A Journey to the Center of the Rustock.B Rootkit

A Journey to the Center
of the Rustock.B Rootkit

Version: 1.0

Last Update: 20th January 2007

Author: Frank Boldewin / www.reconstructer.org

Table of Contents

A Journey to the Center of the Rustock.B Rootkit

1 Abstract

"You try to look innocent, but what's behind the curtain?
Whatever you hide or pretend will be detected - this is certain!"

On 27th December 2006 I found a sample of the Rustock.B Rootkit at www.offensivecomputing.net, which was only sparsely analyzed at this time. I was keen to study its behaviour, as I've heard a lot of stories about this infamous Rootkit. Rustock included several techniques to obfuscate the driver which could be stumbling blocks for the researcher. Analyzing the binary was quite fun. Recalling the work I've done over the last few days, it is clear that Rustock is quite different from most other Rootkits I've seen in the past. It is not much because Rustock uses new techniques, but rather because it combines dozens of known tricks from other malware which makes it very effective.

2 Introduction

This paper is divided into two main parts. In the first part I wanted to extract the native Rootkit driver code but without the use of kernel debuggers or other ring0 tools. The second part covers the extraction over the last three stages but much faster and with lesser efforts using the SoftICE debugger. Each part shows various possibilities for solving the different problems facing the researcher when analyzing Rustock. The techniques can also be useful in future reversing sessions. All the tools I've used can be found in the references. Some of them are free and others again are commercial, like IDA Pro. Further all the binary dumps and IDA .idb files from each stage are included in the package with this paper. Use caution when reproducing the work described here. Consider employing a virtual machine like VMware or Virtual PC and perform the analysis on an isolated network to avoid the damage that could be caused by the Rootkit. Use at your own risk!

3 Stage 1 - Drop from the Mother ship

First thing we have to do is to browse into the directory **stage1** and unzip the file **`Rustock-Rootkit.B-Password-infected.zip`**. The zipfile password is "**`infected`**". Now we are ready to start.

Load the unpacked file **`rustock.exe`** into **`Ollydbg`**.

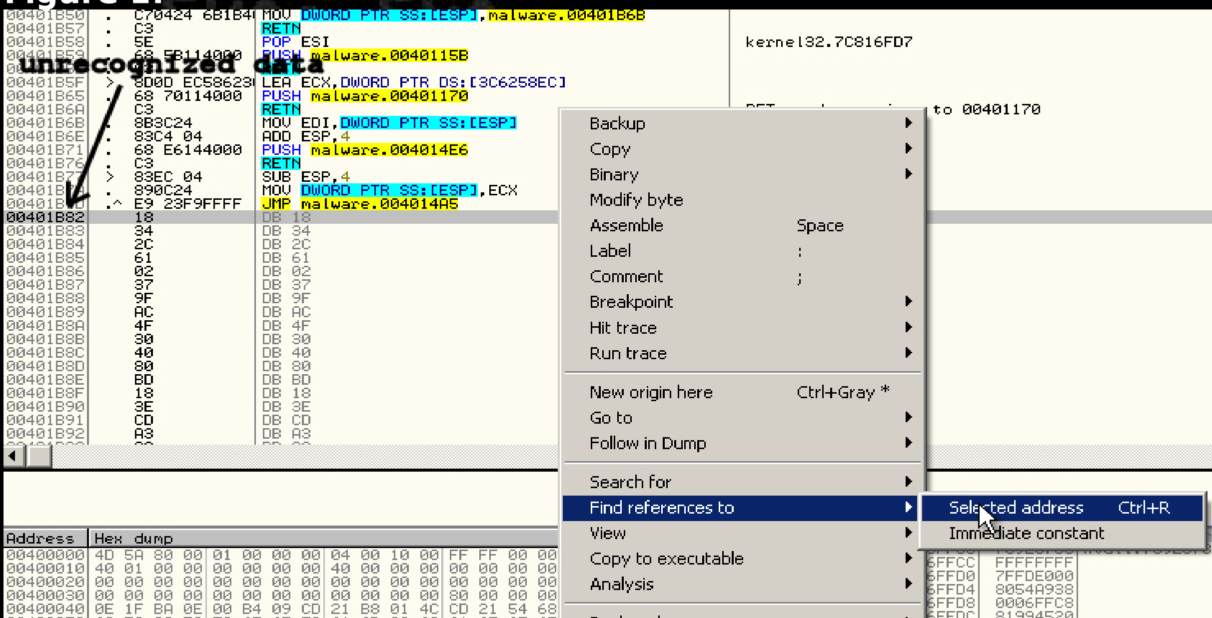
Right click and select:

`Search for ---> All referenced text strings`

The limited result may indicate that the binary is packed or obfuscated in some way. The best idea in this case is often to employ a tool like **`PEID`** or **`Protection-ID`**. Unfortunately, this time both tools cannot determine the Compiler/Packer/Protector. It could be that a proprietary obfuscation technique has been used. One of the indicators that a file is packed or obfuscated often is some unrecognized data, thus we start scrolling down from the entry point at 0x401000 and strike a bonanza at address 0x401b82 (Figure 1). Place the cursor at this position and right click

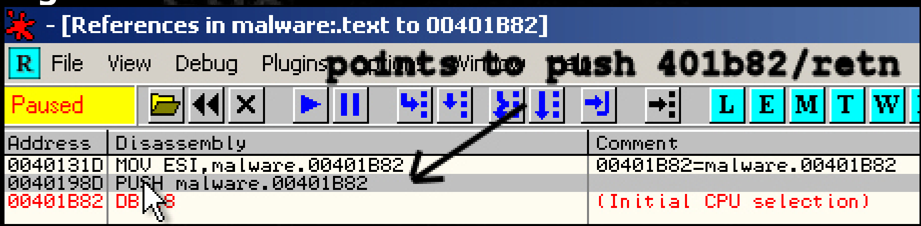
`Find references to ---> Selected address` (or just **`CTRL+R`**)

Figure 1:



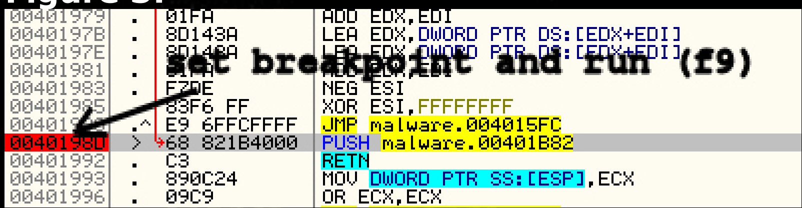
The references window (Figure 2) should show three hits now. We choose the second one at address 0x40198d, because it's the most likely reference that jumps directly to the unrecognized data (**`push 0x401b82/retn`** is the same as **`call 0x401b82`**). A good chance for us, that this is the end of the obfuscation code.

Figure 2:



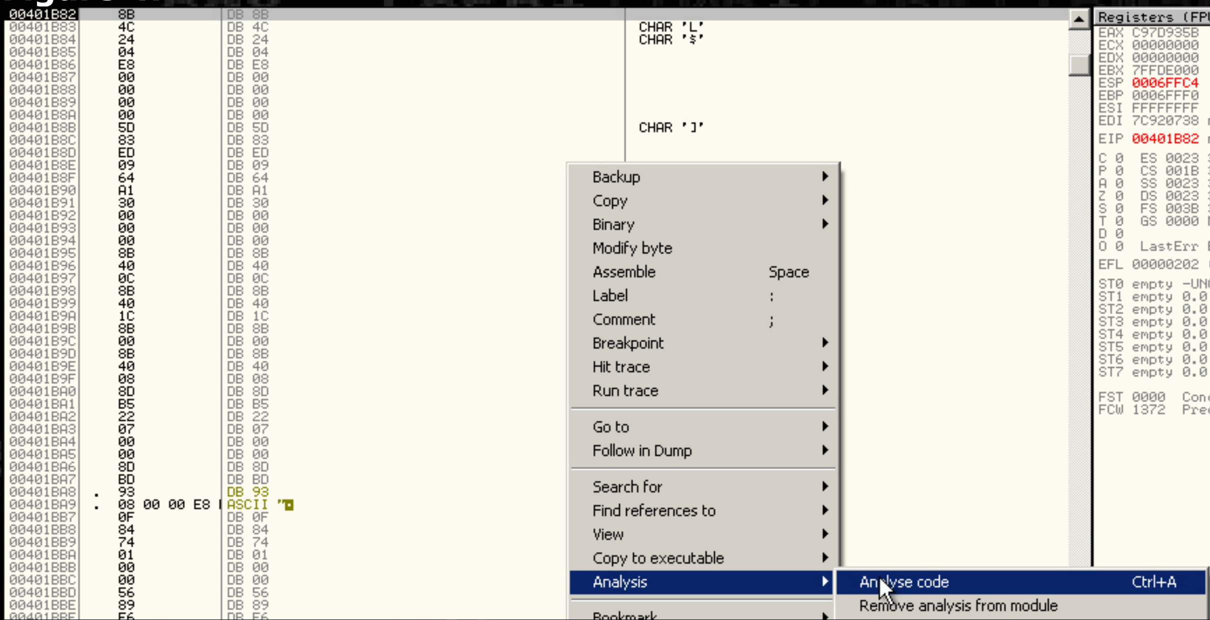
So why not setting a breakpoint (**F2** at cursor position) here and see what happens after we **Run** (**F9**) the code? (Figure 3)

Figure 3:



After running the code, a breakpoint occurred at our address as expected. Now press **Step into** (**F7**) two times and we should be located at address 0x401b82 (Figure 4).

Figure 4:



Hm, still doesn't look like valid code, right?

No problem, right click

Analysis ---> Analyse code (or just **CTRL+A**)

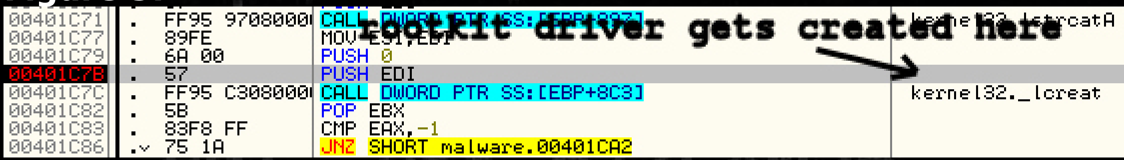
and the result should look much better.

