Lecture 12

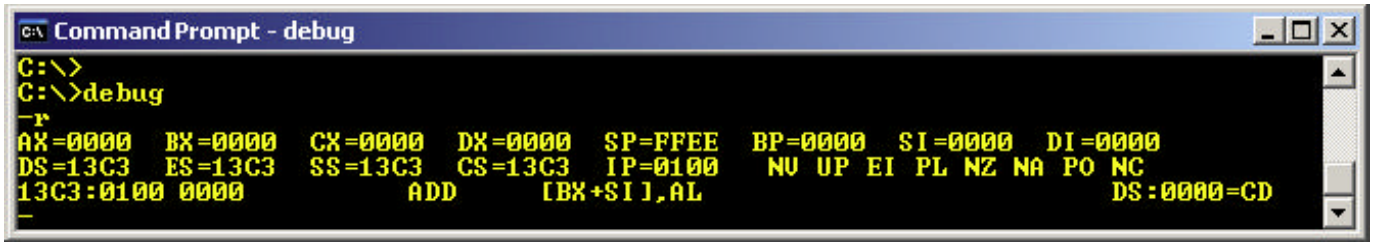
Reversing

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

Primer on x86

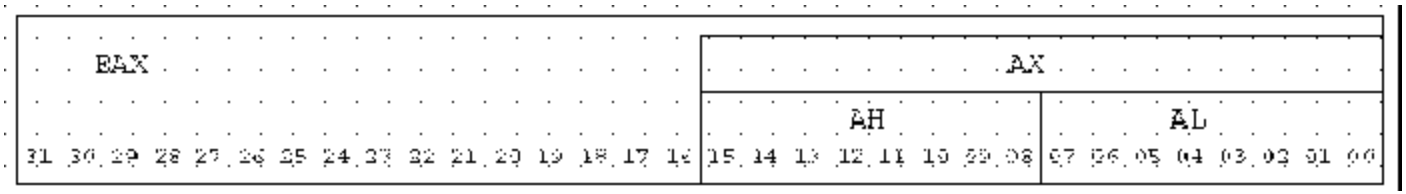
- Registers



Register Name	Size *(in bits)*	Purpose	
AX (EAX)	16 (32)	Main register used in arithmetic calculations. Also known as accumulator, as it holds results of arithmetic operations and **function return values**.	
BX (EBX)	16 (32)	The Base Register. Used to store the base address of the program.	
CX (ECX)	16 (32)	The Counter register is often used to hold a value representing the number of times a process is to be repeated. Used for loop and string operations.	
DX (EDX)	16 (32)	A general purpose register. Also used for I/O operations. Helps extend EAX to 64-bits.	
SI (ESI)	16 (32)	Source Index register. Used as an offset address in string and array operations. It holds the address from where to read data.	
DI (EDI)	16 (32)	Destination Index register. Used as an offset address in string and array operations. It holds the implied write address of all string operations.	
BP (EBP)	16 (32)	Base Pointer. It points to the bottom of the current stack frame. It is used to reference local variables.	
SP (ESP)	16 (32)	Stack Pointer. It points to the top of the current stack frame. It is used to reference local variables.	
IP (EIP)	16 (32)	The instruction pointer holds the address of the next instruction to be executed.	
CS	16	Code segment register. Base location of code section (.text section). Used for fetching instructions.	These registers are used to break up a program into parts. As it executes, the segment registers are assigned the base values of each segment. From here, offset values are used to access each command in the program.*
DS	16	Data segment register. Default location for variables (.data section). Used for data accesses.	
ES	16	Extra segment register. Used during string operations.	
SS	16	Stack segment register. Base location of the stack segment. Used when implicitly using SP or ESP or when explicitly using BP, EBP.	
EFLAGS	32	This register's bits represent several single-bit Boolean values, such as the sign, overflow, carry, and zero flags. It is modified after every mathematical operation. See below for more information.	

Primer on x86

- Registers



- Instructions

Copies the value 1 into the EDX register:

Assembly	Hexadecimal
`MOV EDX, 1`	`BA 01 00 00 00`

Primer on x86

- Instruction rules
  - Source operand can be memory, register or constant
  - Destination can be memory or non-segment register
  - Only one of source and destination can be memory
  - Source and destination must be same size
- Flags set on each instruction
  - EFLAGS
  - Conditional branches handled via EFLAGS

C, x86 example

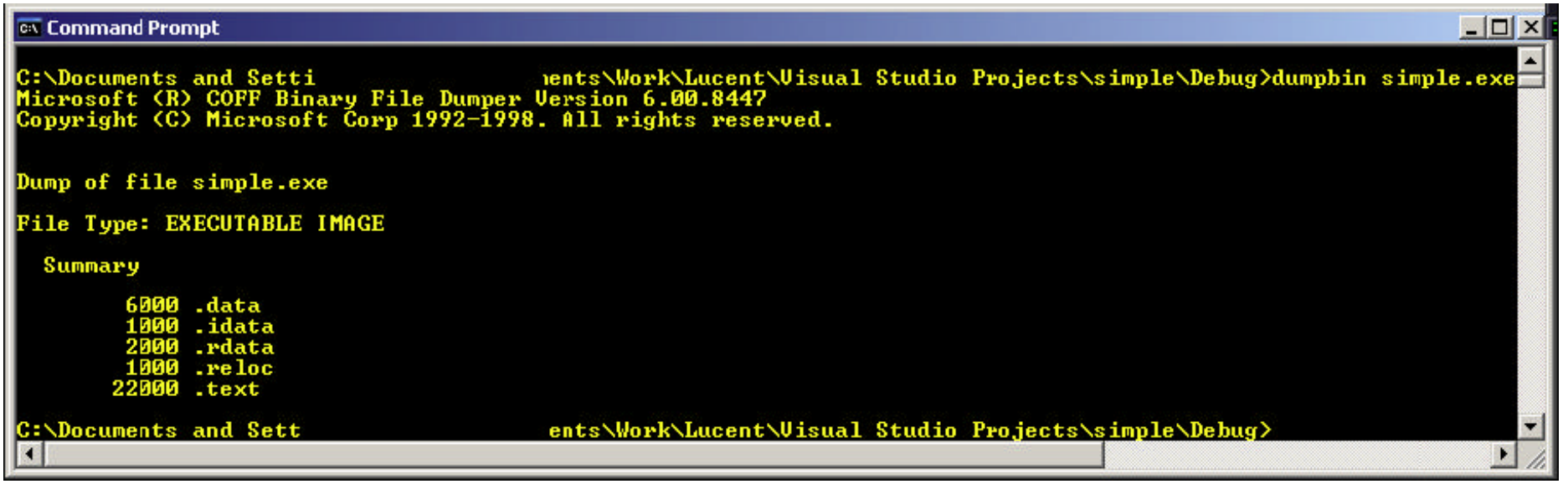
```
int a = 1, b = 3, c;
if (a > b)
     c = a;
else
     c = b;
```

