Rustock.C – Unpacking a Nested Doll

blog.threatexpert.com/2008/05/rustockc-unpacking-nested-doll.html

Rustock.C employs a "Nested Doll Principle"

Unpacking Rustock.C is a challenging task. If you are tired of boring crosswords or Sudoku puzzles and feel like your brain needs a real exercise, think about reversing Rustock.C - satisfaction (or dissatisfaction, depending on the result) is guaranteed.

Rustock.C employs a "Nested Doll Principle"

Rustock.C story began a week ago – when one AV vendor has publicly disclosed the new details about the latest variant of Rustock. As soon as the sample of Rustock.C has been obtained, many researchers started their journey into the center of the rootkit.

First quick look at the driver code reveals a simple decoder. In spite of being simple, it is still a good idea to debug it to see what exactly it produces on its output.

In order to debug a driver, different malware researchers prefer different tools – in our case let's start from WinDbg configured to debug a VMWare session running in debug mode. For more details of this set up, please read this article.

The very first question one might ask is how to put a breakpoint into the very beginning of the driver code?

Some researchers would hook IopLoadDriver() in the kernel to intercept the code before it jumps into the driver, in order to step in it by slowly tracing single instructions.

A simple known trick however, is to build a small driver (and keep it handy) with the first instruction being “int 3”. Once such driver is loaded, the debugger will pop up with the Debug Breakpoint exception. Stepping out from that place leads back into the kernel’s IopLoadDriver() function – right into the point that follows the actual call to the driver. Now, the actual call instruction address is known - a new breakpoint needs to be placed in it.

With the new breakpoint in place, it is time to load Rustock.C driver in the virtual environment controlled by the debugger. Once loaded, the debugger breaks at the call instruction in kernel’s IopLoadDriver(). Stepping into the driver, placing a new breakpoint at the end of its decoder and letting it run until it hits that breakpoint allows to unpack the code that was hidden under that decoder.

The first-layer decoder reveals us a code with a myriad of fake instructions, blocks of code that do nothing, random jumps from one place to another – a huge maze created with only one purpose – to complicate threat analysis by obfuscating and hiding the truly malicious code.

Tracing that code within debugger might be easier with the disassembly listing of that code in the user mode.

One way to get that listing is to reconstruct the driver as a PE-executable by resetting the DLL bit in its PE-header characteristics and changing its subsystem from Native (0x01) to Windows GUI (0x02) to make debugger happy to load it. Another way is to reconstruct a normal PE-executable by building and compiling an Assembler program that includes the top-level Rustock’s decryptor followed by a large stub of encoded data simply copied from the original driver code.

Buidling a PE-executable equivalent of the Rustock.C driver helps to study the code behind the first-layer decoder. Such program can now be loaded into a user-mode debugger, such as OllyDbg, the first-layer decoder can now be debugged in the user mode to unpack the code behind it. Once unpacked, the entire process can be dumped and reloaded into the disassembler.

At this point of analysis, the code behind the first-layer decoder reveals interesting occurrences of DRx registers manipulations, IN/OUT instructions, “sidt/lidt” instructions, and some other interesting code pieces - for example a code that parses an MZ/PE header:

```
00011C0A cmp word ptr [eax], ‘ZM’
00011759 mov bx, [eax+3Ch]
00011E31 cmp dword ptr [eax+ebx], 'EP'
```

The code in general now looks like “spaghetti” – and still, it’s just a second-layer decryptor. The picture below shows you its execution flow – every grey “box” in it represents a stand-alone function:

Placing the breakpoints for all the “interesting” instructions in the driver code is a good idea. The addresses need to offset by a difference between the driver’s entry point reported with a kernel debugger and the entry point of the driver’s PE-executable equivalent, as reported by the user mode debugger.

