

Reversing

Definition

- "Reverse engineering is the process of analyzing a subject system to create representations of the system at a higher level of abstraction."

Chikofsky and Cross

Why Reverse Engineer?

- Sony BMG DRM rootkit!
 - Reverse Engineered by Mark Russinovich of SysInternals
 - First4Internet designed as DRM enforcer
 - Will hide any file / reg key / process whose name begins with \$sys\$
 - Will screw your windows box if you delete the drivers

Why Reverse Engineer?

- Fun / Challenge
- Compatibility / Interoperability
 - IBM PC Clone (Compaq BIOS) (Clean Room / Chinese Wall)
 - OpenOffice.org / Microsoft Office File Formats (SUN Dublin)
 - Windows File Sharing (Samba) Protocol RE
- Lost Source Code (This is common)
- Legacy Code (Original Coder Unavailable, No Source, Y2K)
- Bug Hunting (Again, no source. Popular now with malware authors. New Spl01tz!)
- Virus Analysis

Legalities

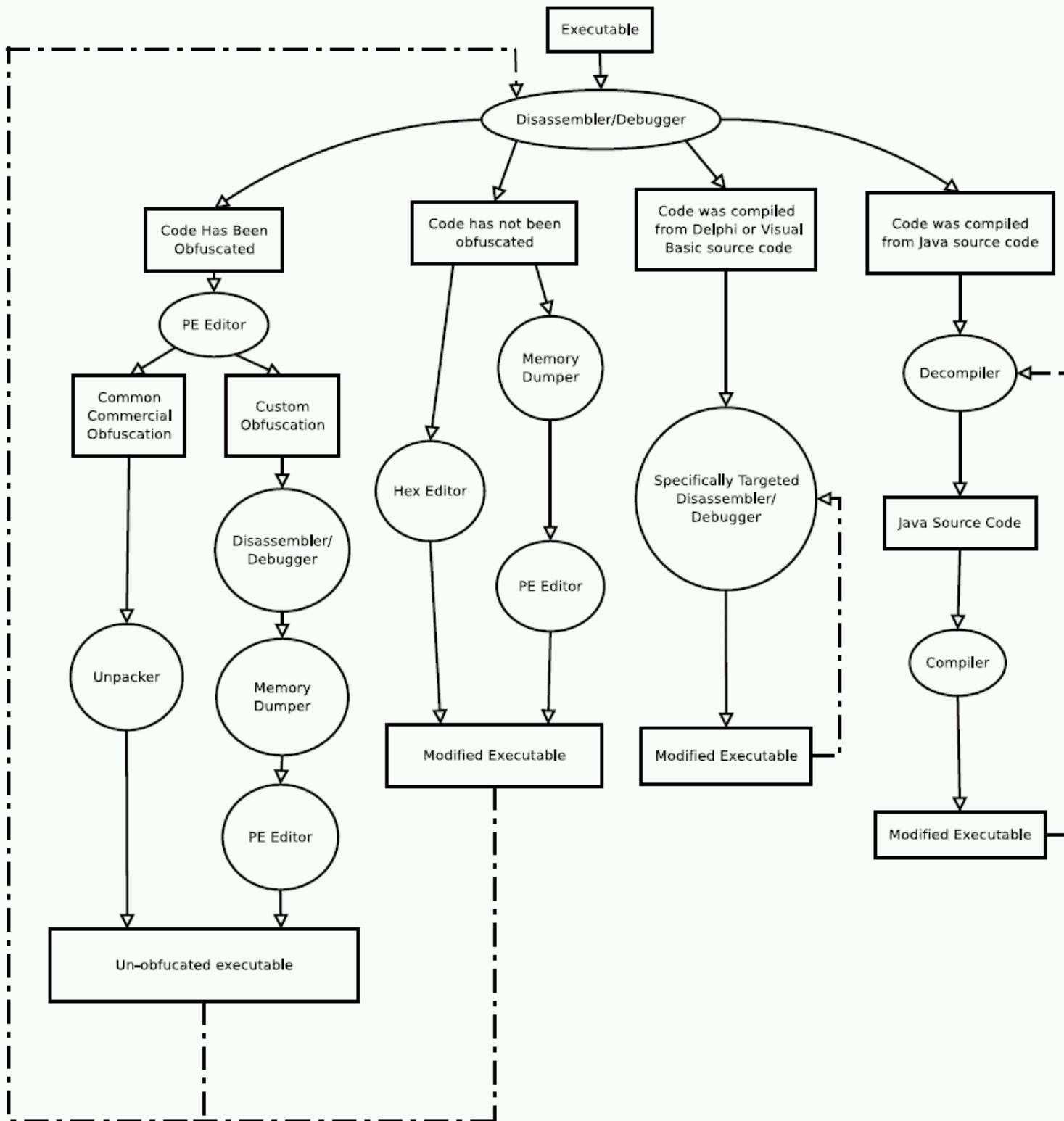
- US DMCA (Danger Here)
 - Exemption acts of reverse engineering aimed at interoperability of file formats and protocols
 - “The exception allows reverse engineering of computer programs if the reverse engineer lawfully obtains the program, seeks permission from the copyright owner, only uses the results of their efforts to create an interoperable computer program and does not publish the results”.
- Dmitry Sklyarov @ DEFCON (July 2001)
 - Jailed for several weeks, detained for 5 months
 - Advanced E-book Processor (ElcomSoft Co. Ltd)
 - Convert Adobe E-book to PDF (Removing Restrictions)
 - Dec 2002 Jury Acquitted Elcomsoft of all charges

Legalities

- EU - COUNCIL DIRECTIVE of 14 May 1991 on the **legal protection of computer programs** (91/250/EEC)
 - Article 5 - Exceptions to the Restricted Acts***
 1. Alterations for where necessary for intended purpose of program, including error correction
 2. The making of a back-up copy
 3. Observe, study or test...to determine the ideas and principles which underlie any element of the program
 - Article 6 - Decompilation***
 - Ok for purposes of Interoperability

Types of RE (Static / Live)

- Live
 - After the fact checks (netstat, directory listings etc)
 - Monitoring Tools (Ethereal, Regmon, Filemon, TDIMon)
 - Debuggers (Softice, Ollydbg, VS, windbg)
 - Memory dumpers
- Static (More Difficult)
 - Hex Editors, Strings tool, PE editors
 - Disassemblers, Decompilers



Live RE: Monitors

- Process Explorer
 - Tells what processes are currently running.
- FileMon
 - Monitors files for operations.
- RegMon
 - Monitors registry for operations.
- RegShot
 - Takes a snapshot of the registry and associated files .

Regmon...

The image shows two windows from the Sysinternals suite. The top window is 'Registry Monitor' and the bottom is 'File Monitor'. Both show a list of system events.

Registry Monitor - Sysinternals: www.sysinternals.com

#	Time	Process	Request	Path	Result	Other
23679	47.42653656	explor...	CloseKey	HKLM\SYSTEM\CurrentControlSet\Se...	SUCCE...	
23680	47.42657852	explor...	OpenKey	HKLM\SYSTEM\CurrentControlSet\Se...	SUCCE...	Access: 0x...
23681	47.42658997	explor...	QueryValue	HKLM\SYSTEM\CurrentControlSet\Se...	SUCCE...	0x0
23682	47.42660904	explor...	QueryValue	HKLM\SYSTEM\CurrentControlSet\Se...	SUCCE...	"255.255.2...
23683	47.42662048	explor...	QueryValue	HKLM\SYSTEM\CurrentControlSet\Se...	SUCCE...	"255.255.2...
23684	47.42663574	explor...	CloseKey	HKLM\SYSTEM\CurrentControlSet\Se...	SUCCE...	
23685	47.42667389	explor...	OpenKey	HKLM\SYSTEM\CurrentControlSet\Se...	SUCCE...	Access: 0x...
23686	47.42668533	explor...	QueryValue	HKLM\SYSTEM\CurrentControlSet\Se...	SUCCE...	0x0
23687	47.42670059	explor...	QueryValue	HKLM\SYSTEM\CurrentControlSet\Se...	SUCCE...	"255.255.2...
23688	47.42671204	explor...	QueryValue	HKLM\SYSTEM\CurrentControlSet\Se...	SUCCE...	"255.255.2...
23689	47.42672729	explor...	CloseKey	HKLM\SYSTEM\CurrentControlSet\Se...	SUCCE...	

File Monitor - Sysinternals: www.sysinternals.com

#	Time	Process	Request	Path
584	5:55:00 PM	svchost.exe:196	OPEN	C:\Documents and Settings\All Us...
585	5:55:00 PM	javaw.exe:2044	READ	G:\Media\Incoming\Video\Family (...)
586	5:55:00 PM	javaw.exe:2044	READ	G:\Media\Incoming\Video\Family (...)
587	5:55:00 PM	javaw.exe:2044	READ	G:\Media\Incoming\Video\Family (...)
588	5:55:00 PM	javaw.exe:2044	QUERY INFORMATION	G:\Media\Incoming\Video\Family (...)
589	5:55:00 PM	msnmsgr.exe:384	QUERY INFORMATION	C:\WINDOWS\system32\advapi32...
590	5:55:00 PM	msnmsgr.exe:384	WRITE	C:\Documents and Settings\Mark\...
591	5:55:01 PM	msnmsgr.exe:384	QUERY INFORMATION	C:\WINDOWS\system32\advapi32...
592	5:55:01 PM	msnmsgr.exe:384	WRITE	C:\Documents and Settings\Mark\...
593	5:55:01 PM	msnmsgr.exe:384	QUERY INFORMATION	C:\WINDOWS\system32\advapi32...
594	5:55:01 PM	msnmsgr.exe:384	WRITE	C:\Documents and Settings\Mark\...
595	5:55:01 PM	msnmsgr.exe:384	QUERY INFORMATION	C:\WINDOWS\system32\advapi32...
596	5:55:01 PM	msnmsgr.exe:384	WRITE	C:\Documents and Settings\Mark\...

Network monitors

- TCPView
 - Displays all TCP and UDP open connections and the process that opened and is using the port.
- TDIMon
 - Logs network connectivity, but not packet contents.
- Ethereal/Wireshark
 - Packet Scanner that captures packets and supports the viewing of contents/payload
- Snort
 - IDS / Packet Sniffer
- netcat
 - Network swiss army knife

Ethereal

The screenshot displays the Ethereal network protocol analyzer interface. The window title is "(Untitled) - Ethereal". The menu bar includes File, Edit, View, Go, Capture, Analyze, Statistics, and Help. The toolbar contains various icons for file operations, capture, and analysis. A filter field is present with a dropdown menu and buttons for "Expression...", "Clear", and "Apply".

The main display area shows a list of captured packets with the following columns: No., Time, Source, Destination, Protocol, and Info. The selected packet (No. 1) is a TCP segment from 62.97.180.11 to 192.168.0.4, port 57385 to 3674, with flags [PSH, ACK], Seq=0, Ack=0, and Win=17520.

Below the packet list, the details pane shows the following information:

- Frame 1 (1514 bytes on wire, 1514 bytes captured)
- Ethernet II, Src: 00:12:17:e7:63:39, Dst: 00:11:50:44:1d:0f
- Internet Protocol, Src Addr: 62.97.180.118 (62.97.180.118), Dst Addr: 192.168.0.4 (192.168.0.4)
- Transmission Control Protocol, Src Port: 57385 (57385), Dst Port: 3674 (3674), Seq: 0, Ack: 0, Len: 1460

The bottom pane shows the raw packet data in hexadecimal and ASCII:

```
0000  00 11 50 44 1d 0f 00 12 17 e7 63 39 08 00 45 00  ..PD.... ..c9..E.
0010  05 dc d3 32 40 00 71 06 7d 65 3e 61 b4 76 c0 a8  ...2@.q. }e>a.v..
0020  00 04 e0 29 0e 5a 4f c2 5e 74 0e 89 76 2f 50 18  ...).ZO. ^t..v/P.
0030  44 70 1c 16 00 00 f9 de 05 7f d0 70 9e 84 8e 7b  Dp..... ..p...{
0040  2c 5a 90 3a b5 15 fe 46 f4 a3 47 e7 2d e8 47 ce  ,Z:....F ..G.-.G.
0050  f9 e3 96 e7 2f 2a 07 d7 f2 af 02 4f 3f cb 7e c4  ..../*... ..0?..~.
0060  54 8b 65 28 7f bb b1 3a a4 82 1e e5 07 83 80 5e  T.e(...: .....^
0070  37 17 ec 14 38 66 44 e9 00 e7 db 9b 28 6c 10 55  7...8fd. ....(l.U
0080  a6 69 ad 40 37 1c 08 21 24 22 a8 cf 46 de ea 64  .i.@7...! $"..F..d
```

The status bar at the bottom indicates: File: (Untitled) 842 KB 00:00:00; P: 1305 D: 1305 M: 0

Live RE: Debuggers

- Example tools
 - OllyDbg
 - IDA Pro
 - SoftICE

The screenshot displays the OllyDbg interface for the application 'calc.exe'. The main window shows assembly code for the 'main thread' at address 01012475. The code includes instructions like PUSH, CALL, XOR, MOV, and CMP. A comment indicates a call to 'kernel32.GetModuleHandleA'. The registers window on the right shows the current state of CPU registers, with EIP at 01012475. The memory dump at the bottom shows the return path from kernel32 to the application's module entry point.

