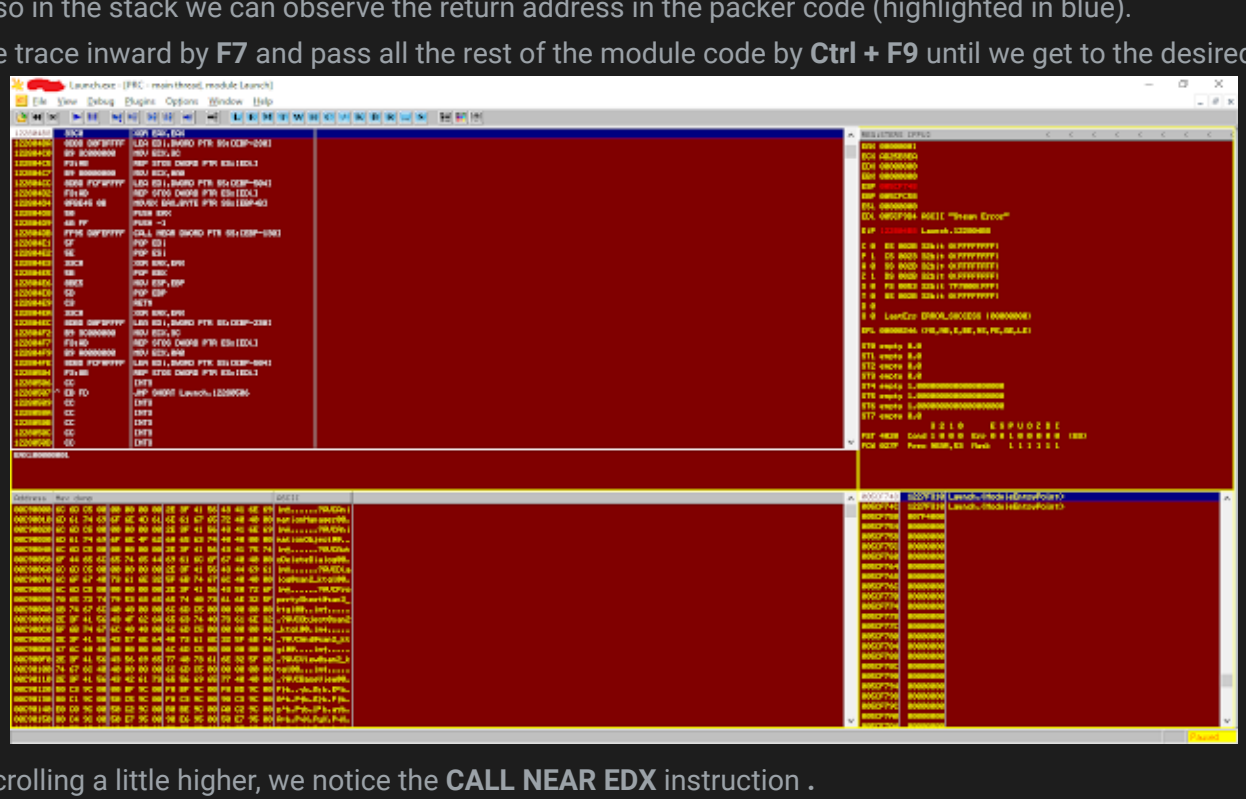
Steam Error

Application load error T:0000065635

Let's put a breakpoint on the prologue of the **MessageBoxA** function , namely on the **Retn 10** instruction, and go to the place in the code in which the check takes place.
After clicking **OK** in the error window, our breakpoint will work.



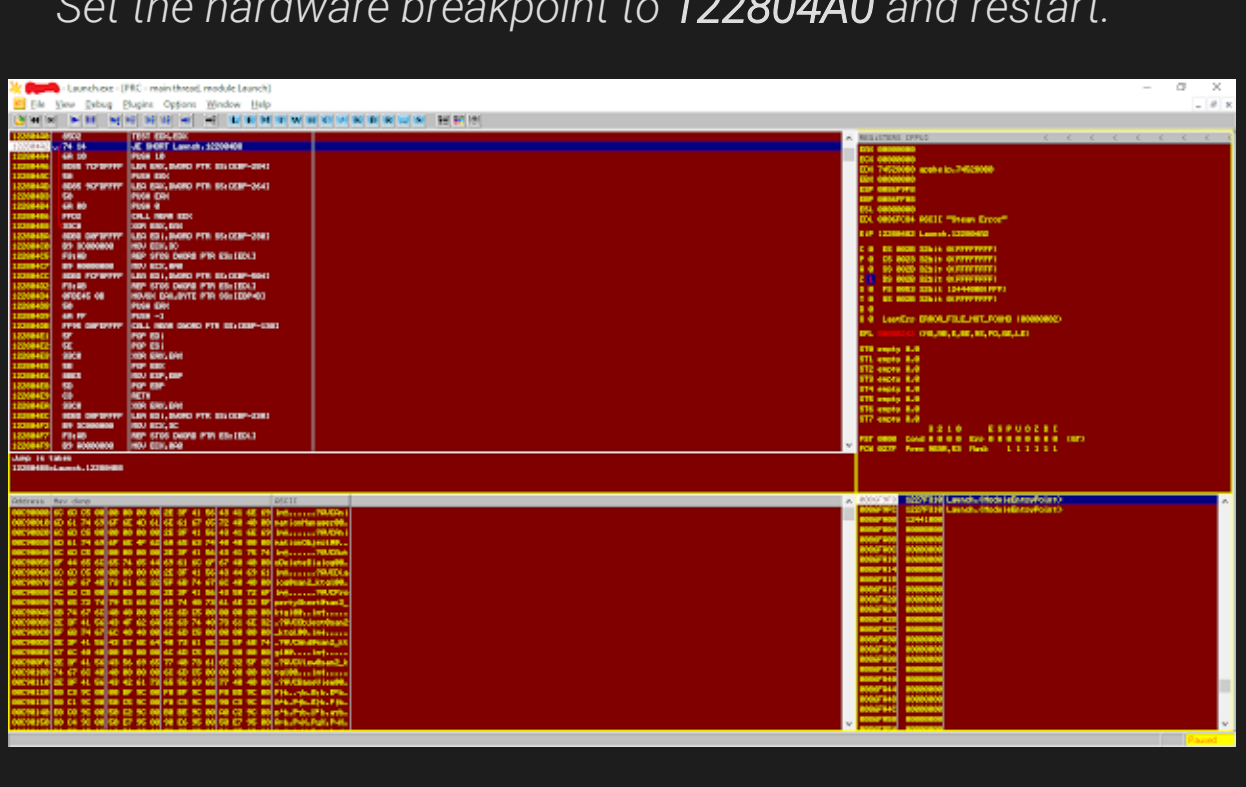
Also in the stack we can observe the return address in the packer code (highlighted in blue).
We trace inward by **F7** and pass all the rest of the module code by **Ctrl + F9** until we get to the desired address.



Scrolling a little higher, we notice the **CALL NEAR EDX** instruction .
We can safely say that this is a function call, the address of which is taken from the **EDX** register .
A little higher, we can observe a check condition of the following type:

> *122804A0 85D2 TEST EDX, EDX*
> *122804A2 74 14 JE SHORT Launch. 122804B8*

> *Set the hardware breakpoint to 122804A0 and restart.*

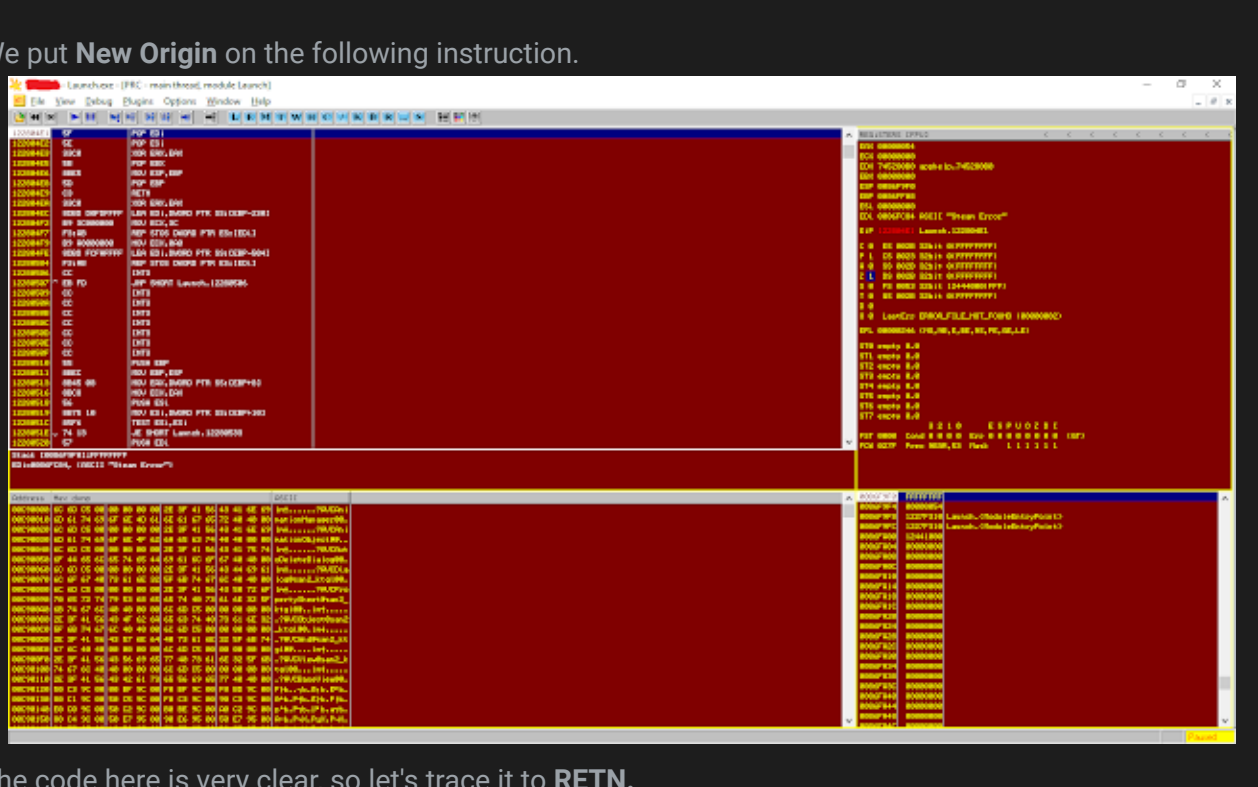


The **EDX** is populated with an address pointing to a **MessageBoxA** call called through the jungle **apphelp.dll** . Fucking **windows10** . Solid redirects and forwards.
Set the flag **Z = 1** to skip the function and thus satisfy the condition.
We continue tracing until we reach this address:

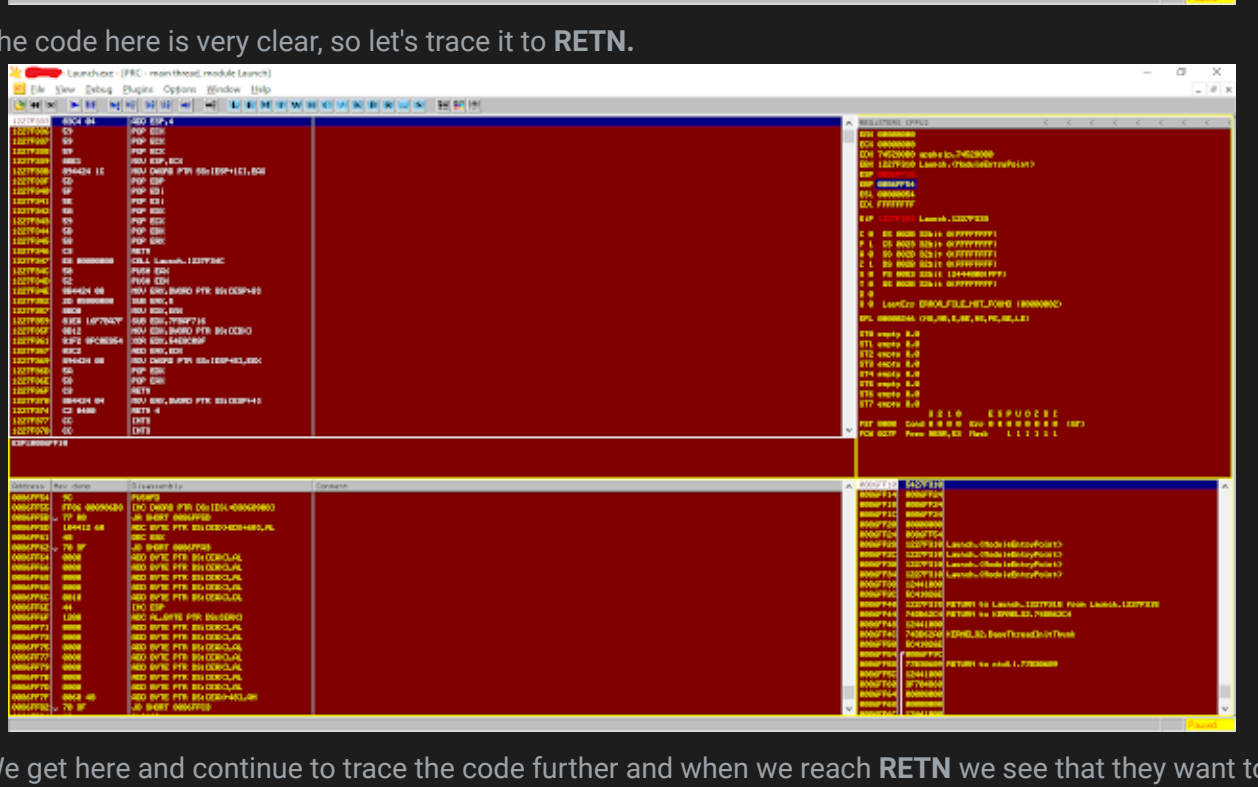
> *122804DB FF95 D0FEFFFF CALL NEAR DWORD PTR SS: [EBP-130]; KERNEL32.TerminateProcess*