With the new breakpoints in place, the code will firstly break on the instruction that searches for an MZ-header of the ntkrnlpa.exe:

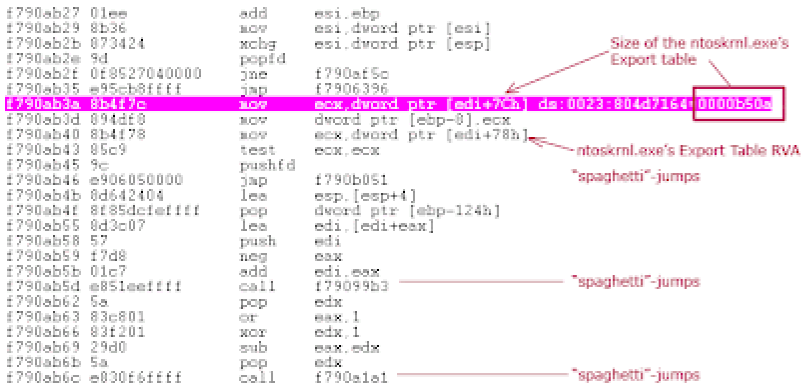
cmp word ptr [eax], ‘ZM’

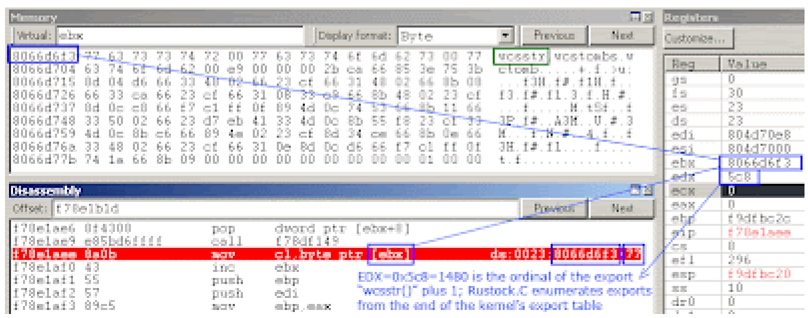
In order to find the image base of ntkrnlpa.exe, Rustock.C looks up the stack to find the return address inside ntkrnlpa.exe. It rounds that address up and starts sliding it backwards by the amount of the section alignment until it reaches the image base of ntkrnlpa.exe.

Once the start of ntkrnlpa.exe is found, the driver then parses its PE-header, locates and parses the export table.

Previous variants of Rustock contained explicit imports from ntkrnlpa.exe. This time, Rustock.C obtains kernel’s exports dynamically, by parsing its memory image – the same trick was widely used by the user-mode malware in the past, when the kernel32.dll’s exports were dynamically obtained during run-time by using the hash values of the export names.

The fragment of Rustock’s second-layer decryptor below parses kernel’s export table:





Now that it knows kernel exports, the driver calls ExAllocatePoolWithQuotaTag() to allocate 228,381 bytes in the non-paged pool (tagged as “Info@”).

The rootkit code then copies itself into that pool and jumps in it to continue its execution from that place.

During the execution, Rustock.C repeats the same trick again – it allocates another 278,528 bytes in the non-paged pool, copies itself into it and transfers there control. This way, the code of the driver "migrates" from one memory location to another. While the "abandoned" areas preserve the severely permutated code, and thus, not easily suitable for scanning, the addresses of the newly allocated areas in the non-paged pool cannot be predicted. Thus, even if the infected driver and its address range in the kernel are established, it is still not clear where the final "detectable" form of Rustock.C code is located.

Following memory allocation tricks, Rustock employs “lidt/sidt” instructions to patch IDT. Executing “lidt” in WinDbg might crash the operating system in the virtual machine. Therefore, “lidt” instruction needs to be skipped (by patching EIP with the address of the next instruction).

Another set of instructions that are better to be skipped with the debugger, are DRx-registers manipulations. By zeroing the debug registers Dr0-Dr3 and the debug control register DR7, the rootkit might attempt to cause trouble for SoftIce – any suspicious instructions need to be skipped for safety reasons.

Following that, Rustock.C driver reads the configuration of devices on a PCI bus by using IN/OUT instructions with the PCI_CONFIG_ADDR and PCI_CONFIG_DATA constants. It then starts a few nested loops to read certain data from the devices attached to a PCI bus.

The read data is then hashed with the purpose of creating a footprint that uniquely identifies hardware of the infected host.

Debugging the Rustock.C driver is easier if the successful code execution path is saved into a map (e.g. a hand-written one). Every successfully terminated loop should be reflected in that map. The relative virtual addresses recorded in it allow skipping long loops when the code is analysed again from the beginning – they should be considered “the milestones” of the code flow. If a wrong move crashes the system – the virtual machine needs to be reverted to a clean snapshot, debugger restarted, and the entire debugging process repeated by using the successful “milestones” from the map.