Registers (FPU)

EAX	00000000
ECX	0007FFB0
EDX	7C90EB94 ntdll.KiFastSystemCallRet
EBX	7FFDF000
ESP	0007FFC4
EBP	0007FFF0
ESI	FFFFFFFF
EDI	7C910738 ntdll.7C910738
EIP	01012475 calc.<ModuleEntryPoint>
C 0	ES 0023 32bit 0(FFFFFFFF)
P 1	CS 001B 32bit 0(FFFFFFFF)
A 0	SS 0023 32bit 0(FFFFFFFF)
Z 1	DS 0023 32bit 0(FFFFFFFF)
S 0	FS 003B 32bit 7FDE000(FFF)
T 0	GS 0000 NULL
D 0	
O 0	LastErr ERROR_ALREADY_EXISTS (0000)
EFL	00000246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0	empty -UNORM D1D8 01050104 00000000
ST1	empty 0.0
ST2	empty 0.0
ST3	empty 0.0
ST4	empty 0.0
ST5	empty 0.0
ST6	empty 1.000000000000000000000000
ST7	empty 1.000000000000000000000000
FST	4020 Cond 1 0 0 0 Err 0 0 1 0 0 0
FCW	027F Prec NEAR,53 Mask 1 1 1 1

Memory Dump

Address	Hex dump	ASCII
01014000	03 00 00 00 01 00 00 00 20 00 00 00 0A 00 00 00@.....
01014010	0A 00 00 00 40 00 00 00 53 00 63 00 69 00 43 00S.c.i.C.
01014020	61 00 6C 00 63 00 00 00 00 00 00 2E 00 00 00	a.l.c.....
01014030	00 00 00 00 00 00 00 00 2C 00 00 00 00 00 00
01014040	00 00 00 00 30 00 00 00 01 00 00 00 00 00 57 00@.....w.
01014050	58 00 56 01 5C 02 5D 02 07 03 59 03 5E 03 5A 03	X.V@]@.v^*2*
01014060	58 03 5F 04 00 00 00 00 FF FF FF FF FF FF FF	[^.....
01014070	FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
01014080	00 00 00 00 00 00 00 00 EC 15 00 01 00 00 00 00*S.@.
01014090	2E 4B 00 00 00 00 00 00 00 FF 00 50 00 00 00 00	..K.....P..
010140A0	FF 00 00 00 51 00 00 00 FF 00 00 00 52 00 00 00R.....
010140B0	FF 00 00 00 53 00 00 00 00 FF 00 00 54 00 00 00S.....T..
010140C0	00 00 FF 00 55 00 00 00 FF 00 00 00 56 00 00 00U.....
010140D0	FF 00 00 00 57 00 00 00 FF 00 00 00 58 00 00 00W.....X..

Analysing calc: 158 heuristical procedures, 273 calls to known, 167 calls to guessed functions

Paused

Static RE: Strings tools, Hex editors

- Basic
 - BinText/Strings
 - Extract strings from executable
 - HHD Hex Editor
 - UltraEdit
- Advanced
 - WinHex (RAM disassembly)
 - Tsearch (RAM disassembly)
 - Hex Workshop
 - Hackman Hex Editor
 - Hiew (Hackers View)

Static RE: PE editors

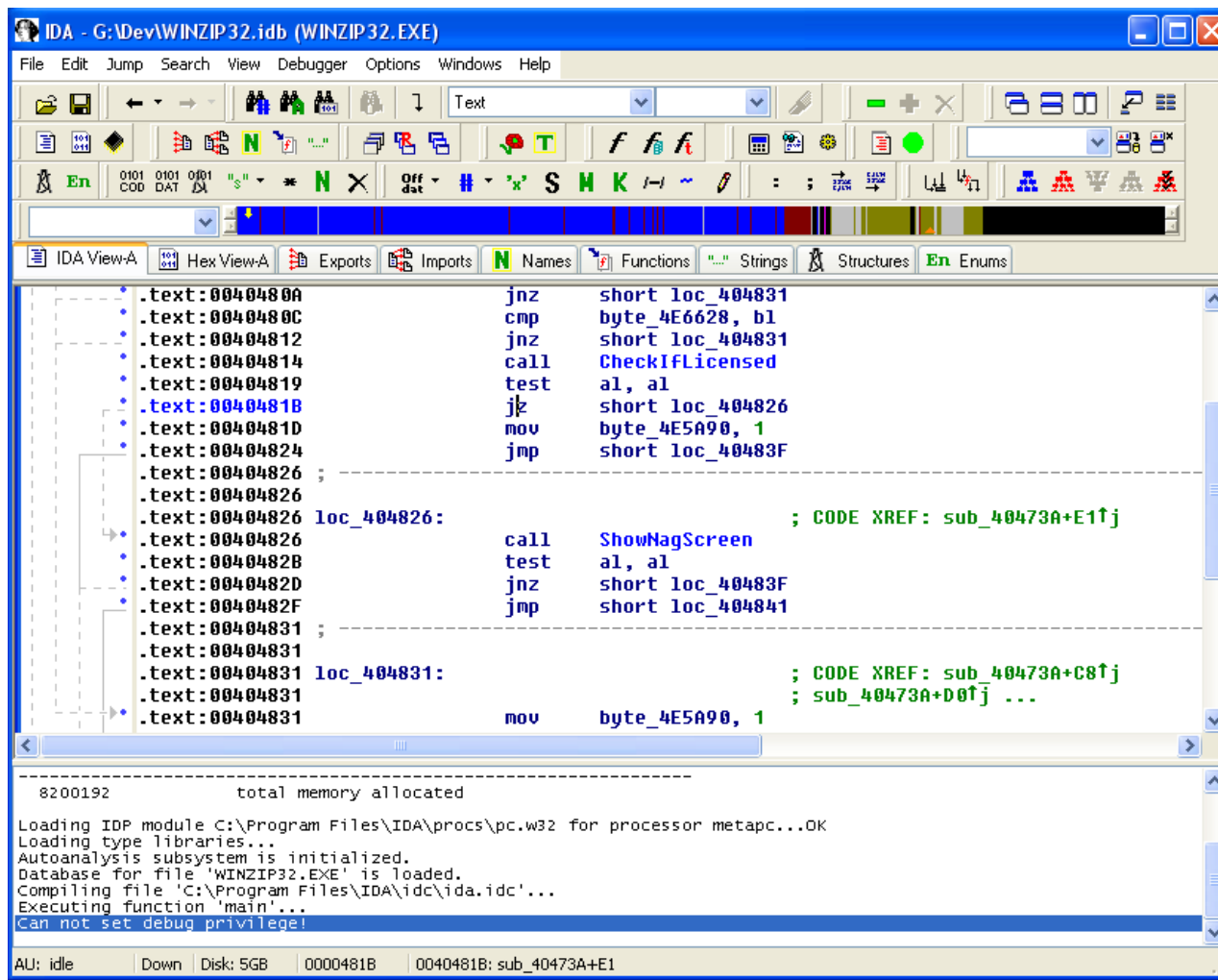
- PE editors parse and interpret binary, create binaries from memory dumps
 - PEditor
 - Procdump
 - LordPE
 - PEDump
 - OllyDump
 - REVirgin
 - ImpRec
 - APISpy

Static RE: Disassemblers

- Take binary as input and produces assembly code as output
- Built-in to debuggers
- Stand-alone disassemblers
 - W32DASM
 - BORG
 - Hackman disassembler
 - Phoenix/DSM Studio
- Language-specific disassemblers and debuggers
 - Delphi: DeDe
 - VB: VBReFormer
 - C: REC, DCC, DisC (Borland TurboC)
 - Java: JAD, JODE, DAVA

Disassemblers (IDA Pro)

- Library Identification/Recognition, parameter tracking, stack variable tracking, call graphs, etc.



The screenshot shows the IDA Pro interface for the file 'G:\Dev\WINZIP32.idb (WINZIP32.EXE)'. The main window displays assembly code in the 'IDA View-A' pane. The code is organized into segments and functions. The current view shows the following assembly instructions:

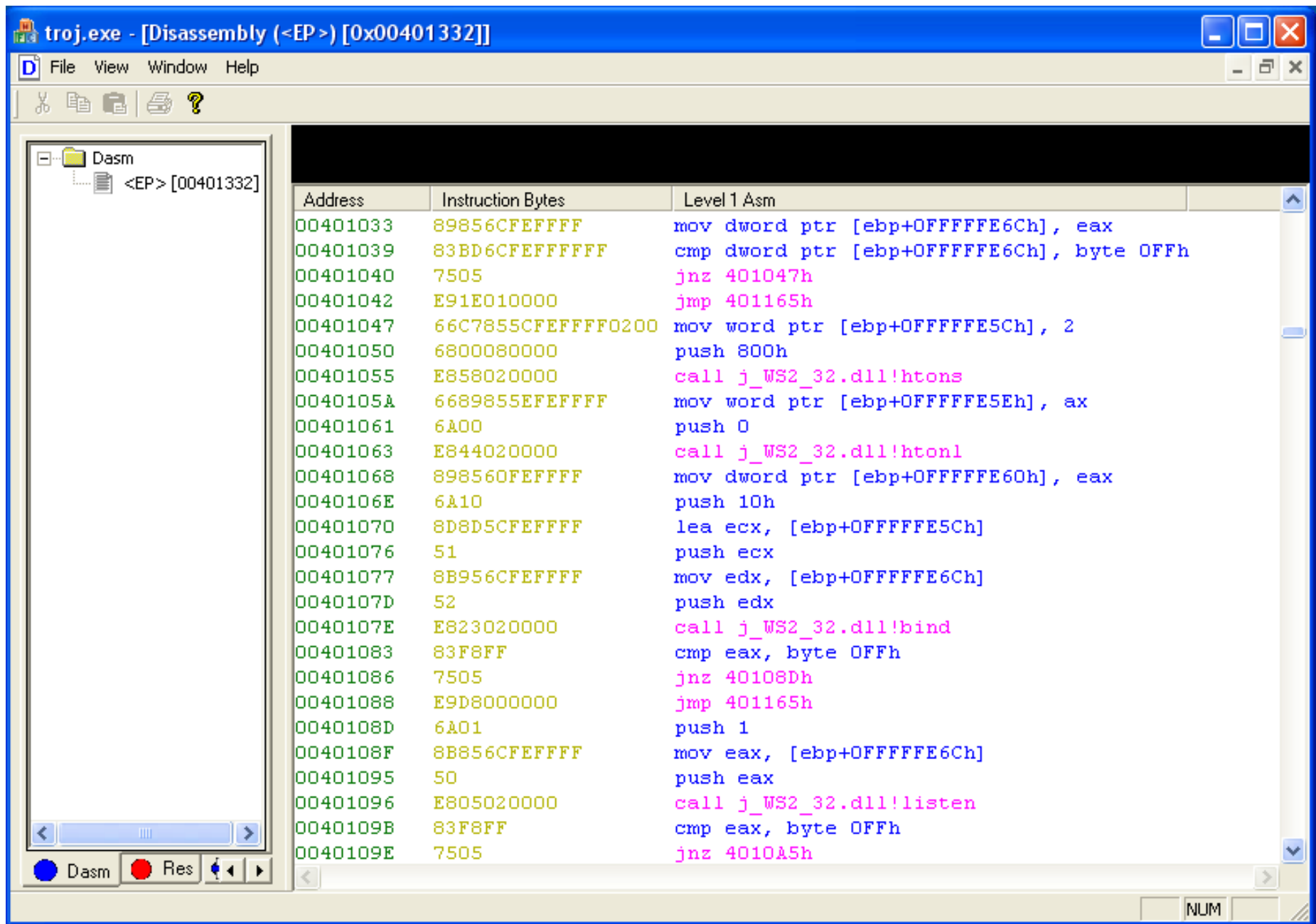
```
.text:0040480A      jnz     short loc_404831
.text:0040480C      cmp     byte_4E6628, b1
.text:00404812      jnz     short loc_404831
.text:00404814      call   CheckIfLicensed
.text:00404819      test   al, al
.text:0040481B      jz     short loc_404826
.text:0040481D      mov    byte_4E5A90, 1
.text:00404824      jmp    short loc_40483F
.text:00404826      ; -----
.text:00404826      loc_404826:
.text:00404826      call   ShowNagScreen      ; CODE XREF: sub_40473A+E1fj
.text:0040482B      test   al, al
.text:0040482D      jnz     short loc_40483F
.text:0040482F      jmp    short loc_404841
.text:00404831      ; -----
.text:00404831      loc_404831:
.text:00404831      ; CODE XREF: sub_40473A+C8fj
.text:00404831      ; sub_40473A+D0fj ...
.text:00404831      mov    byte_4E5A90, 1
```

The bottom pane shows the status bar with the following information:

```
8200192          total memory allocated
Loading IDP module C:\Program Files\IDA\procs\pc.w32 for processor metapc...OK
Loading type libraries..
Autoanalysis subsystem is initialized.
Database for file 'WINZIP32.EXE' is loaded.
Compiling file 'C:\Program Files\IDA\idc\ida.idc'...
Executing function 'main'...
Can not set debug privilege!
```

The status bar at the very bottom indicates: AU: idle | Down | Disk: 5GB | 0000481B | 0040481B: sub_40473A+E1