> *Hmm, the packer was going to extinguish the process, but the dick was floating there.*

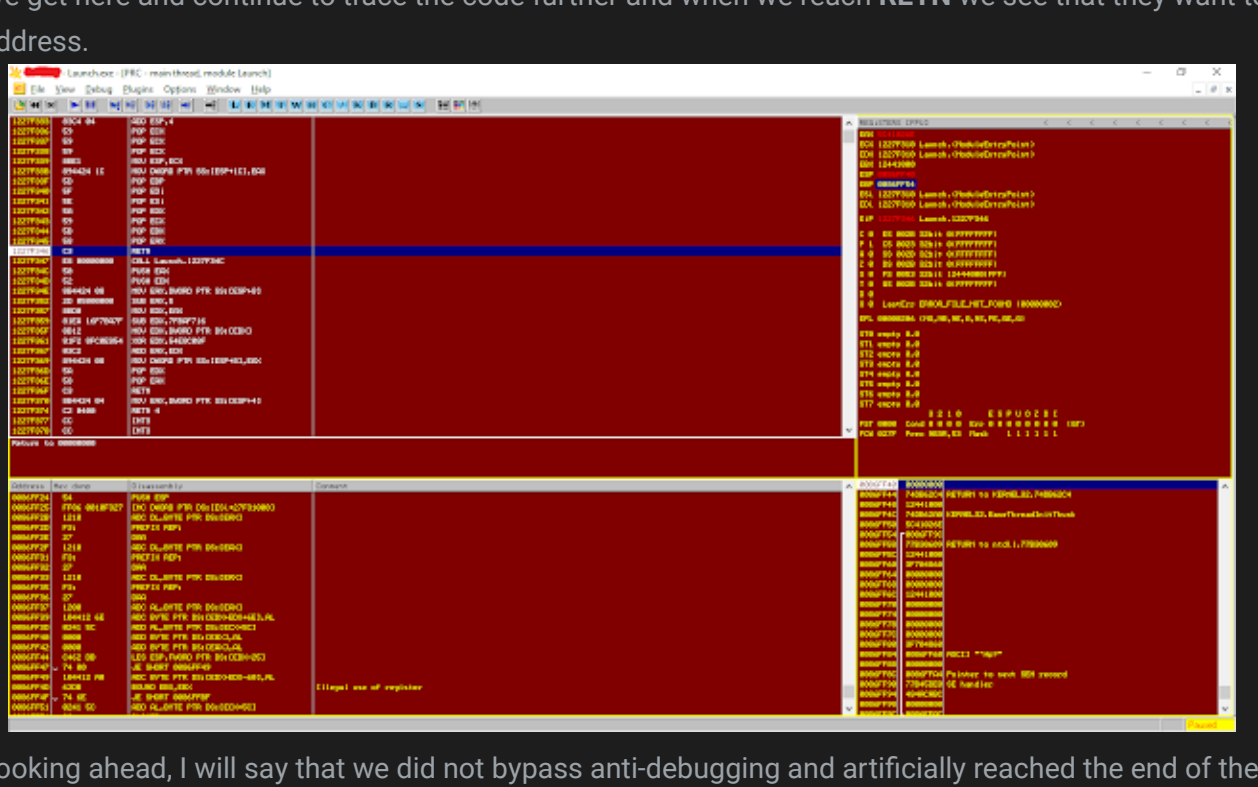
We put **New Origin** on the following instruction.



The code here is very clear, so let's trace it to **RETN.**

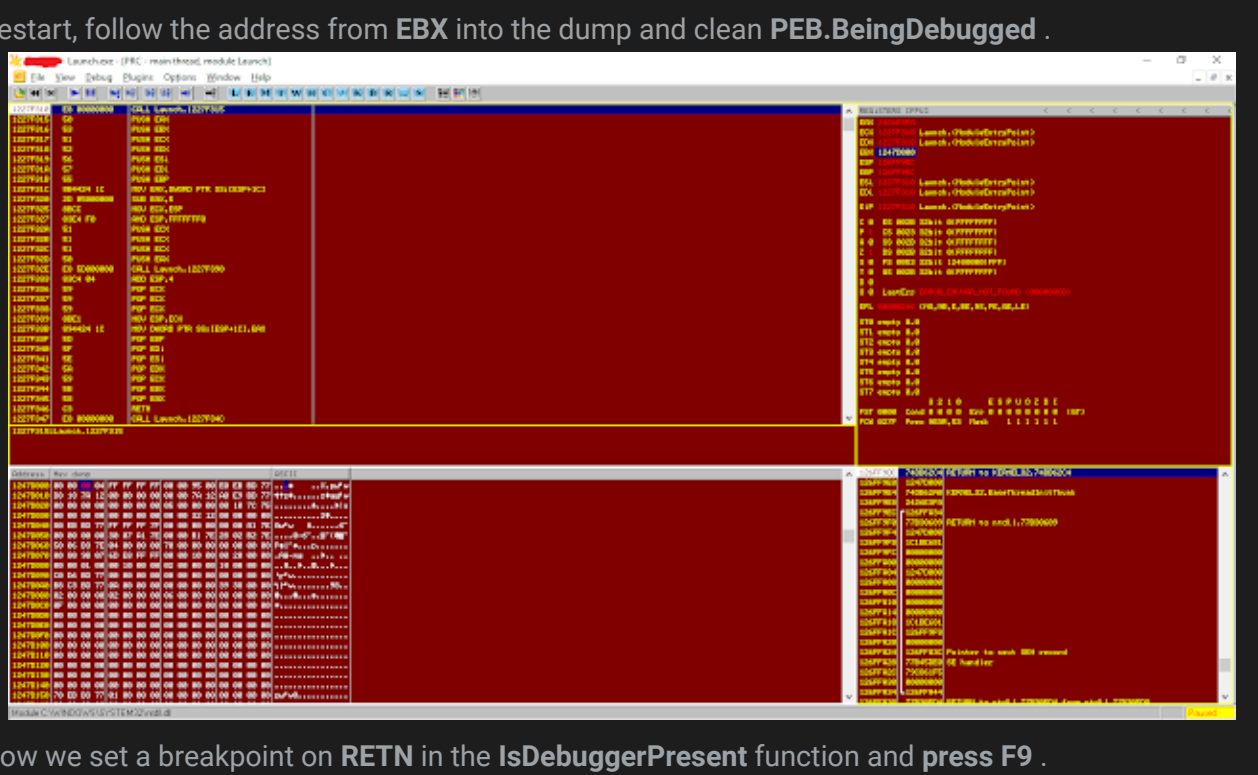


We get here and continue to trace the code further and when we reach **RETN** we see that they want to return us to the zero address.



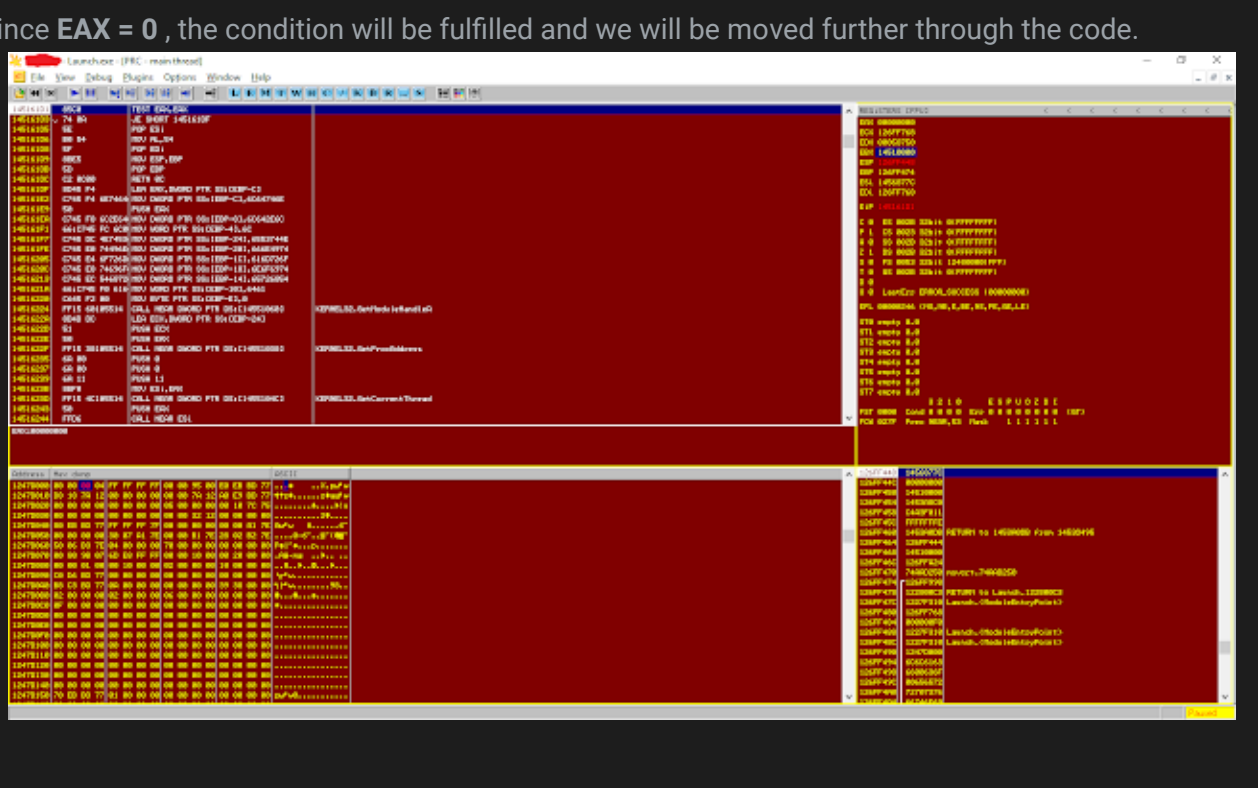
Looking ahead, I will say that we did not bypass anti-debugging and artificially reached the end of the unpacking procedure, but it did not return us to the game's entry point. This will be the case if we try to bypass checks head-on.

Plan b.
Restart, follow the address from **EBX** into the dump and clean **PEB.BeingDebugged** .