The map of the execution “milestones” should tell what to skip, when to break, what to patch, where to jump – in order to navigate the code successfully through all the traps that the authors of Rustock has set against emulators, debuggers, run-time code modifications, etc.

Whenever the driver attempts to access data at a non-existing address, the code needs to be unwound backwards to establish the reason why the address is wrong. In most cases, following the logics of the code helps to understand what values should replace the wrong addresses.

For example, at one point of execution, Rustock.C driver crashes the session under WinDbg by calling the following instruction while the contents of ESI is not a valid address:

```
mov esi, dword ptr [esi]
```

In order to “guide” the code through this crash, the driver needs to be re-analysed from the very beginning to check if this instruction is successfully called before the failure and if it does, what the valid contents of ESI is at that moment of time.

As stated above, the PE-executable equivalent of the driver loaded into the user-mode debugger and disassembler helps to navigate through the code, search instructions in it, search for the code byte sequences, place comments - a good helper for the kernel debugging.

The code of Rustock.C debugged at this stage is a 2nd-layer decryptor that will eventually allocate another buffer in the non-paged pool where it will decrypt the final, but still, ridiculously permutated “spaghetti” code of the driver – this time, with the well-recognizable strings, as shown in the following dumps:

```
0000 0000 0000 0000 58DB 0300 44DB 0300 30DB 0300 1ADB 0300 ........X...D...0.......
04DB 0300 70DB 0300 0000 0000 7ED2 0300 8AD2 0300 A2D2 0300 ....p.......~...........
B8D2 0300 D0D2 0300 F2D2 0300 06D3 0300 1CD3 0300 32D3 0300 ....................2...
48D3 0300 5AD3 0300 6AD3 0300 78D3 0300 90D3 0300 9AD3 0300 H...Z...j...x...........
AED3 0300 BCD3 0300 DAD3 0300 FED3 0300 1CD4 0300 2CD4 0300 ....................,...
3AD4 0300 48D4 0300 62D4 0300 7AD4 0300 8ED4 0300 A4D4 0300 :...H...b...z...........
AED4 0300 B8D4 0300 C2D4 0300 D8D4 0300 ECD4 0300 FAD4 0300 ........................
06D5 0300 24D5 0300 42D5 0300 62D5 0300 84D5 0300 A6D5 0300 ....$...B...b...........
C6D5 0300 DED5 0300 EAD5 0300 FAD5 0300 04D6 0300 0ED6 0300 ........................
22D6 0300 3ED6 0300 56D6 0300 6AD6 0300 86D6 0300 92D6 0300 "...>...V...j...........
AED6 0300 74D2 0300 D0D6 0300 ECD6 0300 04D7 0300 16D7 0300 ....t...................
26D7 0300 36D7 0300 4CD7 0300 5ED7 0300 74D7 0300 88D7 0300 &...6...L...^...t.......
A0D7 0300 B4D7 0300 BED7 0300 DAD7 0300 F4D7 0300 04D8 0300 ........................
14D8 0300 28D8 0300 44D8 0300 56D8 0300 68D8 0300 78D8 0300 ....(...D...V...h...x...
8CD8 0300 A6D8 0300 BCD8 0300 C0D8 0300 D6D8 0300 E8D8 0300 ........................