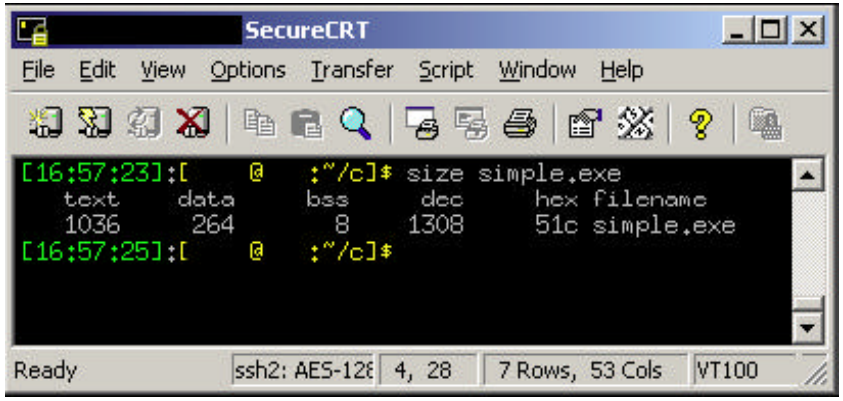
```
00000018: C7 45 FC 01 00 00 00      mov dword ptr [ebp-4],1           ; store a = 1
0000001F: C7 45 F8 03 00 00 00      mov dword ptr [ebp-8],3           ; store b = 3
00000026: 8B 45 FC                  mov eax,dword ptr [ebp-4]         ; move a into EAX register
00000029: 3B 45 F8                  cmp eax,dword ptr [ebp-8]         ; compare a with b (subtraction)
0000002C: 7E 08                     jle 00000036                      ; if (a<=b) jump to line 00000036
0000002E: 8B 4D FC                  mov ecx,dword ptr [ebp-4]         ; else move 1 into ECX register &&
00000031: 89 4D F4                  mov dword ptr [ebp-0Ch],ecx       ; move ECX into c (12 bytes down)
     &&
00000034: EB 06                     jmp 0000003C                      ; unconditional jump to 0000003C
00000036: 8B 55 F8                  mov edx,dword ptr [ebp-8]         ; move 3 into EDX register &&
00000039: 89 55 F4                  mov dword ptr [ebp-0Ch],edx       ; move EDX into c (12 bytes down)
```

Run-time data structures

Segment Name	Segment Description
.text	This segment contains the executable instructions and is shared among every process running the same binary. This segment usually has READ and EXECUTE permissions only. This section is the one most affected by optimization.
.data	Contains the initialized global and static variables and their values. It is usually the largest part of the executable. It usually has READ/WRITE permissions.
.rdata	Sometimes known as .rodata (read-only data) segment. This contains constants and string literals.
.bss	BSS stands for "Block Started by Symbol." It holds un-initialized global and static variables. Since the BSS only holds variables that don't have any values yet, it doesn't actually need to store the image of these variables. The size that BSS will require at runtime is recorded in the object file, but the BSS (unlike the data segment) doesn't take up any actual space in the object file.
.reloc	Stores the information required for relocating the image while loading.
Heap	The heap area is for dynamically allocated memory (malloc(), realloc(), calloc()) and is accessed through a pointer. Everything on a heap is anonymous, thus you can only access parts of it through a pointer. A malloc() request may be rounded up in size to some convenient power of two. Freed memory goes back to the heap, but there is no easy way to give up that memory back to the OS. The heap usually grows up toward the stack. The end of the heap is marked by a pointer known as the "break." You cannot reference past the break. You can, however, move the break pointer (via brk and sbrk system calls) to a new position to increase the amount of heap memory available. This is usually done automatically for you by the system if you use malloc often enough.[9]
Stack	The stack holds local (automatic) variables, temporary information, function parameters, and the like. It acts like a LIFO (Last In First Out) object as it grows downward toward the heap. When a function is called, a stack frame (or a procedure activation record) is created and PUSHed onto the top of the stack. This stack frame contains information such as the address from which the function was called (and where to jump back to when the function is finished (return address)), parameters, local variables, and any other information needed by the invoked function. The order of the information varies by system and compiler, but on Solaris it is described in /usr/include/sys/frame.h. When a function returns, the stack frame is POPped from the stack. The current instruction that is running is pointed to by the IP (Instruction Pointer). The address of the next instruction is held in the PC (Program Counter).