Use **Step Over** (**F8**) until you passed **call 0x402092** at 0x401bac, which does some API importing stuff). Ollydbg now should be able to show you the API names to the relative addresses, e.g.

```
CALL DWORD PTR:SS[EBP+8c3] = kernel32._lcreat
```

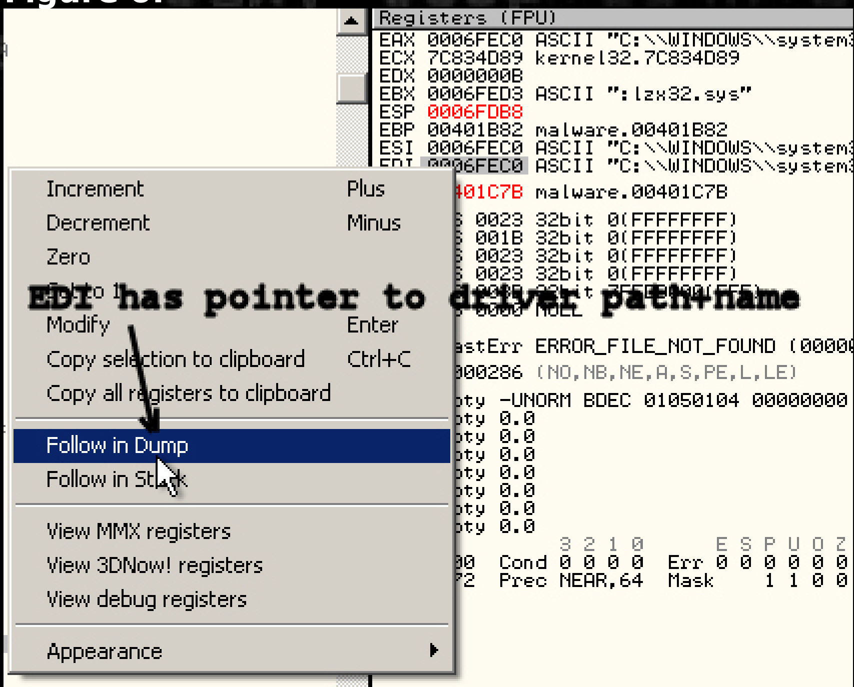
After reading some code, we notice that a file called “lzx32.sys” is created at 0x401c7c (Figure 5). It is fairly telltale that this is the kernel mode driver. So let us set another breakpoint at 0x401c7b (**F2**) and **Run** (**F9**) the code again.

Figure 5:



After the breakpoint occurred select **EDI** in the **Registers** window, then right click **Follow in Dump** (Figure 6)

Figure 6:



The register **EDI** points to the following string:

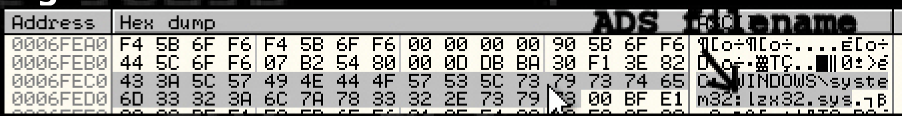
C:\windows\system32:lzx32.sys

Confused of the ":" instead of "\" in the pathname?

Is it a mistake? Surely not, the driver just is created as Alternative Data Stream (ADS). A nice method to hide the driver from easy detection, because neither Windows Explorer nor cmd.exe will show you ADS streams. This is only possible with special tools like Sysinternals **streams.exe**. To simplify our analysis it's a good idea to let the code create a normal file. Therefore select the ": = 0x3a" in the memory map (Figure 7) and patch it to "\ = 0x5c" using right click

Binary ---> Edit (or just **CTRL+E**)

Figure 7:



Lastly **Step Over** (**F8**) until the file was written (**_lwrite**) and closed (**_lclose**) at address 0x401cc7.

As the aim of this paper is to describe how to deobfuscate/unpack the driver, Ollydbg can be closed now and the first stage was mastered.

As a goody here is a short description for the folks, who are interested what else is going on in the dropper code of Rustock.

1. If API Import fails, connect to:
   http://208.66.194.158/index.php?page=main
   Delete lzx32.sys and Exit
2. Try to open the service control manager (if it fails go to 5)
3. Create Service PE386 (if it fails go to 5)
4. Start Service PE386 (if it fails go to 5 – if ok go to 7)
5. Create service registry entries by hand as lzx32
6. Invoke ZwLoadDriver
7. Inject Rustock.exe into the Explorer process, create a remote thread and Exit

4 Stage 2 – Kernel code vs PE-Tools

Welcome to Stage 2! After we have successfully detached the driver we load it into IDA and see what is going on there. You can find my detached driver code and .idb file in the directory "stage1".
Hm, after running down some pages, it seems that there is more code obfuscation fun waiting for us. ;)
When we try to load this binary into Ollydbg now, a Message box pops up and tells us something like this:

`File "original-dropped-lzx32_sys.sys" is a DLL. Windows can't execute DLLs directly. Launch LOADDLL.EXE?`

Usually after clicking '**`yes`**' the DLL gets loaded by **`LOADDLL`** and stops at its entry point. But after selecting **`Run`** (**`F9`**) the next Message box appears and informs us about the following:

`Entry Point Alert`

`Module "ntoskrnl" has entry point outside the code (as specified in the PE-Header). Maybe this file is self-extracting or self-modifying. Please keep it in mind when setting breakpoints!`

And the same Message for HAL.DLL. No problem, but after clicking "**`Ok`**" **`LOADDLL`** terminates with exit code 1001 and that's it. :(

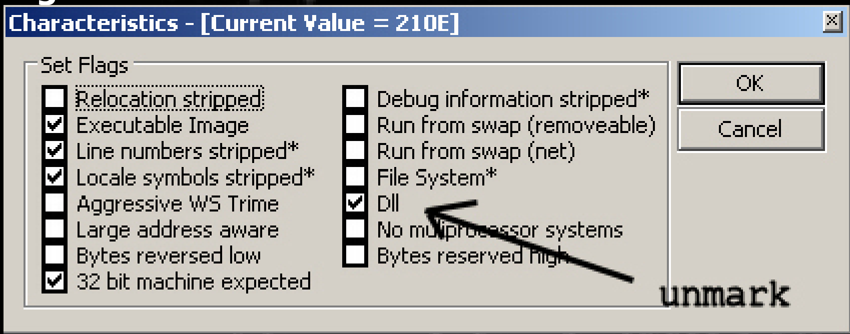
What now?

In these cases the best choice is to "fix" the PE-Files with **`PE-Tools`**.

So fire up PE-Tools and make the following modifications:

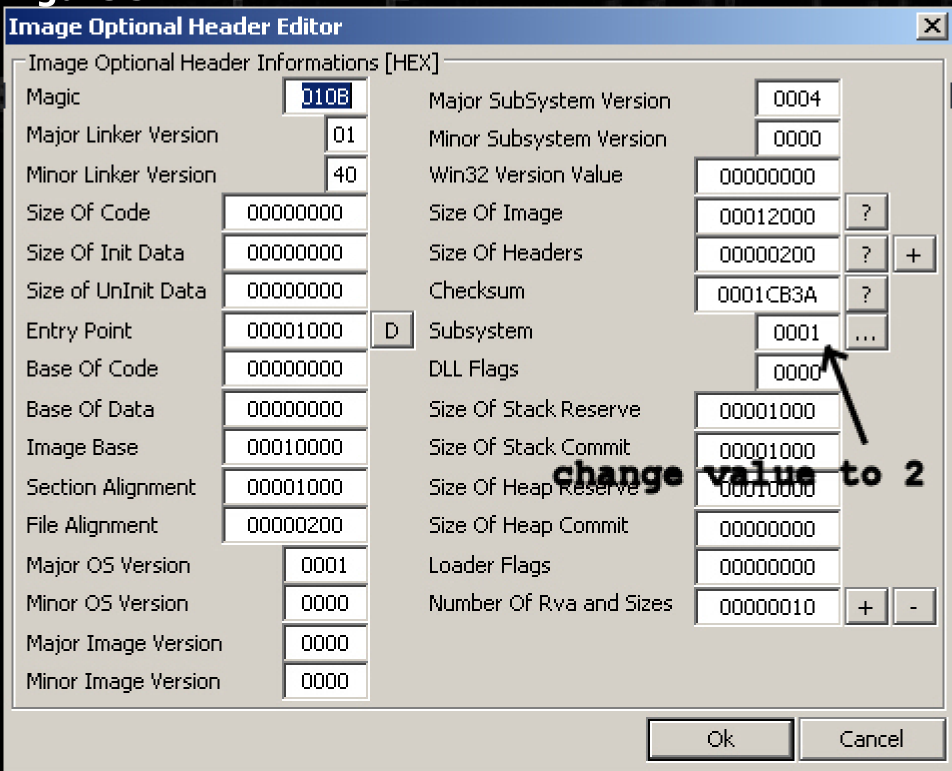
- **`Tools-->PE-Editor-->Select original-dropped lzx32_sys.sys`**
- **`Select "File Header"--->Characteristics--->unmark the "dll" bit---> click OK (Figure 8)`**
- **`Leave "Image File Header Editor"`**

Figure 8:



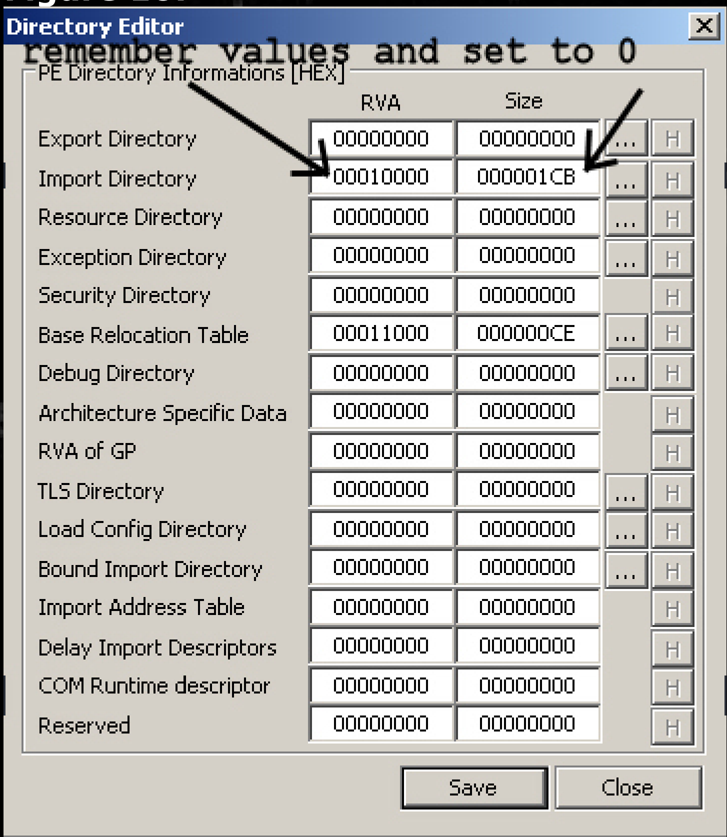
Select "Optional Header" and change the "Subsystem" value to "2" (Windows GUI) ---> click OK (Figure 9)

Figure 9:



Select "Directories"--->"Import Directory" and set its "RVA" and "Size" to "00000000"--->click Save and leave PE-Tools (Remember the old values **0x00010000** and **0x000001cb**. We have to reset these later in order to have a working file!) (Figure 10)

Figure 10:



So what have we done so far?

