Lecture 11

Example Rootkit

Intel internship

- Intel CTG (Corporate Technology Group)
  - Advanced research & development
  - System integrity services using AMT
    - Detecting rootkits
    - C programming experience
    - Low-level OS knowledge
    - Embedded programming
- Who can I send?

Rustock.B rootkit

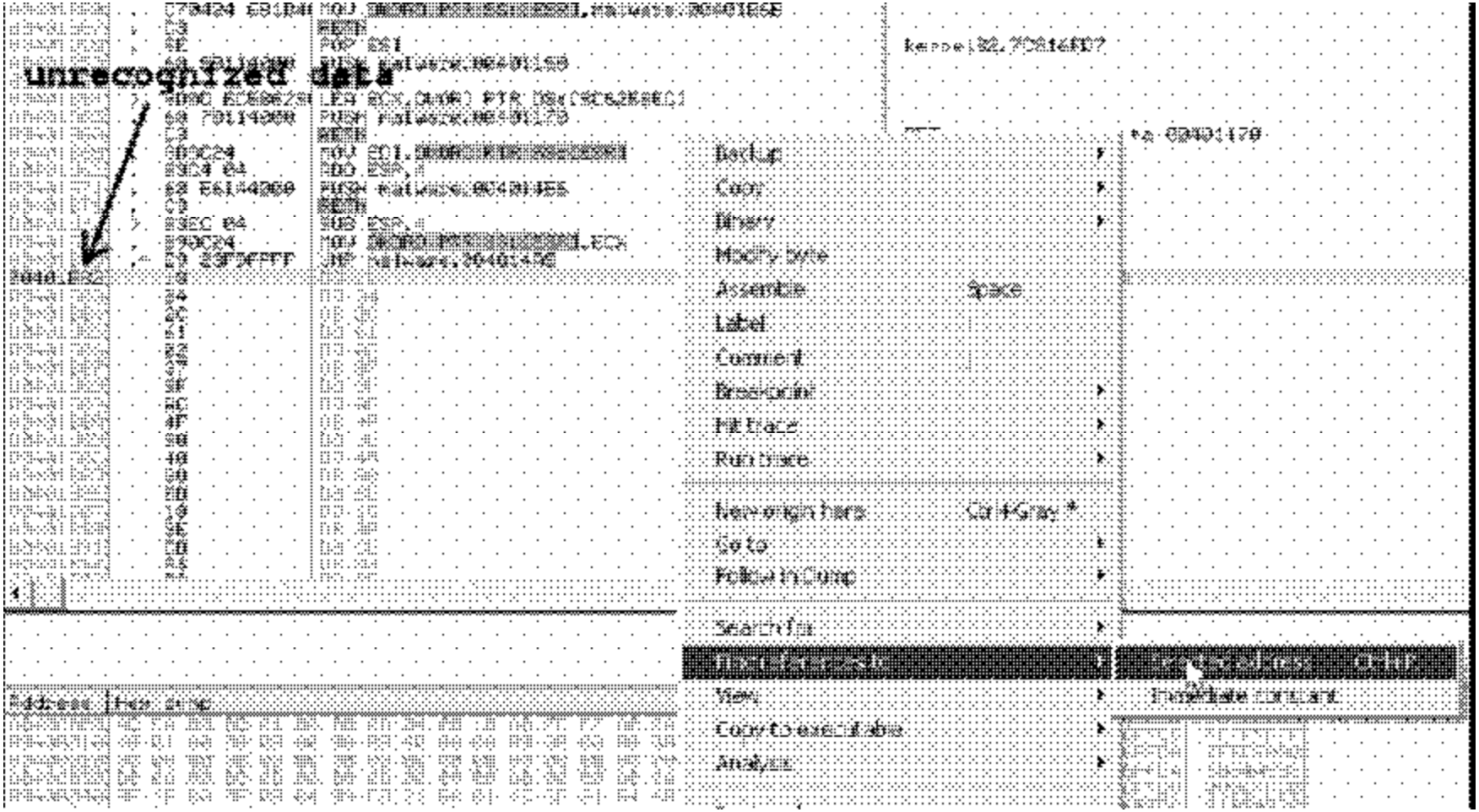
- Frank Boldewin, “A Journey to the Center of the Rustock.B Rootkit”, Jan. 20, 2007
  - http://reconstructer.org
- Combines a number of obfuscation techniques found in other malware

Stage 1: Ollydbg

- Drop from Mother Ship
  - Gives you `rustock.exe`, a Windows PE
- Step 1
  - In `Ollydbg`, search for all referenced text strings
    - Not much shown due to obfuscation/packing
  - Use `PEID` or `Protection-ID` tools to determine packer/compiler/protector
    - Not much shown, perhaps a proprietary packer used
  - Check for unrecognized data in code
    - Code loaded at virtual address of 0x400000
    - Entry point of Rustock.B at 0x401000

