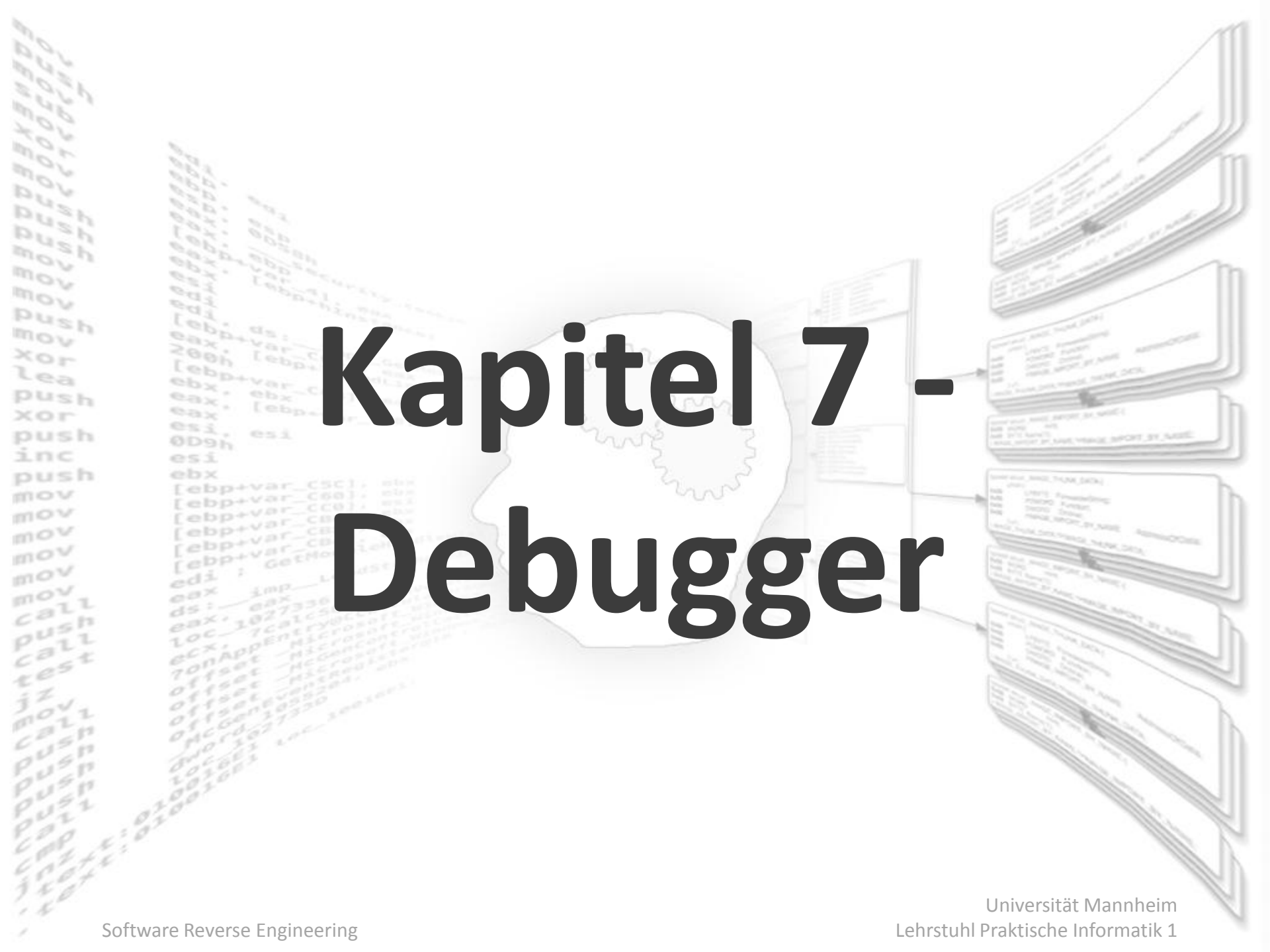


Kapitel 7 - Debugger

Universität Mannheim
Lehrstuhl Praktische Informatik 1

zynamics

- Hersteller von Reverse Engineering Tools
- Fünf Produkte
  - BinDiff
  - BinNavi
  - VxClass (Deutscher IT-Sicherheitspreis 2006)
  - BinCrowd
  - PDF Tool

Was ist ein Debugger (offiziell)?