Before the settings were:

- A DLL
- A Native Executable
- Had Imports from Kernel Libraries NTOSKRNL.EXE and HAL.DLL

Now the settings are:

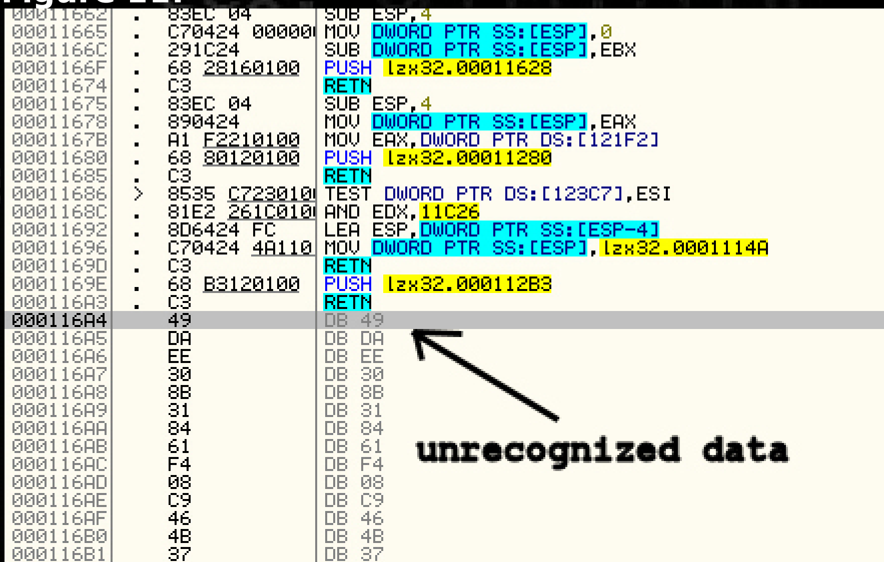
- No DLL
- A Windows GUI Application
- No Imports

Why can we do this?
The answer is easy. As long as the obfuscation code does not expect any special data returned by imported kernel library functions, it does not matter how we declare the PE-FILE. ;)

Therefore, after patching the PE-File behaviour we load up **`original-dropped lzx32_sys.sys`** again and - Eureka!

After running down some pages again we notice some unrecognized data at address 0x116a4 again (Figure 11). Are the bells ringing?

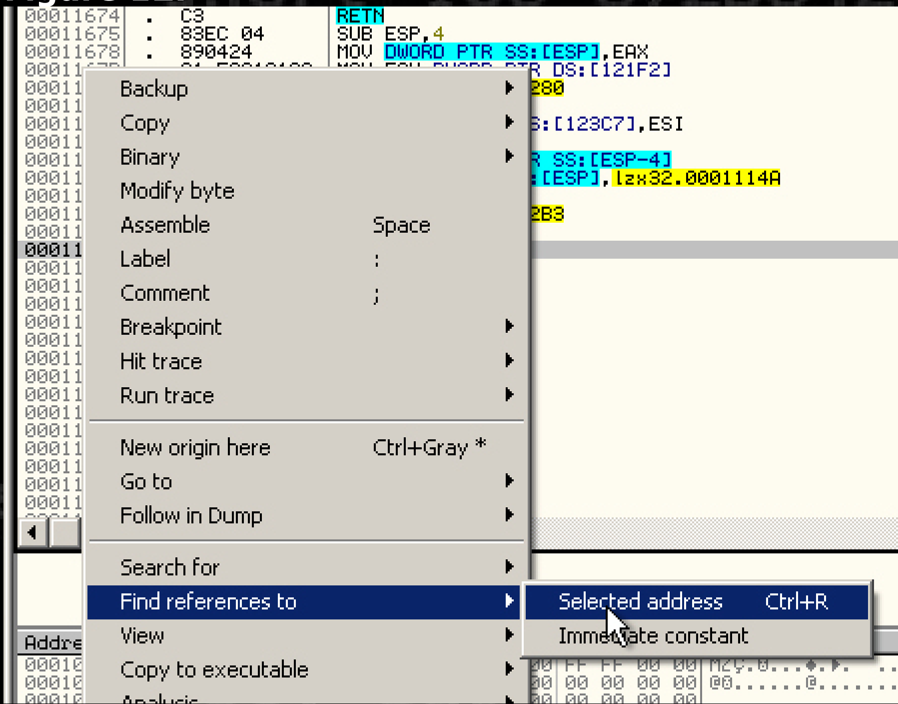
Figure 11:



```
00011662  .  83EC 04          SUB ESP,4
00011665  .  C70424 000000    MOV DWORD PTR SS:[ESP],0
0001166C  .  291C24           SUB DWORD PTR SS:[ESP],EBX
0001166F  .  68 28160100      PUSH lzx32.00011628
00011674  .  C3               RETN
00011675  .  83EC 04          SUB ESP,4
00011678  .  890424           MOV DWORD PTR SS:[ESP],EAX
0001167B  .  A1 F2210100      MOV EAX,DWORD PTR DS:[121F2]
00011680  .  68 80120100      PUSH lzx32.00011280
00011685  .  C3               RETN
00011686  >  8535 C723010     TEST DWORD PTR DS:[123C7],ESI
0001168C  .  81E2 261C010     AND EDX,11C26
00011692  .  8D6424 FC        LEA ESP,DWORD PTR SS:[ESP-4]
00011696  .  C70424 4A110     MOV DWORD PTR SS:[ESP],lzx32.0001114A
0001169D  .  C3               RETN
0001169E  .  68 B3120100      PUSH lzx32.000112B3
000116A3  .  C3               RETN
000116A4     49               DB 49
000116A5     DA               DB DA
000116A6     EE               DB EE
000116A7     30               DB 30
000116A8     8B               DB 8B
000116A9     31               DB 31
000116AA     84               DB 84
000116AB     61               DB 61
000116AC     F4               DB F4
000116AD     08               DB 08
000116AE     C9               DB C9
000116AF     46               DB 46
000116B0     4B               DB 4B
000116B1     37               DB 37
```

Yep, we saw this stuff in the dropper code before. Why do not trying the same trick again? (Figure 12)

Figure 12:



We select address 0x116a4, right click

`Find references to--->Selected address` (Figure 13)

As the first hit in the references looks best (push 0x116a4/retn) we choose this one (Figure 13), set a breakpoint using **`F2`** (Figure 14) at address 0x110ce and `Run` (**`F9`**) the code.

Figure 13:

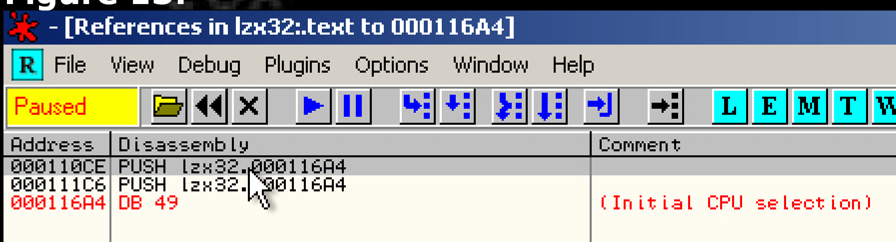
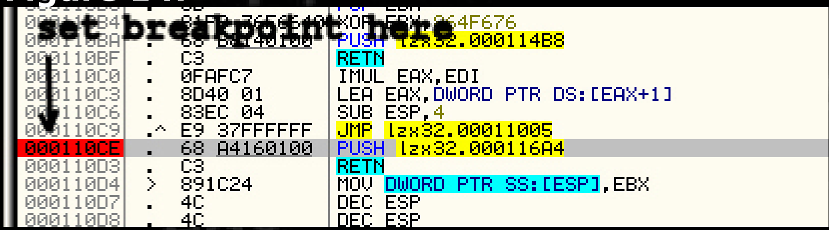
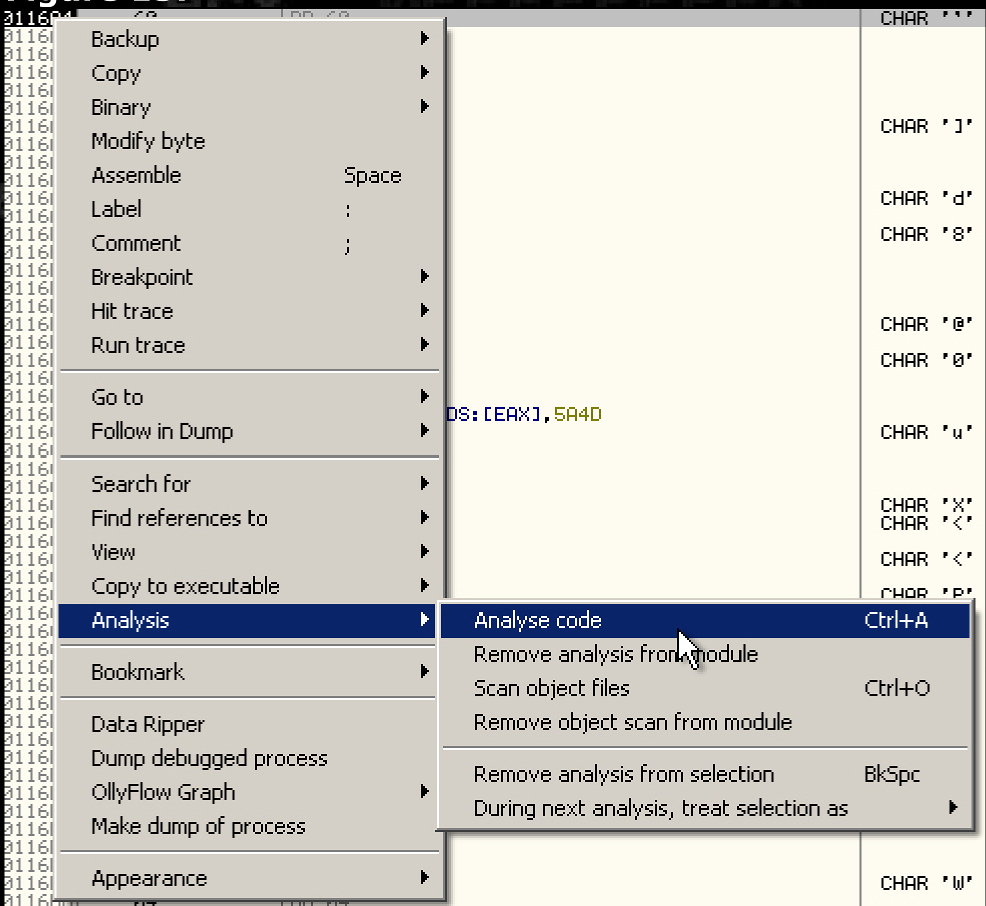