Stage 1: Ollydbg

- Looks like obfuscated code at 0x401B82
  - Find references to this address

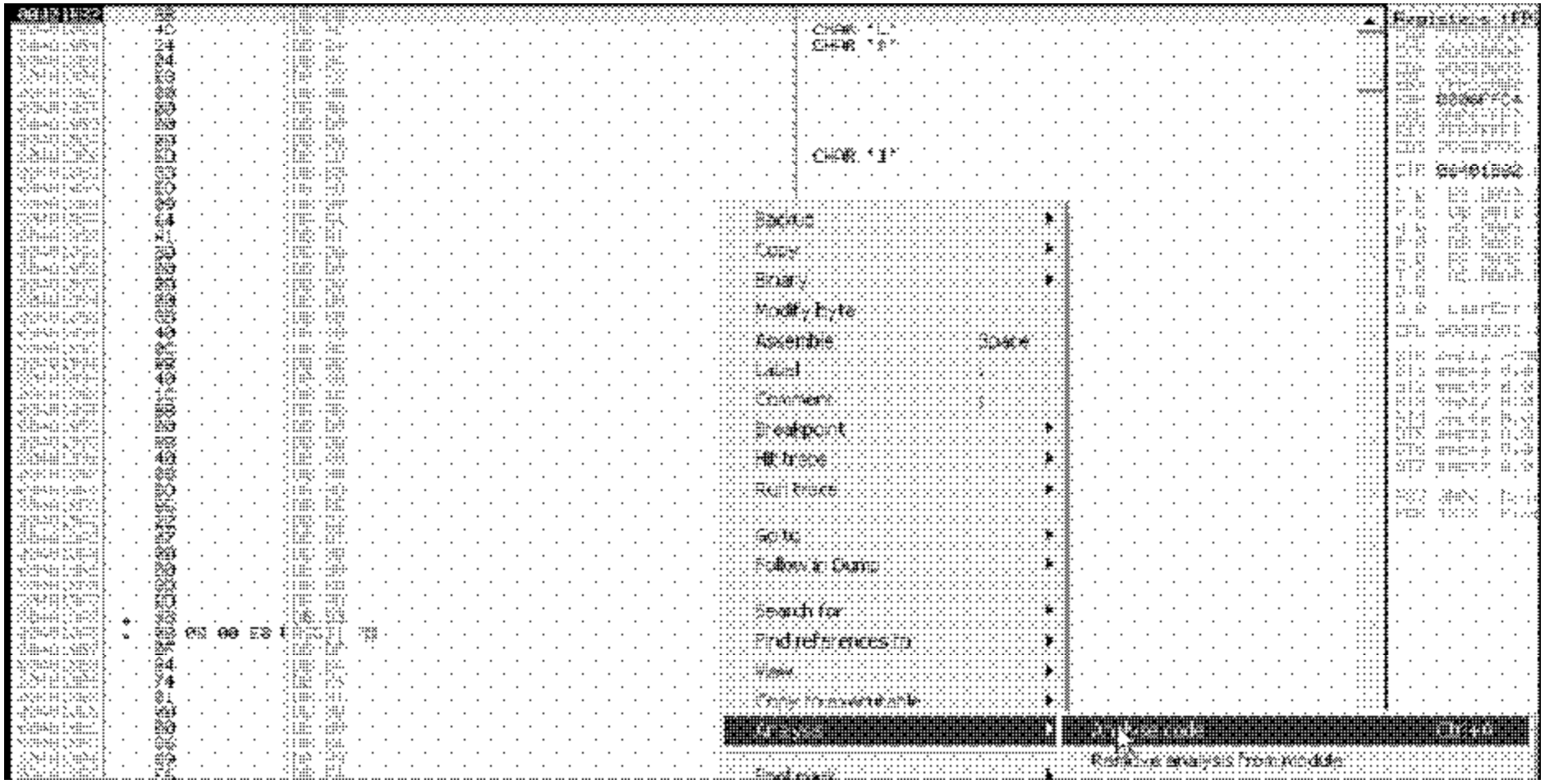


Stage 1: Ollydbg

- At 0x040198D
  - PUSH of 0x401B82 followed by RETN
  - Same as a CALL
  - Set breakpoint and run
  - Obfuscated code should be unobfuscated now