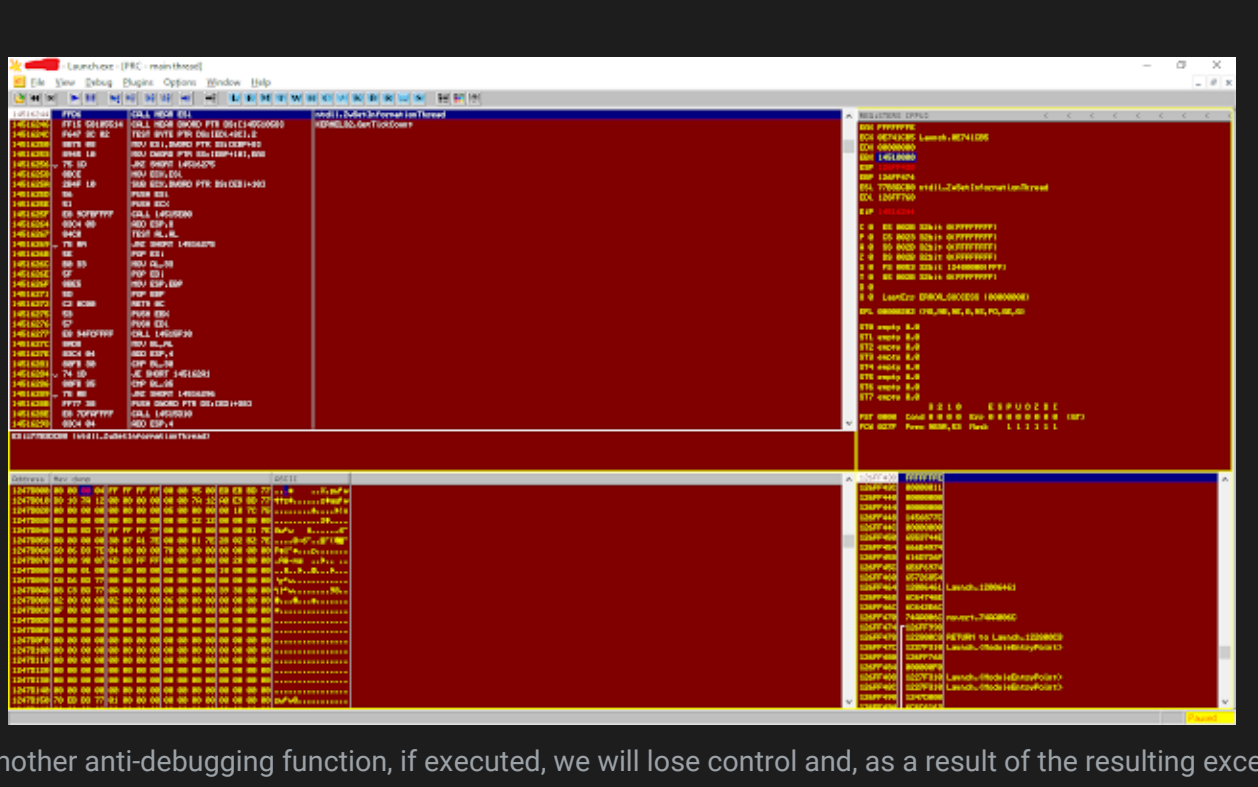
Now we set a breakpoint on **RETN** in the **IsDebuggerPresent** function and **press F9** .
In **EAX** , we have zero because we made the program think that it is not under debug.
We trace inside and get into the allocated memory area.
(This array is created by calls to the **VirtualAlloc** function , and the necessary access rights are set using **VirtualProtect**).

Since **EAX = 0** , the condition will be fulfilled and we will be moved further through the code.



We continue tracing, but we will be very careful.
We can safely trace to this address:

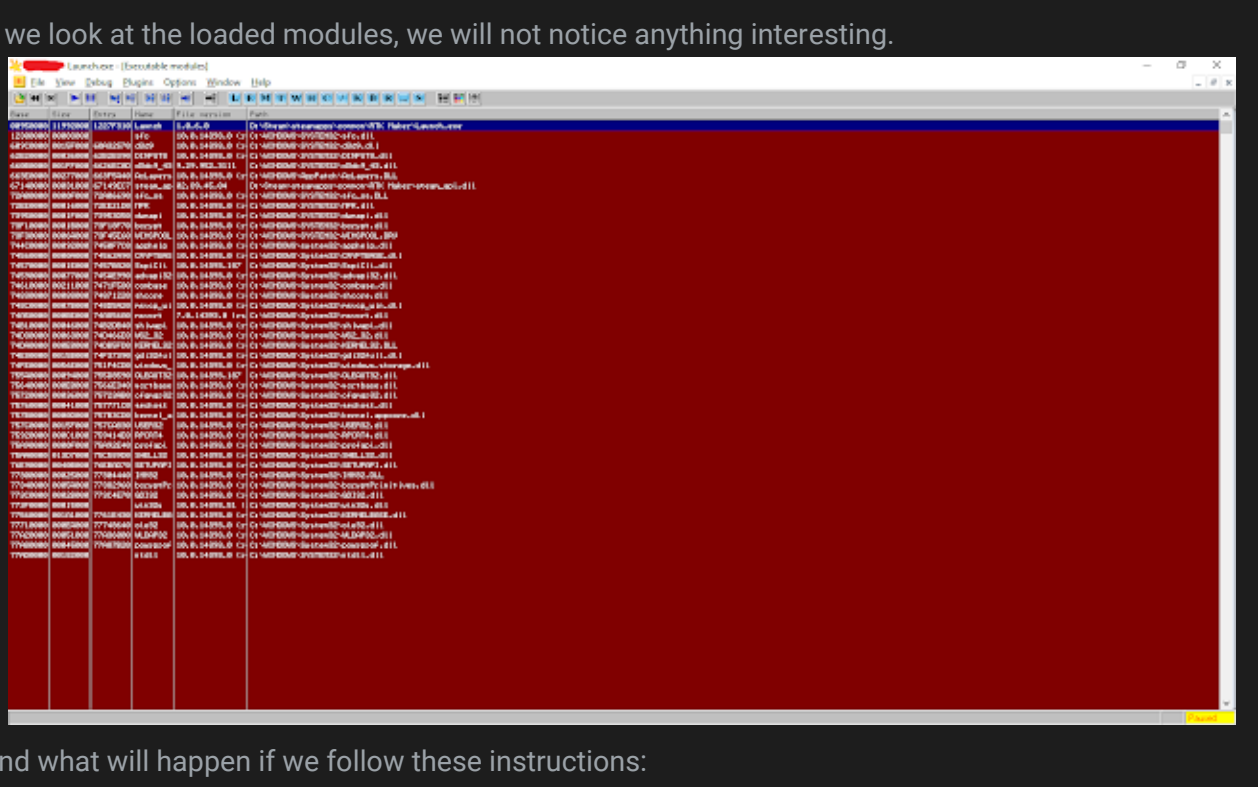
> *14516244 FFD6 CALL NEAR ESi; ntdll.ZwSetInformationThread*



We are greeted by another anti-debugging function, if executed, we will lose control and, as a result of the resulting exception, we will not be able to continue further actions. **Set New Origin** to the next instruction and continue tracing.
We will skip the first **CALL** encountered , but we will stop at the second.

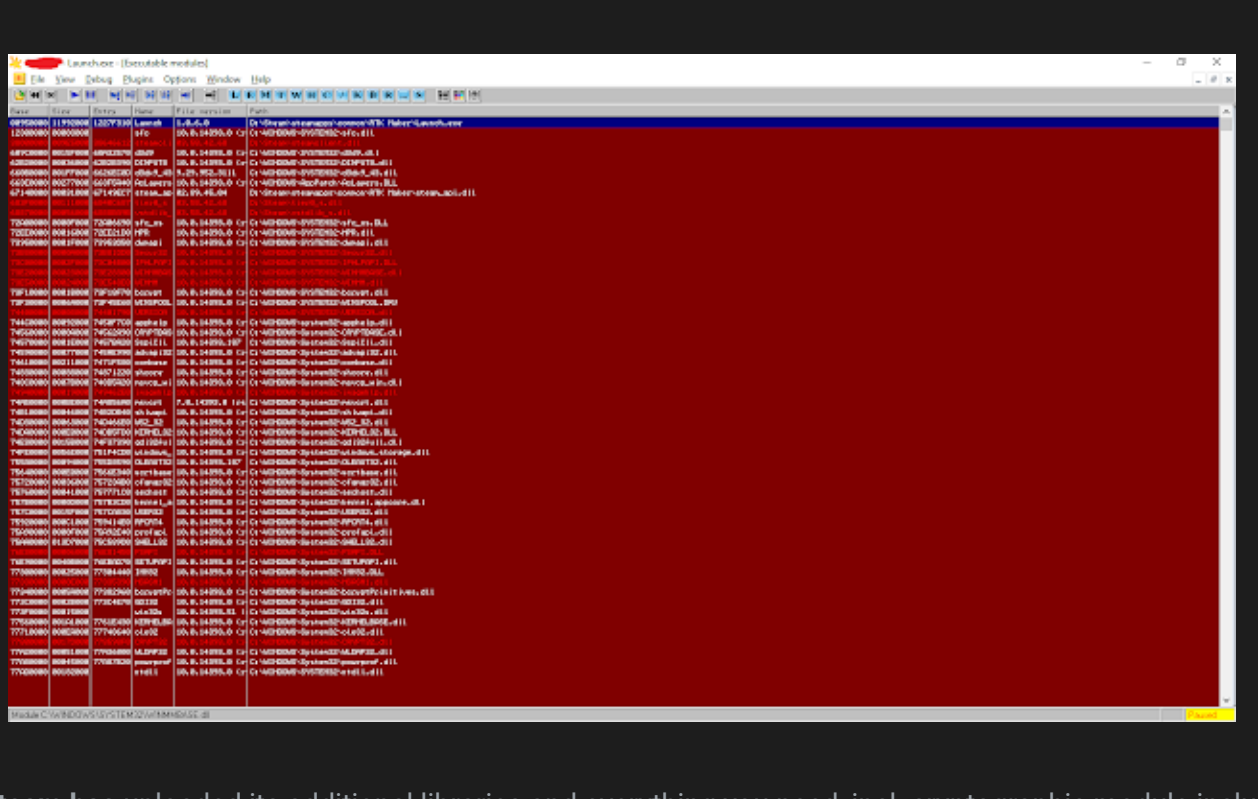
> *Screenshots with different addresses will be shown below, but this is not the essence of how important. Each time you restart* **Steam, it** *changes the memory area from which the decryption process of the original file is performed.*

If we look at the loaded modules, we will not notice anything interesting.



And what will happen if we follow these instructions:

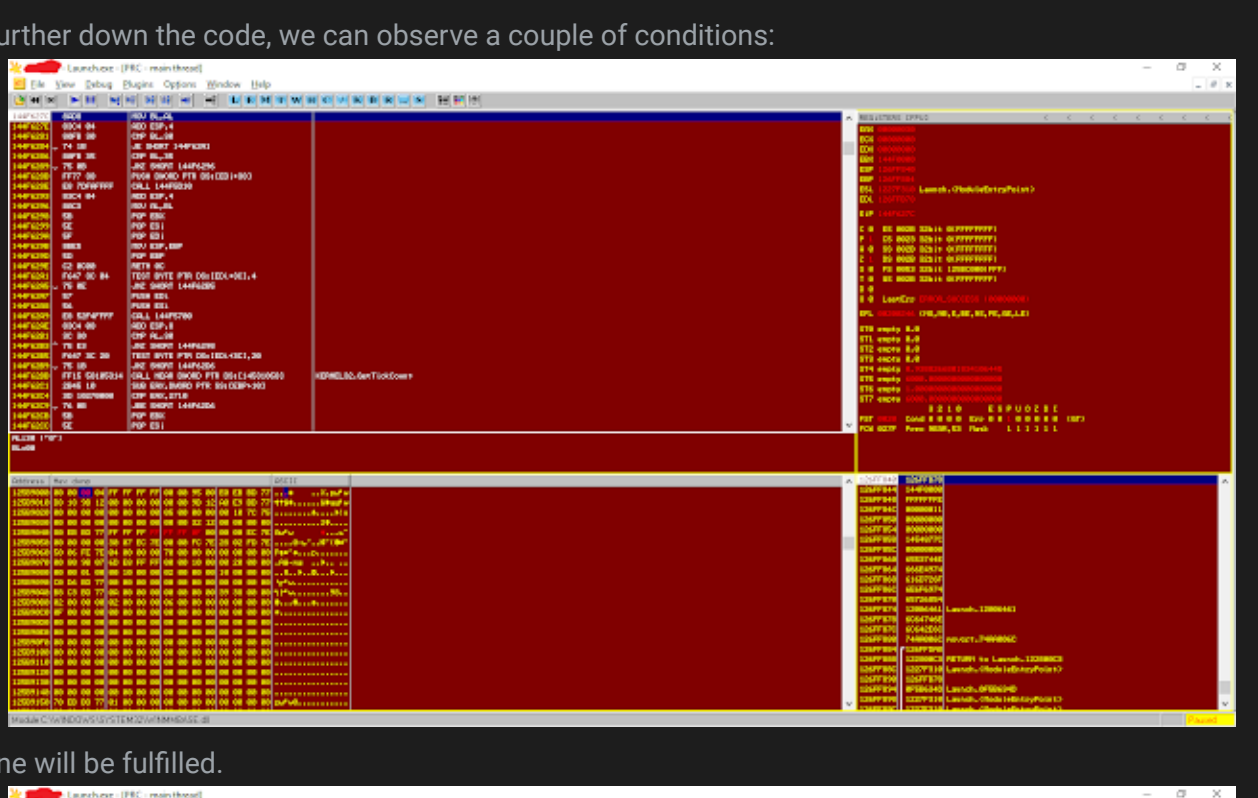
> *144F6277 E8 94FCFFFF CALL 144F5F10*



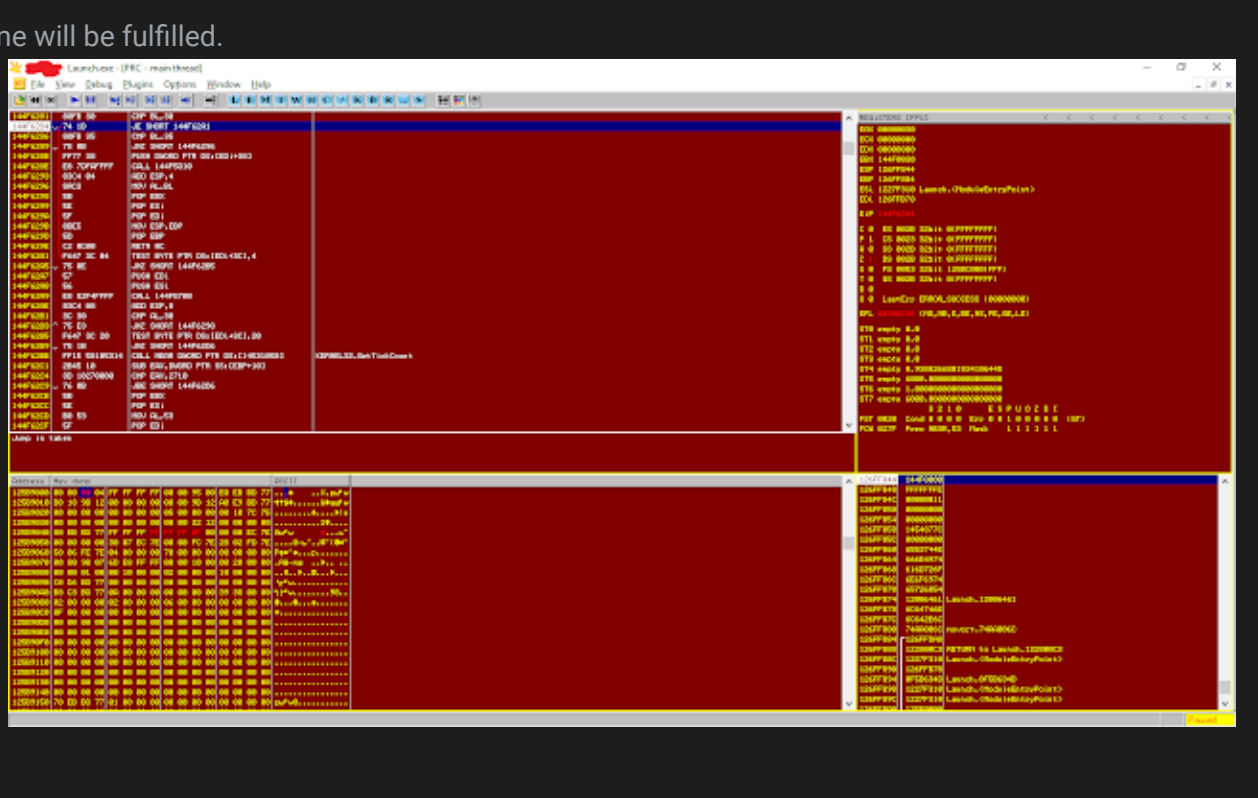
Steam has uploaded its additional libraries and everything you need, incl. cryptographic module included with windows.
This is definitely a decryption procedure. We will not go into detail about her guts, just move on to the next instruction on **F4.**

> *Note:* **Steam itself** *now considers the game to be running, i.e. nothing suspicious happens.*

Further down the code, we can observe a couple of conditions:



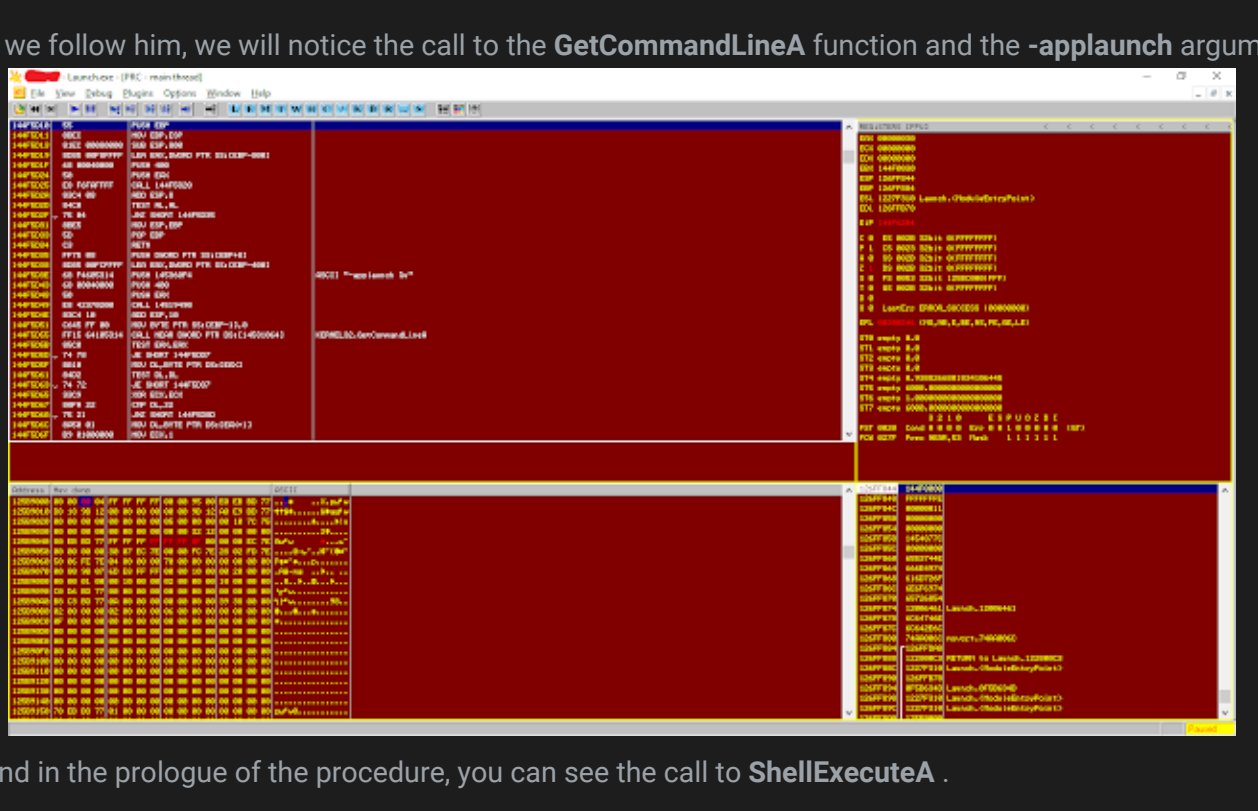
Specifically, the first one will be fulfilled.



In fact, we have it means that the game is unpacked, there is no debugging, but we are missing an important call:

> *144F628E E8 7DFAFFFF CALL 144F5D10*

If we follow him, we will notice the call to the **GetCommandLineA** function and the **-applaunch** argument .



And in the prologue of the procedure, you can see the call to **ShellExecuteA** .

In other words, **Steam** is going to launch the game, but we will not let it do this, and the condition already described above will not allow it, and accordingly we will not lose control over the process and will not get an error.

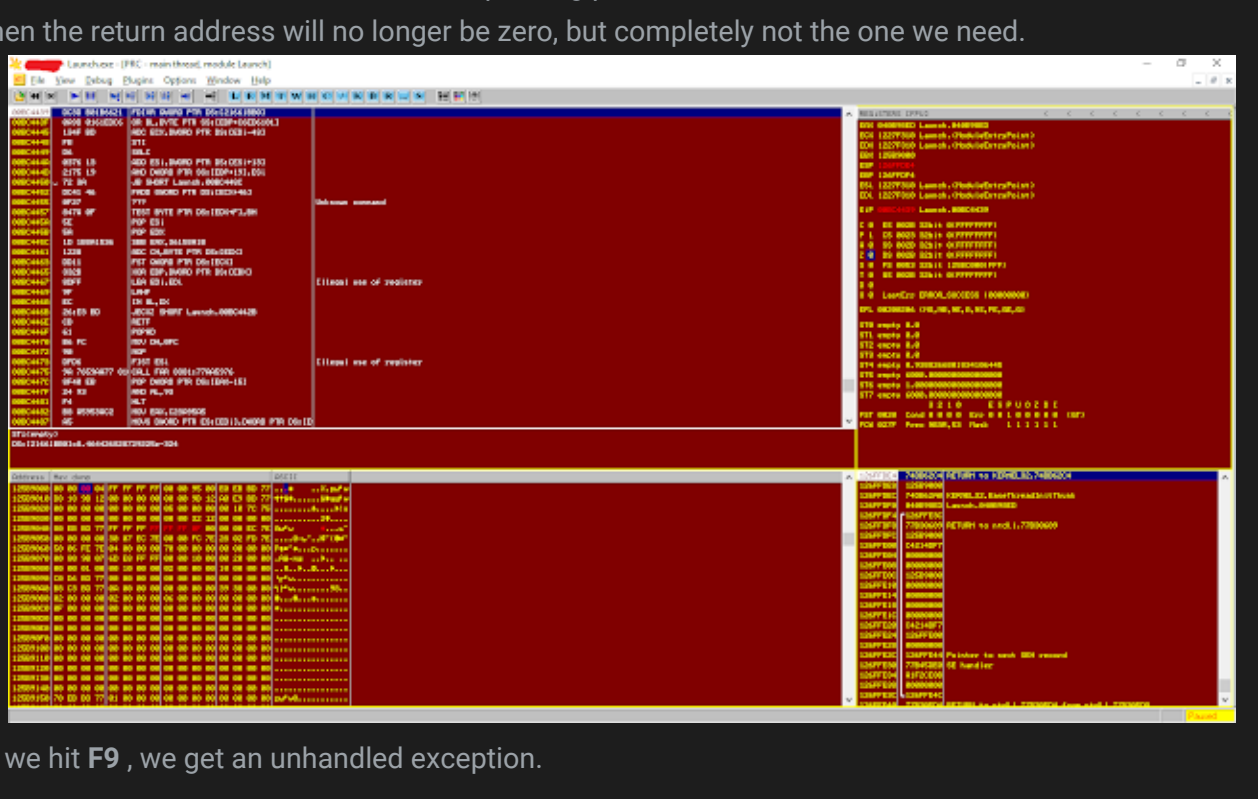
But in case the condition

144F6281 80FB 30 CMP BL, 30
144F6284 74 1B JE SHORT 144F62A1

will not be executed, then the next check will occur:

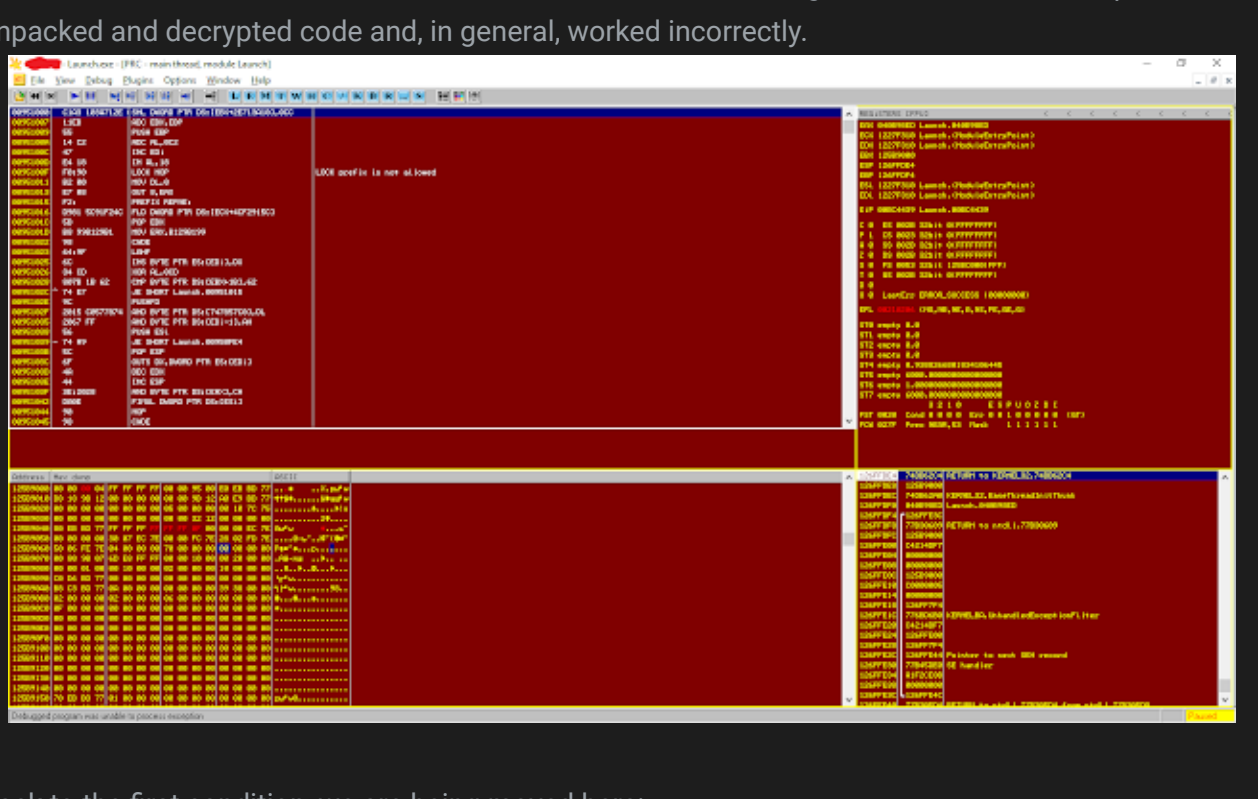
144F6286 80FB 35 CMP BL, 35
144F6289 75 0B JNZ SHORT 144F6296

The condition of which will be satisfied, because **BL = 30** , which is different from **35** .
If we trace the code to the end of the unpacking procedure, the view of which can be seen in the first part of the article, then the return address will no longer be zero, but completely not the one we need.

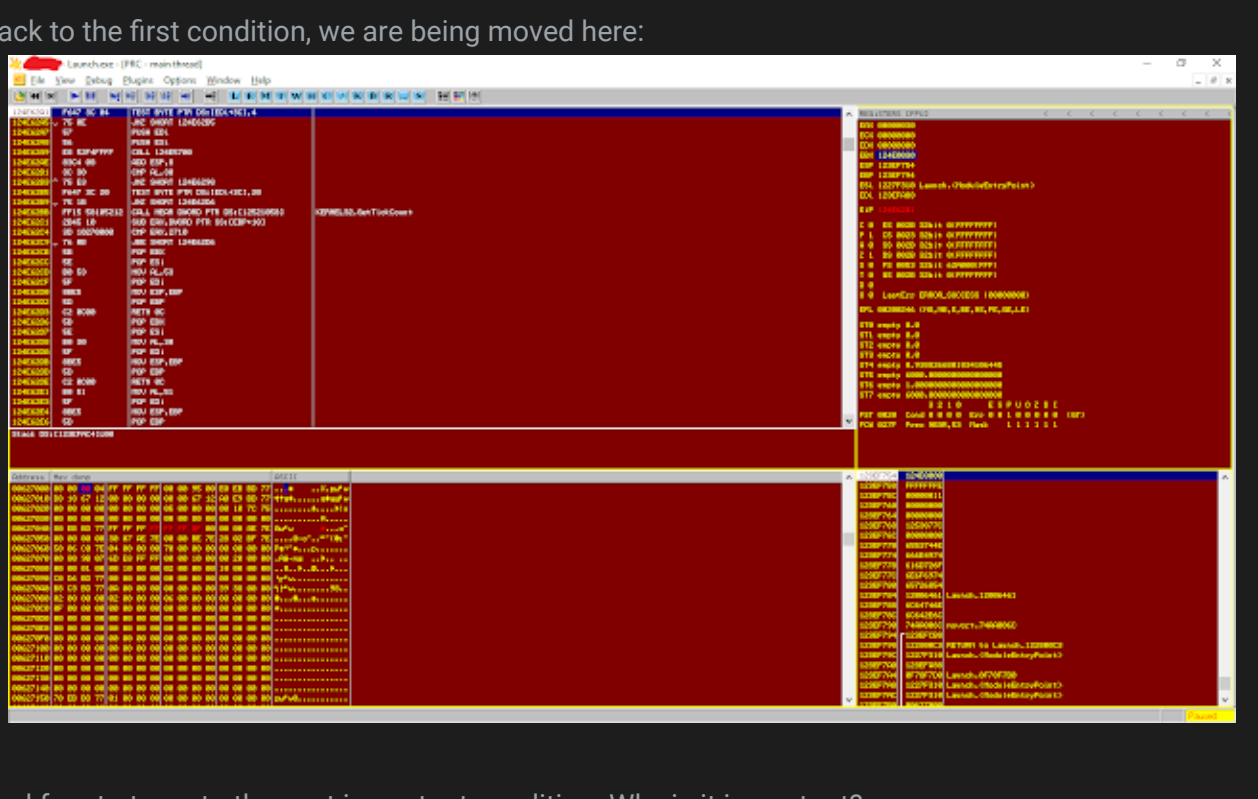


If we hit **F9** , we get an unhandled exception.

And if we send a section of the code to the disassembler, we will get a mess, since the unpacker did not overwrite the unpacked and decrypted code and, in general, worked incorrectly.

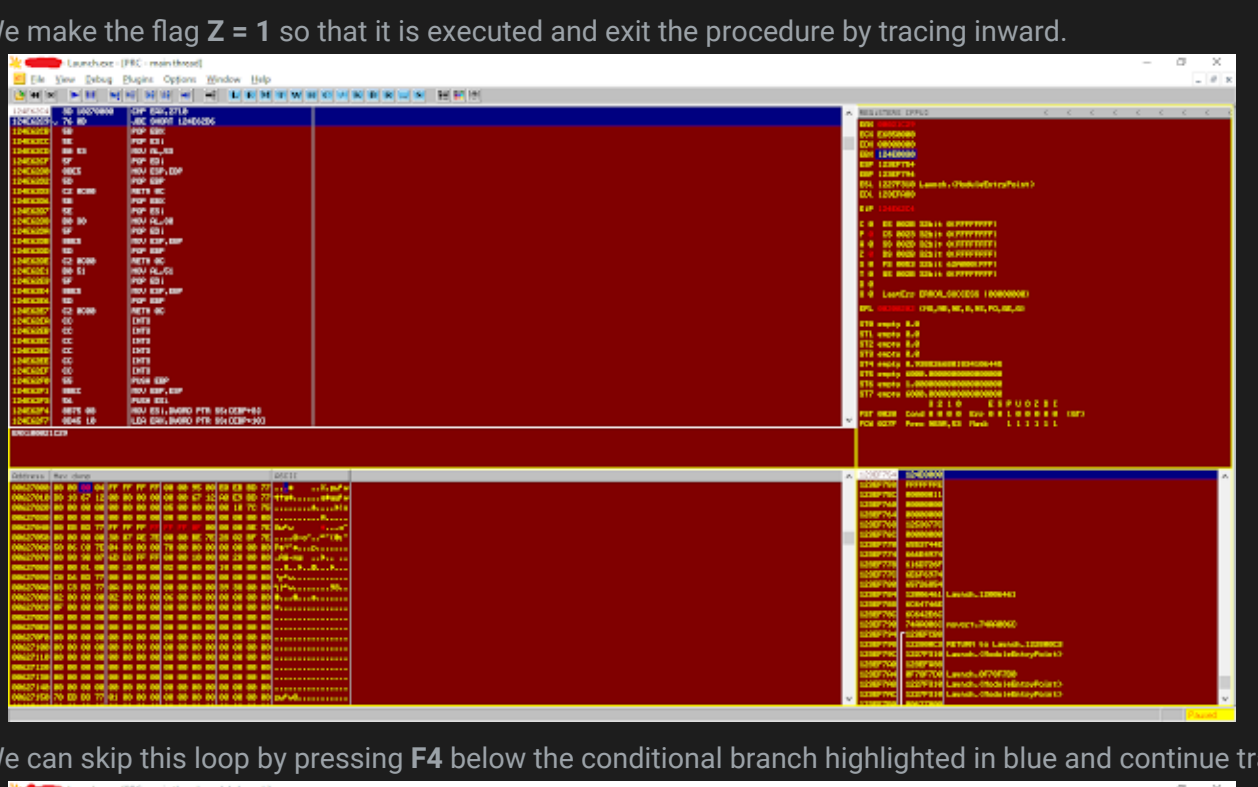


Back to the first condition, we are being moved here:

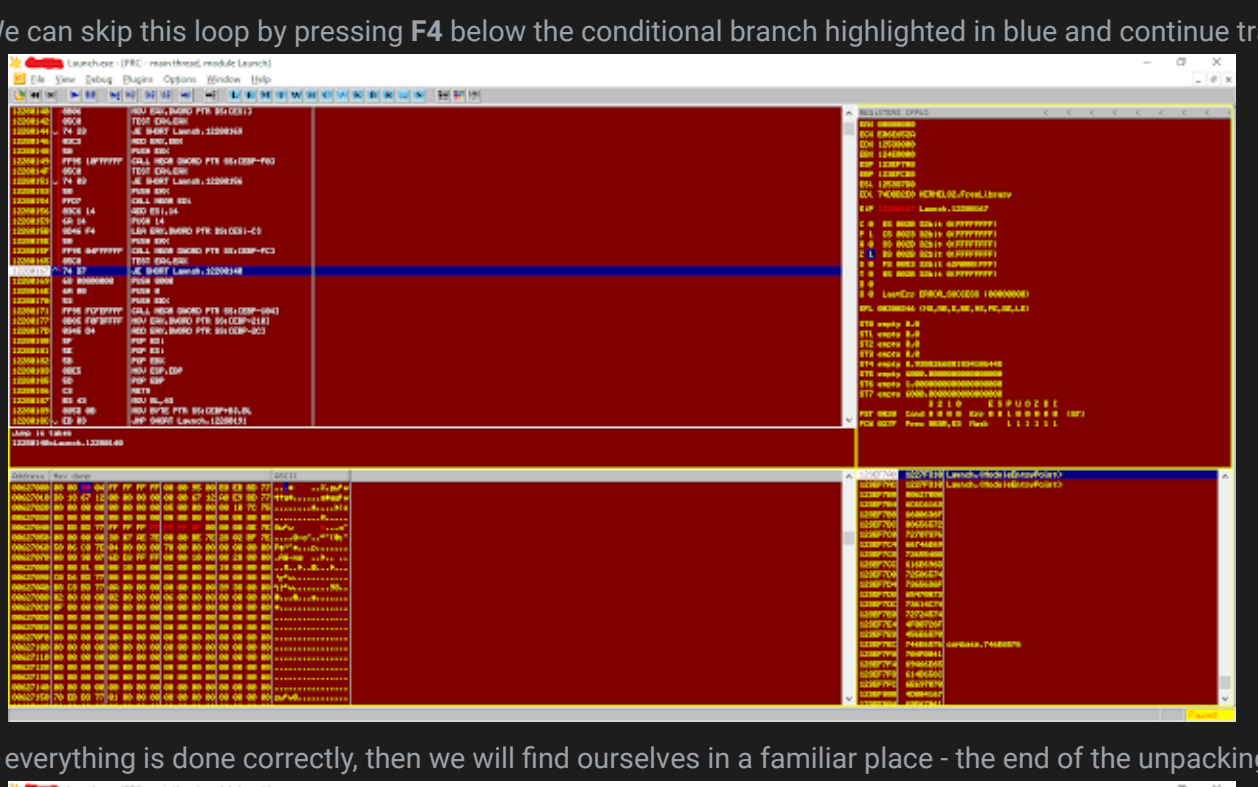


Feel free to trace to the next important condition. Why is it important?
I removed the steam stub of old versions and always during the decryption process and before reaching the entry point this condition was met. This is a kind of signature.