Disassemblers (DSM Studio) aka Phoenix



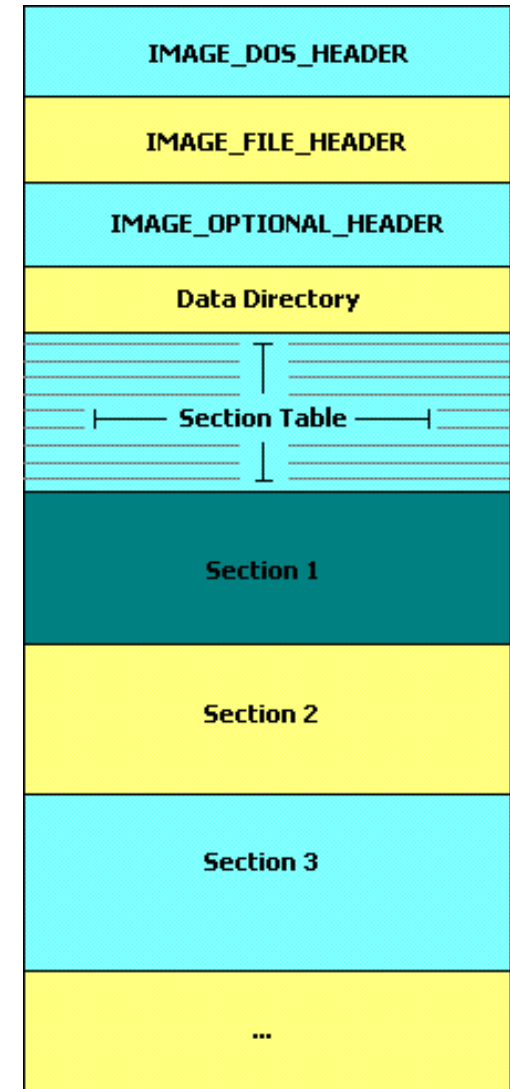
The screenshot shows the DSM Studio disassembler interface. The title bar reads "troj.exe - [Disassembly (<EP>) [0x00401332]]". The menu bar includes "File", "View", "Window", and "Help". A toolbar with icons for copy, paste, and help is visible. On the left, a tree view shows the disassembly structure: "Dasm" > "<EP> [00401332]". The main window displays a table of assembly instructions with columns for Address, Instruction Bytes, and Level 1 Asm.

Address	Instruction Bytes	Level 1 Asm
00401033	89856CFEFFFFFF	mov dword ptr [ebp+OFFF7FE6Ch], eax
00401039	83BD6CFEFFFFFF	cmp dword ptr [ebp+OFFF7FE6Ch], byte 0FFh
00401040	7505	jnz 401047h
00401042	E91E010000	jmp 401165h
00401047	66C7855CFEFFFFFF0200	mov word ptr [ebp+OFFF7FE5Ch], 2
00401050	6800080000	push 800h
00401055	E858020000	call j_WS2_32.dll!htons
0040105A	6689855EFEFFFFFF	mov word ptr [ebp+OFFF7FE5Eh], ax
00401061	6A00	push 0
00401063	E844020000	call j_WS2_32.dll!htonl
00401068	898560FEFFFFFF	mov dword ptr [ebp+OFFF7FE60h], eax
0040106E	6A10	push 10h
00401070	8D8D5CFEFFFFFF	lea ecx, [ebp+OFFF7FE5Ch]
00401076	51	push ecx
00401077	8B956CFEFFFFFF	mov edx, [ebp+OFFF7FE6Ch]
0040107D	52	push edx
0040107E	E823020000	call j_WS2_32.dll!bind
00401083	83F8FF	cmp eax, byte 0FFh
00401086	7505	jnz 40108Dh
00401088	E9D8000000	jmp 401165h
0040108D	6A01	push 1
0040108F	8B856CFEFFFFFF	mov eax, [ebp+OFFF7FE6Ch]
00401095	50	push eax
00401096	E805020000	call j_WS2_32.dll!listen
0040109B	83F8FF	cmp eax, byte 0FFh
0040109E	7505	jnz 4010A5h

At the bottom of the window, there are buttons for "Dasm" (blue) and "Res" (red), and a "NUM" button in the bottom right corner.

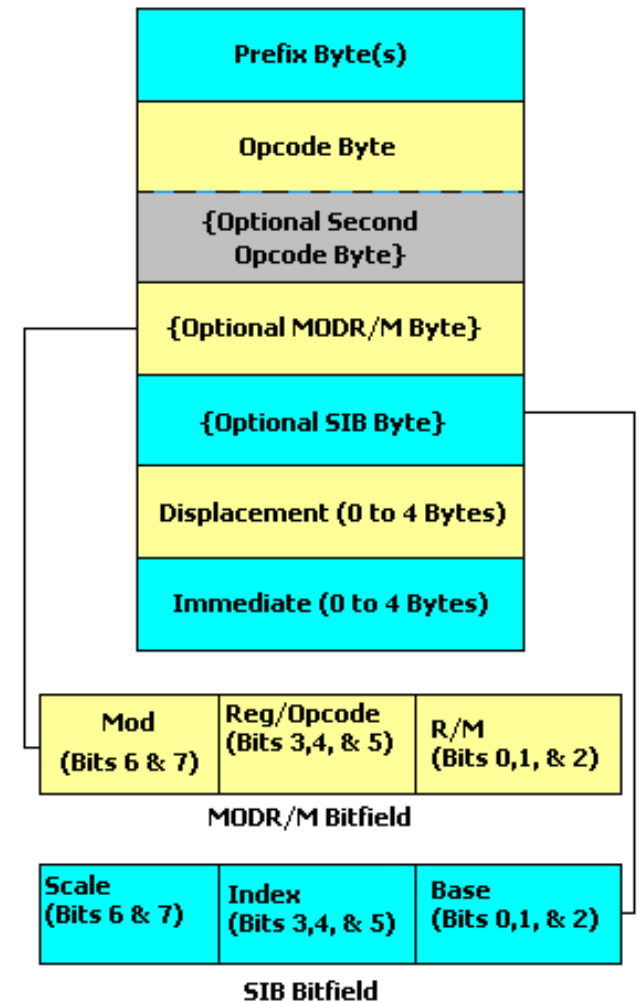
Designing A Disassembler – A Walkthrough

- PE File Format
 - Now Pretty Well Documented... (Google it)
 - Based on COFF Format
 - Magic Numbers, Headers, Tables, Directories, Sections
- Simply Overlay Data with C Structures
- Load File as OS Loader Would
- Identify Entry Points (Default & Exported)



Designing A Disassembler – A Walkthrough

- Intel x86 Instruction Format
 - Excellently Documented
 - Extended From 8086 format
 - Lots of Hacks, Rules & Exceptions
- Tough Parsing Code (ANSI C Conformant)
 - Never want to open that file again (~3K LOC excluding tables)
 - Lots of hard coded tables
 - Standard, FPU, MMX, SSE, SSE2
 - Exception Tables
 - ModRM & SIB encoding Tables
 - Tables allow fast lookup
- Main Function: DisassembleSingleInstruction()



Designing A Disassembler – A Walkthrough

- Disassembly Engine to emulate processor
 - Follow instruction flow in BFS manner until termination
 - Collecting external references (CALL, JMP, Jxx conditional)
 - Termination conditions (RET, JMP)
 - Repeat process for all external references

Designing A Disassembler – A Walkthrough

- Analysis Engine
 - Create Symbols
 - Identify Data Types (DWORD / String etc)
 - Identify High Level Functions
 - Identify Stack Parameters

Disassembly example

```
int main(int argc, char **argv)
{
    WSADATA wsa;
    SOCKET s;
    struct sockaddr_in name;
    unsigned char buf[256];

    // Initialize Winsock
    if(WSAStartup(MAKEWORD(1,1), &wsa))
        return 1;

    // Create Socket
    s = socket(AF_INET, SOCK_STREAM, 0);

    if(INVALID_SOCKET == s)
        goto Error_Cleanup;

    name.sin_family = AF_INET;
    name.sin_port = htons(PORT_NUMBER);
    name.sin_addr.S_un.S_addr = htonl(INADDR_ANY);

    // Bind Socket To Local Port
    if(SOCKET_ERROR == bind(s, (struct
        sockaddr*)&name, sizeof(name)))
        goto Error_Cleanup;

    // Set Backlog parameters
    if(SOCKET_ERROR == listen(s, 1))
        goto Error_Cleanup;
}
```

```
push ebp
mov ebp, esp
sub esp, 2A8h
lea eax, [ebp+0FFFFFFE70h]
push eax
push 101h
call 4012BEh
test eax, eax
jz 401028h
mov eax, 1
jmp 40116Fh
push 0
push 1
push 2
call 4012B8h
mov dword ptr [ebp+0FFFFFFE6Ch], eax
cmp dword ptr [ebp+0FFFFFFE6Ch], byte 0FFh
jnz 401047h
jmp 401165h
mov word ptr [ebp+0FFFFFFE5Ch], 2
push 800h
call 4012B2h
mov word ptr [ebp+0FFFFFFE5Eh], ax
push 0
call 4012ACh
mov dword ptr [ebp+0FFFFFFE60h], eax
push 10h
lea ecx, [ebp+0FFFFFFE5Ch]
push ecx
mov edx, [ebp+0FFFFFFE6Ch]
push edx
call 4012A6h
cmp eax, byte 0FFh
jnz 40108Dh
jmp 401165h
push 1
mov eax, [ebp+0FFFFFFE6Ch]
push eax
call 4012A0h
cmp eax, byte 0FFh
jnz 4010A5h
jmp 401165h
```

Disassembly example

```
// Wait for connection from attacker
if(INVALID_SOCKET == (g_current =
    accept(s, NULL, 0)))
    goto Error_Cleanup;

// Send Welcome Message!
if(SOCKET_ERROR ==
    send(g_current, kszWelcome,
        strlen(kszWelcome), 0))
    goto Error_Cleanup;

// Receive Command
recv(g_current, buf, 255, 0);

// Execute Command
if('v' == buf[0])
    DisplayVersion();
else if('d' == buf[0])
    DisplayDiskSpace();
else if('l' == buf[0])
    LaunchCalculator();

// Cleanup
closesocket(g_current);
WSACleanup();
return 0;

Error_Cleanup:
WSACleanup();
return 1;
}
```

```
push 0
push 0
mov ecx, [ebp+0FFFFFFE6Ch]
push ecx
call 40129Ah
mov [4030C4h], eax
cmp dword ptr [4030C4h], byte 0FFh
jnz 4010C8h
jmp 401165h
mov edx, [ebp+0FFFFFFE6Ch]
push edx
call 401294h
push 0
mov eax, [403088h]
push eax
call 4012D0h
add esp, byte 4
push eax
mov ecx, [403088h]
push ecx
mov edx, [4030C4h]
push edx
call 40128Eh
cmp eax, byte 0FFh
jnz 4010FFh
jmp 401165h
push 0
push 0FFh
```

```
lea eax, [ebp+0FFFFFFD58h]
push eax
mov ecx, [4030C4h]
push ecx
call 401288h
movzx edx, byte ptr [ebp+0FFFFFFD58h]
cmp edx, byte 76h
jnz 40112Ch
call 401180h
jmp 401150h
movzx eax, byte ptr [ebp+0FFFFFFD58h]
cmp eax, byte 64h
jnz 40113Fh
call 401200h
jmp 401150h
movzx ecx, byte ptr [ebp+0FFFFFFD58h]
cmp ecx, byte 6Ch
jnz 401150h
call 401270h
mov edx, [4030C4h]
push edx
call 401294h
call 401282h
xor eax, eax
jmp 40116Fh
call 401282h
mov eax, 1
mov esp, ebp
pop ebp
ret
```

Anti-debugging

Anti-debuggers

- Making reverse-engineering and disassembly painful
 - Polymorphism
 - Metamorphism
 - Encryption
 - Interrupt disabling
 - Debugger detection
 - Behavior modification
 - Crashing debugger
- Links
 - <http://www.textfiles.com/virus/adebgtut.txt>
 - <http://anti-debugging.qarchive.org>
 - <http://rdist.root.org/2007/04/19/anti-debugger-techniques-are-c>
 - http://www.openrce.org/reference_library/anti_reversing
 - <http://reconstructor.org>

Modify Interrupts

- Overwrite Interrupt Vector of INT 1/3 to point to garbage code
 - INT 1 = Debug Single Step
 - INT 3 = Debug Breakpoint
 - Point to garbage so debugger crashes
 - Must skip instructions to counter
- Run long loop of INT 3 instructions
 - Slows down AV or debugging process
 - Must NOP out INT 3 to counter
- Turn off keyboard interrupts

```
IN AL,20h
OR AL,02
OUT AL,20
<virus code>
IN AL,20
AND AL, NOT 2
OUT AL,20
```

Modify interrupts

- Detect INT 1 tracing by stack inspection
 - INT 1 overwrites 6 bytes below SP with return values for IP, CS, and Flags

```
PUSH AX  
POP AX  
DEC SP  
DEC SP  
POP BX  
CMP AX, BX  
JNE CODE_IS_TRACED
```
- Force INT 1/INT 3 tracing to disable code
 - Hide critical value (i.e. decryption key) on stack directly without modifying stack pointer
 - Debugger will overwrite value if it runs

Protecting anti-debug code

- Use anti-debug routines as decryption key
 - Retrieve a byte from routines to modify decryption routine
 - Prevents AV and anti-debugger from “NOP”ing out your traps
- Running line
 - Hook INT 1
 - Decrypt each instruction just before it is run
 - Re-encrypt each instruction just after it is run
 - Only one instruction at a time is decrypted in memory