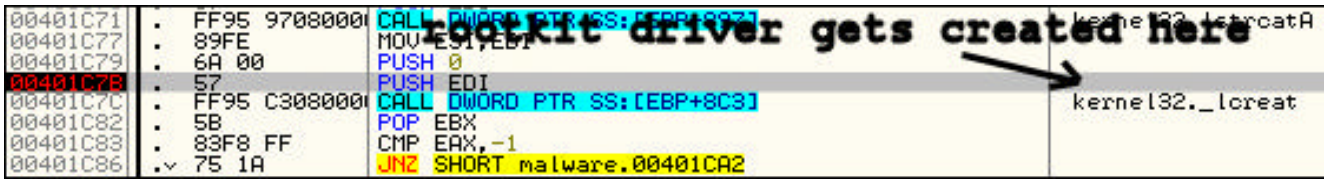
Stage 1: Ollydbg

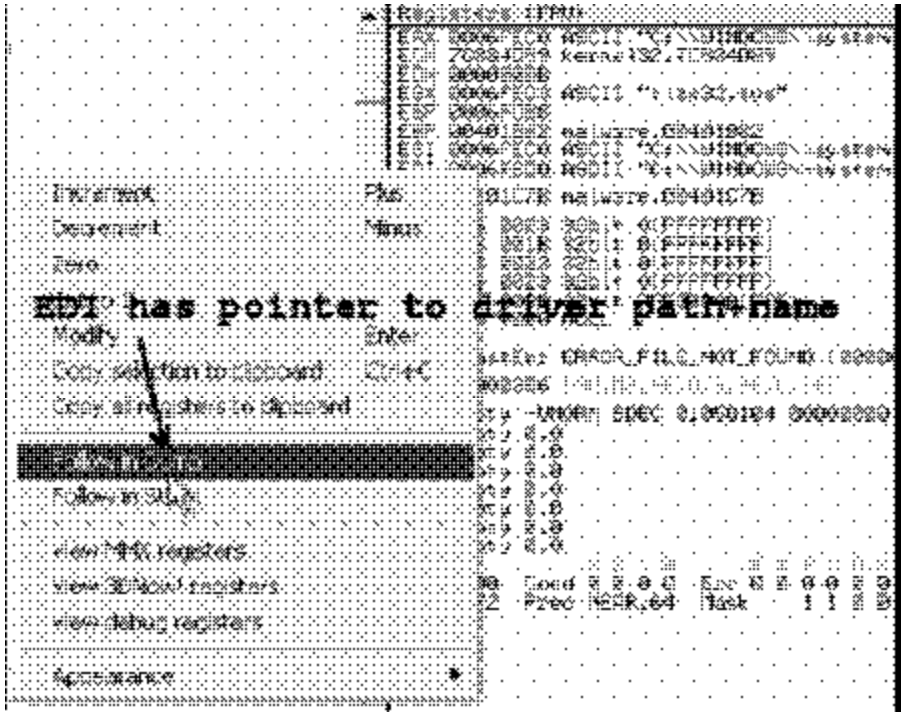
- Not quite, have `Ollydbg` analyze code
  - Step through API importing code to obtain API names for subsequent call instructions



Stage 1: Ollydbg

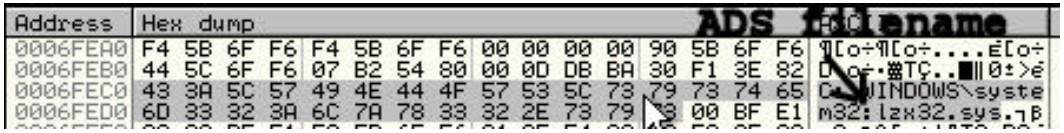
- Find call to kernel32._lcreat
  - Creates a file called lzx32.sys (kernel mode driver)
  - Set breakpoint and run again
  - Select EDI in Registers window and follow it





Stage 1: Ollydbg

- EDI points to C:\windows\system32:lzx32.sys
  - Use of : instead of \
  - Alternative Data Stream (ADS)
    - Hides the driver from easy detection
    - Windows Explorer and cmd.exe do not show ADS
    - Change memory to replace “:” (0x3a) to a “\” (0x5c)
    - Attach ADS to directory since ADS viewers do not show this
  - Rerun code and step through driver creation
  - Stop code at lclose at address 0x401cc7
- Driver has been deobfuscated and unpacked now

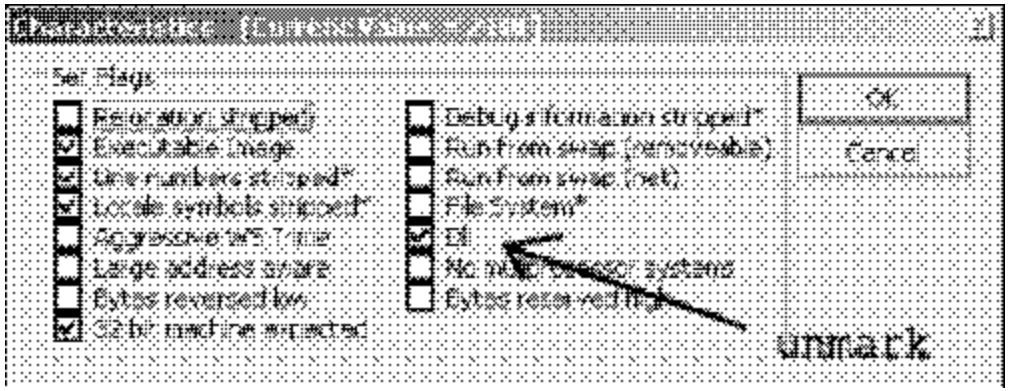


Stage 2: PE-Tools

- Driver now detached
  - Analyze it in IDA to find obfuscated code
  - Detached driver code and .idb file in “stage1” directory
  - Attempt to load in `Ollydbg`
    - Launch using `LOADDLL.EXE` fails