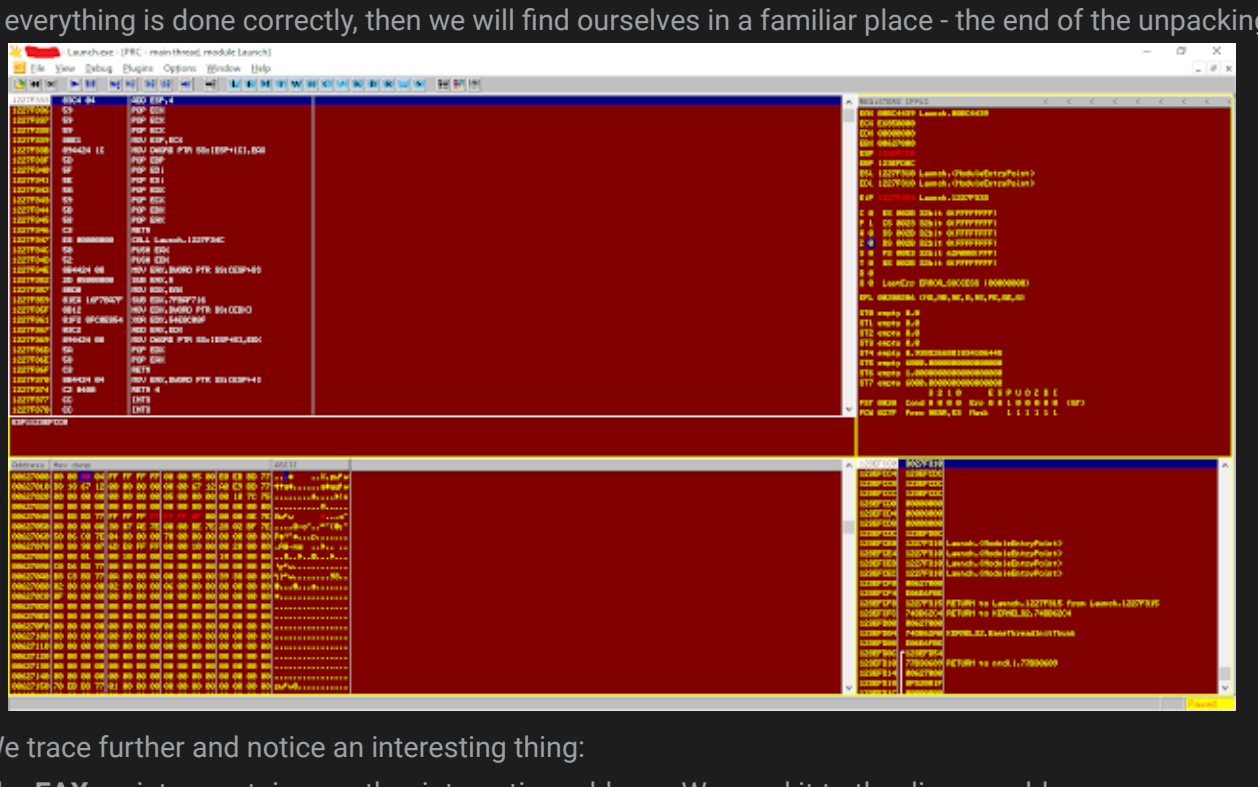
We make the flag **Z = 1** so that it is executed and exit the procedure by tracing inward.



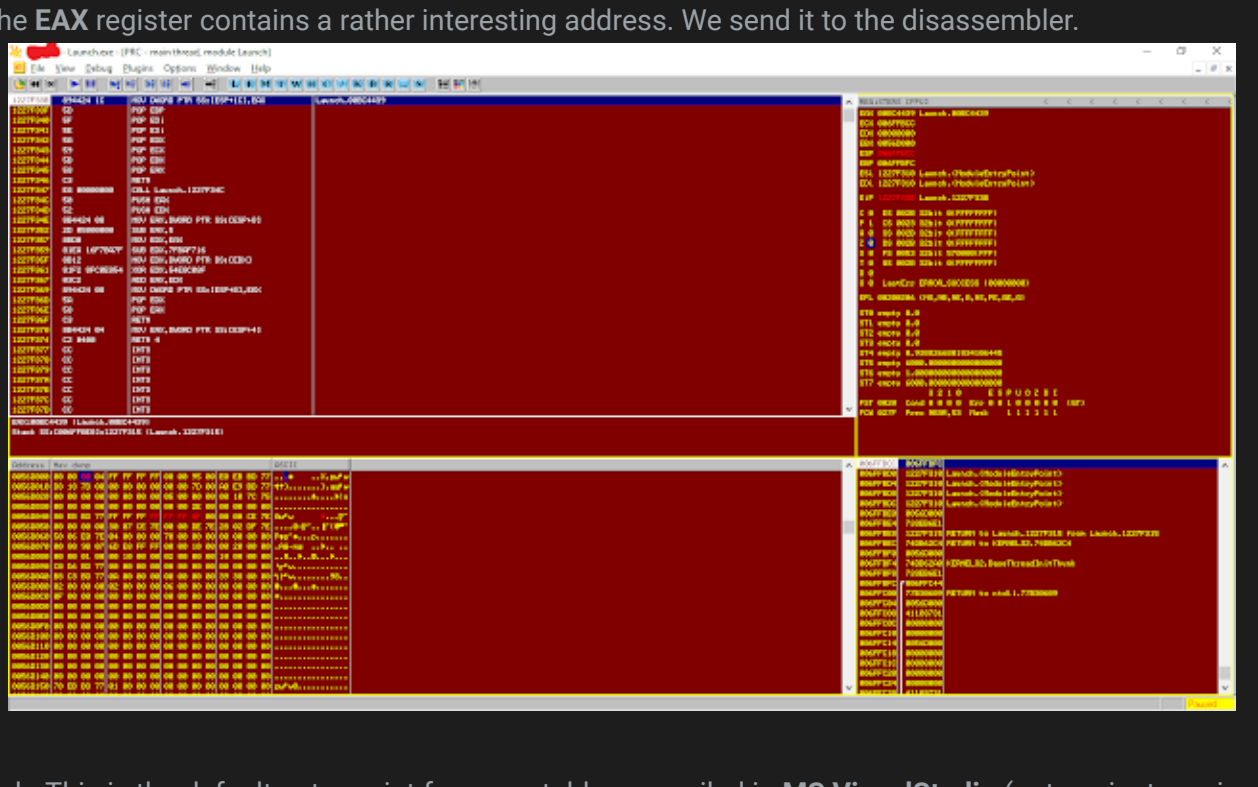
We can skip this loop by pressing **F4** below the conditional branch highlighted in blue and continue tracing.



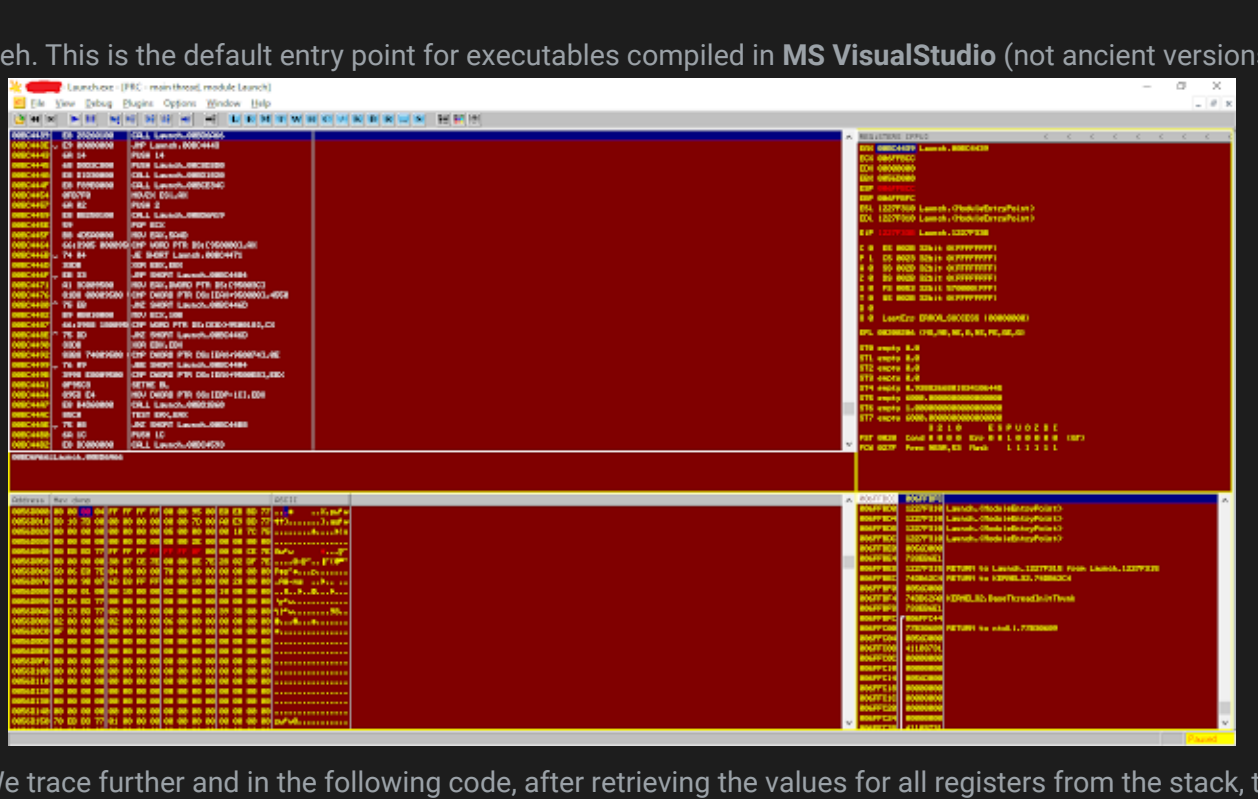
If everything is done correctly, then we will find ourselves in a familiar place - the end of the unpacking procedure.



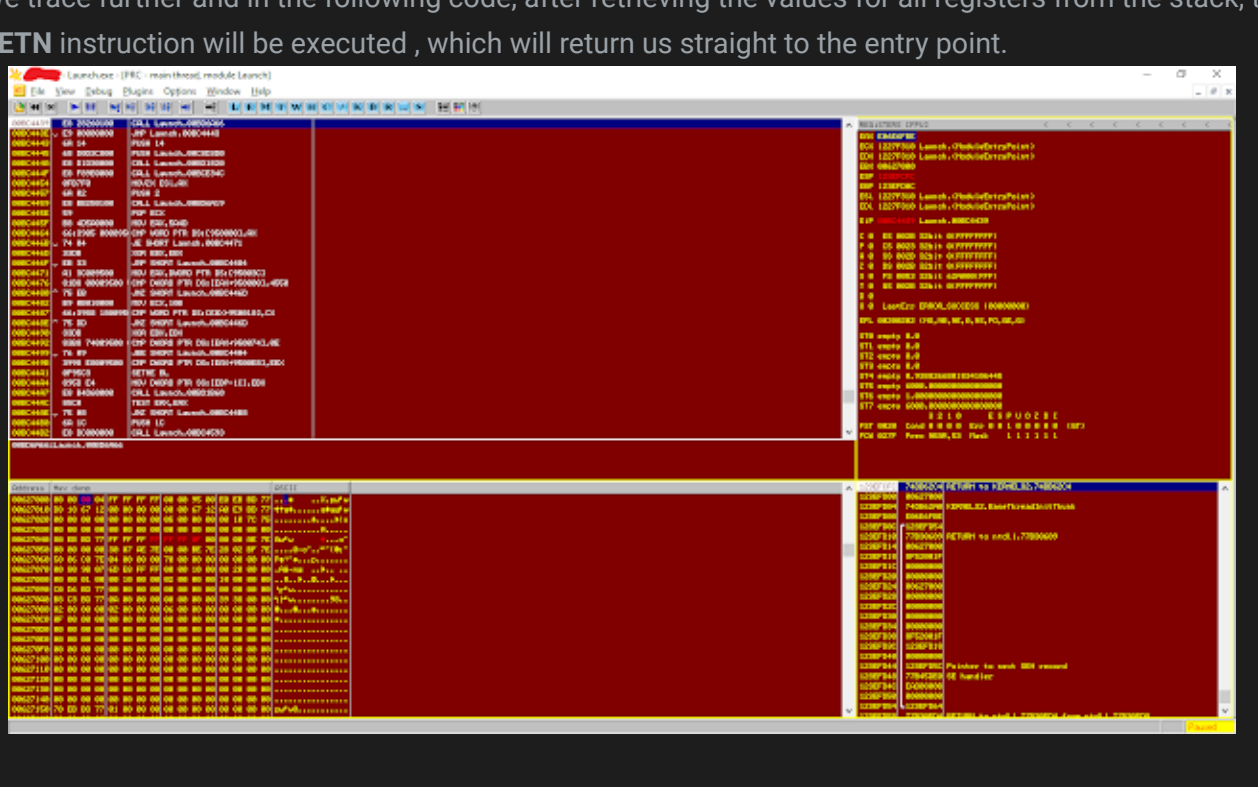
We trace further and notice an interesting thing:
The **EAX** register contains a rather interesting address. We send it to the disassembler.