Detection tricks

- `IsDebuggerPresent()`
 - Returns 0 if current process is not running in the context of a debugger
 - Search for PEB (Process Environment Block) for the field `IsDebugged`
 - `CheckRemoteDebuggerPresent()` checks if debugger is a separate, parallel process that is attached to process
 - Windows NT
 - `NTGlobalFlag` for WindowsNT
 - `NtQueryInformationProcess`
 - Can be easily subverted

Detection tricks

- Detect presence of INT 41/INT 2dh
 - Used by Windows debugging interface
- Detect INT 68h hooking (SoftICE)
- Detect files and registry keys created by debugger
 - szOllyKey, szIsSICEKey
 - Attempt to create SoftICE driver to detect presence
 - Look for \system32\drivers\WINICE.dat
 - `\\.\SICE`, `\\.\SIWDEBUG`, etc.
- Detect debug processes or hooks
 - Check for NTice service via `OpenService()`
 - OllyInvisible hooking of `CsrGetProcessId`
 - Olly HideDebugger detour modifications to `OpenProcess()`
 - Scan every running process and read out memory address `0x004B064B` (where OllyDB stores some of its strings)

Detection tricks

- Scan and detect 0xCC opcode (int 3)
- Scan low memory for specific names installed by SoftICE such as “WINICE.BR”, “SOFTICE1”
- Examine timing
 - Use rdtsc to measure cycles
 - Measure system calls to detect virtualization or syscall hijacks
 - Vmware, Sebek, UML, etc.
 - VMEDetect using rdtsc trick
- Detecting VMware
 - Execute VMware backdoor I/O function call
 - Check location of IDT

Detection tricks

- Check registers and flags used for debug purposes
 - TRAP bit in EFLAGS register
 - DR0-DR7
 - Set handler apriori and cause an error in code (divide by zero)
 - When context switching to handler, debug registers saved on user stack
 - Read and write these values
 - PEB ProcessHeap flag
- Execute exception with Trap flag set
 - No debugger = SEH will occur
 - Debugger = SEH will not occur

Crashing tricks (landmines)

- Exploit software errors in debuggers
 - Confuse LordPE and ProcDump dumping by reporting inconsistent process image size
 - Crash SoftICE with poorly formatted strings
 - Crash SoftICE with illegal form of instructions
 - Crash OllyDBG with format string vulnerability
 - OutputDebugString()
 - Crash OllyDBG with bogus PE Headers

Confusion tricks

- OllyDBG does not handle instruction prefixes well
 - Insert prefix one byte before SEH
 - OllyDBG skips it, but SEH runs with no debugger
- OllyDBG mishandling INT 3
 - Set an SEH before INT3
- OllyDBG memory handling
 - PAGE_GUARD used as memory break-point
 - Set SEH and execute PAGE_GUARD exception
 - With OllyDBG, goes to MemBpx and continues execution
 - Without OllyDBG, SEH code executed

Packers

Packers

- <http://www.woodmann.com/fravia/packers.htm>
- Packers compress executables into an equivalent self-extracting one
 - Make smaller
 - Resist analysis and modification
 - Must unpack and rebuild to do so
- Examples
 - Protect!, ICE (COM only), TinyProg, PkTiny, Microsoft EXE Pack, LZEXE, PKLite, PROPACKER, DIET
 - SEA-AXE, PGMPak, OPTLink, DeltaPack
 - AsPack <http://www.aspack.com>

Unpacking

- Packing identifiers
 - PE iDentifier (PEiD)
- Unpackers
 - GUW (Generic Unpacker for Windows)
 - ProcDump unpacking wizard
 - Upx
 - Tron
 - Xopen
 - Unp
 - StickBuster
 - UnNFO
- Reversing tools with unpacking facilities
 - HBGary (www.hbgary.com)
- Note: some packers are highly configurable and require memory dump after unpacking operation

Unpacking

- Example target
 - <http://www.shelltoys.com/files/cmset.exe>
 - Packed with ASProtect
- Steps
 - Dumping the App (IceDump, Peditor)
 - Adding a section for new IAT (Peditor)
 - Create new IAT (Revirgin)
 - Combine dump and IAT (HexWorkshop)
 - Update Entrypoint, update IAT rva and size (Peditor)
 - Run (SoftICE)

Examples

- http://www.woodmann.com/fravia/text/eb_tut31.txt
 - Unpacking notepad.exe packed with Shrinker 3.4
 - ProcDump, SoftICE, Symbol Loader
- http://www.woodmann.com/fravia/text/eb_tut32.txt
 - Unpacking notepad.exe packed with NeoLite v2.0
 - ProcDump, SoftICE, Symbol Loader
- http://www.woodmann.com/fravia/predator_unpacking.htm

Example #1

- http://www.woodmann.com/fravia/text/eb_tut33.txt
- Unpacking notepad.exe packed with PECompact
 - ProcDump, SoftICE, Symbol Loader
- Open with Symbol Loader and SoftICE

```
0040AC44  FFFF                INVALID
0040AC4C  9C                 PUSHFD
0040AC4D  60                 PUSHAD
0040AC4E  E802000000        CALL     0040AC55
```

**If you step over this CALL using F10, the program will run.

Thus, reload the program and step into this CALL using F8 next time.

- Breaks at entry point of unpacking code
- First call is unpacking routine
 - Step through it

Example #1

- Lots of conditional loops

```
aaaaaaaa
...
wwwwwwww
xxxxxxx JNZ zzzzzzzz      <-- Loop back to aaaaaaaaa
yyyyyyyy JMP aaaaaaaaa
zzzzzzz New Instructions
```

- Set bpx on zzzzzzzz

```
0040CA83 8BBD2E744000      MOV     EDI, [EBP+0040742E]
0040CA89 E85E040000        CALL   0040CEEC
0040CA8E 61                POPAD
0040CA8F 9D                POPFD
0040CA90 50                PUSH   EAX
0040CA91 68CC104000        PUSH   004010CC
0040CA96 C20400           RET     0004
```

- POPAD and POPFD used by unpackers a lot
 - POPAD: Pops all registers off stack and restores them
 - POPFD: Pops EFLAGS register from stack
- Push of 0x004010cc like other examples
- Set breakpoint there and dump memory image

Example #2

- http://www.woodmann.com/fravia/volati_s.htm
- Unpacking calc.exe packed with ASPack
 - Much of the same as before
 - Slightly different entry into unpacked code

```
015F:01017554 MOV     [ESP+1C],EAX
015F:01017558 POPAD
015F:01017559 JNZ     01017563          (JUMP )
015F:01017563 PUSH    EAX    *** Take note of the value
of EAX!
015F:01017564 RET     *** Stop here!!!
```

Highlights of other examples

- AsProtect
 - http://www.woodmann.com/fravia/tsehp_asprotect10.htm
 - http://www.woodmann.com/fravia/tsehp_asprotect105.htm
- Anti-SoftICE code in packer
 - Call createfileA on known SoftICE driver
 - Call getlocaltime to see if being debugged
 - SoftICE uses int 3 for breakpoints
 - Check by triggering an int 3
- Getting to entry point
 - Look for POPAD followed by JMP
 - Dump via ProcDump
 - Fix IAT

Reversing

- FAQ
 - <http://www.woodmann.com/fravia/rce-faq.htm>
- Unpacking
 - <http://www.woodmann.com/fravia/projunpa.htm>
- Other resources
 - http://en.wikibooks.org/wiki/Reverse_Engineering

Obfuscation

Objectives

- Slow reverse engineering process
 - Make automated analysis difficult
 - Make code more complicated
 - Make decompilation difficult
 - Make code unreadable by human

Metrics

- Resilience
 - Irreversibility
- Cost
 - Added run-time or code size
- Stealth
 - Similarity to rest of code

Techniques

- Data obfuscation
- Control flow obfuscation
- Advanced techniques

Data obfuscation

- Renaming variables, procedures, classes, methods
- Deleting comments and spaces
- Inserting dead code
- Variable splitting
- Scalar/object conversion
- Change variable lifetime
- Split/fold/merge arrays
- Change encoding
- Merge scalar variables

Control-flow obfuscation

- Break basic blocks
- Inline methods
- Unroll loops
- Reorder statements
- Reorder loops
- Merge all functions into one

Advanced techniques

- Reuse identifiers
- Misleading comments
- Modify inheritance relations
- Convert static data to procedural data
- Store part of program as text and interpret it only during runtime
- Remove library calls
- Attack specific decompilers and debuggers

Code obfuscators

- Source code
 - Semantic Designs
 - PreEmptive Solutions
- Binary code
 - Y0da's Cryptor
 - NFO
 - ASProtect
 - Armadillo

Shiva (Mehta/Clowes 2003)

- Outer encryption layer
 - Defeats “strings”
 - Slows access to protected code
- TRAP flag detection
 - Defeat single-stepping
- “checkme” data check (canary)
- ptrace defense
 - Exits if ptrace active
 - Clones itself and two copies ptrace each other to prevent additional PTRACE_ATTACH “inter-pttrace”
- Timing checks
- AES, password protected middle encryption layer
- Inner encryption layer
 - Run-time protection

Shiva (Mehta/Clowes 2003)

- /proc defenses
 - Only portions of binary decrypted at a given time
- INT 3 instruction replacement
 - Some instructions replaced with INT 3
 - Instructions emulated in INT 3 handler
 - If debugger uses INT 3, code will be missing
- Jumping into middle of instructions
- Polymorphic code generation

Reversing Shiva

- Use similar techniques to run partially and dump images
 - Scripted decryption via IDA scripts
 - Virtual x86 plugin for IDA

.NET reversing

- Reversing tutorial on .NET
 - <http://accessroot.com>
 - <http://www.blong.com/Conferences/DCon2003/ReverseEngineering/ReverseEngineering.htm>
- Tools
 - ILDASM
 - Disassembler that comes with .NET framework SDK
 - Reflector
 - Dis#

.NET obfuscation

- <http://www.codebreakers-journal.com/index.php?>
- StrongName
 - Verifies code integrity via cryptographic hash calculation
 - Prevents patching
 - Easily bypassed via ildasm edits to remove signature scheme
 - <http://www.andreabertolotto.net/>
 - StrongName Remove
 - Or patch system DLL to make check return valid all the time

.NET obfuscation

- Name obfuscation
 - Change metadata saved with binary to make names either unprintable or random



The screenshot shows the Visual Studio IDE with a disassembler window open. The left pane displays a tree view of the assembly's metadata, including base types, derived types, and various methods and fields, all with obfuscated names. The right pane shows the disassembled code for a method named `x73e711f48aa0238b`. The code is written in C# and includes several labels and control flow statements, demonstrating the effect of name obfuscation on the assembly's metadata.

```
private void x73e711f48aa0238b(object x89797597ff45d640, EventArgs x92a31911cd18ca81)
{
    string text1 = this.xed2399b4564968f9.Text.Trim();
    while (0 == 0)
    {
        if (Operators.CompareString(text1, "", false) != 0)
        {
            break;
        }
        if (-1 != 0)
        {
            return;
        }
    }
    Label_002F:
        x83a7a42c48984168.x2e6643035572cbe8[0].AppendChild(x83a7a42c48984168.xcedf2cc56311efb9);
    Label_0045:
        x83a7a42c48984168.xe4fbed097abe9199.Save(x83a7a42c48984168.x927fbb62f7b835fd);
    return;
    Label_005B:
        x83a7a42c48984168.x6e6254a15847ee4f.Value = text1;
    if (0 != 0)
    {
        goto Label_008F;
    }
}
```

.NET obfuscation

- Flow obfuscation
 - Make msil reading hard by preventing its translation into a HLL
 - Adding boolean checks that are always true or false
 - Splitting source into many segments and connecting them using various branches
 - Mess with stack
 - MSIL is stack-based and will not allow unballanced stack
 - Insert a “pop” that will never run
 - Breaks Reflector

```
.assembly extern mscorlib { }
.assembly extern System{}
.assembly sample {}
.method public hidebysig void Main()
{
    .entrypoint
    br.s start_here
    pop
    start_here:
    ldstr "hello!"
    call void [mscorlib]System.Console::WriteLine(string)
    ret
}
```

.NET obfuscation

- Metadata encryption
 - String references stored as metadata in managed PE file
 - Often the key in reversing
 - Encrypt to hide and decrypt just before use

Dotfuscator

- <http://www.preemptive.com/products/dotfuscator/>
- Uses a variety of mechanisms to obfuscate
- All done after compilation (i.e. does not modify source code)

Dotfuscator

- <http://www.preemptive.com/products/dotfuscator/FAQ.htm>
- “Overload Induction” renaming
 - Identify colliding sets of methods across inheritance hierarchies
 - Rename such sets according to some enumeration (e.g. the alphabet or unprintable characters).
 - Method overloading is induced on a grand scale
 - OI algorithm determines all opportunities for name reuse and takes advantage of them.
 - Can use return type to determine method uniqueness as well
 - Anecdotal evidence
 - 33% of ALL methods were renamed to a single character (such as "a").
 - Typically, 10% more are renamed to "b", etc.
 - overload induction reduces the final program size of obfuscated code.
 - Up to 10% of the size savings in Dotfuscated and DashO'd programs

Dotfuscator

- Undoing Dotfuscator renaming
 - Decompiler needs to implement overload induction themselves (ironically, violating Preemptive's patent in the process) to undo it.
 - Overload induction is provably irreversible
 - The best reversing will come out with a different number of unique methods than the original source code contained
 - Overload induction destroys original overloading relationships
 - In reversed state, there will be no overloaded methods.
 - Grand designers of OO technology implemented overloaded methods as a way of creating "more readable code"
 - By removing that ability, the code has less information in it than before.