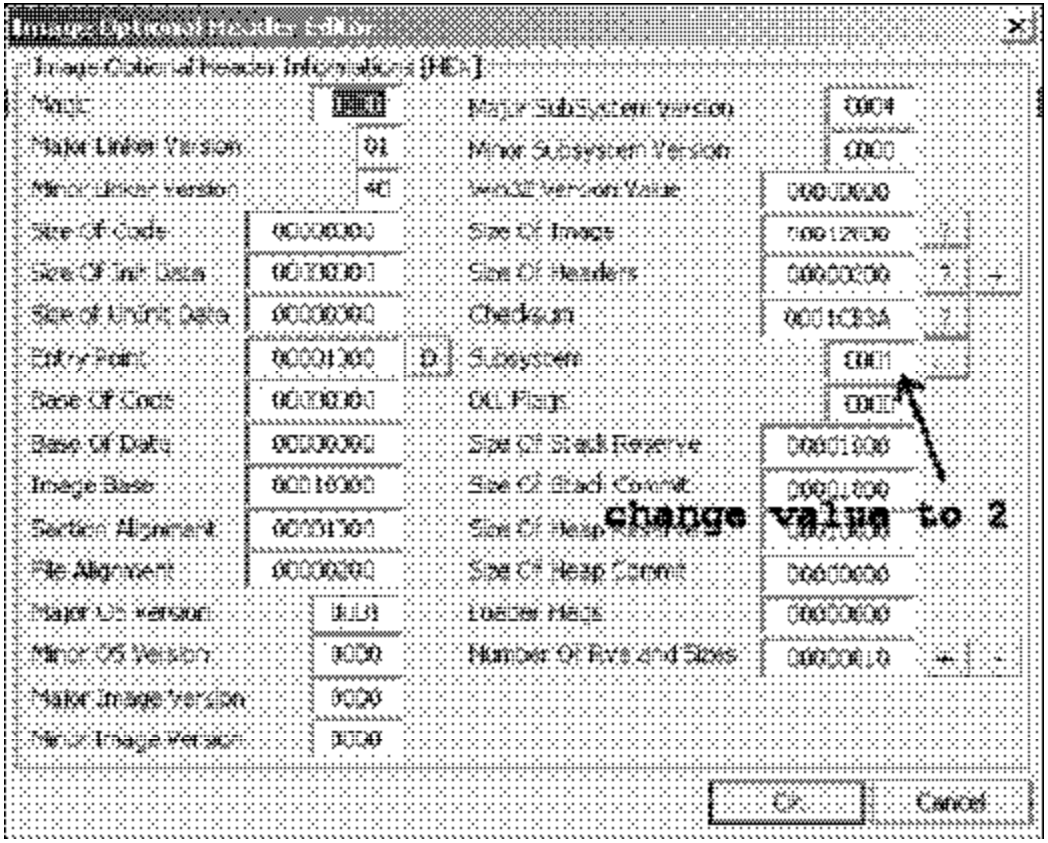
Stage 2: PE-Tools

- Change driver
  - Currently a DLL, a native executable, and contains imports from kernel libraries (NTOSKRNL.EXE and HAL.DLL)
  - Change to no DLL, a Windows GUI application, and no imports
  - Fix PE-files using `PE-Tools`
    - Unmark DLL bit in `PE-Tools`



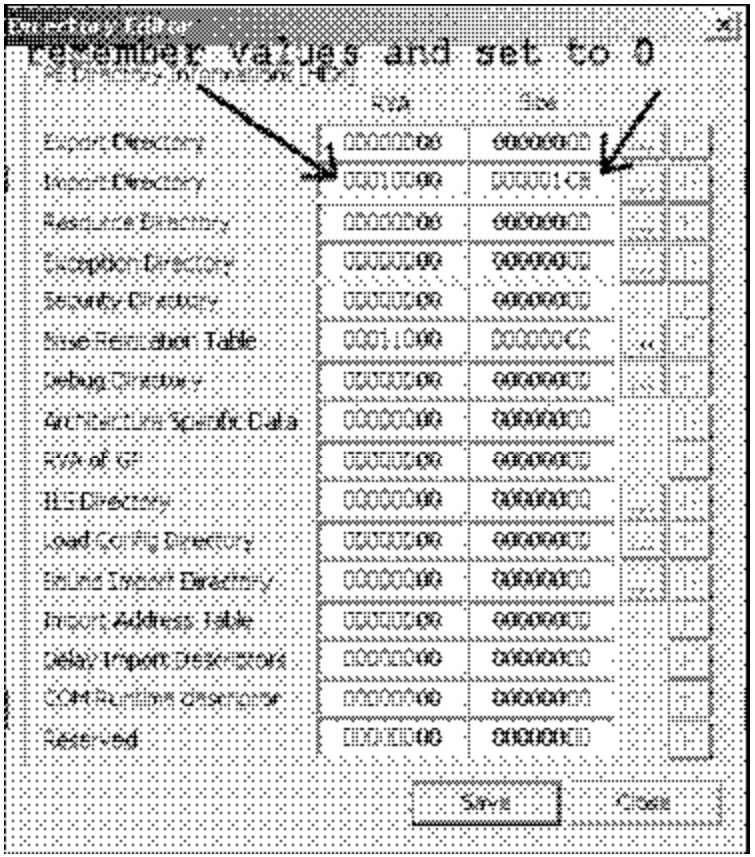
Stage 2: PE-Tools

- Change driver
  - In Optional Header of `PE-Tools`, change Subsystem value from 1 to 2 (Windows GUI)