Ein Debugger (von engl. bug im Sinne von Programmfehler) ist ein Werkzeug zum Diagnostizieren, Auffinden und Beheben von Fehlern in Computersystemen, dabei vor allem in Programmen, aber auch in der für die Ausführung benötigten Hardware.

Wikipedia

Was ist ein Debugger (RE)?

Ein Debugger (von engl. bug im Sinne von Programmfehler) ist ein Werkzeug zum Herausfinden der Funktionalität von Programmen für die kein Quellcode vorliegt.

Ich

Debugging im RE-Kontext

- Malware-Analyse
- Aufdeckung von Sicherheitslücken
- Entdeckung von Lizenzverstößen
- Behebung von Interoperabilitätsproblemen
- Erweiterung von Programmen nach Quellcodeverlust

Warum Debugger für RE?

- Statische Analyse oft schwierig
  - Verschlüsselung, Packer, ...
- Statische Analyse hat oft nicht alle benötigten Informationen
- Manchmal kapiert mans einfach nicht ohne Programmausführung

Verschiedene Debuggertypen

Ziel

Quelltextdebugger vs. Assemblerdebugger

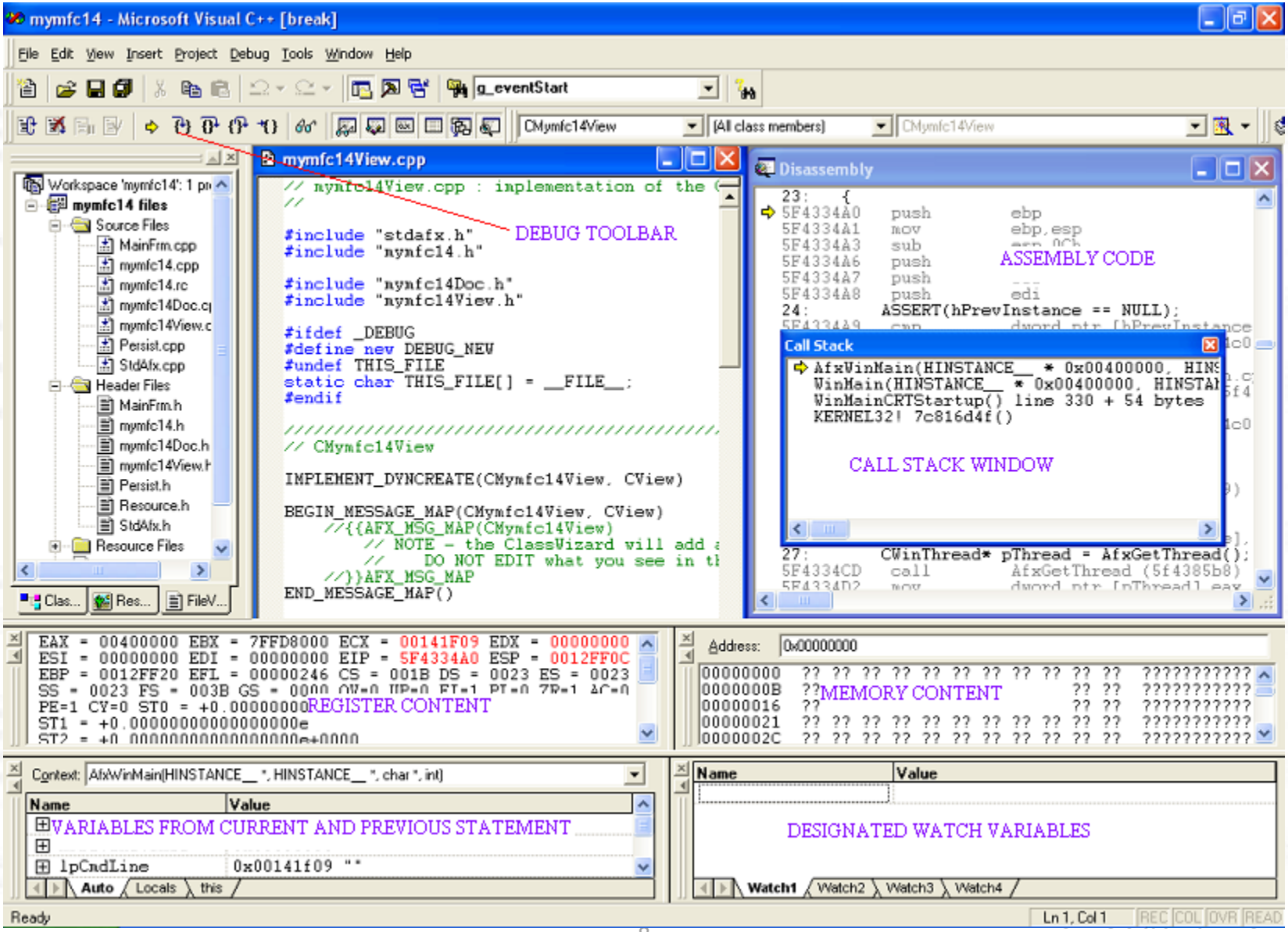
Benutzung

Lokaler Debugger vs. Remote Debugger

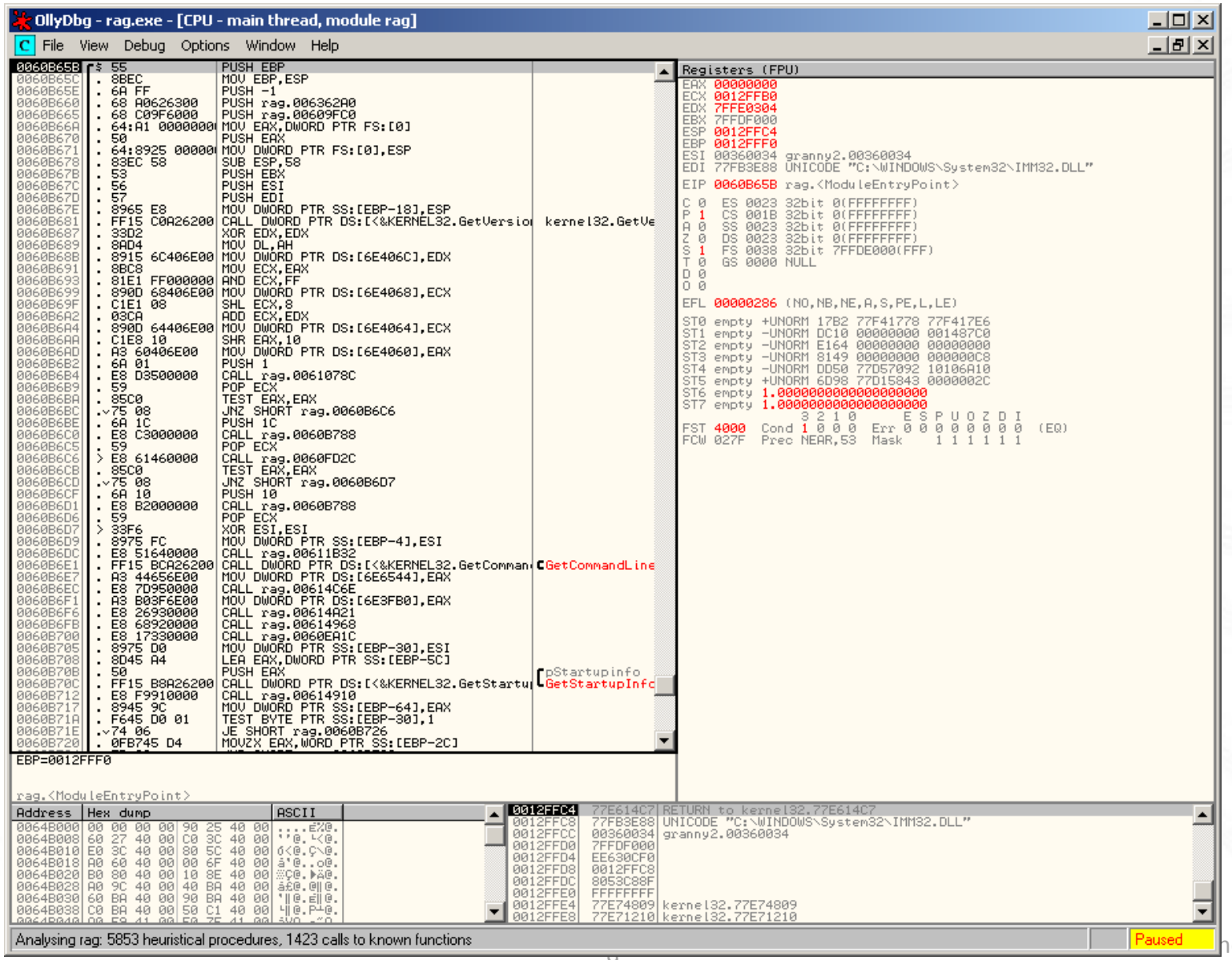
Mächtigkeit

Userland Debugger vs. Kernel Debugger