Dotfuscator

- Also supports
 - String encryption
 - Incremental obfuscation (for patches)
 - Control-flow obfuscation
 - Breaks loops and other HLL control structures up

Example: Rustock.B

Rustock.B rootkit

- Frank Boldewin, “A Journey to the Center of the Rustock.B Rootkit”, Jan. 20, 2007
 - <http://reconstructor.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

The screenshot shows the OllyDbg interface with assembly code loaded. The address 00401B82 is highlighted, and a context menu is open over it. The menu options include Backup, Copy, Binary, Modify byte, Assemble (Space), Label (:), Comment (;), Breakpoint, Hit trace, Run trace, New origin here (Ctrl+Gray *), Go to, Follow in Dump, Search for, Find references to (Selected address Ctrl+R, Immediate constant), View, Copy to executable, and Analysis.

The assembly code is as follows:

```
00401B50 .: C70424 6B1B41 MOV DWORD PTR SS:[ESP],malware.00401B6B
00401B57 .: C3 RETN
00401B58 .: 5E POP ESI
00401B59 .: 68 5B114000 PUSH malware.0040115B
00401B5F > 8D0D EC586231 LEA ECX,DWORD PTR DS:[3C6258EC]
00401B65 .: 68 70114000 PUSH malware.00401170
00401B6A .: C3 RETN
00401B6B .: 8B3C24 MOV EDI,DWORD PTR SS:[ESP]
00401B6E .: 83C4 04 ADD ESP,4
00401B71 .: 68 E6144000 PUSH malware.004014E6
00401B76 .: C3 RETN
00401B77 > 83EC 04 SUB ESP,4
00401B7D .: 890C24 MOV DWORD PTR SS:[ESP],ECX
00401B7E .: E9 23F9FFFF JMP malware.004014A5
00401B82 18 DB 18
00401B83 34 DB 34
00401B84 2C DB 2C
00401B85 61 DB 61
00401B86 02 DB 02
00401B87 37 DB 37
00401B88 9F DB 9F
00401B89 AC DB AC
00401B8A 4F DB 4F
00401B8B 30 DB 30
00401B8C 40 DB 40
00401B8D 80 DB 80
00401B8E BD DB BD
00401B8F 18 DB 18
00401B90 3E DB 3E
00401B91 CD DB CD
00401B92 A3 DB A3
```

The context menu is open over the address 00401B82, showing the following options:

- Backup
- Copy
- Binary
- Modify byte
- Assemble Space
- Label :
- Comment ;
- Breakpoint
- Hit trace
- Run trace
- New origin here Ctrl+Gray *
- Go to
- Follow in Dump
- Search for
- Find references to Selected address Ctrl+R, Immediate constant
- View
- Copy to executable
- Analysis

The hex dump at the bottom of the window shows the following data:

Address	Hex dump
00400000	4D 5A 80 00 01 00 00 00 04 00 10 00 FF FF 00 00
00400010	40 01 00 00 00 00 00 00 40 00 00 00 00 00 00
00400020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00400030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00400040	0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68

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

points to push 401b82/retn

Address	Disassembly	Comment
0040198D	PUSH malware.00401B82	
00401B82	DB 08	(Initial CPU selection)

set breakpoint and run (f9)

Address	Disassembly	Comment
0040198D	PUSH malware.00401B82	(Initial CPU selection)
00401992	RETN	

Stage 1: Ollydbg

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

The screenshot displays the OllyDbg interface. The main window shows assembly code with addresses from 00401B82 to 00401BFF. The registers window on the right shows the current state of CPU registers, with EIP at 00401B82. A context menu is open over the assembly code, listing various actions such as Backup, Copy, Binary, Modify byte, Assemble, Label, Comment, Breakpoint, Hit trace, Run trace, Go to, Follow in Dump, Search for, Find references to, View, Copy to executable, Analysis, and Bookmark. The 'Analysis' option is highlighted, and a sub-menu is visible with 'Analyse code' (Ctrl+A) and 'Remove analysis from module'.

Address	Hex	Disassembly
00401B82	8B	DB 8B
00401B83	4C	DB 4C
00401B84	24	DB 24
00401B85	04	DB 04
00401B86	E8	DB E8
00401B87	00	DB 00
00401B88	00	DB 00
00401B89	00	DB 00
00401B8A	00	DB 00
00401B8B	5D	DB 5D
00401B8C	83	DB 83
00401B8D	ED	DB ED
00401B8E	09	DB 09
00401B8F	64	DB 64
00401B90	A1	DB A1
00401B91	30	DB 30
00401B92	00	DB 00
00401B93	00	DB 00
00401B94	00	DB 00
00401B95	8B	DB 8B
00401B96	40	DB 40
00401B97	0C	DB 0C
00401B98	9B	DB 9B
00401B99	40	DB 40
00401B9A	1C	DB 1C
00401B9B	8B	DB 8B
00401B9C	00	DB 00
00401B9D	8B	DB 8B
00401B9E	40	DB 40
00401B9F	08	DB 08
00401BA0	8D	DB 8D
00401BA1	B5	DB B5
00401BA2	22	DB 22
00401BA3	07	DB 07
00401BA4	00	DB 00
00401BA5	00	DB 00
00401BA6	8D	DB 8D
00401BA7	BD	DB BD
00401BA8	93	DB 93
00401BA9	08 00 00 E8	ASCII 'E'
00401BB7	0F	DB 0F
00401BB8	84	DB 84
00401BB9	74	DB 74
00401BBA	01	DB 01
00401BBB	00	DB 00
00401BBC	00	DB 00
00401BBD	56	DB 56
00401BBE	89	DB 89
00401BBF	FA	DB FA

Registers (FPU):
EAX: C97D935B
ECX: 00000000
EDX: 00000000
EBX: 7FFDE000
ESP: 0006FFC4
EBP: 0006FFF0
ESI: FFFFFFFF
EDI: 7C920738
EIP: 00401B82

Context Menu:
Backup
Copy
Binary
Modify byte
Assemble Space
Label :
Comment ;
Breakpoint
Hit trace
Run trace
Go to
Follow in Dump
Search for
Find references to
View
Copy to executable
Analysis
Bookmark

Analysis Sub-menu:
Analyse code (Ctrl+A)
Remove analysis from module

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

00401C71	. FF95 97080000	CALL DWORD PTR DS:[EBP+8C31]	kernel32._lcreatA
00401C77	. 89FE	MOV ESI,EBI	
00401C79	. 6A 00	PUSH 0	
00401C7B	. 57	PUSH EDI	
00401C7C	. FF95 C3080000	CALL DWORD PTR SS:[EBP+8C31]	kernel32._lcreat
00401C82	. 5B	POP EBX	
00401C83	. 83F8 FF	CMP EAX,-1	
00401C86	.v 75 1A	JNZ SHORT malware.00401CA2	

rootkit driver gets created here →

The screenshot shows the Registers (FPU) window in OllyDbg. The registers are listed as follows:

EAX	0006FEC0	ASCII "C:\\WINDOWS\\system:
ECX	7C834D89	kernel32.7C834D89
EDX	00000000	
EBX	0006FED3	ASCII ":lzx32.sys"
ESP	0006FDB8	
EBP	00401B82	malware.00401B82
ESI	0006FEC0	ASCII "C:\\WINDOWS\\system:
EDI	0006FEC0	ASCII "C:\\WINDOWS\\system:

A context menu is open over the EDI register, with the following options:

- Increment Plus
- Decrement Minus
- Zero
- Follow in Dump (highlighted)
- Follow in Stack
- View MMX registers
- View 3DNow! registers
- View debug registers
- Appearance

EDI has pointer to driver path+name →

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

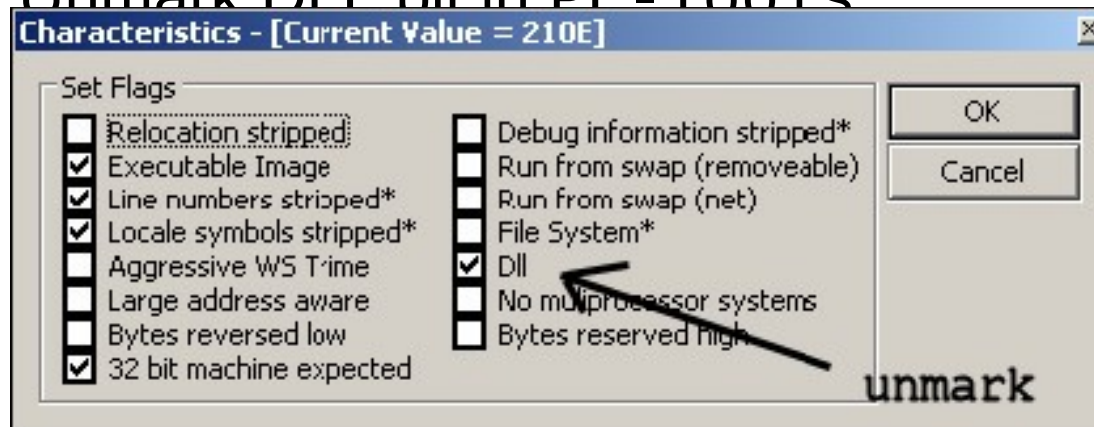
Address	Hex dump	ADS filename
0006FEA0	F4 5B 6F F6 F4 5B 6F F6 00 00 00 00 90 5B 6F F6	Co+Co+....ECo+
0006FEB0	44 5C 6F F6 07 B2 54 80 00 00 DB BA 30 F1 3E 82	Co+TÇ.. 0:>é
0006FEC0	43 3A 5C 57 49 4E 44 4F 57 53 5C 73 79 73 74 65	C:\WINDOWS\sysme
0006FED0	6D 33 32 3A 6C 7A 78 33 32 2E 73 79 30 00 BF E1	m32:lzx32.sys.7B