Stage 2: PE-Tools

- Change driver
  - Set RVA and size to 0
  - Will be reset later

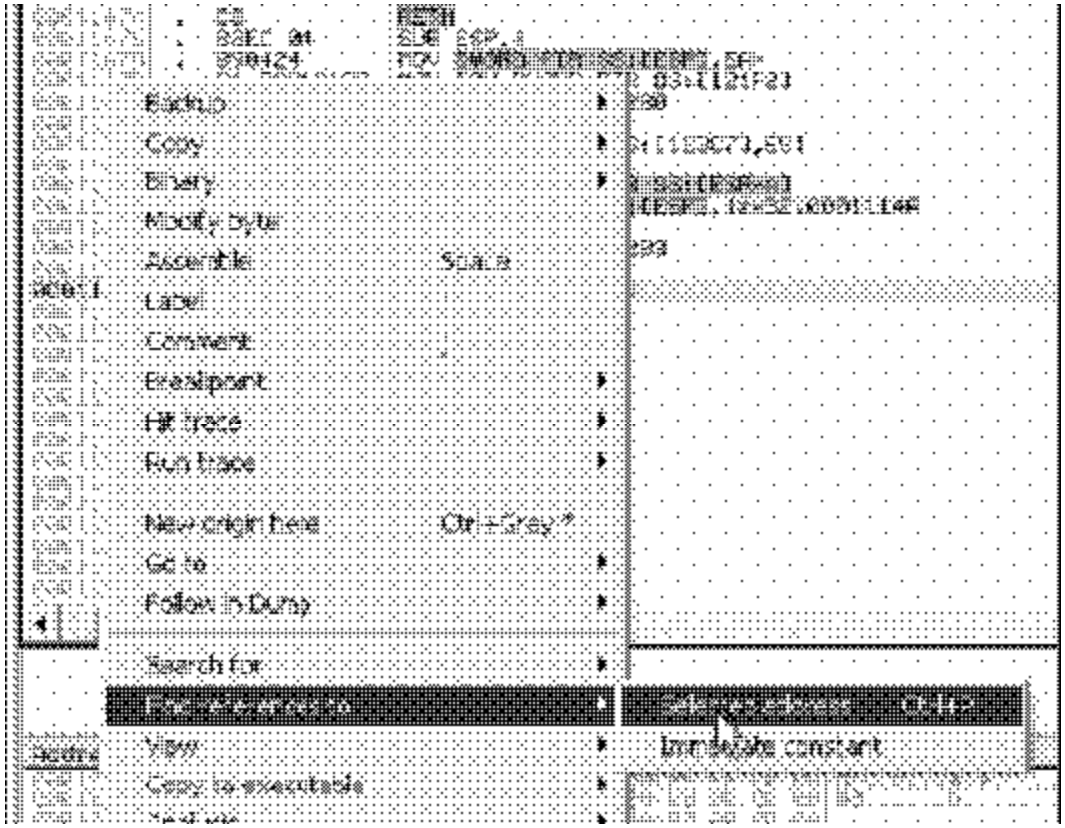


Stage 2: Ollydbg

- Driver loads now
  - Same as Stage 1: obfuscated code at 0x116a4

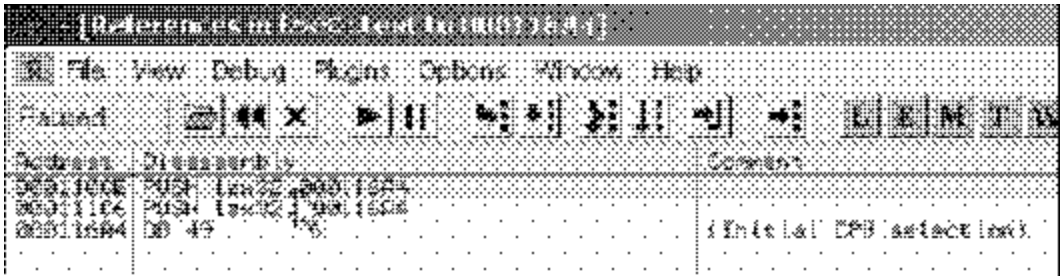
Stage 2: Ollydbg

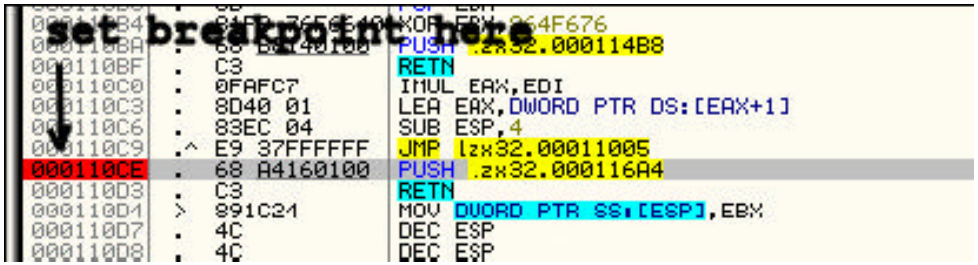
- Find references to this address 0x116a4