FAD8 0300 0AD9 0300 18D9 0300 2ED9 0300 4AD9 0300 5CD9 0300 ................J...\...
70D9 0300 88D9 0300 9ED9 0300 B0D9 0300 C0D9 0300 CED9 0300 p.......................
EED9 0300 0ADA 0300 1ADA 0300 30DA 0300 40DA 0300 60DA 0300 ............0...@...`...
7CDA 0300 96DA 0300 AADA 0300 BEDA 0300 D0DA 0300 E0DA 0300 |.......................
60D2 0300 54D2 0300 48D2 0300 3CD2 0300 30D2 0300 C6D6 0300 `...T...H...<...0.......
1CD2 0300 0000 0000 0000 0000 2CD1 0C47 0000 0000 0200 0000 ............,..G........
5000 0000 FC31 0000 FC09 0000 5253 4453 387E ABBB 70A5 8B4A P....1......RSDS8~..p..J
91F3 0940 582C 6E7E 0300 0000 5A3A 5C4E 6577 5072 6F6A 6563 ...@X,n~....Z:\NewProjec
7473 5C73 7061 6D62 6F74 5C72 7573 746F 636B 2E63 5C64 7269 ts\spambot\rustock.c\dri
7665 725C 6173 6D5F 5C64 7269 7665 722E 7064 6200 0000 0000 ver\asm_\driver.pdb.....
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 ........................
```

```
7200 7600 6500 7200 0000 5C00 5300 7900 7300 7400 6500 6D00 r.v.e.r...\.S.y.s.t.e.m.
5200 6F00 6F00 7400 5C00 5300 7900 7300 7400 6500 6D00 3300 R.o.o.t.\.S.y.s.t.e.m.3.
3200 5C00 6E00 7400 6400 6C00 6C00 2E00 6400 6C00 6C00 0000 2.\.n.t.d.l.l...d.l.l...
5C00 5300 7900 7300 7400 6500 6D00 5200 6F00 6F00 7400 0000 \.S.y.s.t.e.m.R.o.o.t...
4900 6D00 6100 6700 6500 5000 6100 7400 6800 0000 2E74 6578 I.m.a.g.e.P.a.t.h....tex
7400 4578 5261 6973 6545 7863 6570 7469 6F6E 002E 7465 7874 t.ExRaiseException..text
0050 4147 4500 FFFF FFFF 6B27 4781 4124 4881 494E 4954 0074 .PAGE.....k'G.A$H.INIT.t
0063 0070 0069 0070 002E 0073 0079 0073 0000 002E 6461 7461 .c.p.i.p...s.y.s....data
002E 7465 7874 006E 0064 0069 0073 002E 0073 0079 0073 0000 ..text.n.d.i.s...s.y.s..
006E 6469 732E 7379 7300 7700 6100 6E00 6100 7200 7000 2E00 .ndis.sys.w.a.n.a.r.p...
7300 7900 7300 0000 5200 5500 5300 5400 4F00 4300 4B00 0000 s.y.s...R.U.S.T.O.C.K...
5400 4300 5000 4900 5000 0000 5400 4300 5000 4900 5000 5F00 T.C.P.I.P...T.C.P.I.P._.
5700 4100 4E00 4100 5200 5000 0000 4E64 6973 5265 6769 7374 W.A.N.A.R.P...NdisRegist
6572 5072 6F74 6F63 6F6C 004E 6469 7344 6572 6567 6973 7465 erProtocol.NdisDeregiste
7250 726F 746F 636F 6C00 4E64 6973 416C 6C6F 6361 7465 4275 rProtocol.NdisAllocateBu
6666 6572 004E 6469 7341 6C6C 6F63 6174 6542 7566 6665 7250 ffer.NdisAllocateBufferP
6F6F 6C00 4E64 6973 416C 6C6F 6361 7465 5061 636B 6574 004E ool.NdisAllocatePacket.N
6469 7341 6C6C 6F63 6174 6550 6163 6B65 7450 6F6F 6C45 7800 disAllocatePacketPoolEx.
4E64 6973 4672 6565 5061 636B 6574 004E 6469 734D 5265 6769 NdisFreePacket.NdisMRegi
7374 6572 4D69 6E69 706F 7274 004E 6469 734F 7065 6E41 6461 sterMiniport.NdisOpenAda
7074 6572 004E 6469 7343 6C6F 7365 4164 6170 7465 7200 4E64 pter.NdisCloseAdapter.Nd
6973 494D 436F 7079 5365 6E64 5065 7250 6163 6B65 7449 6E66 isIMCopySendPerPacketInf
6F00 4E64 6973 494D 436F 7079 5365 6E64 436F 6D70 6C65 7465 o.NdisIMCopySendComplete