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

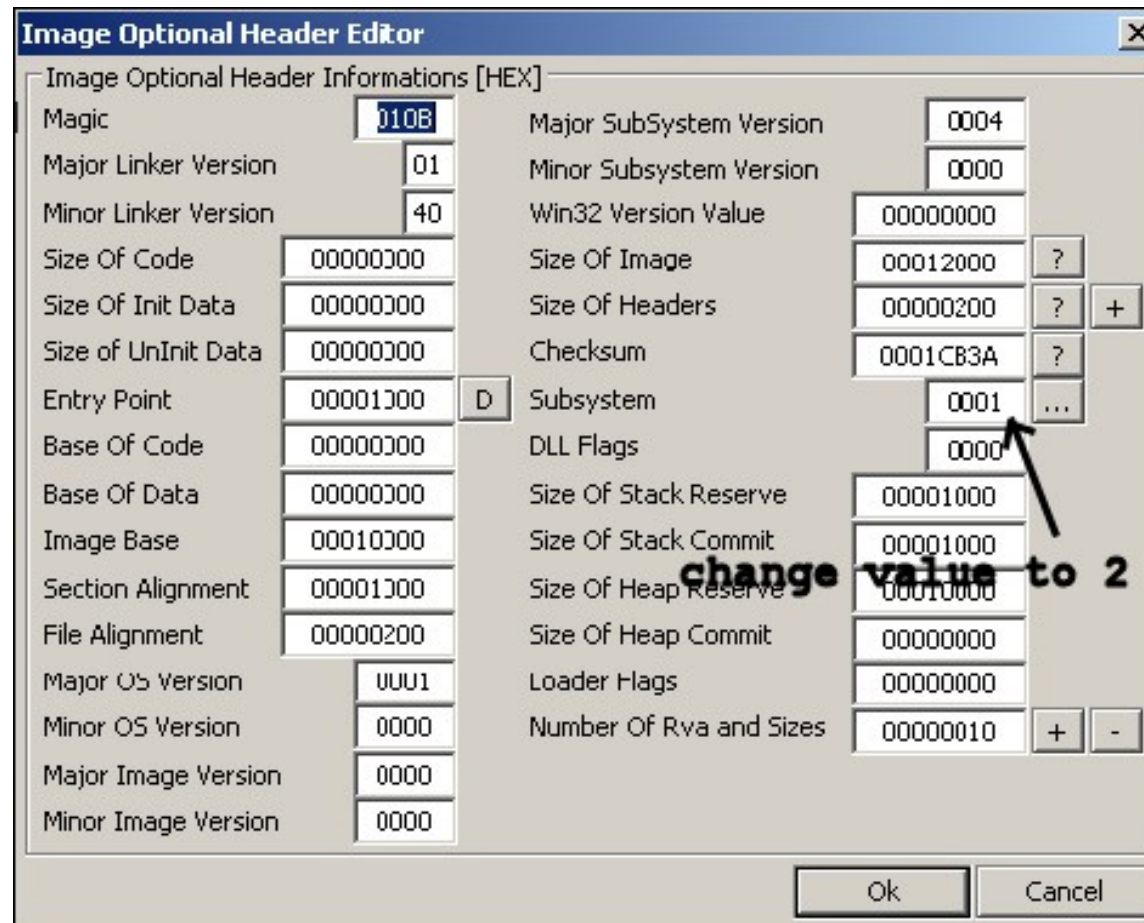
Stage 2: PE-Tools

- 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



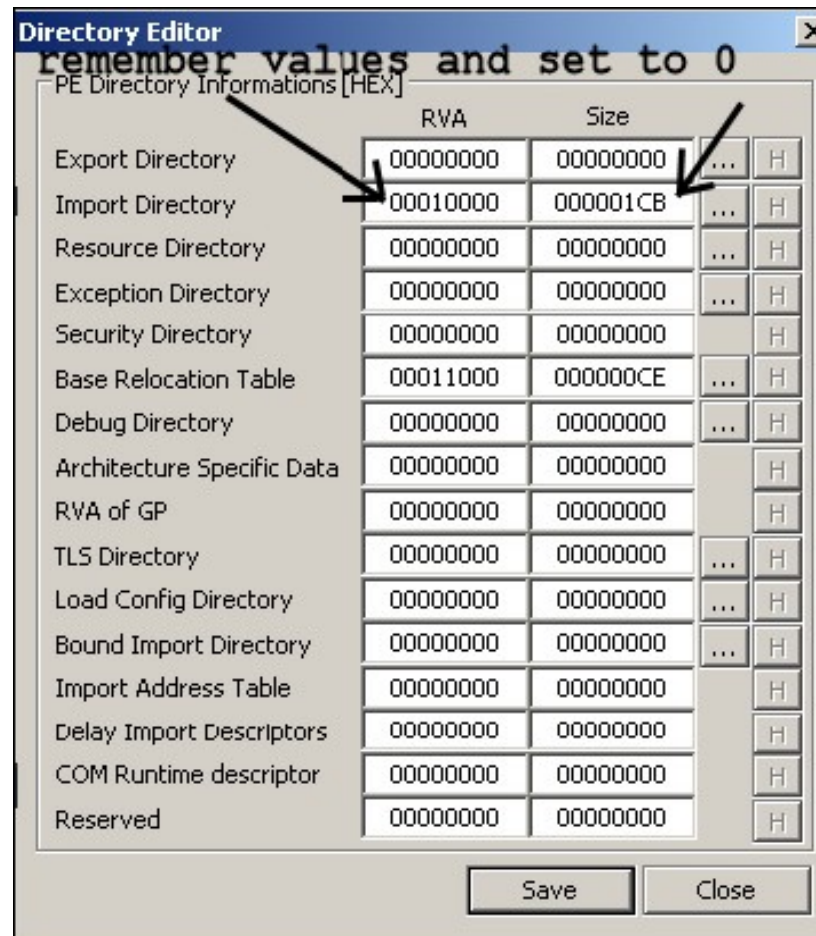
Stage 2: PE-Tools

- 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

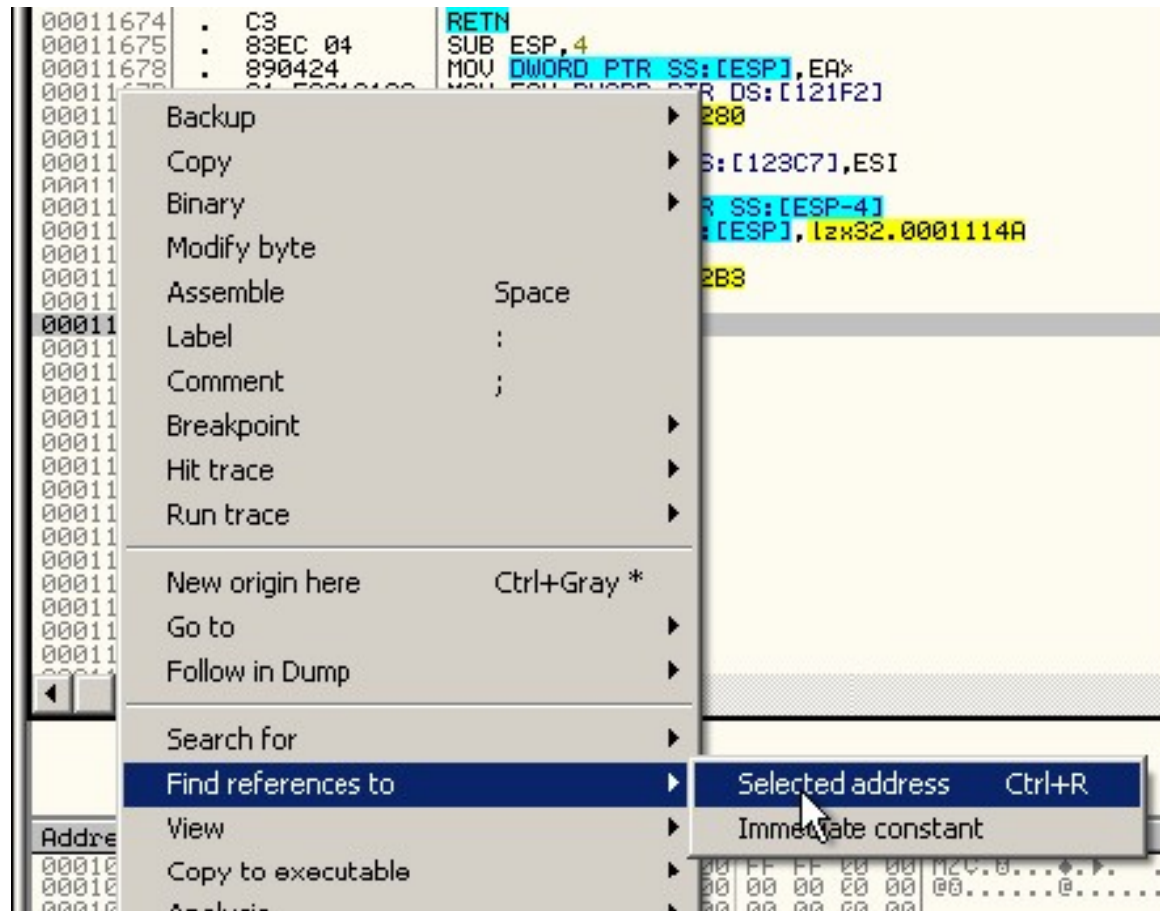
```
00011662 . 83EC 04          SUB ESP,4
00011665 . C70424 000000    MOV DWORD PTR SS:[ESP],0
0001166C . 291C24          SUB DWORD PTR SS:[ESP],EBX
0001166F . 68 28160100     PUSH [zx32.00011628]
00011674 . C3             RETN
00011675 . 83EC 04          SUB ESP,4
00011678 . 890424          MOV DWORD PTR SS:[ESP],EAX
0001167B . A1 F2210100     MOV EAX,DWORD PTR DS:[121F21]
00011680 . 68 80120100     PUSH [zx32.00011280]
00011685 . C3             RETN
00011686 > 8535 C7230100   TEST DWORD PTR DS:[123C7],ESI
0001168C . 81E2 261C0100  AND EDX,11C26
00011692 . 8D6424 FC       LEA ESP,DWORD PTR SS:[ESP-4]
00011696 . C70424 4A1100   MOV DWORD PTR SS:[ESP],[zx32.0001114A]
0001169D . C3             RETN
0001169E . 68 B3120100     PUSH [zx32.000112B3]
000116A3 . C3             RETN
000116A4 49              DB 49
000116A5 DA              DB DA
000116A6 FF              DB FF
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
```



unrecognized data

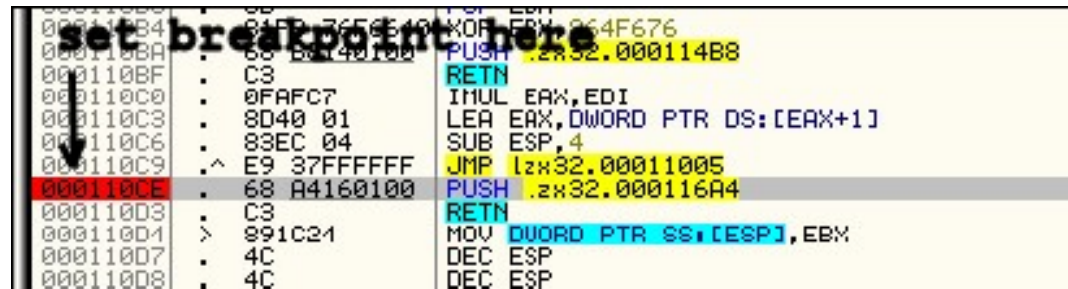
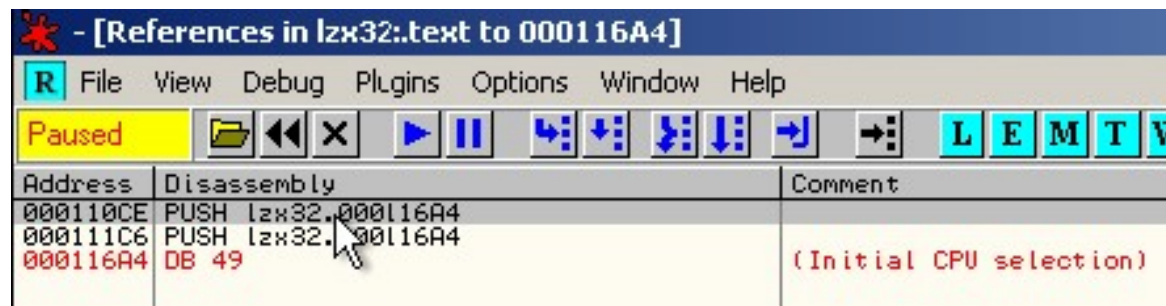
Stage 2: Ollydbg