Figure 14:



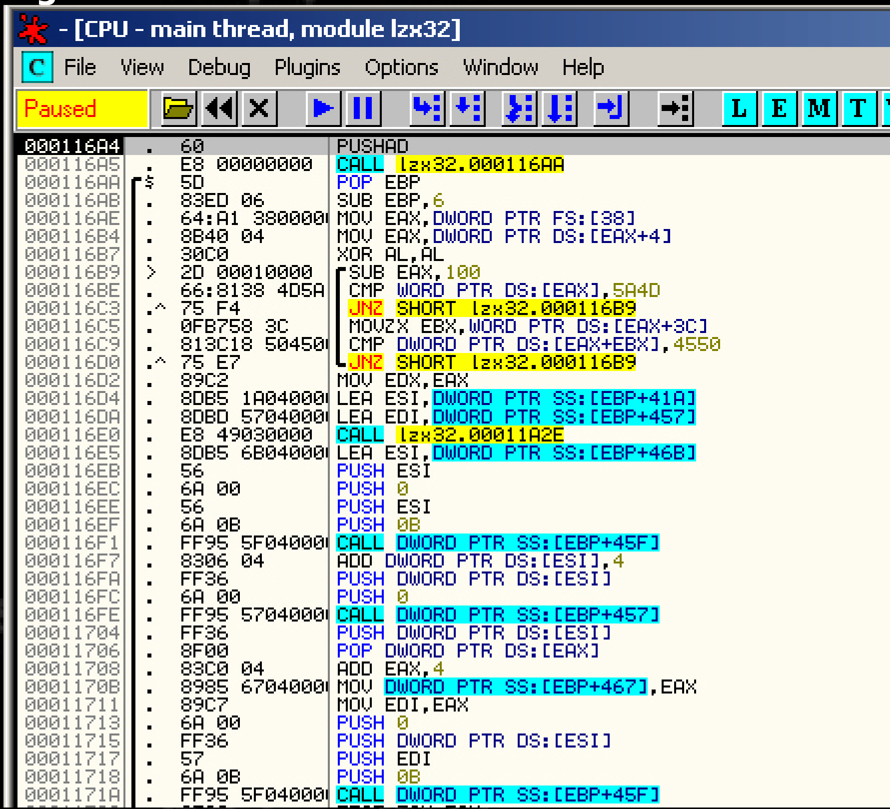
When the breakpoint is reached press **Step into (F7)** two times. We should be arrived at address 0x116a4 (Figure 15).

Figure 15:



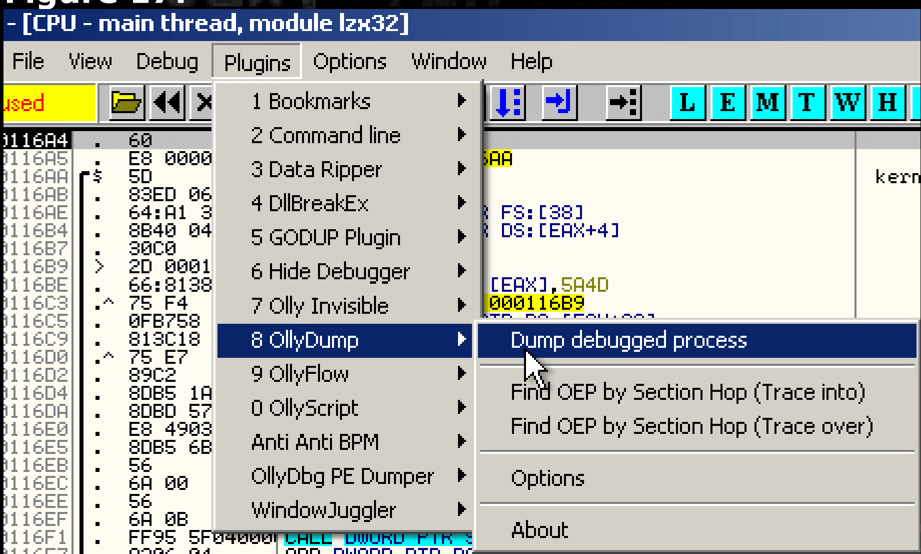
Use hotkey **CTRL+A** to display some human readable code.
As the code looks clearly less obfuscated now (Figure 16), it's time to dump the current state.

Figure 16:



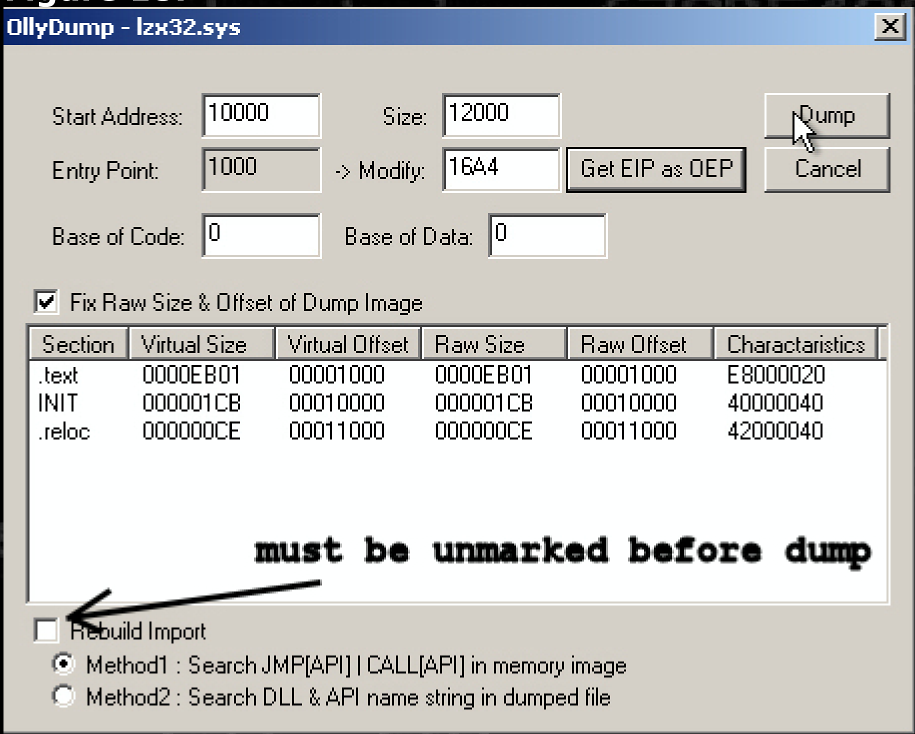
Click Plugins--->OllyDump--->Dump debugged process (Figure 17)

Figure 17:



Unmark "Rebuild Import" and click "Dump" (Figure 18)

Figure 18:



Cool, after saving the dump it is important to reset the old PE-File settings:

- **Setting the "DLL" bit in the "Characteristics" area**
- **Setting the "Subsystem" to "Native" in the "Optional Header" area**
- **Setting the "RVA" and "Size" of the Import Directory field in the "Directories" area to 0x00010000 and 0x000001cb**

Ok, that is it for the second stage.

5 Stage 3 – Naked looks best

For the last stage, load your dumped file from stage 2 into IDA or just use mine: **`stage2/lzx32-unobfuscated.idb`**

As you can see in Figure 19, you can find some comments within the .idb file. This allows a better understanding about the code.

Figure 19:

```
000116A4                 pusha
000116A5                 call    $+5
000116AA                 pop     ebp
000116AB                 sub     ebp, 6          ; standard "what's my current base address" trick
000116AE                 mov     eax, large fs:38h
000116B4                 mov     eax, [eax+4]
000116B7                 xor     al, al
000116B9
000116B9 loc_116B9:                              ; CODE XREF: DllEntryPoint+1F↓j
000116B9                                         ; DllEntryPoint+2C↓j
000116B9                 sub     eax, 100h
000116BE                 cmp     word ptr [eax], 5A4Dh ; MZ
000116C3                 jnz     short loc_116B9
000116C5                 movzx   ebx, word ptr [eax+3Ch]
000116C9                 cmp     dword ptr [eax+ebx], 4550h ; PE
000116D0                 jnz     short loc_116B9 ; Scan for NTOSKRNL base
000116D2                 mov     edx, eax
000116D4                 lea     esi, [ebp+41Ah] ; First Entry is ExAllocatePool
000116DA                 lea     edi, [ebp+457h] ; Buffer for API Addresses
000116E0                 call    sub_11A2E       ; Scan for several APIs
000116E5                 lea     esi, [ebp+46Bh]
000116EB                 push    esi
000116EC                 push    0
000116EE                 push    esi
000116EF                 push    0Bh             ; 0xb = SystemModuleInformation
000116F1                 call    dword ptr [ebp+45Fh] ; ZwQuerySystemInformation
000116F7                 add     dword ptr [esi], 4
000116FA                 push    dword ptr [esi]
000116FC                 push    0
000116FE                 call    dword ptr [ebp+457h] ; ExAllocatePool
00011704                 push    dword ptr [esi]
00011706                 pop     dword ptr [eax]
00011708                 add     eax, 4
0001170B                 mov     [ebp+467h], eax
```

Before I start explaining what the code between 0x116a4 and 0x11abc basically does, let's have a short look at Figure 20.

Again, we have unrecognized data beginning at 0x11afc.