Stage 2: Ollydbg

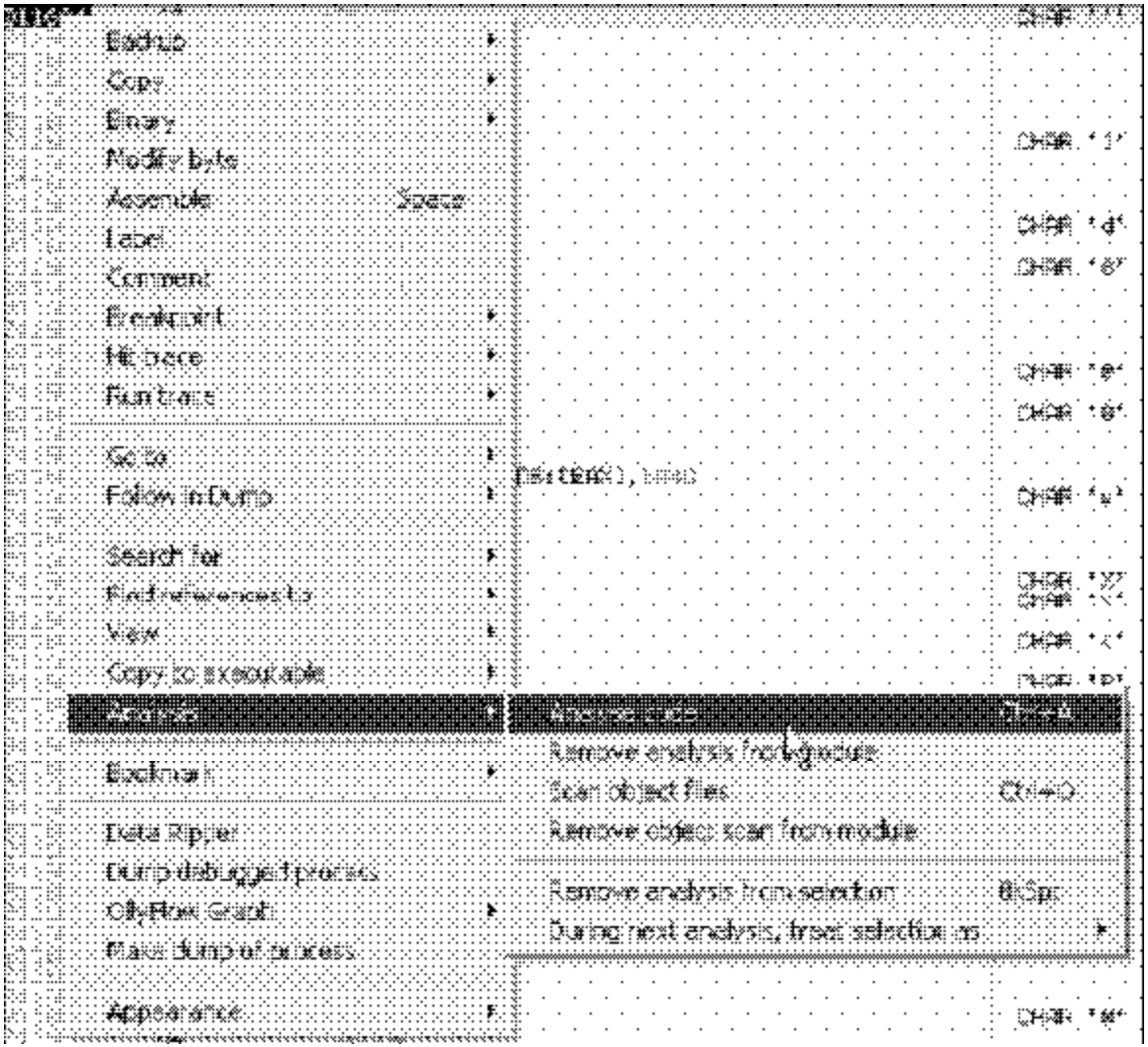
- Two places with PUSH 0x116a4/RETN
  - Set breakpoint and run