Getting contents of executables



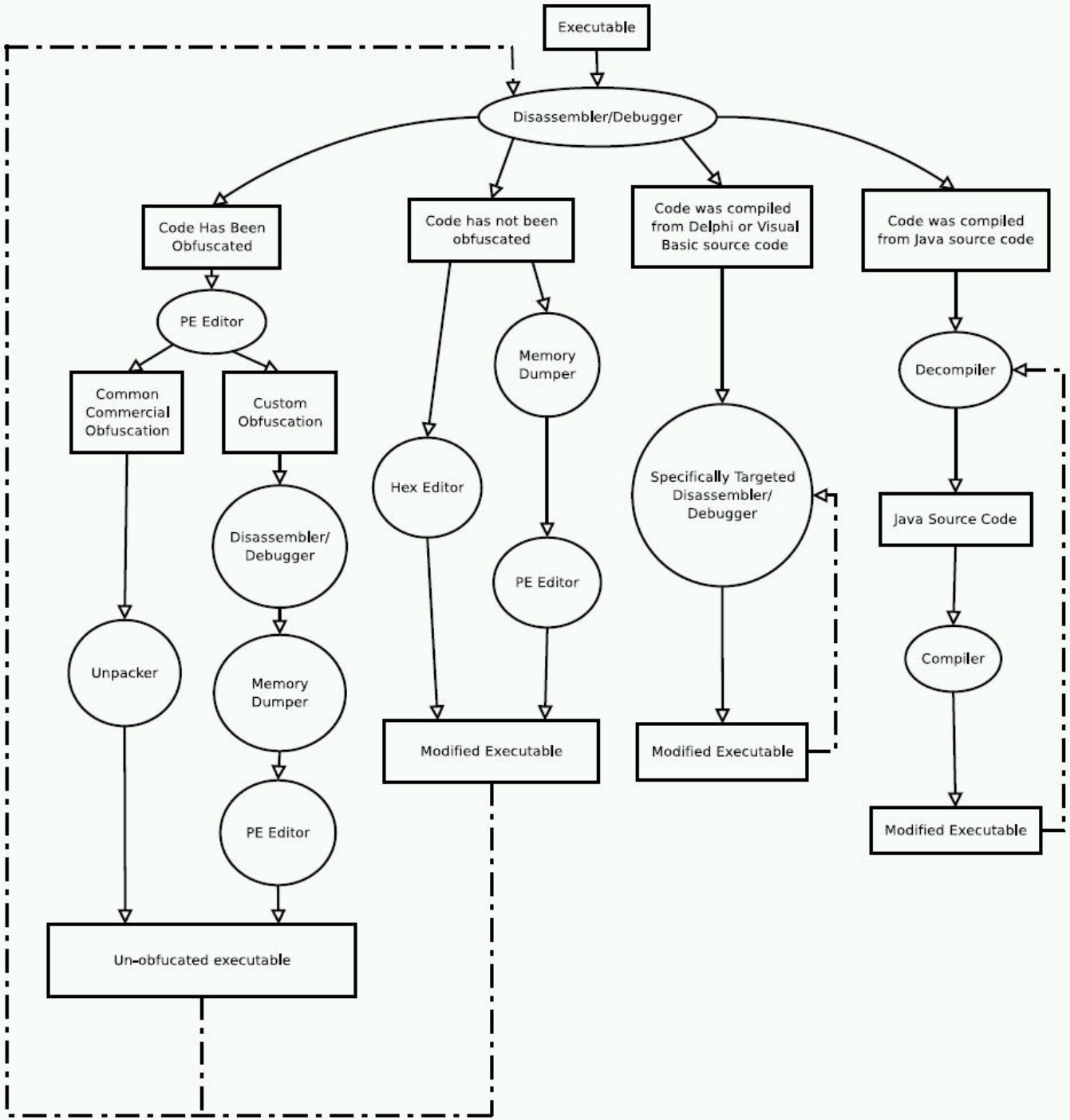


Function calls and stacks

- See CS 200 lecture on procedures
  - Return IP and function parameters pushed onto stack
  - call instruction invokes function
  - Local variables stored on stack
  - Old stack frame pointer stored on stack so that it may be later restored when function returns

Tools round-up

- A Survey of Reverse Engineering Tools for the 32-bit Microsoft Windows Environment" by R. Canzanese Jr., Matthew Oyer, S. Mancoridis, M. Kam.
  - From http://www.cs.drexel.edu/~spiros/teaching/CS675/index.html
- Tools
  - Hex editors
  - Disassemblers/Debuggers
  - Decompilers
  - Code obfuscators
  - Unpackers
  - PE editors
  - Memory dumpers


Executable
Disassembler/Debugger
Code Has Been Obfuscated
Code has not been obfuscated
Code was compiled from Delphi or Visual Basic source code
Code was compiled from Java source code
PE Editor
Common Commercial Obfuscation
Custom Obfuscation
Memory Dumper
Hex Editor
Specifically Targeted Disassembler/ Debugger
Decompiler
Java Source Code
Compiler
Disassembler/ Debugger
Unpacker
Memory Dumper
PE Editor
Modified Executable
Modified Executable
Modified Executable
PE Editor
Un-obfucated executable