Figure 20:

```
00011AAC loc_11AAC:                              ; CODE XREF: sub_11A44+4F↑j
00011AAC                 add     esi, 4
00011AAF                 inc     ecx
00011AB0                 jmp     short loc_11A77
00011AB2 ; ---------------------------------------------------------------------------
00011AB2
00011AB2 loc_11AB2:                              ; CODE XREF: sub_11A44+66↑j
00011AB2                                         ; sub_11A44+78↓j
00011AB2                 pop     edx
00011AB3                 pop     ebx
00011AB4                 pop     edi
00011AB5                 pop     esi
00011AB6                 leave
00011AB7                 retn    8
00011ABA ; ---------------------------------------------------------------------------
00011ABA
00011ABA loc_11ABA:                              ; CODE XREF: sub_11A44+17↑j
00011ABA                                         ; sub_11A44+36↑j
00011ABA                 xor     eax, eax
00011ABC                 jmp     short loc_11AB2
00011ABC sub_11A44       endp
00011ABC
00011ABC ; ---------------------------------------------------------------------------
00011ABE aExallocatepool db 'ExAllocatePool',0      unrecognized data
00011ACD aExfreepool     db 'ExFreePool',0
00011AD8 aZwquerysystemi db 'ZwQuerySystemInformation',0
00011AF1 a_stricmp       db '_stricmp',0
00011AFA                 align 4
00011AFC                 dd 6 dup(0)
00011B14                 dd 80000000h, 4D000088h, 38905A38h, 4026603h, 81FF7109h
00011B14                 dd 191C2B8h, 0C615C240h, 0E1C09E0h, 0F8BA1Fh, 21CD09B4h
00011B14                 dd 0C04C01B8h, 6968540Ah, 700E2073h, 67676F72h, 63876D61h
00011B14                 dd 4F1F6E47h, 6562E774h, 5F75CFAFh, 44066998h, 3537E4Fh
```

Unfortunately, there are no direct references in the code to this area, like in the two stages before. :(

Now we need a strategy.

We should start reading some code, to get a clue how to solve our problem.

Here's a short description:

- Import some APIs from NTOSKRNL (ExAllocatePool, ExFreePool, ZwQuerySystemInformation, _stricmp)
- Query all running System modules using ZwQuerySystemInformation/Subfunction 0x0b
- Allocate Kernel memory
- Unpacking routine at 0x11788 ---> call sub_117d3 unpacks code to new allocated Kernel memory
- Move unpacked code over packed code area, grab imports from ntoskrnl.exe and hal.dll, destroy PE-Header (MZ, PE, e_lfanew) and rebase API calls
- Free the kernel memory, that is no longer used
- JMP EAX at address 0x117c8 ---> execute the real naked driver

So, how can we grab the real driver without any kernel debugging now?

Why not just ripping the unpacking code at address 0x117d3 and then dumping the whole data as a file?

A good idea especially before the PE-Header gets destroyed and the driver code rebased. ;)

My small C Program called **lzx32-laststage-unpacker** in directory **stage3** exactly does this job (Figure 21).

Figure 21:

```
X:\paper\stage3>lzx32-laststage-unpacker.exe

+----------------------------------------------+
|      Rustock lzx32.sys last stage unpacker    |
| coding Frank Boldewin / www.reconstructer.org |
+----------------------------------------------+

[*] Opening source file lzx32-unobfuscated.sys
[*] Opening destination file lzx32-native-unpacked.sys
[*] Setting filepointer to offset: 0x00001b1b
[*] Reading packed driver data
[*] Unpacking now...
[*] Writing unpacked data to lzx32-native-unpacked.sys
[*] Job done!

X:\paper\stage3>
```

Voila, last but not least we have a clean native driver that can be analyzed very easily now. As a whole analysis of the complete rootkit would go beyond the scope of this paper, here's just a link to an analysis of Rustock.B:

http://www.sarc.com/avcenter/venc/data/backdoor.rustock.b.html#technicaldetails

Basically the paper may end here, but I thought it might be also of interest, how to do the same action like in all 3 stages with a kernel debugger, but faster.

6 Speeddumping with SoftICE+ICEEXT

6.1 Preparation

To fully understand what's going on in the preparation, you need to know that a special function in NTOSKNRL.EXE called IopLoadDriver isn't exported by default (next to others). If exported, this function could be a very useful breakpoint for us.

To solve this problem, we need the proper .pdb file of NTOSKRNL.EXE from the Microsoft server. Further, the downloaded .pdb file need conversion to the proprietary SoftICE format .nms. Normally not a big task, as SoftICE has its own Symbol retriever. The bad news is that this tool always sucks for me. :(

But why despair, if there's another way!

The first thing we have to do is to leech the "Windows Debugging Tools" from the Microsoft website (Link can found in the references) and installing them.

Based on the fact that you have installed Driverstudio/SoftICE as well, edit the file **%systemroot%\system32\drivers\winice.dat** now.

Set **NTSYMBOLS=ON**
Set **LOAD=SystemRoot\ntoskrnl.nms**
Change value **SYM=2048** (default is 512)

If you are not familiar with SoftICE you should read my paper:

The big SoftICE howto (see references)

Or if you are a WinDBG freak, use this one.

Do the following next steps:

```
md %systemroot%\symbols

cd %ProgramFiles%\Debugging Tools for Windows

symchk.exe %systemroot%\system32\ntoskrnl.exe /s
SRV*%systemroot%\symbols*http://msdl.microsoft.com/download/symbols

copy
%systemroot%\Symbols\ntoskrnl.pdb\<some_hash_value>\ntoskrnl.pdb
%systemroot%\system32

cd %ProgramFiles%\Compuware\DriverStudio\SoftICE

net start iceext

nmsym.exe %systemroot%\system32\ntoskrnl.exe
/OUTPUT:%systemroot%\system32\ntoskrnl.nms
```

So, what have we done so far?

- Created a symbol directory
- Switched to the Windows Debugging Tools directory
- Retrieved the proper NTOSKRNL.PDB from the Windows website with symchk.exe
- Copied the .pdb from the symbol directory into the Windows system32 directory
- Switched to the SoftICE directory
- Started the SoftICE extension ICEEXT and thus SoftICE too (which is quite tautological)
- Converted the .pdb file to a .nms file

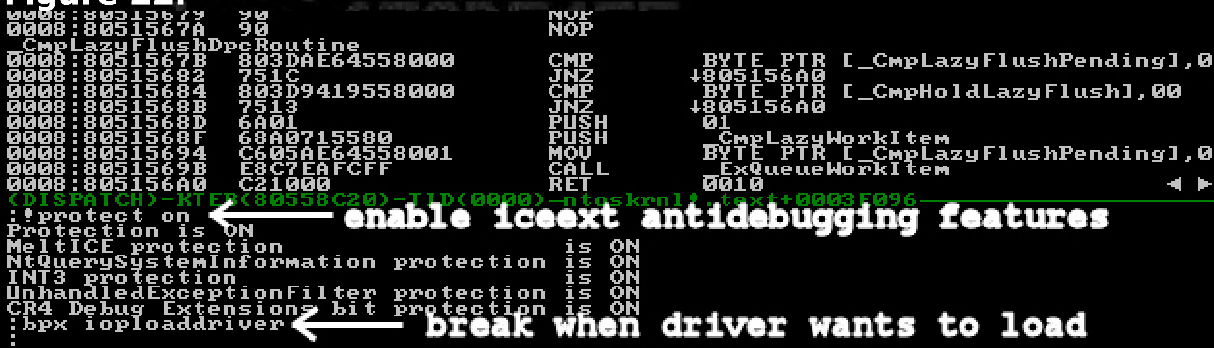
Ok, we are now prepared to start the debugging session now.

6.2 Debugging and Dumping

Before we fire up the **Rustock.exe** we need to adjust two settings in the SoftICE window first. So enter SoftICE using **CTRL+D** and set (Figure 22)

```
!protect on
bpx iopl1oaddriver
```

Figure 22:



Leave the debugger using **x** and execute **Rustock.exe**
The debugger window should have been popped up now at the entry point of IopLoadDriver (Figure 23)

Figure 23:

```
———————_IopGetDriverNameFromKeyNode+0154—————————————————————————————
0008:805A038F  009090909090        ADD     [EAX+90909090],DL
_IopLoadDriver
0008:805A0395  8BFF                MOV     EDI,EDI
0008:805A0397  55                  PUSH    EBP
0008:805A0398  8BEC                MOV     EBP,ESP
0008:805A039A  81ECB4000000        SUB     ESP,000000B4
0008:805A03A0  A120135580          MOV     EAX,[___security_cookie]
0008:805A03A5  8B4D08              MOV     ECX,[EBP+08]
0008:805A03A8  53                  PUSH    EBX
0008:805A03A9  33DB                XOR     EBX,EBX
0008:805A03AB  56                  PUSH    ESI
0008:805A03AC  8945FC              MOV     [EBP-04],EAX
0008:805A03AF  8B4514              MOV     EAX,[EBP+14]
0008:805A03B2  57                  PUSH    EDI
0008:805A03B3  898568FFFFFF        MOV     [EBP-0098],EAX
0008:805A03B9  8918                MOV     [EAX],EBX
0008:805A03BB  8D4588              LEA     EAX,[EBP-78]
0008:805A03BE  50                  PUSH    EAX
0008:805A03BF  53                  PUSH    EBX
0008:805A03C0  53                  PUSH    EBX
0008:805A03C1  53                  PUSH    EBX
0008:805A03C2  51                  PUSH    ECX
0008:805A03C3  894D8C              MOV     [EBP-74],ECX
0008:805A03C6  899D6CFFFFFF        MOV     [EBP-0094],EBX
0008:805A03CC  66895D98            MOV     [EBP-68],BX
0008:805A03D0  66895D9A            MOV     [EBP-66],BX
0008:805A03D4  895D9C              MOV     [EBP-64],EBX
0008:805A03D7  899D78FFFFFF        MOV     [EBP-0088],EBX
0008:805A03DD  895DA8              MOV     [EBP-58],EBX
0008:805A03E0  E88CE7FCFF          CALL    _NtQueryKey
0008:805A03E5  3D05000080          CMP     EAX,80000005 ; STATUS_BUF
0008:805A03EA  740B                JZ      805A03F7
0008:805A03EC  3D230000C0          CMP     EAX,C0000023 ; STATUS_BUF
0008:805A03F1  0F8503690400        JNZ     805E6CFA
0008:805A03F7  8B4588              MOV     EAX,[EBP-78]
0008:805A03FA  BF496F2020          MOV     EDI,20206F49
0008:805A03FF  57                  PUSH    EDI
0008:805A0400  83C008              ADD     EAX,08
0008:805A0403  50                  PUSH    EAX
0008:805A0404  53                  PUSH    EBX
0008:805A0405  E83AA5FAFF          CALL    _ExAllocatePoolWithTag
0008:805A040A  8BF0                MOV     ESI,EAX
0008:805A040C  3BF3                CMP     ESI,EBX
0008:805A040E  89B56CFFFFFF        MOV     [EBP-0094],ESI
0008:805A0414  0F84096A0400        JZ      805E6E23
0008:805A041A  8D4588              LEA     EAX,[EBP-78]
0008:805A041D  50                  PUSH    EAX
0008:805A041E  FF7588              PUSH    DWORD PTR [EBP-78]
0008:805A0421  56                  PUSH    ESI
0008:805A0422  53                  PUSH    EBX
(PASSIVE)-KTEB(823C6B20)-TID(002C)——ntoskrnl!PAGE+0003D90F————————————
Protection is ON
MeltICE protection                    is ON
NtQuerySystemInformation protection is ON
INT3 protect
UnhandledExc
CR4 Debug Extensions bit protection is ON
:bpx iopload driver
:x
NTICE: Load32 START=00400000  SIZE=14000  KPEB=821A7558  MOD=rustock
NTICE: Load32 START=7C910000  SIZE=B7000  KPEB=821A7558  MOD=ntdll
NTICE: Load32 START=7C800000  SIZE=106000  KPEB=821A7558  MOD=kernel32
NTICE: Load32 START=77DA0000  SIZE=AA000  KPEB=821A7558  MOD=advapi32
NTICE: Load32 START=77E50000  SIZE=910  KPEB=821A7558  MOD=rpcrt4
Break due to BP 00: BPX _IopLoadDriver (ET=2.36 seconds)
:
```