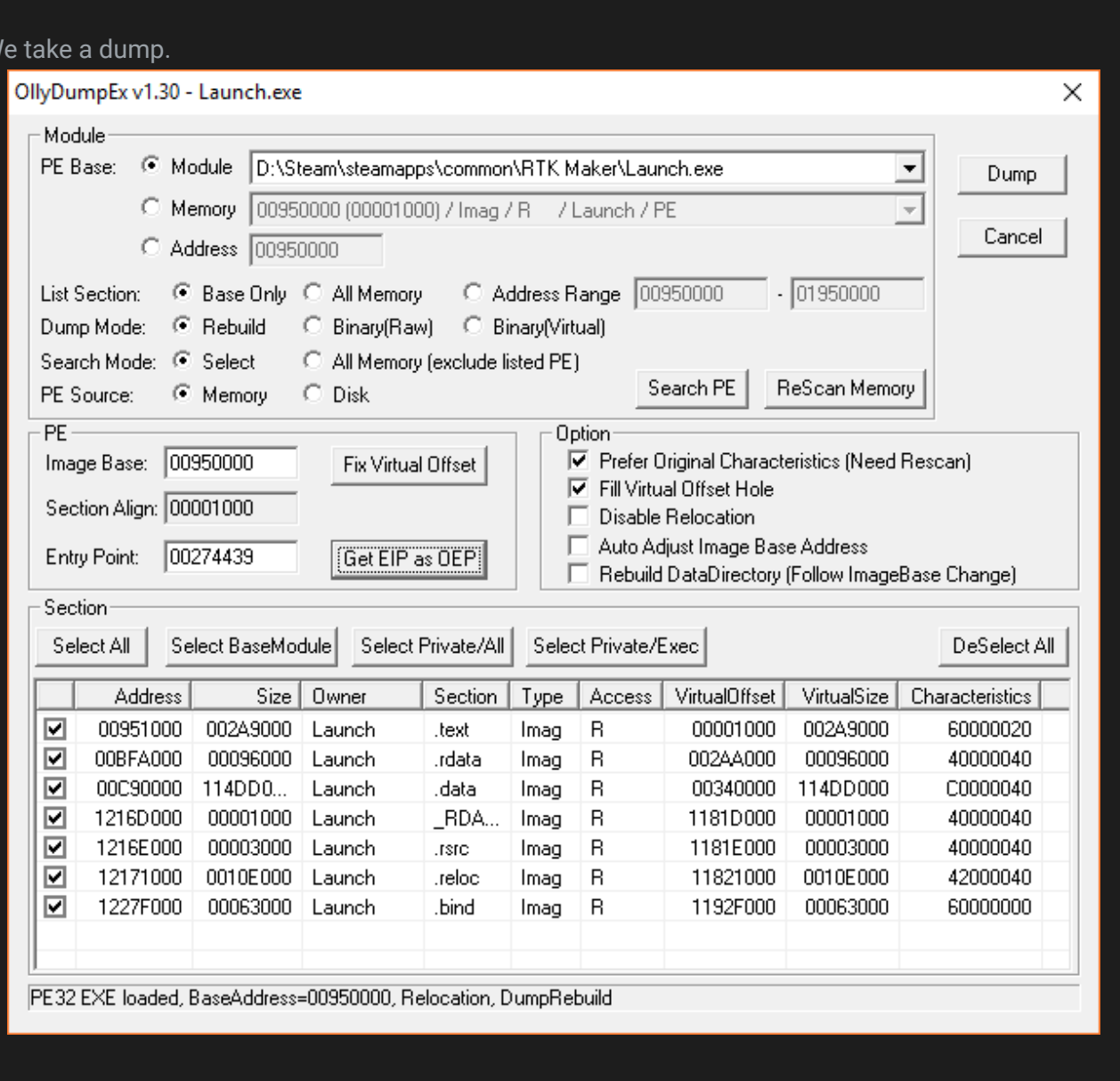
Heh. This is the default entry point for executables compiled in **MS VisualStudio** (not ancient versions).



We trace further and in the following code, after retrieving the values for all registers from the stack, the long-awaited **RETN** instruction will be executed , which will return us straight to the entry point.



We take a dump.



OllyDumpEx v1.30 - Launch.exe

Module
PE Base: ● Module D:\Steam\steamapps\common\RTK Maker\Launch.exe
○ Memory 00950000 (00001000) / Imag / R / Launch / PE
○ Address 00950000
List Section: ● Base Only ○ All Memory ○ Address Range 00950000 - 01950000
Dump Mode: ● Rebuild ○ Binary(Raw) ○ Binary(Virtual)
Search Mode: ● Select ○ All Memory (exclude listed PE)
PE Source: ● Memory ○ Disk

PE
Image Base: 00950000
Section Align: 00001000
Entry Point: 00274439

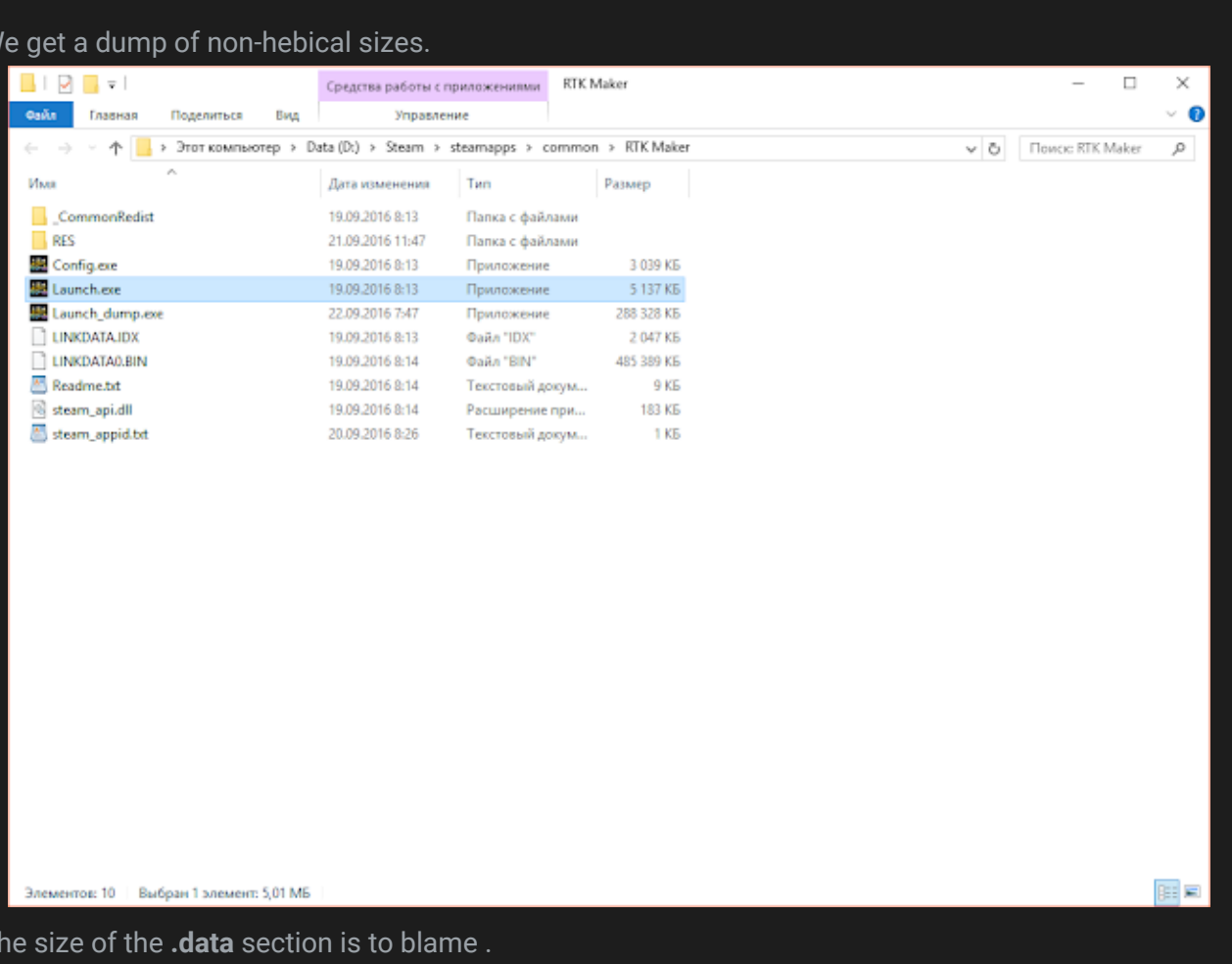
Option
☑ Prefer Original Characteristics (Need Rescan)
☑ Fill Virtual Offset Hole
☐ Disable Relocation
☐ Auto Adjust Image Base Address
☐ Rebuild DataDirectory (Follow ImageBase Change)

Section

	Address	Size	Owner	Section	Type	Access	VirtualOffset	VirtualSize	Characteristics
☑	00951000	002A9000	Launch	.text	Imag	R	00001000	002A9000	60000020
☑	00BFA000	00096000	Launch	.rdata	Imag	R	002AA000	00096000	40000040
☑	00C90000	114DD0...	Launch	.data	Imag	R	00340000	114DD000	C0000040
☑	1216D000	00001000	Launch	_RDA...	Imag	R	1181D000	00001000	40000040
☑	1216E000	00003000	Launch	.rsrc	Imag	R	1181E000	00003000	40000040
☑	12171000	0010E000	Launch	.reloc	Imag	R	11821000	0010E000	42000040
☑	1227F000	00063000	Launch	.bind	Imag	R	1192F000	00063000	60000000

PE32 EXE loaded, BaseAddress=00950000, Relocation, DumpRebuild

We get a dump of non-hebical sizes.

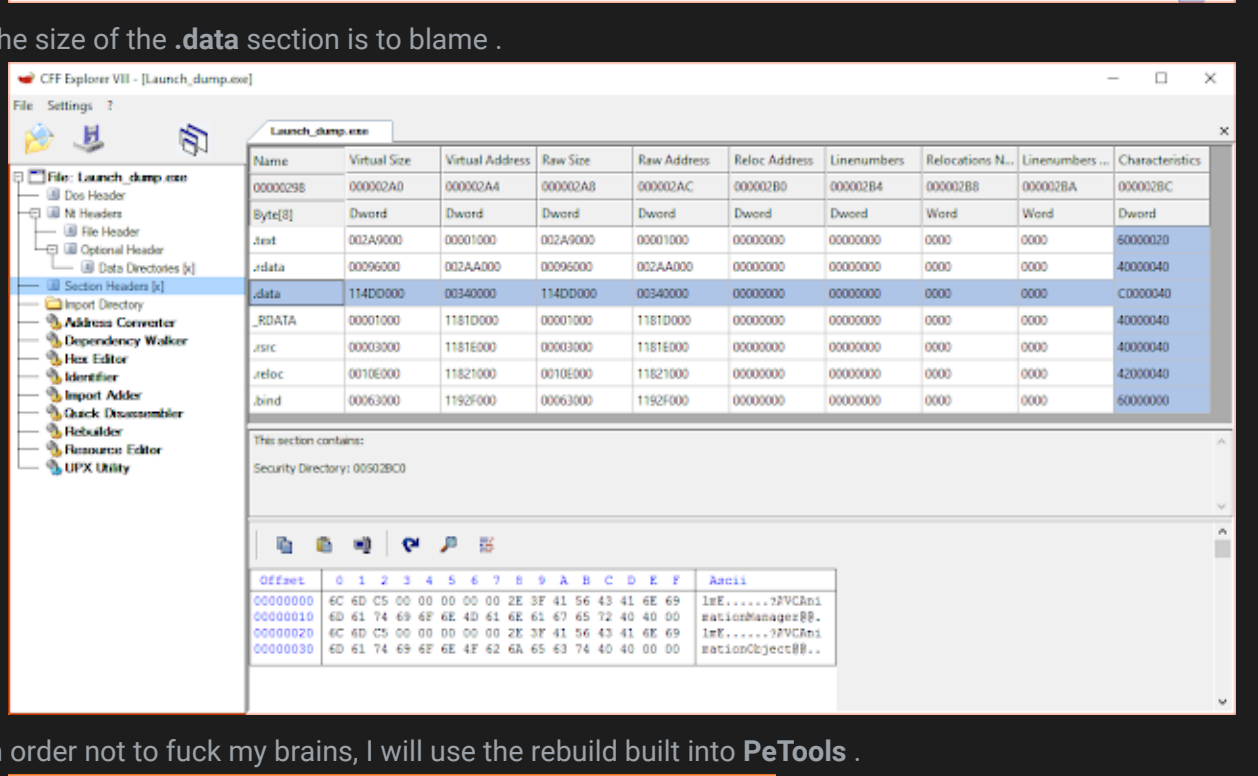


Этот компьютер › Data (D:) › Steam › steamapps › common › RTK Maker

Имя	Дата изменения	Тип	Размер
_CommonRedist	19.09.2016 8:13	Папка с файлами	
RES	21.09.2016 11:47	Папка с файлами	
Config.exe	19.09.2016 8:13	Приложение	3 039 КБ
Launch.exe	19.09.2016 8:13	Приложение	5 137 КБ
Launch_dump.exe	22.09.2016 7:47	Приложение	288 328 КБ
LINKDATA.IDX	19.09.2016 8:13	Файл "IDX"	2 047 КБ
LINKDATA0.BIN	19.09.2016 8:14	Файл "BIN"	485 389 КБ
Readme.txt	19.09.2016 8:14	Текстовый докум...	9 КБ
steam_api.dll	19.09.2016 8:14	Расширение при...	183 КБ
steam_appid.txt	20.09.2016 8:26	Текстовый докум...	1 КБ

Элементов: 10 Выбран 1 элемент: 5,01 МБ

The size of the **.data** section is to blame .