5065 7250 6163 6B65 7449 6E66 6F00 4E64 6973 5363 6865 6475 PerPacketInfo.NdisSchedu
6C65 576F 726B 4974 656D 004B 6652 656C 6561 7365 5370 696E leWorkItem.KfReleaseSpin
4C6F 636B 004B 6549 6E69 7469 616C 697A 6545 7665 6E74 002E Lock.KeInitializeEvent..
7465 7874 006E 746F 736B 726E 6C00 5274 6C47 6574 5665 7273 text.ntoskrnl.RtlGetVers
696F 6E00 6B73 2E73 7973 0076 6964 656F 7072 742E 7379 7300 ion.ks.sys.videoprt.sys.
776D 696C 6962 2E73 7973 0068 616C 2E64 6C6C 006E 746F 736B wmilib.sys.hal.dll.ntosk
726E 6C2E 6578 6500 6600 7600 6500 7600 6F00 6C00 0000 5C00 rnl.exe.f.v.e.v.o.l...\.
5200 6500 6700 6900 7300 7400 7200 7900 5C00 4D00 6100 6300 R.e.g.i.s.t.r.y.\.M.a.c.
6800 6900 6E00 6500 5C00 5300 7900 7300 7400 6500 6D00 5C00 h.i.n.e.\.S.y.s.t.e.m.\.
4300 7500 7200 7200 6500 6E00 7400 4300 6F00 6E00 7400 7200 C.u.r.r.e.n.t.C.o.n.t.r.
6F00 6C00 5300 6500 7400 5C00 5300 6500 7200 7600 6900 6300 o.l.S.e.t.\.S.e.r.v.i.c.
6500 7300 0000 5C00 5200 6500 6700 6900 7300 7400 7200 7900 e.s...\.R.e.g.i.s.t.r.y.
5C00 4D00 6100 6300 6800 6900 6E00 6500 5C00 5300 7900 7300 \.M.a.c.h.i.n.e.\.S.y.s.
7400 6500 6D00 5C00 4300 7500 7200 7200 6500 6E00 7400 4300 t.e.m.\.C.u.r.r.e.n.t.C.
6F00 6E00 7400 7200 6F00 6C00 5300 6500 7400 5C00 4300 6F00 o.n.t.r.o.l.S.e.t.\.C.o.
6E00 7400 7200 6F00 6C00 5C00 5300 6100 6600 6500 4200 6F00 n.t.r.o.l.\.S.a.f.e.B.o.
6F00 7400 5C00 4D00 6900 6E00 6900 6D00 6100 6C00 0000 4D00 o.t.\.M.i.n.i.m.a.l...M.
6900 6300 7200 6F00 7300 6F00 6600 7400 2000 4300 6F00 7200 i.c.r.o.s.o.f.t. .C.o.r.
7000 4D00 6900 6300 7200 6F00 7300 6F00 6600 7400 AE00 2000 p.M.i.c.r.o.s.o.f.t... .
5700 6900 6E00 6400 6F00 7700 7300 AE00 4D00 6900 6300 7200 W.i.n.d.o.w.s...M.i.c.r.
6F00 7300 6F00 6600 7400 2800 5200 2900 2000 5700 6900 6E00 o.s.o.f.t.(.R.). .W.i.n.
6400 6F00 7700 7300 2000 2800 5200 2900 0013 E645 810B E645 d.o.w.s. .(.R.)....E...E
8100 E645 81F3 E545 81EC E545 8149 006D 0061 0067 0065 0050 ...E...E...E.I.m.a.g.e.P
0061 0074 0068 0000 0047 0072 006F 0075 0070 0000 0053 0074 .a.t.h...G.r.o.u.p...S.t
0061 0072 0074 0000 0054 0079 0070 0065 0000 0025 0077 0073 .a.r.t...T.y.p.e...%.w.s
005C 0025 0077 0073 0000 0080 8480 8480 8480 8480 8880 8880 .\.%.w.s................
```

```
8181 8109 1809 0900 0012 0010 1010 1001 0101 0109 0902 0008 |........................
0809 1803 0002 0000 0100 0001 0100 0050 5012 8120 0020 2000 |...............PP..  .  .
0800 0908 0000 0008 0008 0009 0909 0908 0808 0050 5050 5000 |....................PPPP.
0009 0808 0809 085C 0053 0079 0073 0074 0065 006D 0052 006F |.......\.S.y.s.t.e.m.R.o
006F 0074 005C 0073 0079 0073 0074 0065 006D 0033 0032 005C |.o.t.\.s.y.s.t.e.m.3.2.\
0064 0072 0069 0076 0065 0072 0073 005C 0025 0077 0073 002E |.d.r.i.v.e.r.s.\.%.w.s..
0073 0079 0073 0000 005C 0053 0079 0073 0074 0065 006D 0052 |.s.y.s...\.S.y.s.t.e.m.R