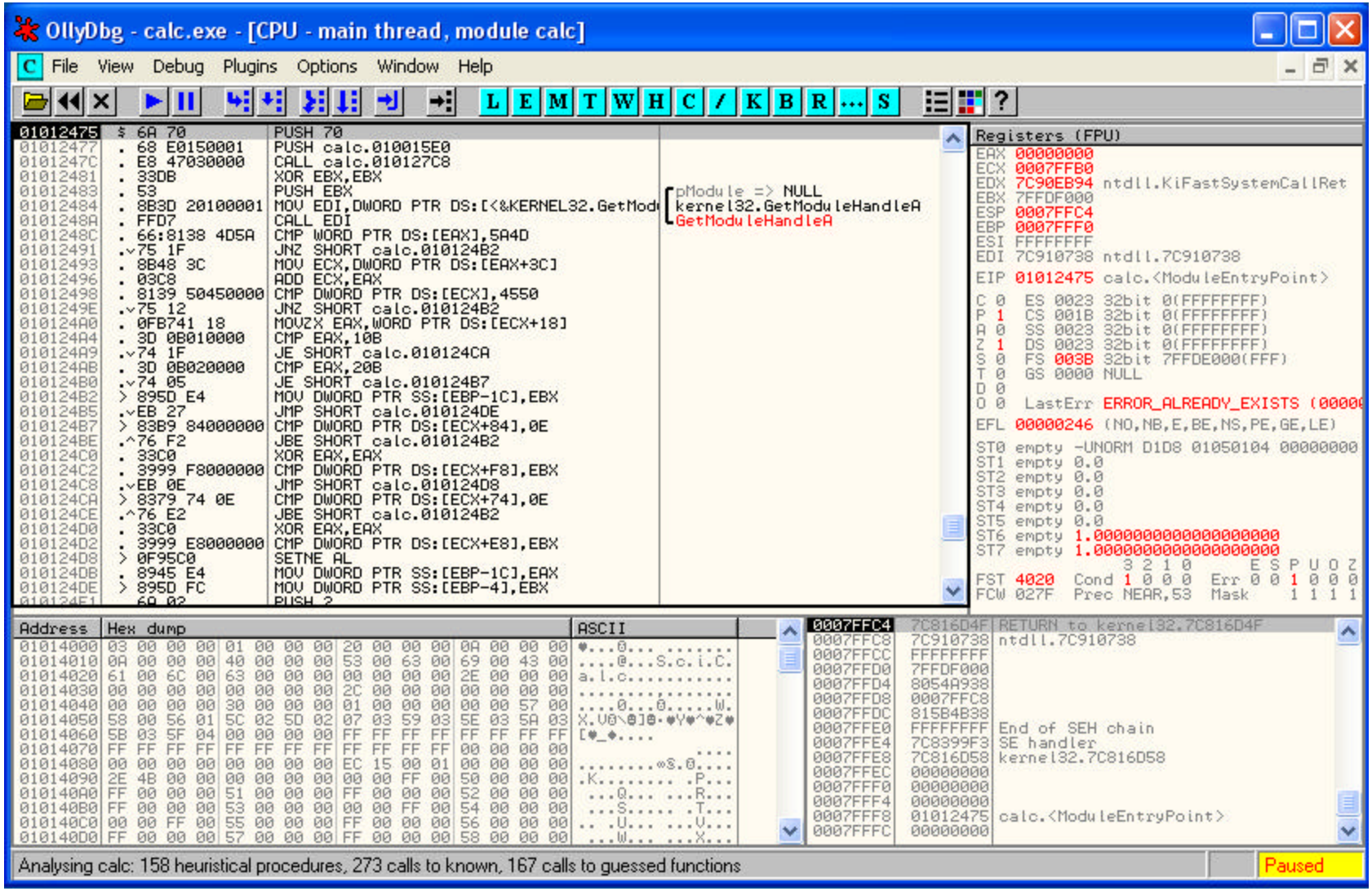
Hex editors

- Basic
  - HHD Hex Editor
  - UltraEdit
- Advanced
  - WinHex (RAM disassembly)
  - Tsearch (RAM disassembly)
  - Hex Workshop
  - Hackman Hex Editor
  - Hiew (Hackers View)

Disassemblers/debuggers

- Debuggers
  - OllyDbg
  - IDA Pro
  - SoftICE
  - TRW
  - Debuggy
  - Hackman debugger
- Disassemblers
  - W32DASM
  - BORG
  - Hackman disassembler
  - Phoenix/DSM Studio
- Language-specific disassemblers and debuggers
  - DeDe
  - VBReFormer

Disassemblers/debuggers



Decompilers

- C
  - REC
  - DCC
  - DisC (Borland TurboC)
- Java
  - JAD
  - JODE
  - DAVA

Code obfuscators

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

Unpackers

- Universal
  - GUW (Generic Unpacker for Windows)
  - ProcDump
  - PeiD
- Packer-specific unpackers
  - unNFO
  - upx
  - Some packers are highly configurable (ASProtect) and do not have unpackers
  - Must do memory dump after unpacking at run-time