breakpoint occured after starting rustock!

Next switch to the code window using **F6** and scroll down until you see code like this (Figure 24)

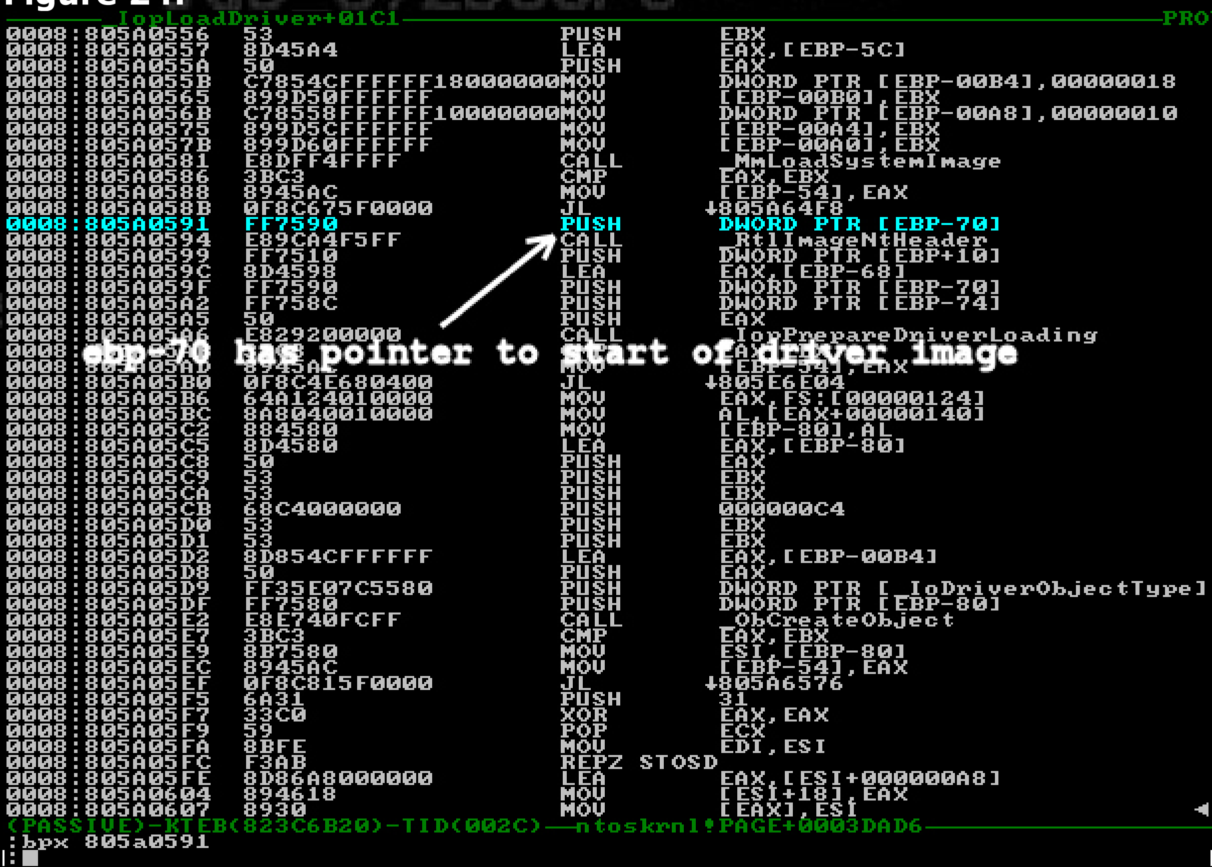
```
CALL MmLoadSystemImage
CMP EAX, EBX
MOV [EBP-54], EAX
JL somewhere
PUSH DWORD PTR [EBP-70]
CALL RtlImageNtHeader
```

On my machine IopLoadDriver+1c1 (Windows XP SP2 German) and address 0x805a0591.

Leave the code windows with F6 and set a breakpoint at the address were the following instruction is found and run the code using **x**.

PUSH DWORD PTR [EBP-70]

Figure 24:




```
———_IopLoadDriver+01C1———————————————————————————————PRO
0008:805A0556  53                    PUSH   EBX
0008:805A0557  8D45A4                LEA    EAX,[EBP-5C]
0008:805A055A  50                    PUSH   EAX
0008:805A055B  C7854CFFFFFF18000000  MOV    DWORD PTR [EBP-00B4],00000018
0008:805A0565  899D50FFFFFF          MOV    [EBP-00B0],EBX
0008:805A056B  C78558FFFFFF10000000  MOV    DWORD PTR [EBP-00A8],00000010
0008:805A0575  899D5CFFFFFF          MOV    [EBP-00A4],EBX
0008:805A057B  899D60FFFFFF          MOV    [EBP-00A0],EBX
0008:805A0581  E8DFF4FFFF            CALL   _MmLoadSystemImage
0008:805A0586  3BC3                  CMP    EAX,EBX
0008:805A0588  8945AC                MOV    [EBP-54],EAX
0008:805A058B  0F8C675F0000          JL     805A64F8
0008:805A0591  FF7590                PUSH   DWORD PTR [EBP-70]
0008:805A0594  E89CA4F5FF            CALL   _RtlImageNtHeader
0008:805A0599  FF7510                PUSH   DWORD PTR [EBP+10]
0008:805A059C  8D4598                LEA    EAX,[EBP-68]
0008:805A059F  FF7590                PUSH   DWORD PTR [EBP-70]
0008:805A05A2  FF758C                PUSH   DWORD PTR [EBP-74]
0008:805A05A5  50                    PUSH   EAX
0008:805A05A6  E829200000            CALL   _IopPrepareDriverLoading
0008:805A05AD  8945AC                MOV    [EBP-54],EAX
0008:805A05B0  0F8C4E680400          JL     805E6E04
0008:805A05B6  64A124010000          MOV    EAX,FS:[00000124]
0008:805A05BC  8A8040010000          MOV    AL,[EAX+00000140]
0008:805A05C2  884580                MOV    [EBP-80],AL
0008:805A05C5  8D4580                LEA    EAX,[EBP-80]
0008:805A05C8  50                    PUSH   EAX
0008:805A05C9  53                    PUSH   EBX
0008:805A05CA  53                    PUSH   EBX
0008:805A05CB  68C4000000            PUSH   000000C4
0008:805A05D0  53                    PUSH   EBX
0008:805A05D1  53                    PUSH   EBX
0008:805A05D2  8D854CFFFFFF          LEA    EAX,[EBP-00B4]
0008:805A05D8  50                    PUSH   EAX
0008:805A05D9  FF35E07C5580          PUSH   DWORD PTR [_IoDriverObjectType]
0008:805A05DF  FF7580                PUSH   DWORD PTR [EBP-80]
0008:805A05E2  E8E740FCFF            CALL   _ObCreateObject
0008:805A05E7  3BC3                  CMP    EAX,EBX
0008:805A05E9  8B7580                MOV    ESI,[EBP-80]
0008:805A05EC  8945AC                MOV    [EBP-54],EAX
0008:805A05EF  0F8C815F0000          JL     805A6576
0008:805A05F5  6A31                  PUSH   31
0008:805A05F7  33C0                  XOR    EAX,EAX
0008:805A05F9  59                    POP    ECX
0008:805A05FA  8BFE                  MOV    EDI,ESI
0008:805A05FC  F3AB                  REPZ STOSD
0008:805A05FE  8D86A8000000          LEA    EAX,[ESI+000000A8]
0008:805A0604  894618                MOV    [ESI+18],EAX
0008:805A0607  8930                  MOV    [EAX],ESI
(PASSIVE)-KTEB(823C6B20)-TID(002C)——ntoskrnl!PAGE+0003DAD6———————
:bpx 805a0591
:
```

When the breakpoint occurred at **PUSH DWORD PTR [EBP-70]** enter:

d *(ebp-70)

As you can see in Figure 25, **ebp-70** has a pointer to the start of the image to the Rootkit driver.