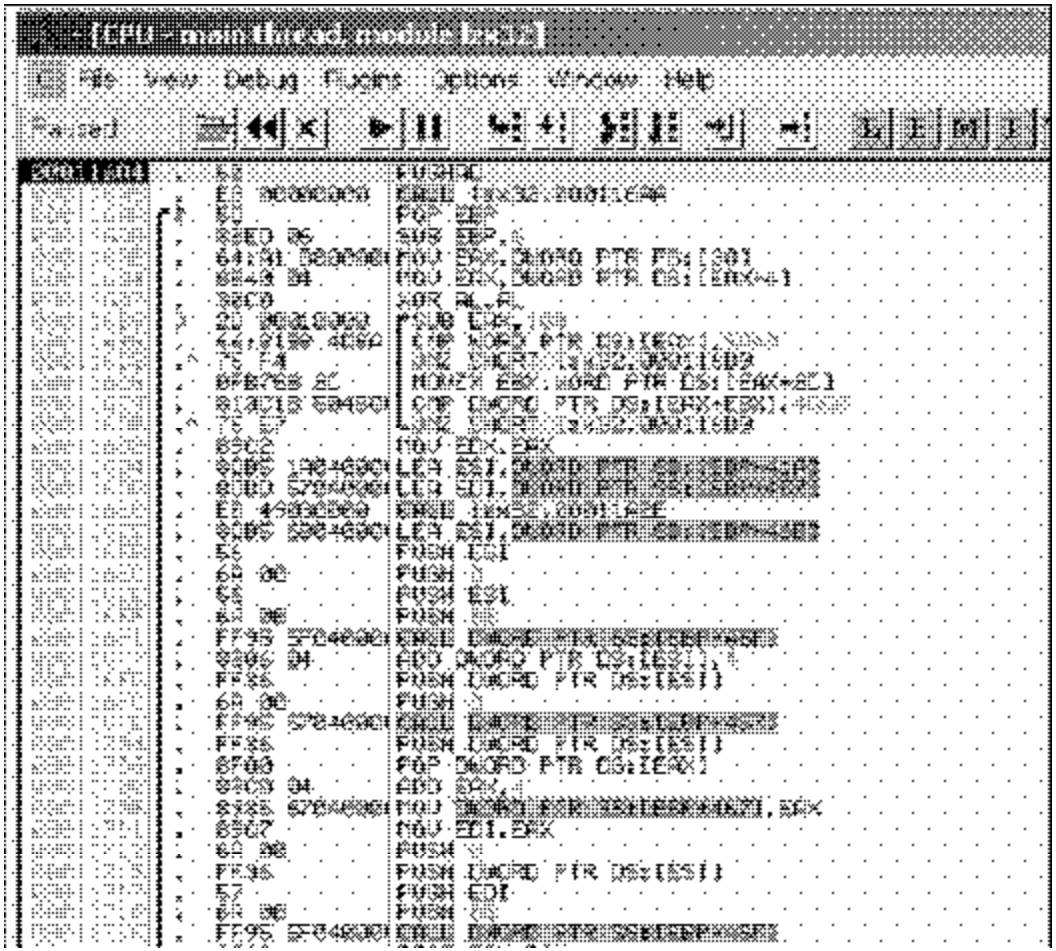


Stage 2: Ollydbg

- Analyze code now…



Stage 2: Ollydbg

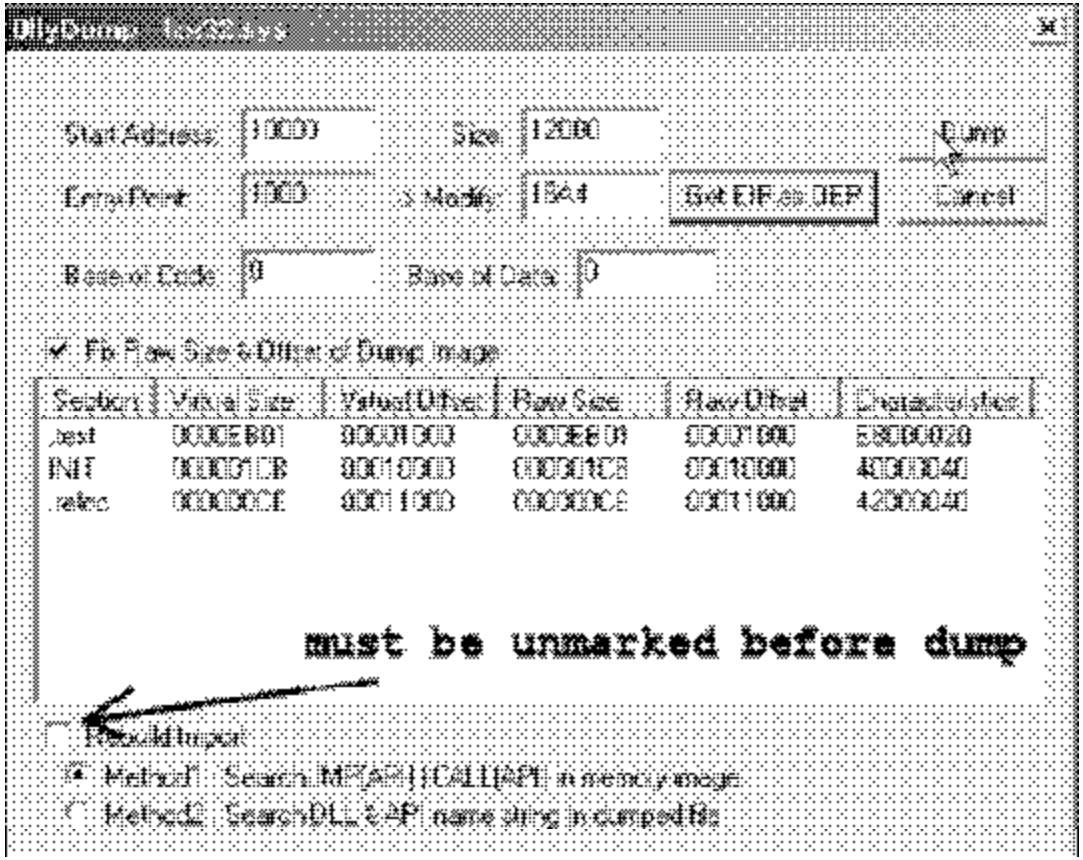


Stage 2: Ollydbg

- Dump debugged process

Stage 2: Ollydbg

- Dump debugged process
  - Unmark “Rebuild Import”



Stage 2

- After dump, restore PE-File settings
  - DLL bit
  - Subsystem native
  - RVA and Size of Import directory field