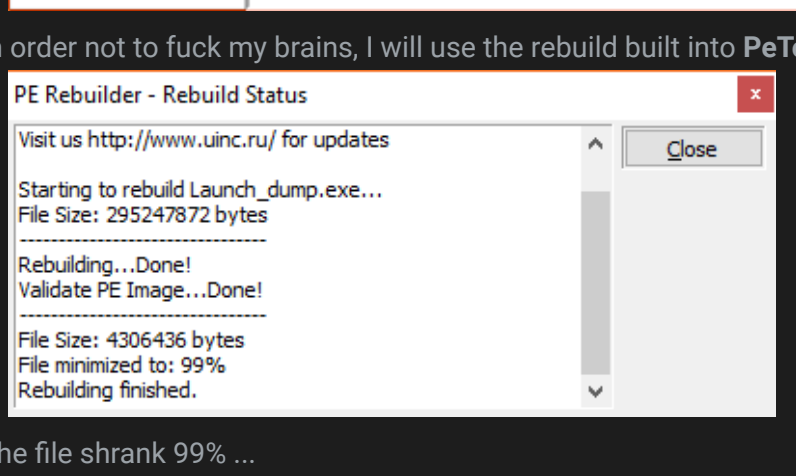
CFF Explorer VIII - [Launch_dump.exe]

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N...	Linenumbers ...	Characteristics
00000298	000002A0	000002A4	000002A8	000002AC	000002B0	000002B4	000002B8	000002BA	000002BC
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	002A9000	00001000	002A9000	00001000	00000000	00000000	0000	0000	60000020
.rdata	00096000	002AA000	00096000	002AA000	00000000	00000000	0000	0000	40000040
.data	114DD000	00340000	114DD000	00340000	00000000	00000000	0000	0000	C0000040
_RDATA	00001000	1181D000	00001000	1181D000	00000000	00000000	0000	0000	40000040
.rsrc	00003000	1181E000	00003000	1181E000	00000000	00000000	0000	0000	40000040
.reloc	0010E000	11821000	0010E000	11821000	00000000	00000000	0000	0000	42000040
.bind	00063000	1192F000	00063000	1192F000	00000000	00000000	0000	0000	60000000

This section contains:
Security Directory: 00502BC0

```
Offset    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   Ascii
00000000  6C 6D C5 00 00 00 00 00 2E 3F 41 56 43 41 6E 69  lmE......?AVCAni
00000010  6D 61 74 69 6F 6E 4D 61 6E 61 67 65 72 40 40 00  mationManager@@.
00000020  6C 6D C5 00 00 00 00 00 2E 3F 41 56 43 41 6E 69  lmE......?AVCAni
00000030  6D 61 74 69 6F 6E 4F 62 6A 65 63 74 40 40 00 00  mationObject@@..
```

In order not to fuck my brains, I will use the rebuild built into **PeTools** .

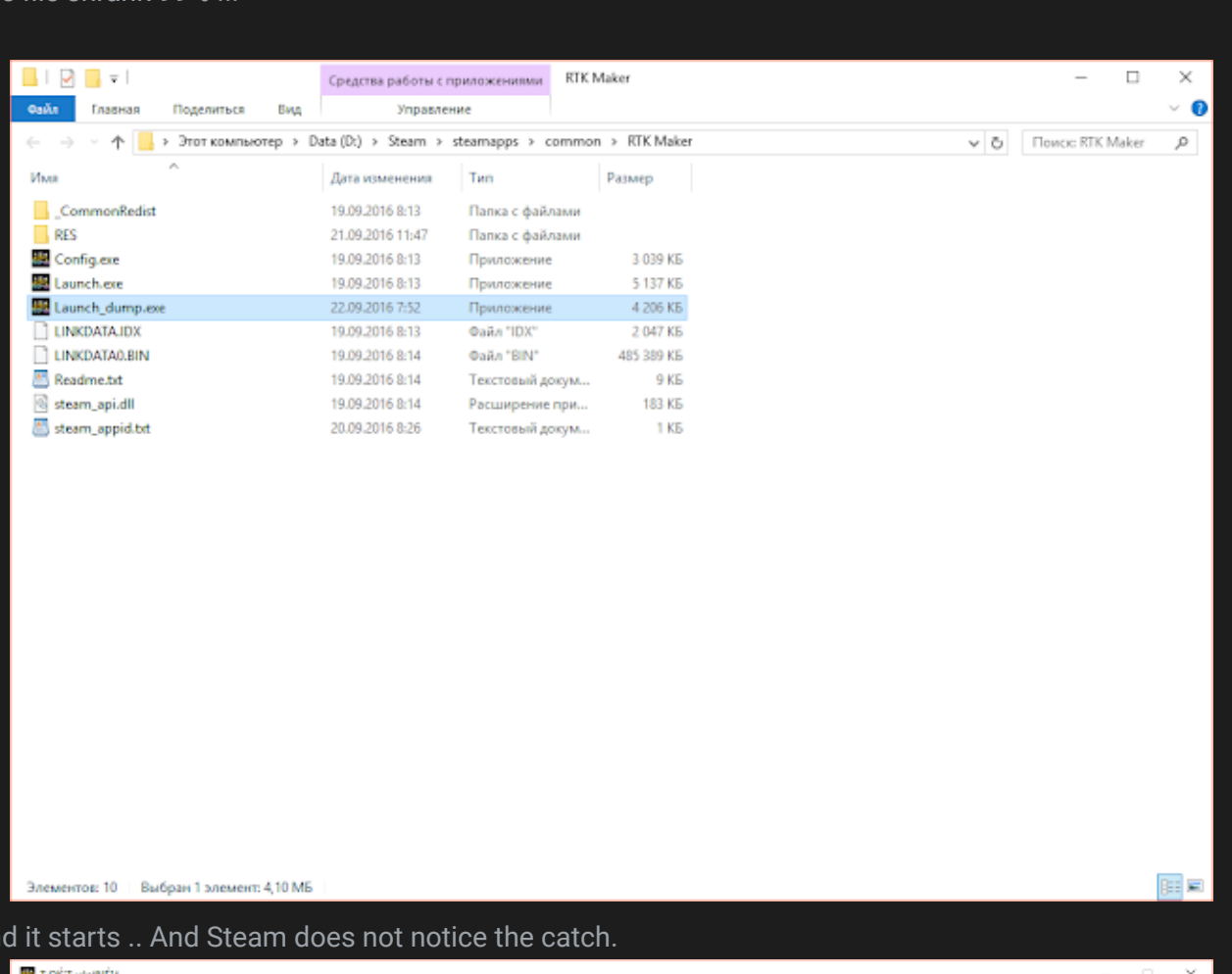


PE Rebuilder - Rebuild Status

```
Visit us http://www.uinc.ru/ for updates

Starting to rebuild Launch_dump.exe...
File Size: 295247872 bytes
------------------------------
Rebuilding...Done!
Validate PE Image...Done!
------------------------------
File Size: 4306436 bytes
File minimized to: 99%
Rebuilding finished.
```

The file shrank 99% ...

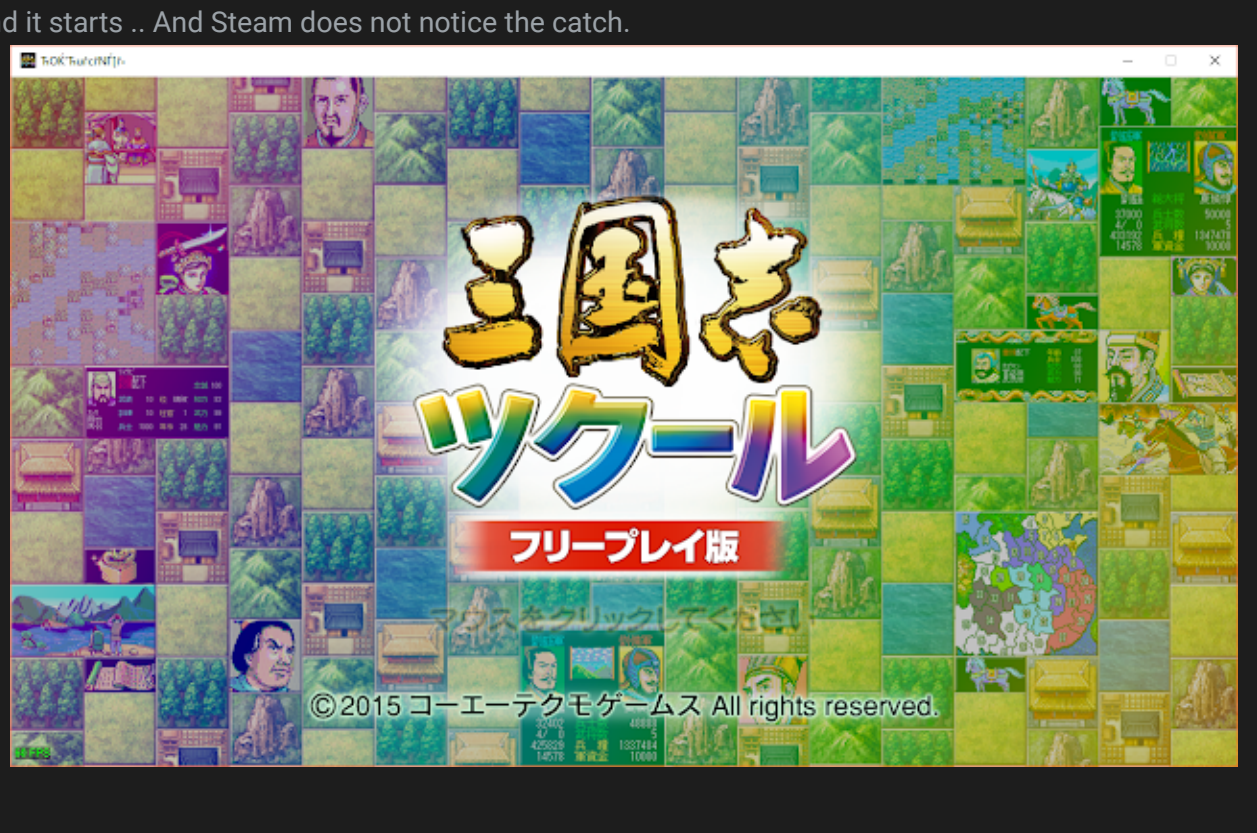


Этот компьютер › Data (D:) › Steam › steamapps › common › RTK Maker

Имя	Дата изменения	Тип	Размер
_CommonRedist	19.09.2016 8:13	Папка с файлами	
RES	21.09.2016 11:47	Папка с файлами	
Config.exe	19.09.2016 8:13	Приложение	3 039 КБ
Launch.exe	19.09.2016 8:13	Приложение	5 137 КБ
Launch_dump.exe	22.09.2016 7:52	Приложение	4 206 КБ
LINKDATA.IDX	19.09.2016 8:13	Файл "IDX"	2 047 КБ
LINKDATA0.BIN	19.09.2016 8:14	Файл "BIN"	485 389 КБ
Readme.txt	19.09.2016 8:14	Текстовый докум...	9 КБ
steam_api.dll	19.09.2016 8:14	Расширение при...	183 КБ
steam_appid.txt	20.09.2016 8:26	Текстовый докум...	1 КБ

Элементов: 10 Выбран 1 элемент: 4,10 МБ

And it starts .. And Steam does not notice the catch.



To be continued...