- Find references to this address 0x116a4



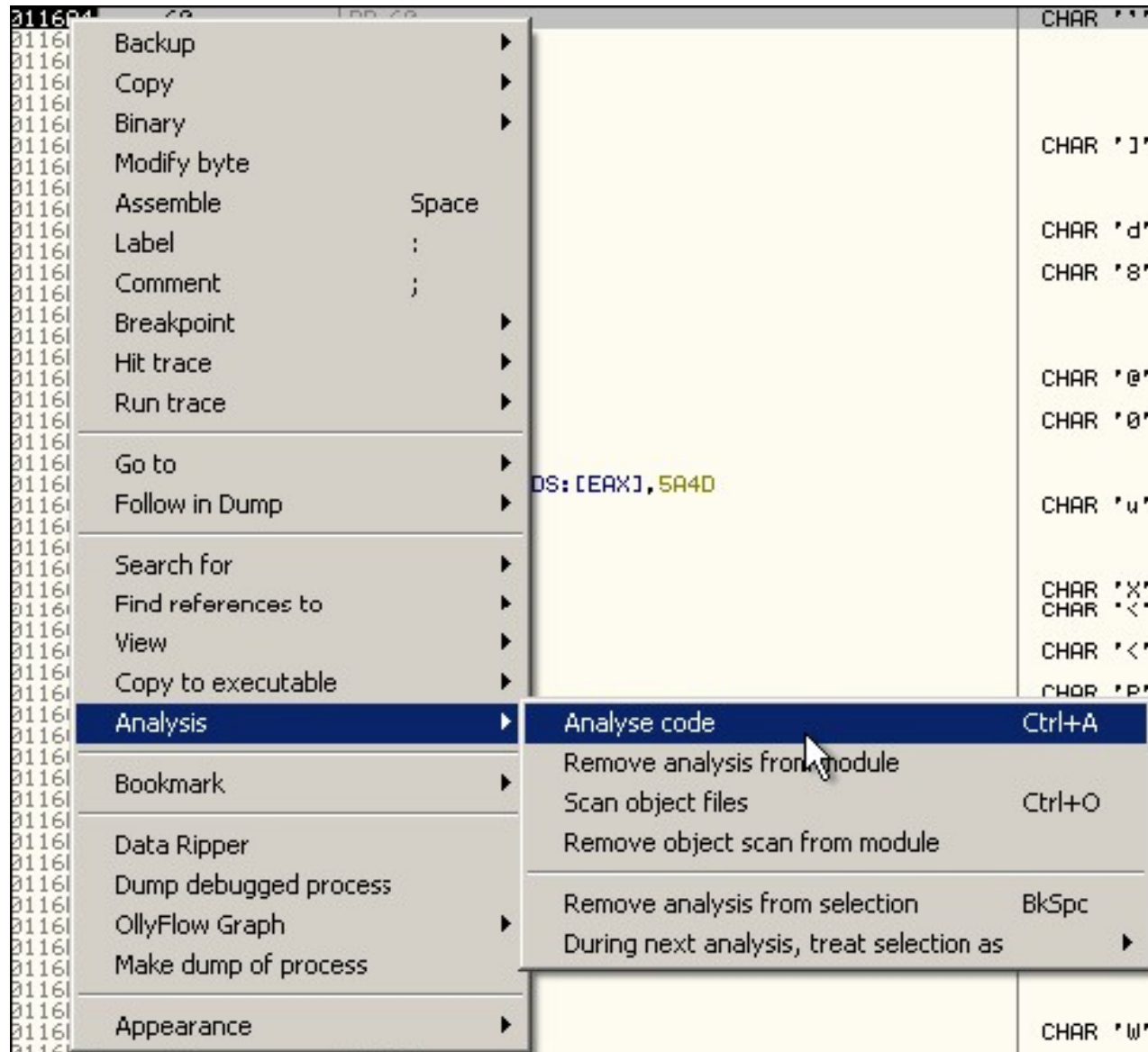
Stage 2: Ollydbg

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



Stage 2: Ollydbg

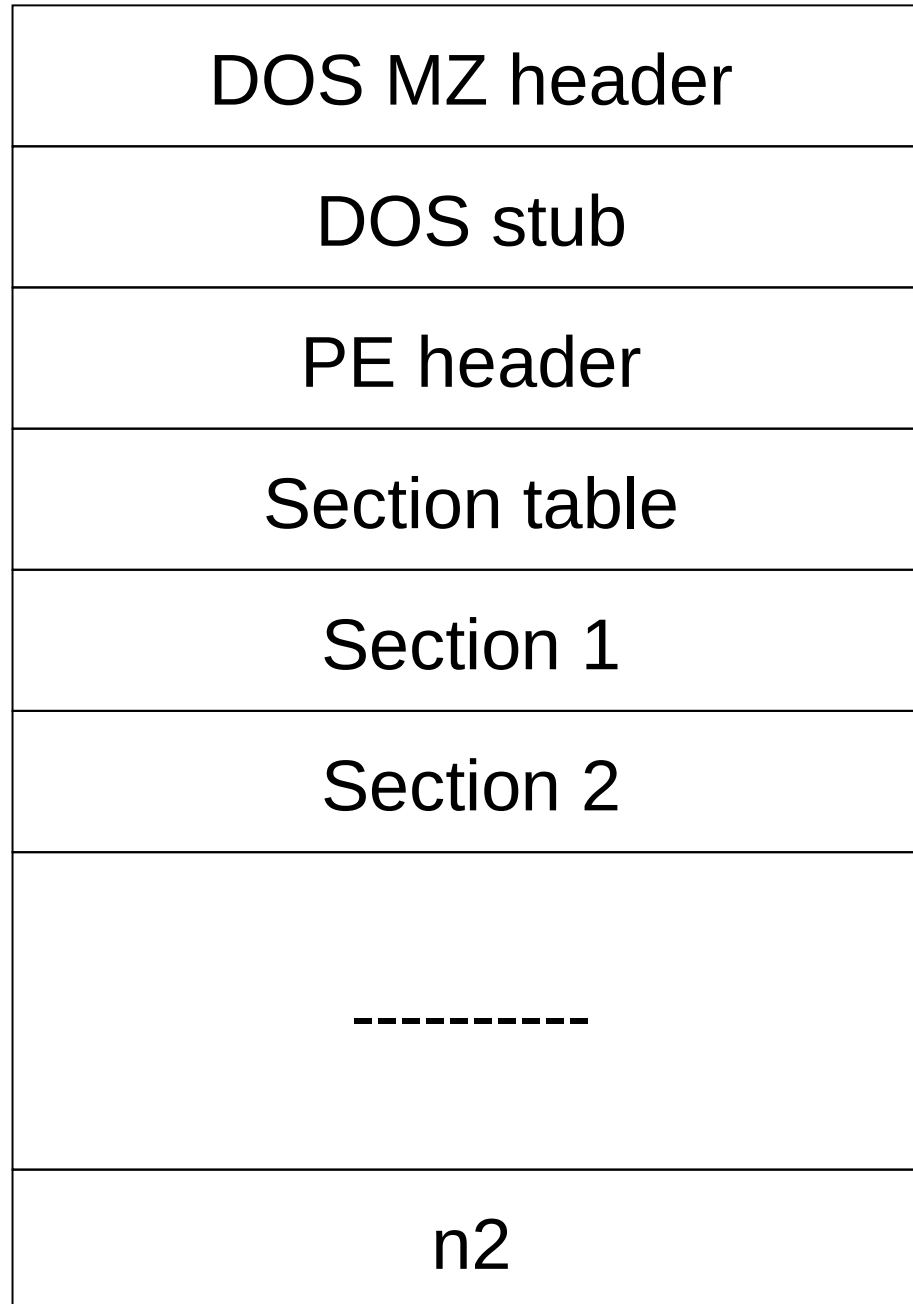
- Analyze code now...



System call analysis

- “Black box” analysis via system call tracing (Windows)
 - Windows system calls
 - <http://www.metasploit.com/users/opcode/syscalls.html>
 - Strace for Windows
 - http://www.bindview.com/Services/RAZOR/Utilities/Windows/strace_readme.cfm
 - Sebek
 - <http://www.honeynet.org/tools/sebek/>
 - Snare
 - <http://www.intersectalliance.com/projects/SnareWindows/index.html>
 - Holodeck (fault injector)
 - <http://tejasconsulting.com/open-testware/feature/holodeck-2.0.173.html>

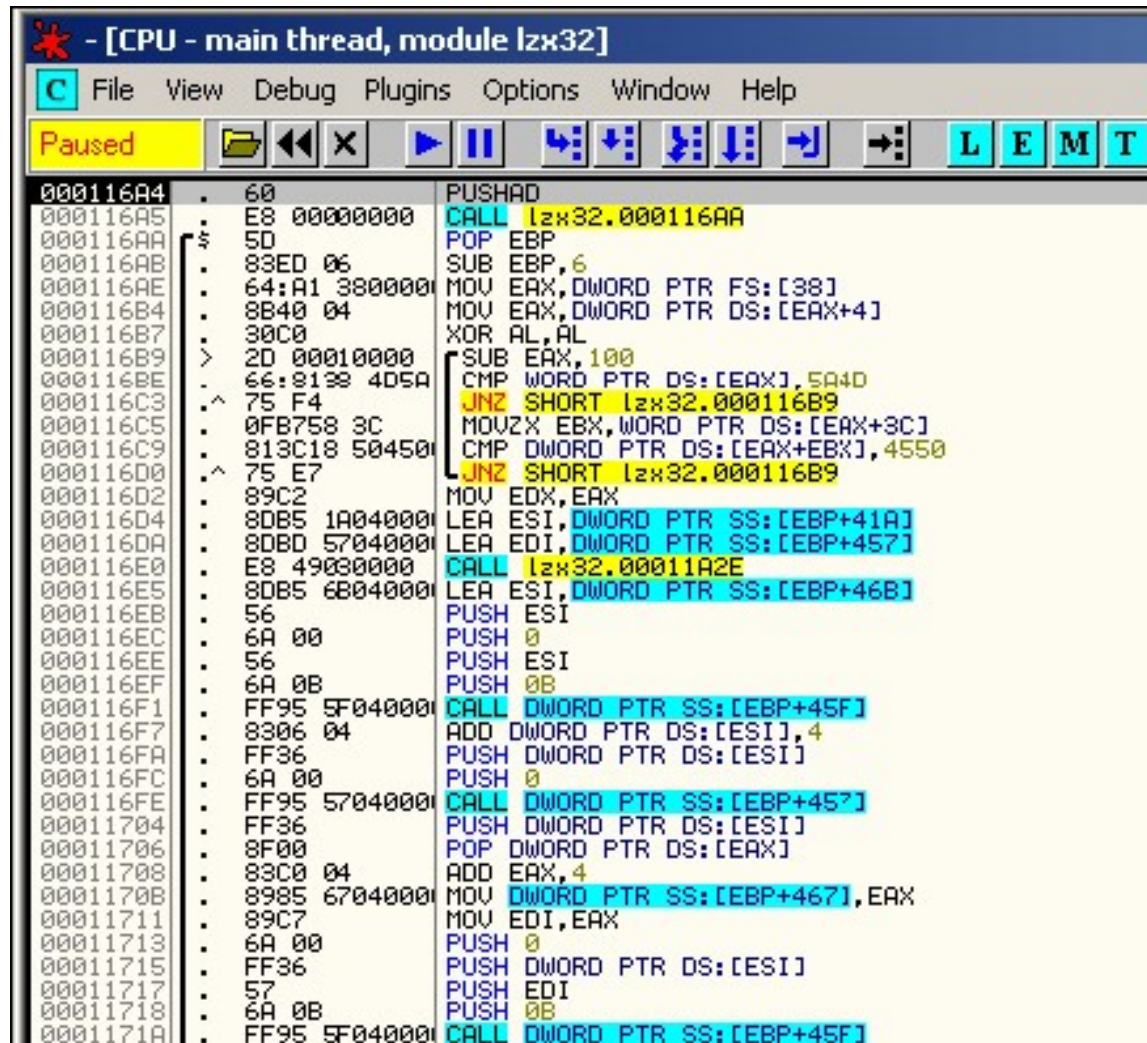
PE file format



PE file format

- DOS MZ header
 - So DOS can recognize program
- DOS stub
 - Built-in DOS executable to display “This program cannot run in MS-DOS mode”
- PE header
 - PE loader uses DOS MZ header to find starting offset of the PE header (skipping stub)
- Sections
 - Blocks of code/data organized on a logical basis

Stage 2: Ollydbg

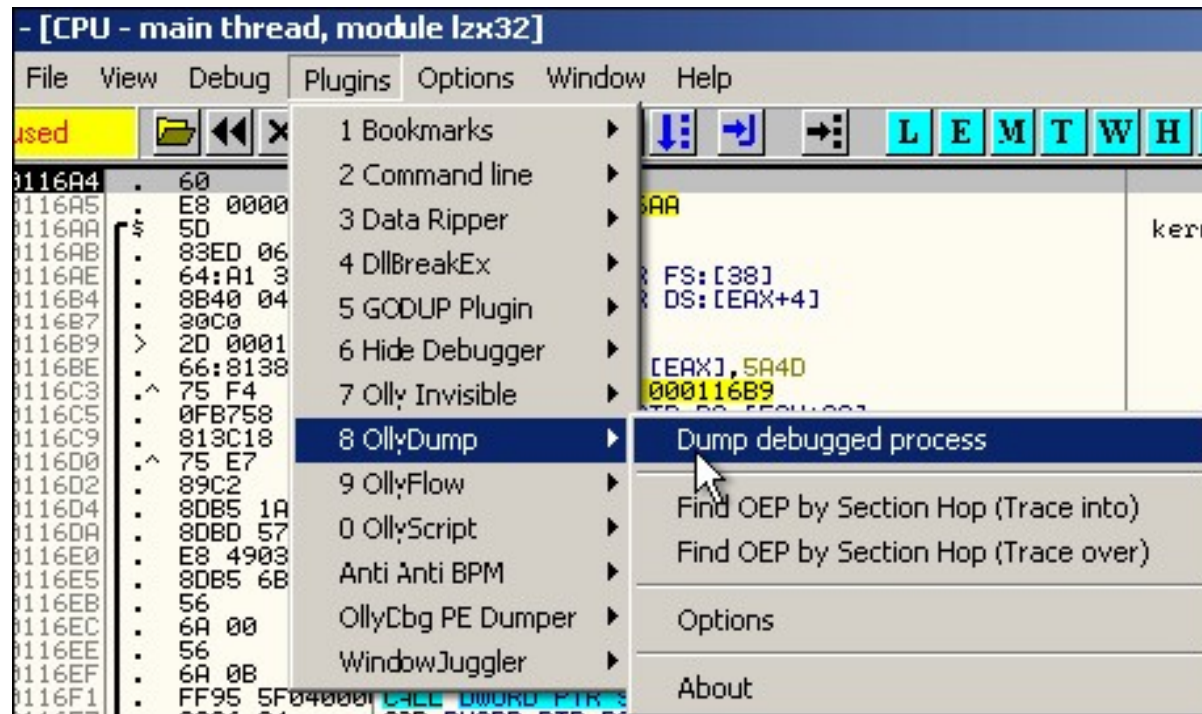


The screenshot shows the OllyDbg interface with the assembly window open. The title bar reads "[CPU - main thread, module lz32]". The menu bar includes File, View, Debug, Plugins, Options, Window, and Help. The status bar shows "Paused" and various navigation icons. The assembly window displays the following code:

```
000116A4 . 60 PUSHAD
000116A5 . E8 00000000 CALL lz32.000116AA
000116AA . 5D POP EBP
000116AB . 83ED 06 SUB EBP,6
000116AE . 64:A1 380000 MOV EAX,DWORD PTR FS:[38]
000116B4 . 8B40 04 MOV EAX,DWORD PTR DS:[EAX+4]
000116B7 . 30C0 XOR AL,AL
000116B9 > 2D 00010000 SUB EAX,100
000116BE . 66:8138 4D5A CMP WORD PTR DS:[EAX],5A4D
000116C3 . ^ 75 F4 JNZ SHORT lz32.000116B9
000116C5 . 0FB758 3C MOVZX EBX,WORD PTR DS:[EAX+3C]
000116C9 . 813C18 504501 CMP DWORD PTR DS:[EAX+EBX],4550
000116D0 . ^ 75 E7 JNZ SHORT lz32.000116B9
000116D2 . 89C2 MOV EDX,EAX
000116D4 . 8DB5 1A040001 LEA ESI,DWORD PTR SS:[EBP+41A]
000116DA . 8DB0 57040001 LEA EDI,DWORD PTR SS:[EBP+457]
000116E0 . E8 49000000 CALL lz32.00011A2E
000116E5 . 8DB5 6B040001 LEA ESI,DWORD PTR SS:[EBP+46B]
000116EB . 56 PUSH ESI
000116EC . 6A 00 PUSH 0
000116EE . 56 PUSH ESI
000116EF . 6A 0B PUSH 0B
000116F1 . FF95 5F040001 CALL DWORD PTR SS:[EBP+45F]
000116F7 . 8306 04 ADD DWORD PTR DS:[ESI],4
000116FA . FF36 PUSH DWORD PTR DS:[ESI]
000116FC . 6A 00 PUSH 0
000116FE . FF95 57040001 CALL DWORD PTR SS:[EBP+457]
00011704 . FF36 PUSH DWORD PTR DS:[ESI]
00011706 . 8F00 POP DWORD PTR DS:[EAX]
00011708 . 83C0 04 ADD EAX,4
0001170B . 8985 67040001 MOV DWORD PTR SS:[EBP+467],EAX
00011711 . 89C7 MOV EDI,EAX
00011713 . 6A 00 PUSH 0
00011715 . FF36 PUSH DWORD PTR DS:[ESI]
00011717 . 57 PUSH EDI
00011718 . 6A 0B PUSH 0B
0001171A . FF95 5F040001 CALL DWORD PTR SS:[EBP+45F]
```