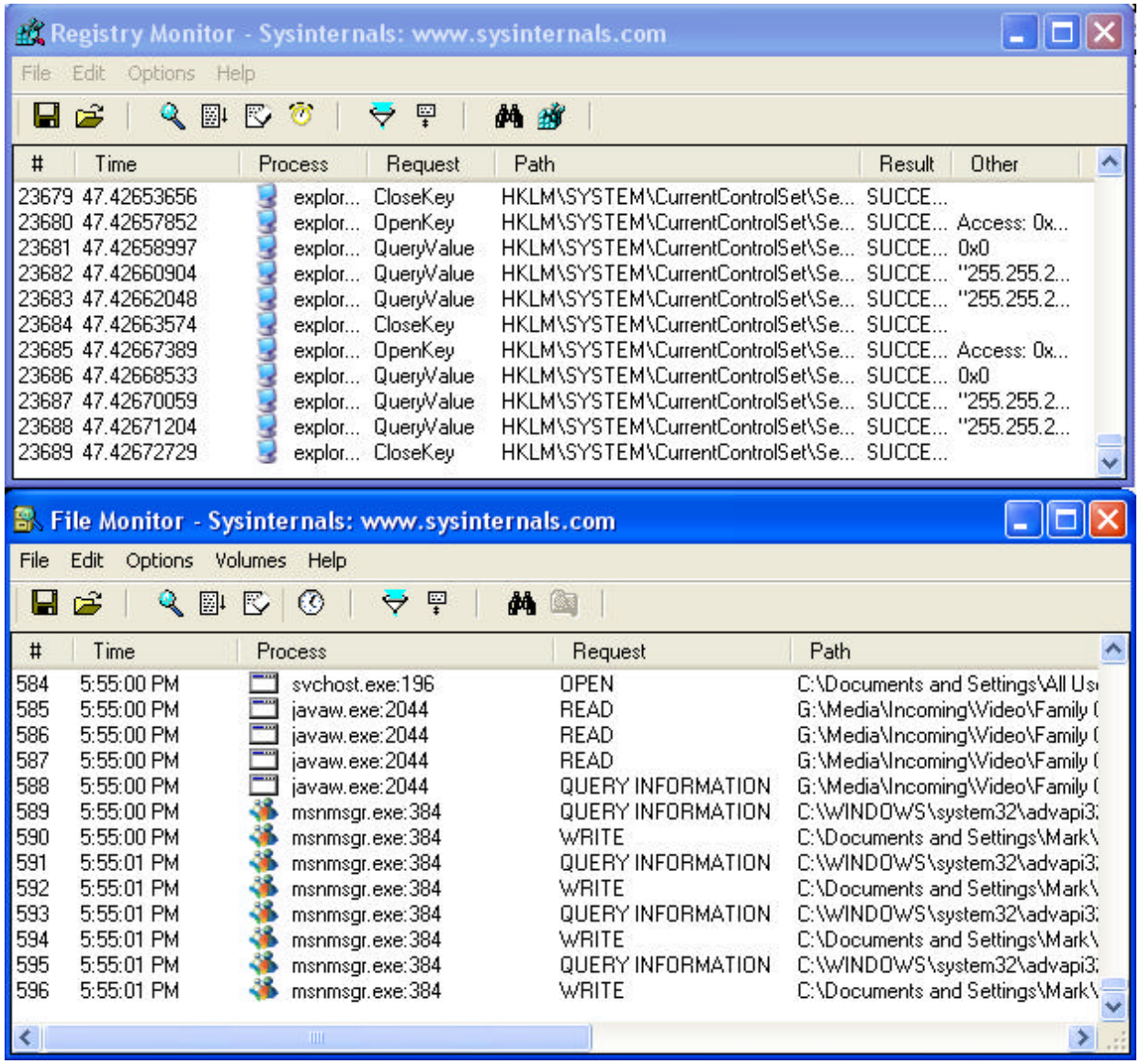
PE editors

- PE headers and IATs must be repaired after dumping since relative addresses translated at load time
  - PEditor
  - Procdump
  - LordPE
  - PEDump
- Restoring IAT
  - OllyDump
  - REVirgin
  - ImpRec
- Listing imports
  - APISpy

Other useful tools

- BinText/Strings
  - extract strings from executable
- 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 .