A Journey to the Center of the Rustock.B Rootkit

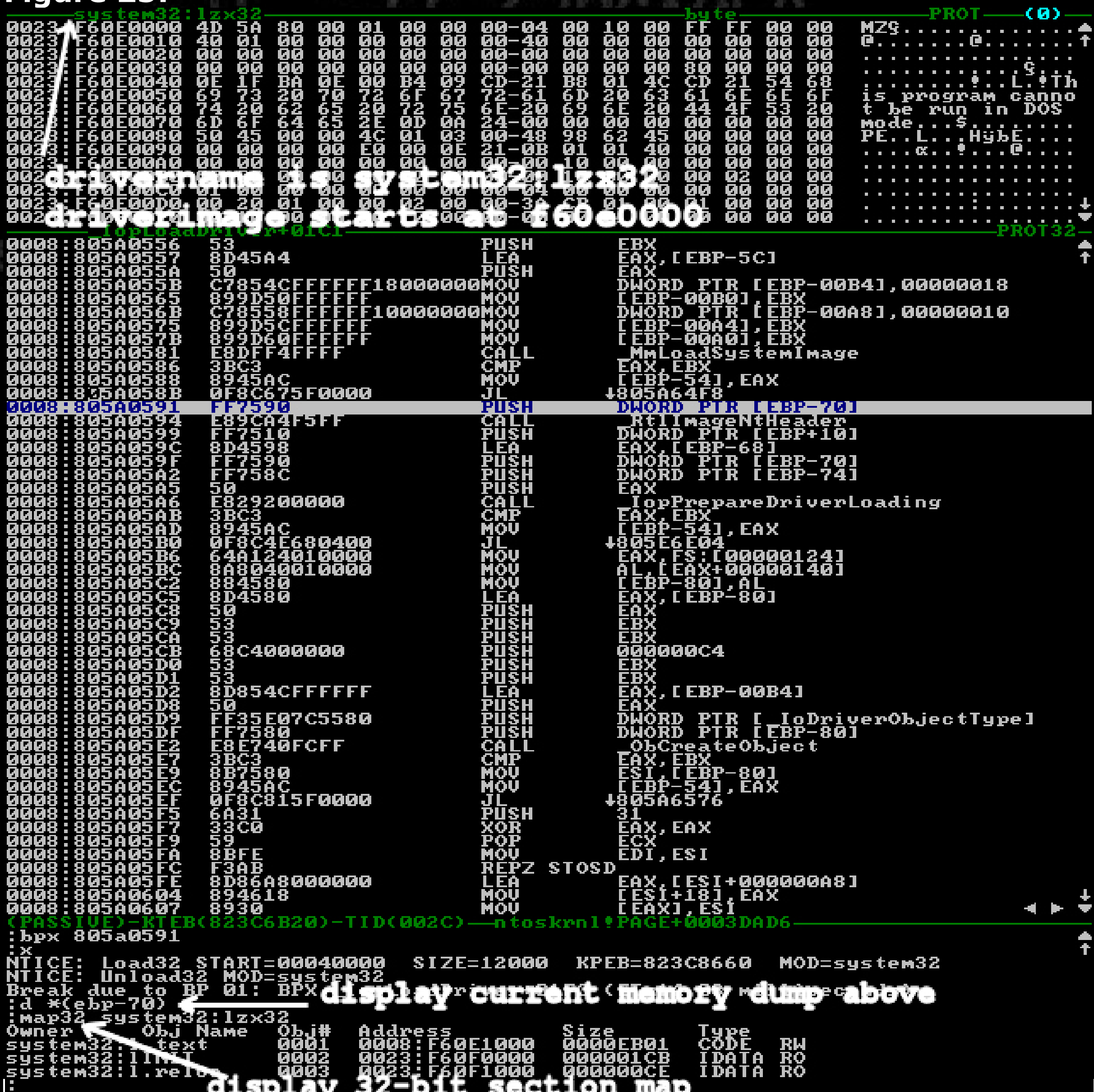
Let us grab some PE section info about the driver.

`map32 system32:lzx32`

So what do we know about the image now?

```
ImageStart ---> 0xf60ee000
Size: 0x110ce ---> (reloc_addr+size) - ImageStart
EntryPoint ---> ImageStart + (*(e_lfanew)+28h) = 0xf60e1000
```

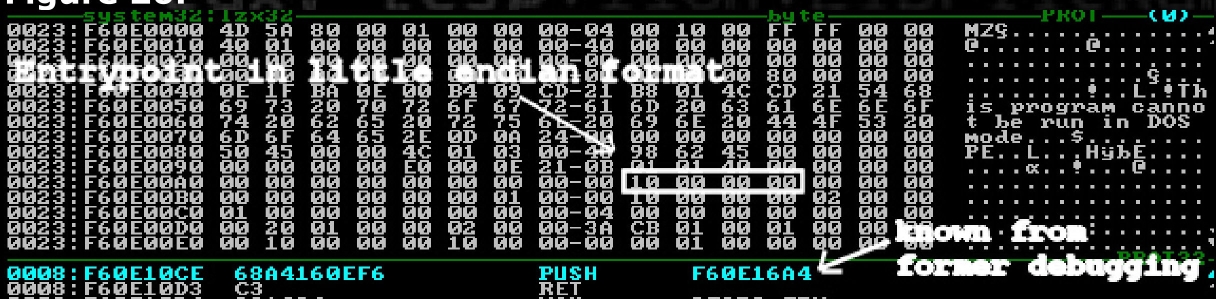
Figure 25:



```
--system32:lzx32---------------------------------------byte----------PROT---(0)--
0023:F60E0000 4D 5A 80 00 01 00 00 00-04 00 10 00 FF FF 00 00  MZ
0023:F60E0010 40 01 00 00 00 00 00 00-40 00 00 00 00 00 00 00  @.......@.......
0023:F60E0020 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0023:F60E0030 00 00 00 00 00 00 00 00-00 00 00 00 80 00 00 00  ................
0023:F60E0040 0E 1F BA 0E 00 B4 09 CD-21 B8 01 4C CD 21 54 68  ..........L..!Th
0023:F60E0050 69 73 20 70 72 6F 67 72-61 6D 20 63 61 6E 6E 6F  is program canno
0023:F60E0060 74 20 62 65 20 72 75 6E-20 69 6E 20 44 4F 53 20  t be run in DOS
0023:F60E0070 6D 6F 64 65 2E 0D 0A 24-00 00 00 00 00 00 00 00  mode...$........
0023:F60E0080 50 45 00 00 4C 01 03 00-48 98 62 45 00 00 00 00  PE..L...HbE....
0023:F60E0090 00 00 00 00 E0 00 0E 21-0B 01 01 40 00 00 00 00  ...............
0023:F60E00A0 00 00 00 00 00 00 00 00-00 10 00 00 00 00 00 00  ................
drivername is system32:lzx32
driverimage starts at f60e0000
--_IopLoadDriver+01C1-------------------------------------------------PROT32--
0008:805A0556  53                  PUSH    EBX
0008:805A0557  8D45A4              LEA     EAX,[EBP-5C]
0008:805A055A  50                  PUSH    EAX
0008:805A055B  C7854CFFFFFF18000000MOV     DWORD PTR [EBP-00B4],00000018
0008:805A0565  899D50FFFFFF        MOV     [EBP-00B0],EBX
0008:805A056B  C78558FFFFFF10000000MOV     DWORD PTR [EBP-00A8],00000010
0008:805A0575  899D5CFFFFFF        MOV     [EBP-00A4],EBX
0008:805A057B  899D60FFFFFF        MOV     [EBP-00A0],EBX
0008:805A0581  E8DFF4FFFF          CALL    _MmLoadSystemImage
0008:805A0586  3BC3                CMP     EAX,EBX
0008:805A0588  8945AC              MOV     [EBP-54],EAX
0008:805A058B  0F8C675F0000        JL      805A64F8
0008:805A0591  FF7590              PUSH    DWORD PTR [EBP-70]
0008:805A0594  E89CA4F5FF          CALL    _RtlImageNtHeader
0008:805A0599  FF7510              PUSH    DWORD PTR [EBP+10]
0008:805A059C  8D4598              LEA     EAX,[EBP-68]
0008:805A059F  FF7590              PUSH    DWORD PTR [EBP-70]
0008:805A05A2  FF758C              PUSH    DWORD PTR [EBP-74]
0008:805A05A5  50                  PUSH    EAX
0008:805A05A6  E829200000          CALL    _IopPrepareDriverLoading
0008:805A05AB  3BC3                CMP     EAX,EBX
0008:805A05AD  8945AC              MOV     [EBP-54],EAX
0008:805A05B0  0F8C4E680400        JL      805E6E04
0008:805A05B6  64A124010000        MOV     EAX,FS:[00000124]
0008:805A05BC  8A8040010000        MOV     AL,[EAX+00000140]
0008:805A05C2  884580              MOV     [EBP-80],AL
0008:805A05C5  8D4580              LEA     EAX,[EBP-80]
0008:805A05C8  50                  PUSH    EAX
0008:805A05C9  53                  PUSH    EBX
0008:805A05CA  53                  PUSH    EBX
0008:805A05CB  68C4000000          PUSH    000000C4
0008:805A05D0  53                  PUSH    EBX
0008:805A05D1  53                  PUSH    EBX
0008:805A05D2  8D854CFFFFFF        LEA     EAX,[EBP-00B4]
0008:805A05D8  50                  PUSH    EAX
0008:805A05D9  FF35E07C5580        PUSH    DWORD PTR [_IoDriverObjectType]
0008:805A05DF  FF7580              PUSH    DWORD PTR [EBP-80]
0008:805A05E2  E8E740FCFF          CALL    _ObCreateObject
0008:805A05E7  3BC3                CMP     EAX,EBX
0008:805A05E9  8B7580              MOV     ESI,[EBP-80]
0008:805A05EC  8945AC              MOV     [EBP-54],EAX
0008:805A05EF  0F8C815F0000        JL      805A6576
0008:805A05F5  6A31                PUSH    31
0008:805A05F7  33C0                XOR     EAX,EAX
0008:805A05F9  59                  POP     ECX
0008:805A05FA  8BFE                MOV     EDI,ESI
0008:805A05FC  F3AB                REPZ STOSD
0008:805A05FE  8D86A8000000        LEA     EAX,[ESI+000000A8]
0008:805A0604  894618              MOV     [ESI+18],EAX
0008:805A0607  8930                MOV     [EAX],ESI
(PASSIVE)-KTEB(823C6B20)-TID(002C)--ntoskrnl!PAGE+0003DAD6--------------------
:bpx 805a0591
:x
NTICE: Load32 START=00040000  SIZE=12000  KPEB=823C8660  MOD=system32
NTICE: Unload32 MOD=system32
Break due to BP 01: BPX
:d *(ebp-70)        display current memory dump above
:map32 system32:lzx32
Owner         Obj Name   Obj#  Address        Size      Type
system32:     .text      0001  0008:F60E1000  0000EB01  CODE   RW
system32:                0002  0023:F60F0000  000001CB  IDATA  RO
system32:l.rel           0003  0023:F60F1000  000000CE  IDATA  RO
                  display 32-bit section map
:
```

As we know from the former debugging session with Ollydbg the end address of the first deobfuscation was at EntryPoint+0xce. So, it's a good idea to set our next breakpoint at this address (Figure 26) and run the code again.