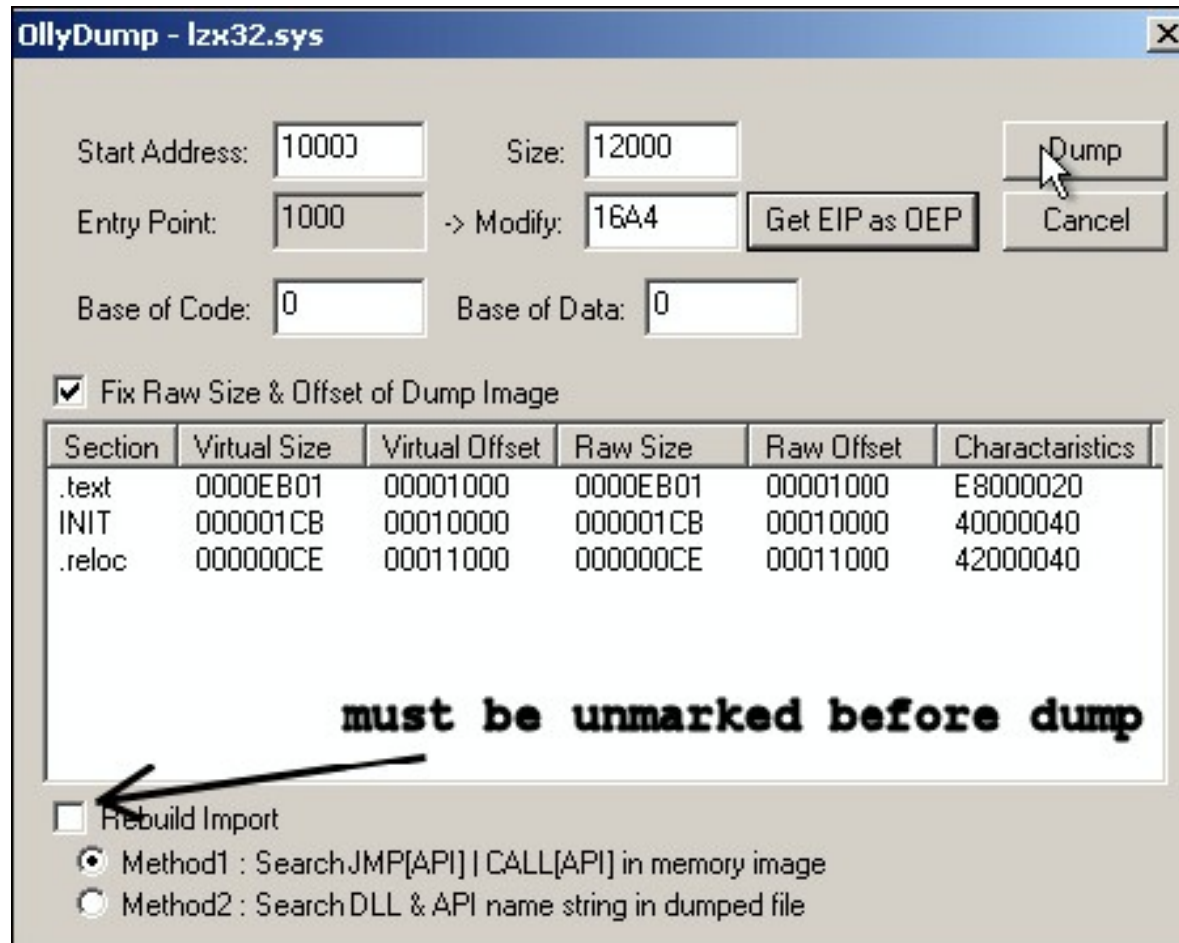
Stage 2: Ollydbg

- 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+468h]
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

```
00011AAC loc_11AAC: ; CODE XREF: sub_11A44+4F↑j
00011AAC      add     esi, 4
00011AAF      inc     ecx
00011AB0      jmp     short loc_11A77
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 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 ; -----
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
```

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

```
X:\paper\stage3>lzx32-laststage-unpacker.exe
+-----+
|           Rustock lzx32.sys last stage unpacker           |
| coding Frank Boldewin / www.reconstructor.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>
```

Stage 3: Reversing Rustock.B

<http://www.sarc.com/avcenter/venc/data/backdoor.r>

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://reconstructor.org>
- Alternative
 - Leech Windows Debugging Tools from MSFT
 - Read paper for recipe

Cleanup

- Run RkUnhooker

Finishing up Rustock.B

- Details of rootkit
 - Hooks MSR_SYSENTER
 - System call entry routine
 - Changes the following APIs
 - ZwOpenKey
 - ZwEnumerateKey
 - ZwQueryKey
 - ZwCreateKey
 - ZwSaveKey
 - ZwDeviceIoControlFile
 - ZwQuerySystemInformation
 - ZwInitializeRegistry
 - Hides based on process name issuing call (RookitRevealer, Rkdetector, etc)
 - Scans Windows kernel image for sting
 - FATAL_UNHANDLED_HARD_ERROR
 - Replaces it with rootkit functions
 - Creates bypasses for network communication
 - Modifies tcpip.sys, wanarp.sys, ndis.sys

Example: svrccp.exe

Example: `svcp.exe`

svrvc.exe

- <http://www.sans.org/resources/malwarefaq/srvcp>
- Use filemon, regmon
 - Flag new keys and files being created

The image shows two windows from Sysinternals: Registry Monitor and File Monitor, both displaying activity from the process svrvc.exe.

Registry Monitor - Sysinternals: www.sysinternals.com

#	Time	Process	Request	Path
58	47.19233696	svrvc.exe...	CloseKey	HKCU\Control Panel\International\Sorting Order
59	47.20654770	svrvc.exe...	OpenKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
60	47.24618541	svrvc.exe...	SetValue	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\Service Profiler
61	47.24655501	svrvc.exe...	CloseKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
62	47.24745596	svrvc.exe...	OpenKey	HKLM\System\CurrentControlSet\Services\WinSock2\Parameters

File Monitor - Sysinternals: www.sysinternals.com

#	Time	Process	Request	Path	Result
46	6:19:46 AM	System: 42	IRP_MJ_SET_INFORMATION	C:\WINNT\system32\config\software.LOG	SUCCESS
47	6:19:46 AM	svrvc.exe:...	FSCTL_IS_VOLUME_MOUN...	C:\Download	SUCCESS
48	6:19:46 AM	svrvc.exe:...	IRP_MJ_CREATE	C:\WINNT\System32\gus.ini	FILE NOT FOL
49	6:19:46 AM	svrvc.exe:...	FSCTL_IS_VOLUME_MOUN...	C:\Download	SUCCESS
50	6:19:46 AM	svrvc.exe:...	IRP_MJ_CREATE	C:\WINNT\System32\gus.ini	FILE NOT FOL

svrvc.exe

- tcpdump to identify communication to IRC C&C

Domain Name Resolution Request

03/16-06:19:46.414790 172.16.198.131:1046 -> 172.16.198.1:53

UDP TTL:128 TOS:0x0 ID:4354 IpLen:20 DgmLen:57Len: 37

```
00 01 01 00 00 01 00 00 00 00 00 00 03 69 72 63 ..... irc
03 6D 63 73 03 6E 65 74 00 00 01 00 01          .mcs.net .....
```

svrvc.exe

- Encrypted code
 - svrvc.exe contains no strings
 - Encrypted with XOR-based algorithm
 - Some instructions not encrypted
 - Repeated pushes to the stack of a string that looks random followed by calls to the routine “sub_4012C6”

Probable Calls to String Decryption Routine

```
.text:0040141D push offset aNhlPwf ; " nhl*pwf "  
.text:00401422 call sub_4012C6  
.text:00401427 push offset aAhkl ; "|ahkli"  
.text:0040142C call sub_4012C6  
.text:00401431 push offset aWtwgr ; "wtwgr"  
.text:00401436 call sub_4012C6  
.text:0040143B push offset aCdkk ; "|cdkk"  
.text:00401440 call sub_4012C6  
.text:00401445 push offset aMfqece ; "mfqEce"  
.text:0040144A call sub_4012C6
```

srvcp.exe

- Encrypted code
 - Emits gus.ini file that also contains encrypted code that is decrypted at run-time
 - Look through code to find calls to fopen()
 - Verify via SoftICE

Opening gus.ini File

```
.text:00403662 push offset aR ; " r "  
.text:00403667 push [ebp+arg_0]  
.text:0040366A call fopen
```

```
403662  PUSH      004083DB  
403667  PUSH      DWORD PTR [EBP+08]  
40366A  CALL     CRTDLL!fopen  
40366F  ADD      ESP,08  
403672  MOV      EBX,EAX  
403674  OR       EBX,EBX  
403676  JNZ     0040367F  
403678  XOR     EAX,EAX  
-----KTEB(804A5180)-TID(0082)-----srvcp!.text+265B-----  
406400  43 3A 5C 57 49 4E 4E 54-5C 53 79 73 74 65 6D 33  C:\WINNT\System3  
406410  32 5C 67 75 73 2E 69 6E-69 00 00 00 00 00 00 00  2\gus.ini.....
```

svrvc.exe

- Encrypted code
 - Look for reads from file
 - After read, call code at loc_4036B2
 - Probably leads to the decryption routine, let's go examine

Reading Lines from gus.ini File

```
.text:00403738 push eax
.text:00403739 push offset asc_4083D1 ; " %[\n]\n "
.text:0040373E push ebx
.text:0040373F call fscanf
.text:00403744 add esp, 0Ch
.text:00403747 cmp eax, 0FFFFFFFh
.text:0040374A jnz loc_4036B2
```

srvcp.exe

- Encrypted code
 - .ini files usually have PARAMNAME=value format parsed with associated sscanf string
 - call sub_405366 must be decryption routine

Parsing gus.ini Lines

```
.text:004036B2 lea eax, [ebp+var_414]
.text:004036B8 push eax
.text:004036B9 push esi
.text:004036BA call sub_405366
.text:004036BF mov edi, eax
.text:004036C1 lea eax, [ebp+var_9F0]
.text:004036C7 push eax
.text:004036C8 lea eax, [ebp+var_14]
.text:004036CB push eax
.text:004036CC push offset asc_4083C5 ; " %[^]=%[^ "
.text:004036D1 push edi
.text:004036D2 call sscanf
```

svrvc.exe

- Encrypted code
 - Verify via debugger (SoftICE)
 - Examine memory pointed to by EAX before and after call (using “d EAX” command)

```
4036B2 LEA      EAX,[EBP+FFFFFFBEC]
4036B8 PUSH     EAX
4036B9 PUSH     ESI
4036BA CALL     00405366
4036BF MOV     EDI,EAX
-----KTEB(804B6600)-TID(0077)-svrvc!.text+26A2-----
D8EAEC 4A 65 78 4F 32 31 35 57-75 4B 36 30 48 37 48 67  Jex0215WuK60H7Hg
D8EAFc 49 2E 6A 31 31 76 68 31-00 00 00 00 00 00 00 00  I.j11vh1.....
```

```
4036B2 LEA      EAX,[EBP+FFFFFFBEC]
4036B8 PUSH     EAX
4036B9 PUSH     ESI
4036BA CALL     00405366
4036BF MOV     EDI,EAX
-----KTEB(804AE020)-TID(0027)-svrvc!.text+26A2-----
136518 4E 49 43 4B 3D 6D 69 6B-65 79 00 00 00 00 00 00  NICK=mikey.....
136528 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
```

srvcp.exe

- Encrypted code
 - Decrypted gus.ini file
 - Parameters to srvcp.exe are overwritten here

Decrypted gus.ini File

```
NICK=mikey
MODE=AGGRESSIVE
SETCOMMAND=setpr
COMMAND=fuckedup
CHANNEL=mikag soup
SOUPCHANNEL=alphasoup ah
SERVER0=irc.mcs.net:6666
SERVER1=efnet.cs.hut.fi:6666
SERVER2=efnet.demon.co.uk:6666
SERVER3=irc.concentric.net:6666
SERVER4=irc.etsmtl.ca:6666
SERVER5=irc.fasti.net:6666
... cut for brevity ...
```

svrvc.exe

- More information at...
 - HTML version
 - <http://www.sans.org/resources/malwarefaq/srvcp.php>
 - PDF version
 - <http://www.zeltser.com/reverse-malware-paper/reverse-malware.pdf>
 - Disassembled code
 - <http://www.zeltser.com/sans/gcih-practical/srvcp-asm.pdf>