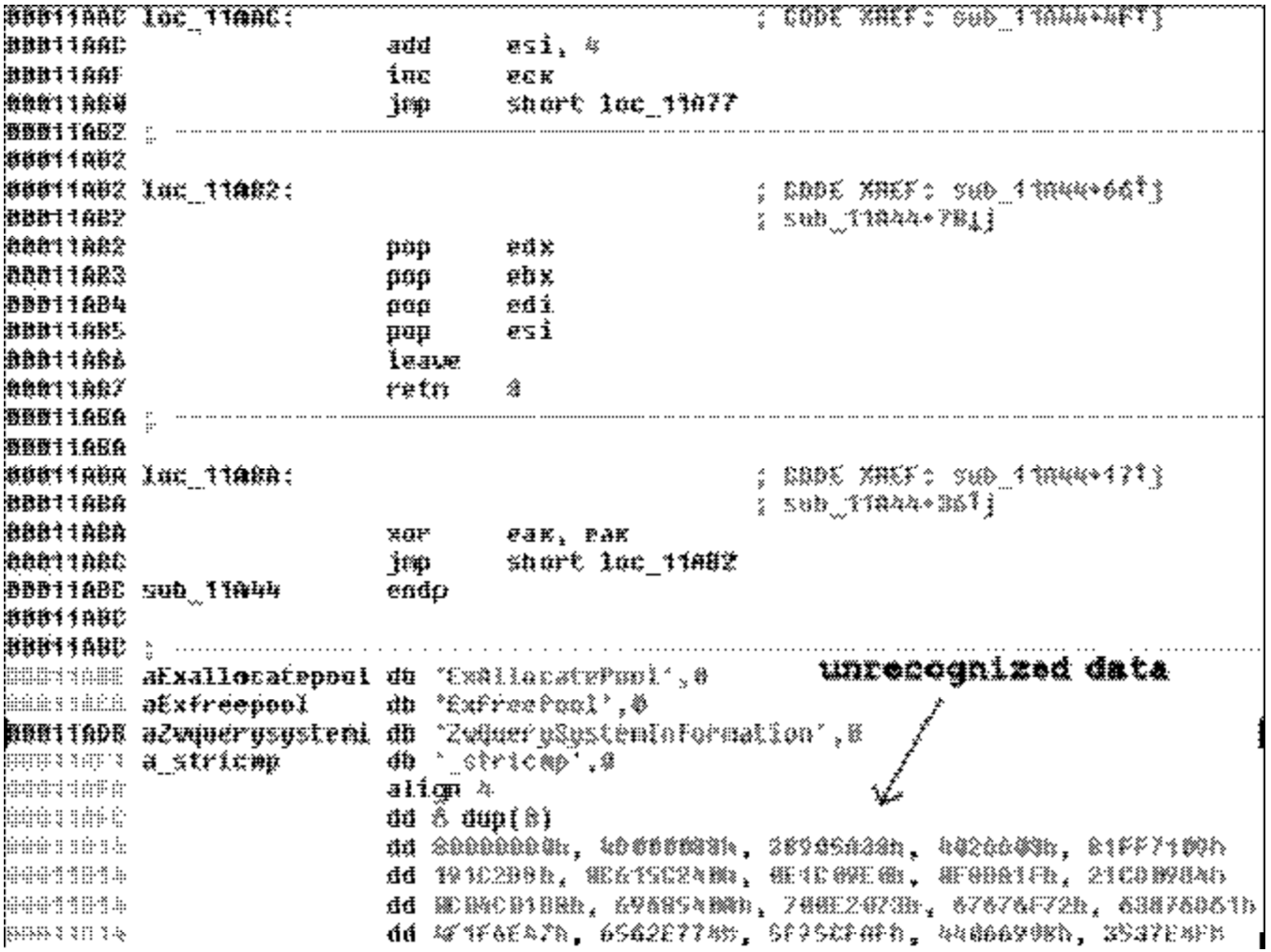
Stage 3: IDA

- Load dumped file into IDA

```
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
```

Stage 3: IDA

- Obfuscated data
  - Can not use the previous approach

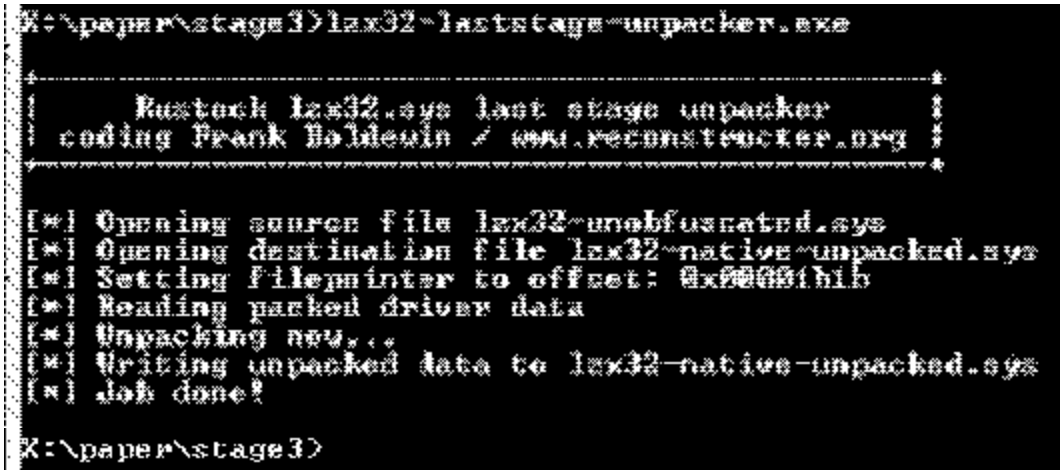


Stage 3

- Read code at 0x116a4
  - Import APIs from NTOSKRNL
  - Query system modules running
  - Allocate kernel memory
  - Unpack routine (0x11788)
    - Unpacks code to kernel memory
  - Move unpacked code over packed code area
  - Grab imports from NTOSKRNL and HAL.DLL, destroy PE-Header, rebase API calls
  - Free unused kernel memory
  - JMP EAX at address 0x117c8
- Real driver created dynamically
  - Must rip the unpacking code at 0x117d3 and dump whole data as file before PE-Header destroyed and driver code rebased

Stage 3

- Program included



Stage 3: Reversing Rustock.B

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

Doing it faster with a kernel debugger

- SoftICE+ICEEXT
  - Special function in NTOSKRNL.EXE to load driver
    - IopLoadDriver
    - Is not exported by default
  - Need proper .pdb file of NTOSKRNL.EXE from Microsoft server
  - Need to convert it to SoftICE format .nms
  - Problem: SoftICE symbol retriever unreliable
  - Read Frank Boldewin's SoftICE howto
    - http://reconstructer.org
- Alternative
  - Leech Windows Debugging Tools from MSFT
  - Read paper for recipe

Cleanup

- Run RkUnhooker