Other useful tools

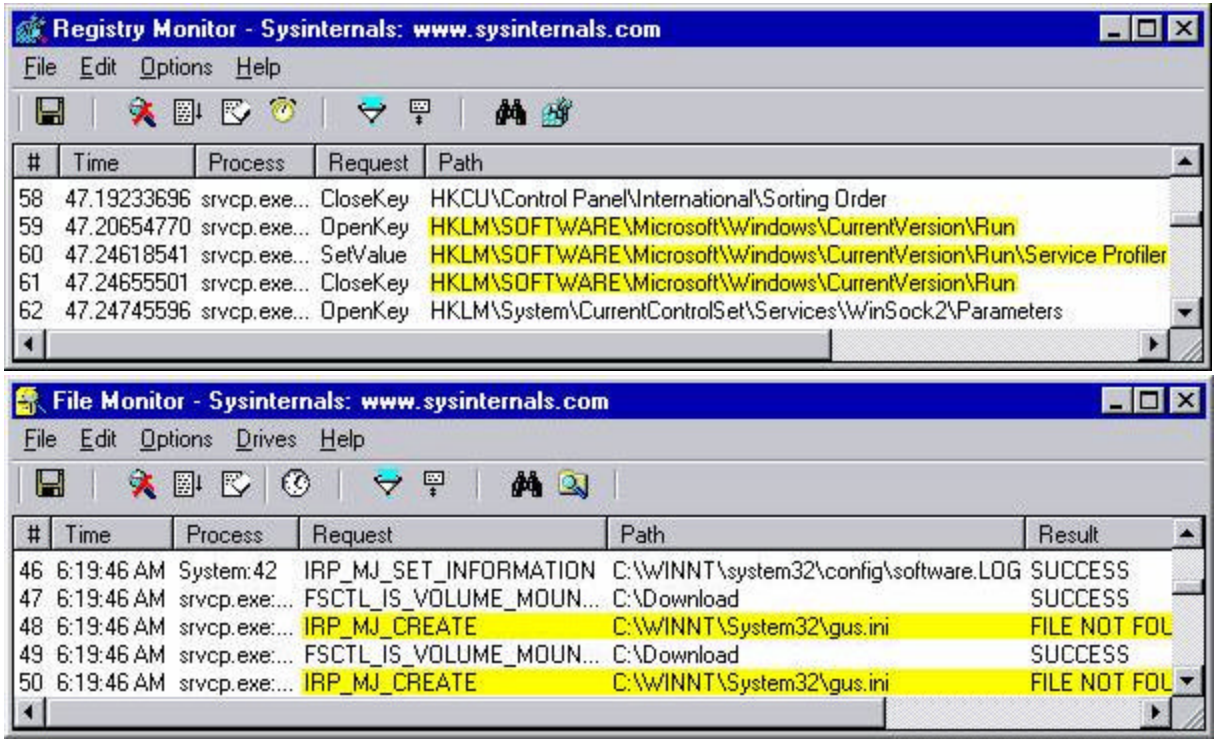


Other useful tools

- 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

Example #1 srvcp.exe

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



Example #1 srvcp.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 .....
```

Example #1 srvcp.exe

- Encrypted code
  - srvcp.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

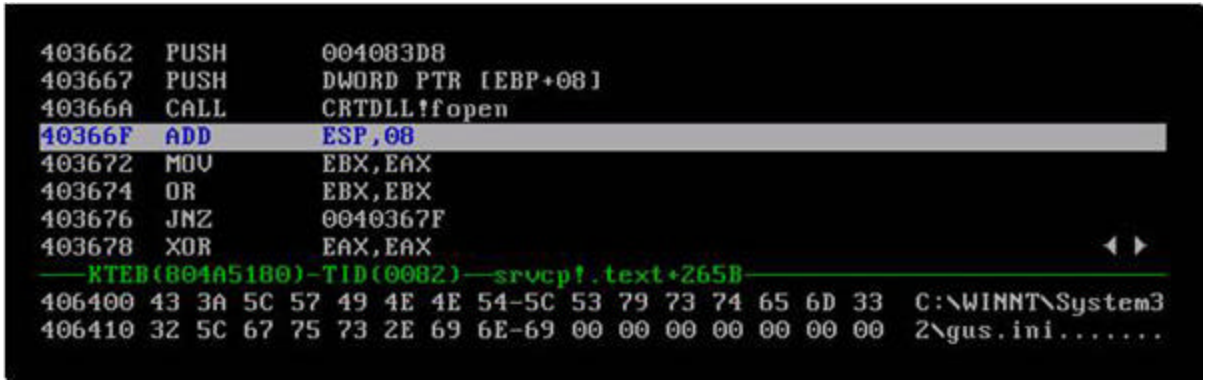
```
.text:0040141D push offset aNhlPwf ; " nhl*pwf "
.text:00401422 call sub_4012C6
.text:00401427 push offset aAhkl ; "|ahkli"
.text:0040142C call sub_4012C6
.text:00401431 push offset aWtwgr ; "wtwgr"
.text:00401436 call sub_4012C6
.text:0040143B push offset aCdkk ; "|cdkk"
.text:00401440 call sub_4012C6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

Example #1 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
```



Example #1 srvcp.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, 0FFFFFFFFh
.text:0040374A jnz loc_4036B2
```

Example #1 srvcp.exe

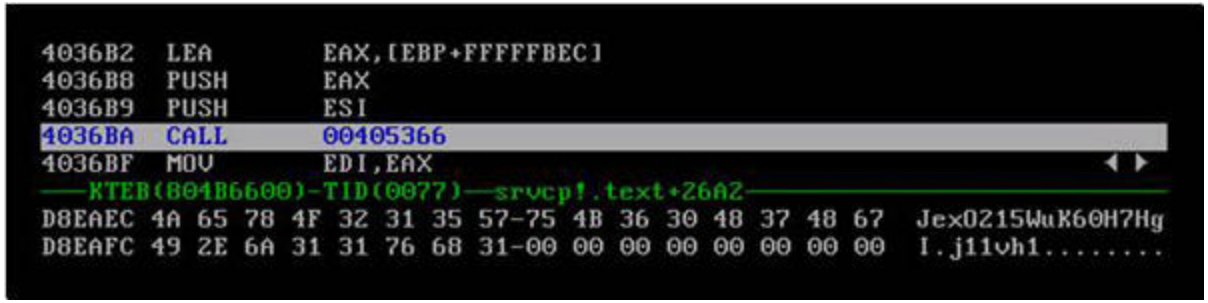
- Encrypted code
  - .ini files usually have PARAMNAME=value format parsed with associated sscanf string
  - call sub_405366 must be decryption routine

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
```

Example #1 srvcp.exe

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



Example #1 srvcp.exe

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

Decrypted gus.ini File

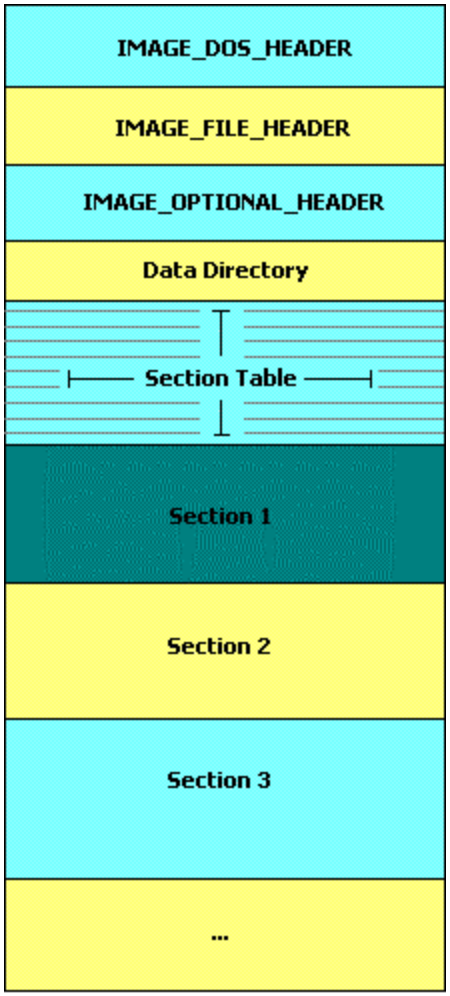
```
NICK=mikey
MODE=AGGRESSIVE
SETCOMMAND=setpr
COMMAND=fuckedup
CHANNEL=mikag soup
SOUPCHANNEL=alphasoup ah
SERVER0=irc.mcs.net:6666
SERVER1=efnet.cs.hut.fi:6666
SERVER2=efnet.demon.co.uk:6666
SERVER3=irc.concentric.net:6666
SERVER4=irc.etsmtl.ca:6666
SERVER5=irc.fasti.net:6666
... cut for brevity ...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