Figure 26:



After the breakpoint is reached, use the trace option 2 times using `t` and you should see some well-known code (Figure 27).

Figure 27:

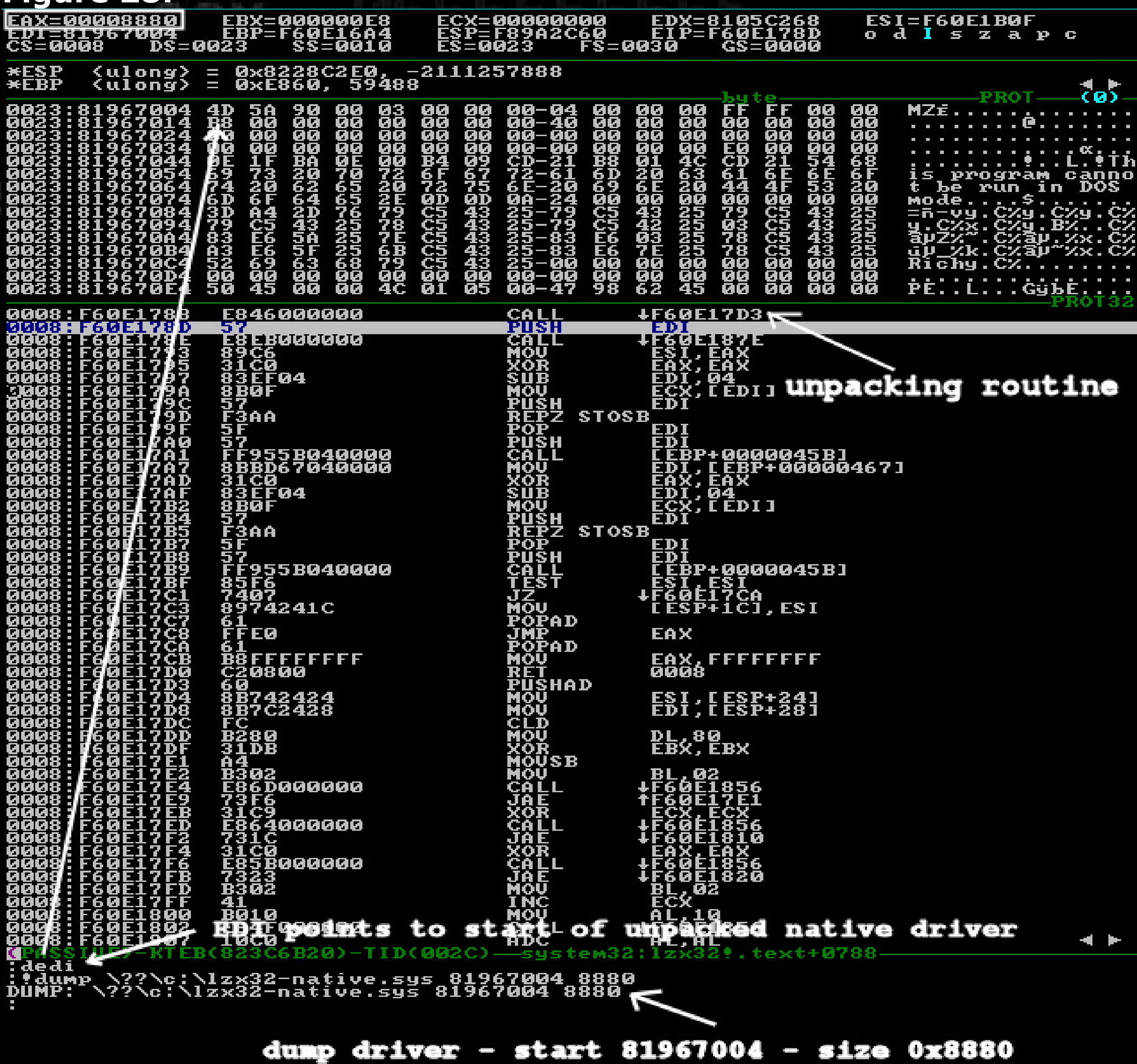
```
0008:F60E16A4  60                PUSHAD
0008:F60E16A5  E800000000        CALL    ↓F60E16AA
0008:F60E16AA  5D                POP     EBP
0008:F60E16AB  83ED06            SUB     EBP,06
0008:F60E16AE  64A138000000      MOV     EAX,FS:[00000038]
0008:F60E16B4  8B4004            MOV     EAX,[EAX+04]
0008:F60E16B7  30C0              XOR     AL,AL
0008:F60E16B9  2D00010000        SUB     EAX,00000100
0008:F60E16BE  6681384D5A        CMP     WORD PTR [EAX],5A4D
0008:F60E16C3  75F4              JNZ     ↑F60E16B9
0008:F60E16C5  0FB7583C          MOVZX   EBX,WORD PTR [EAX+3C]
0008:F60E16C9  813C1850450000    CMP     DWORD PTR [EBX+EAX],00004550
0008:F60E16D0  75E7              JNZ     ↑F60E16B9
0008:F60E16D2  89C2              MOV     EDX,EAX
0008:F60E16D4  8DB51A040000      LEA     ESI,[EBP+0000041A]
0008:F60E16DA  8DBD57040000      LEA     EDI,[EBP+00000457]
0008:F60E16E0  E849030000        CALL    ↓F60E1A2E
0008:F60E16E5  8DB56B040000      LEA     ESI,[EBP+0000046B]
0008:F60E16EB  56                PUSH    ESI
0008:F60E16EC  6A00              PUSH    00
0008:F60E16EE  56                PUSH    ESI
0008:F60E16EF  6A0B              PUSH    0B
0008:F60E16F1  FF955F040000      CALL    [EBP+0000045F]
0008:F60E16F7  830604            ADD     DWORD PTR [ESI],04
0008:F60E16FA  FF36              PUSH    DWORD PTR [ESI]
0008:F60E16FC  6A00              PUSH    00
0008:F60E16FE  FF9557040000      CALL    [EBP+00000457]
0008:F60E1704  FF36              PUSH    DWORD PTR [ESI]
0008:F60E1706  8F00              POP     DWORD PTR [EAX]
0008:F60E1708  83C004            ADD     EAX,04
0008:F60E170B  898567040000      MOV     [EBP+00000467],EAX
0008:F60E1711  89C7              MOV     EDI,EAX
0008:F60E1713  6A00              PUSH    00
0008:F60E1715  FF36              PUSH    DWORD PTR [ESI]
0008:F60E1717  57                PUSH    EDI
0008:F60E1718  6A0B              PUSH    0B
0008:F60E171A  FF955F040000      CALL    [EBP+0000045F]
0008:F60E1720  85C0              TEST    EAX,EAX
0008:F60E1722  0F85A2000000      JNZ     ↓F60E17CA
0008:F60E1728  83C704            ADD     EDI,04
0008:F60E172B  0FB7471A          MOVZX   EAX,WORD PTR [EDI+1A]
0008:F60E172F  8D44071C          LEA     EAX,[EAX+EDI+1C]
0008:F60E1733  C7006E746F73      MOV     DWORD PTR [EAX],736F746E
0008:F60E1739  C740046B726E6C    MOV     DWORD PTR [EAX+04],6C6E726B
0008:F60E1740  C740082E657865    MOV     DWORD PTR [EAX+08],6578652E
0008:F60E1747  C7400C00000000    MOV     DWORD PTR [EAX+0C],00000000
0008:F60E174E  81C71C010000      ADD     EDI,0000011C
0008:F60E1754  0FB7471A          MOVZX   EAX,WORD PTR [EDI+1A]
0008:F60E1758  8D44071C          LEA     EAX,[EAX+EDI+1C]
(PASSIVE)-KTEB(823C6B20)-TID(002C)--system32:lzx32!.text+06A3
:df60e0000
:uf60e1000
:uf60e1000+ce
:bpx f60e10ce
:x
Break due to BP 02: BPX 0008:F60E10CE (ET=86.40 milliseconds)
:t
:t
:
```

The last breakpoint that is left, is the one behind the unpacking routine, we discussed in stage 3. And if you scroll down a little bit in the SoftICE code window you should find the unpacker at offset 0x1788, thus we set the breakpoint right behind it, on my machine at 0xf60e178d and run code the again.

So, after the breakpoint occurred and unpacking was done let's see what we find at EDI now (Figure 28).

Yep, it points to the unpacked and untouched image we want to dump.

Figure 28:



Therefore the last thing what's left is using the dumping tool of IceExt.

```
!dump \??\c:\lzx32-native.sys 81967004 8880
```

That's it folks! To clean your drive from Rustock.B again, just use the fine Rootkit-Detection-Tool called RkUnhooker (see References).

7 Conclusion

After studying this paper the reader now should have a better understanding what different approaches can lead to success when analyzing an obfuscated/packed driver. You may rest assured that reverse engineering Malware is getting harder in future. Therefore, being prepared with some armory and tricks is essential. I hope you enjoyed this paper a little and I would be glad about some constructive reviews.

Happy reversing!

8 References

PE-Tools 1.5.800.2006-RC7
http://neox.iatp.by/updates/pt_update_08_rc7.zip
Ollydbg v1.10
http://www.ollydbg.de/odbg110.zip
Ollydump v3.00.110
http://dd.x-eye.net/file/ollydump300110.zip
IDA Pro 5.0.0.879
http://www.datarescue.com/idabase/
Windows Debugging Tools v6.6.7.5
http://www.microsoft.com/whdc/devtools/debugging/default.mspx
Driverstudio v3.21
http://www.compuware.com/products/driverstudio/782_ENG_HTML.htm
IceExt v0.70
http://sourceforge.net/projects/iceext/
PEID v0.94
http://peid.has.it
ProtectionID 5.2b
http://protectionid.owns.it
The big SoftICE howto v1.1
http://www.reconstructer.org/papers/The%20big%20SoftICE%20howto.pdf
RkUnhooker v3.0
http://rku.xell.ru/?l=e&a=dl

Big thanks go to Val Smith, Marc Schönefeld, FX and Olli Koen for reviewing this paper!