006F 006F 0074 005C 0025 0077 0073 0000 006E 746F 736B 726E |.o.o.t.\.%.w.s...ntoskrn
6C00 8767 4681 90CC 4781 1309 4981 AC00 4881 B1FA 4881 026F |l..gF...G...I...H...H..o
4681 FCA4 4681 5507 4981 D13F 4781 5CA1 4681 63D9 4681 C8FA |F...F.U.I..?G.\.F.c.F...
4581 5A77 5175 6572 7953 7973 7465 6D49 6E66 6F72 6D61 7469 |E.ZwQuerySystemInformati
6F6E 005A 7752 6561 6456 6972 7475 616C 4D65 6D6F 7279 005A |on.ZwReadVirtualMemory.Z
7757 7269 7465 5669 7274 7561 6C4D 656D 6F72 7900 5A77 5072 |wWriteVirtualMemory.ZwPr
6F74 6563 7456 6972 7475 616C 4D65 6D6F 7279 005A 7743 7265 |otectVirtualMemory.ZwCre
6174 6554 6872 6561 6400 5A77 5465 726D 696E 6174 6554 6872 |ateThread.ZwTerminateThr
6561 6400 5A77 4F70 656E 5468 7265 6164 005A 7744 7570 6C69 |ead.ZwOpenThread.ZwDupli
6361 7465 4F62 6A65 6374 005A 7744 656C 6179 4578 6563 7574 |cateObject.ZwDelayExecut
696F 6E00 5A77 5365 7445 7665 6E74 005A 7753 6574 496E 666F |ion.ZwSetEvent.ZwSetInfo
726D 6174 696F 6E54 6872 6561 6400 5A77 5265 7375 6D65 5468 |rmationThread.ZwResumeTh
7265 6164 005A 7754 6572 6D69 6E61 7465 5072 6F63 6573 7300 |read.ZwTerminateProcess.
5A77 4372 6561 7465 5573 6572 5072 6F63 6573 7300 5A77 4372 |ZwCreateUserProcess.ZwCr
6561 7465 5468 7265 6164 4578 0077 696E 6C6F 676F 6E2E 6578 |eateThreadEx.winlogon.ex
6500 7365 7276 6963 6573 2E65 7865 00FF FFFF FF3E 3D47 81D1 |e.services.exe.....>=G..
6148 81FF FFFF FFC8 3F46 819A 1F47 812E 7465 7874 004D 6D55 |aH......?F...G..text.MmU
7365 7250 726F 6265 4164 6472 6573 7300 2E64 6174 6100 7300 |serProbeAddress..data.s.
6500 7200 7600 6900 6300 6500 7300 2E00 6500 7800 6500 0000 |e.r.v.i.c.e.s...e.x.e...
FFFF FFFF F192 4881 800A 4881 0000 0000 8160 4681 D299 4781 |......H...H......`F...G.
5C00 4200 6100 7300 6500 4E00 6100 6D00 6500 6400 4F00 6200 |\.B.a.s.e.N.a.m.e.d.O.b.
6A00 6500 6300 7400 7300 5C00 2500 3000 2E00 3800 5800 2D00 |j.e.c.t.s.\.%.0...8.X.-.
2500 3000 2E00 3400 5800 2D00 2500 3000 2E00 3400 5800 2D00 |%.0...4.X.-.%.0...4.X.-.
2500 3000 2E00 3400 5800 2D00 2500 3000 2E00 3800 5800 2500 |%.0...4.X.-.%.0...8.X.%.
3000 2E00 3400 5800 0000 5C00 4400 6500 7600 6900 6300 6500 |0...4.X...\.D.e.v.i.c.e.
5C00 5400 6300 7000 0000 5C00 4400 6500 7600 6900 6300 6500 |\.T.c.p...\.D.e.v.i.c.e.
5C00 5500 6400 7000 0000 5472 616E 7370 6F72 7441 6464 7265 |\.U.d.p...TransportAddre
7373 0043 6F6E 6E65 6374 696F 6E43 6F6E 7465 7874 00FF FFFF |ss.ConnectionContext....
FF75 FA48 8116 1447 8100 0000 0000 0000 0000 0000 0000 0000 |.u.H...G................
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 |........................
0000 0000 0000 0000 0000 0000 0000 0000 0043 7265 6174 6554 |.................CreateT
6872 6561 6400 4C6F 6164 4C69 6272 6172 7941 0047 6574 5072 |hread.LoadLibraryA.GetPr
6F63 4164 6472 6573 7300 FFFF FFFF 4E42 4881 EBA1 4881 0000 |ocAddress.....NBH...H...
```

PS: Special thanks to Frank Boldewin for exchanging his tips and ideas with me.