Example #1 srvcp.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

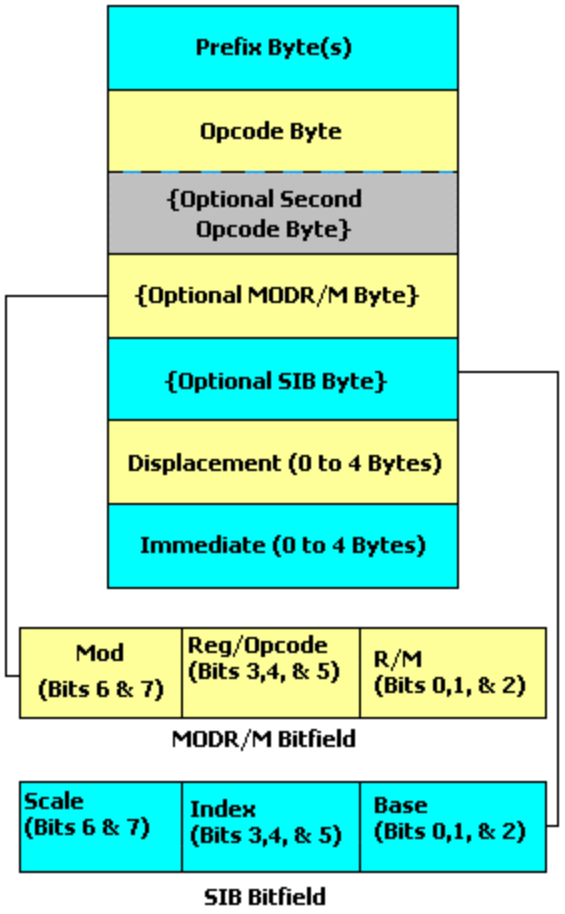
PE file format

- Based on COFF Format
  - Magic Numbers, Headers, Tables, Directories, Sections
- Disassemblers
  - Overlay Data with C Structures
  - Load File as OS Loader Would
  - Identify Entry Points (Default & Exported)



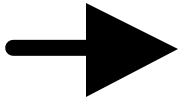
Disassembling

- 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()



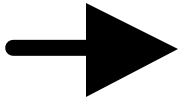
Disassembly example

```
0000    mov ecx, 5
0003    push aHello
0009    call printf
000E    loop 00000003h
0014    ...
```



```
for(int i=0;i<5;i++)
{
   printf("Hello");
}
```

```
0000    cmp ecx, 100h
0003    jnz 001Bh
0009    push aYes
000F    call printf
0015    jmp 0027h
001B    push aNo
0021    call printf
0027    ...
```



```
if(x == 256)
{
   printf("Yes");
}
else
{
   printf("No");
}
```

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+0FFFFFE70h]
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+0FFFFFE6Ch], eax
cmp dword ptr [ebp+0FFFFFE6Ch], byte
      0FFh
jnz 401047h
jmp 401165h
mov word ptr [ebp+0FFFFFE5Ch], 2
push 800h
call 4012B2h
mov word ptr [ebp+0FFFFFE5Eh], ax
push 0
call 4012ACh
mov dword ptr [ebp+0FFFFFE60h], eax
push 10h
lea ecx, [ebp+0FFFFFE5Ch]
push ecx
mov edx, [ebp+0FFFFFE6Ch]
push edx
call 4012A6h
cmp eax, byte 0FFh
jnz 40108Dh
jmp 401165h
push 1
mov eax, [ebp+0FFFFFE6Ch]
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;
}
```

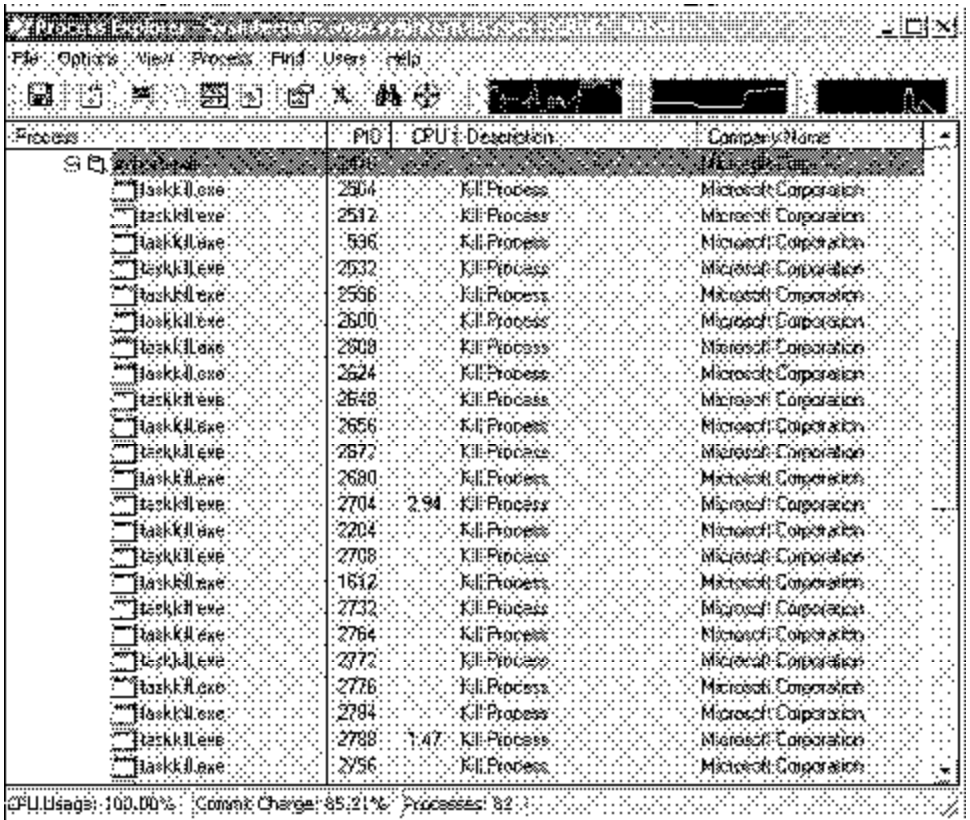
```
push 0
push 0
mov ecx, [ebp+0FFFFFE6Ch]
push ecx
call 40129Ah
mov [4030C4h], eax
cmp dword ptr [4030C4h], byte 0FFh
jnz 4010C8h
jmp 401165h
mov edx, [ebp+0FFFFFE6Ch]
push edx
call 401294h
push 0
mov eax, [403088h]
push eax
call 4012D0h
add esp, byte 4
push eax
mov ecx, [403088h]
push ecx
mov edx, [4030C4h]
push edx
call 40128Eh
cmp eax, byte 0FFh
jnz 4010FFh
jmp 401165h
push 0
push 0FFh
```

```
lea eax, [ebp+0FFFFFD58h]
push eax
mov ecx, [4030C4h]
push ecx
call 401288h
movzx edx, byte ptr [ebp+0FFFFFD58h]
cmp edx, byte 76h
jnz 40112Ch
call 401180h
jmp 401150h
movzx eax, byte ptr [ebp+0FFFFFD58h]
cmp eax, byte 64h
jnz 40113Fh
call 401200h
jmp 401150h
movzx ecx, byte ptr [ebp+0FFFFFD58h]
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
```

More code snippets

- Kills anti-virus, zone-alarm, firewall processes

```
.text:00403212                  lea     ecx, [ebp-34h]
.text:00403215                  call    ds:__vbaFreeObj
.text:0040321B                  mov     edi, ds:__vbaVarDup
.text:00403221                  lea     edx, [ebp-54h]
.text:00403224                  lea     ecx, [ebp-44h]
.text:00403227                  mov     dword ptr [ebp-4Ch], offset aTaskkillImBkav ; "taskkill /im bkav2006.exe"
.text:0040322E                  mov     dword ptr [ebp-54h], 8
```

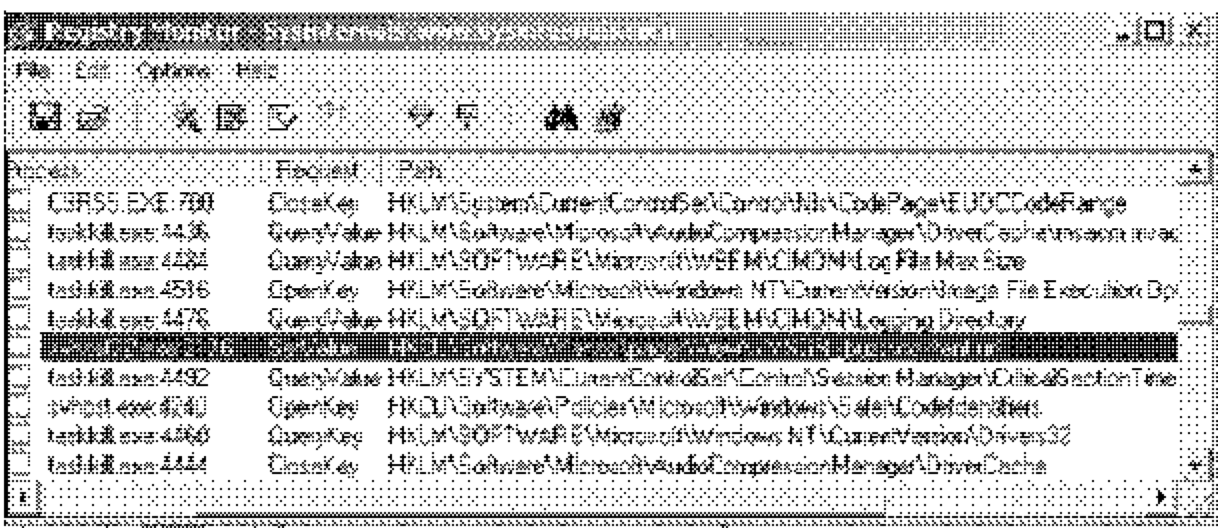


More code snippets

- Registry modifications for disabling task manager and changing browser default page

```
                          unicode 0, <Software\Microsoft\Windows\CurrentVersion\Po>
                          unicode 0, <licies\System>,0
                          unicode 0, <{}>,0
         aDisableregistr:
                          unicode 0, <DisableRegistryTools>,0
```

HKEY_CURRENT_USER\Software\Policies\Microsoft\Internet Explorer\Control Panel,Homepage
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\SystemDisableRegistryTools
HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\MainStart Page
HKEY_CURRENT_USER\Software\Yahoo\pager\View\YMSGR_buzz content url
HKEY_CURRENT_USER\Software\Yahoo\pager\View\YMSGR_Launchcast DisableTaskMgr



More code snippets

- New variants
  - Download worm update files and register them as services
  - regsvr32 MSINET.OCX
    - Internet Transfer ActiveX Control

```
text:004021C8                   lea     edx, [ebp-4Ch]
text:004021CB                   lea     ecx, [ebp-3Ch]
text:004021CE                   mov     dword ptr [ebp-44h], offset aRegsvr32Msinet ; "regsvr32 MSINET.OCX"
text:004021D5                   mov     dword ptr [ebp-4Ch], 8
```

  - Check for updates

```
.text:00[redacted] aHttpGiftshop_v:
.text:00[redacted]                  unicode 0, <http://gift[redacted]/update.txt>,0
```