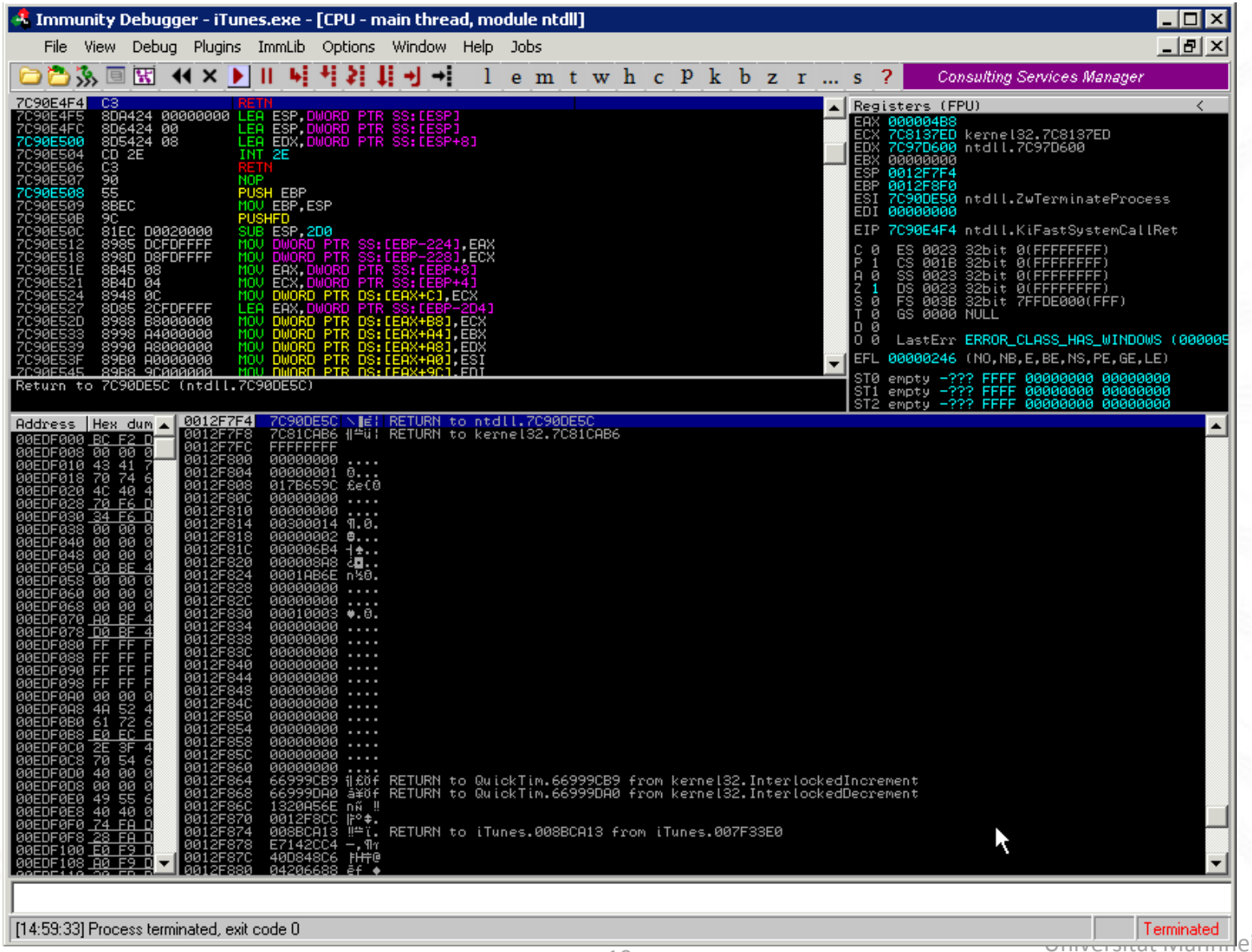
Visual Studio Debugger



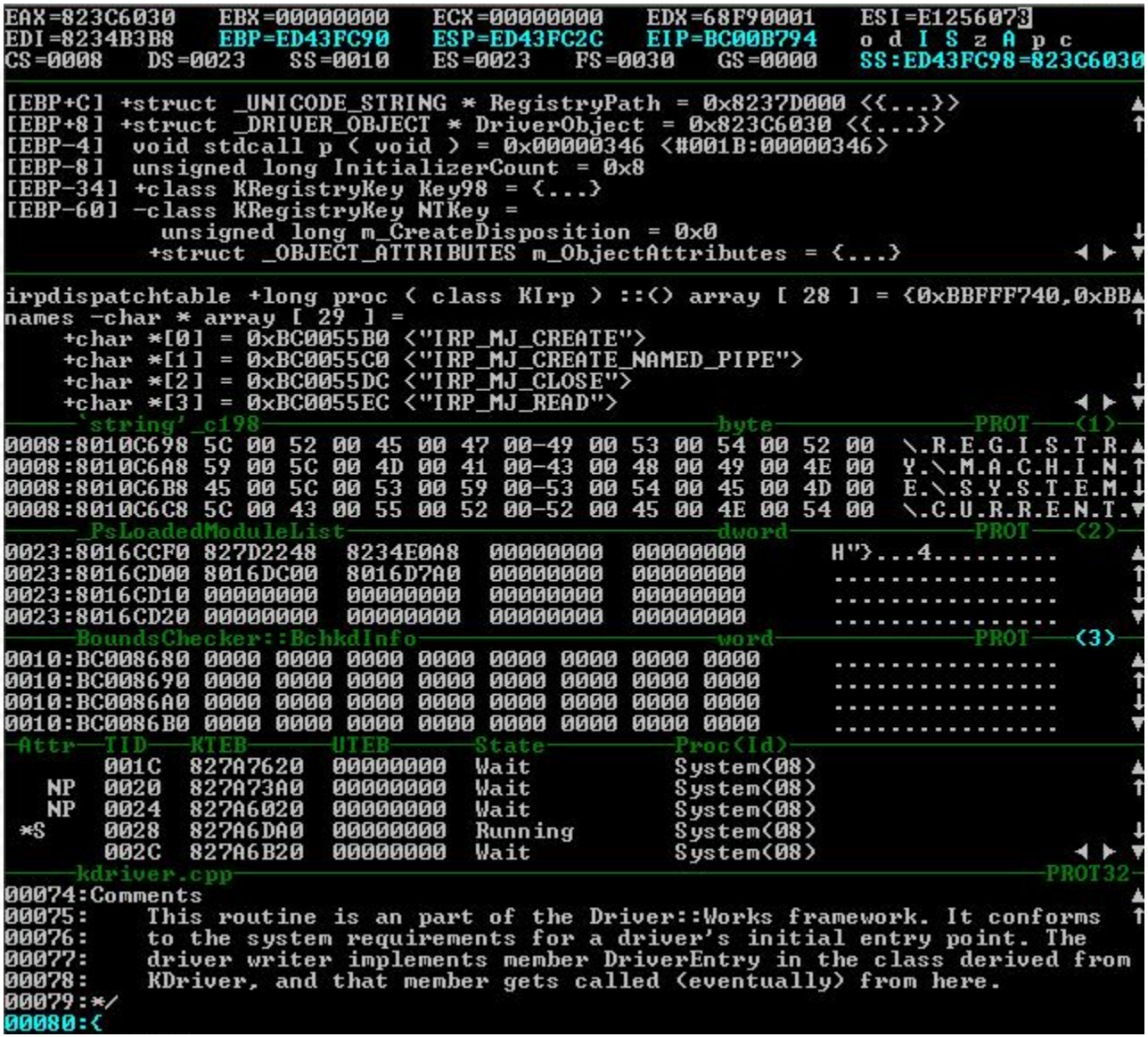
OllyDbg


OllyDbg - rag.exe - [CPU - main thread, module rag]
File View Debug Options Window Help
0060B65B $ 55 PUSH EBP
0060B65C . 8BEC MOV EBP,ESP
0060B65E . 6A FF PUSH -1
0060B660 . 68 A0626300 PUSH rag.006362A0
0060B665 . 68 C09F6000 PUSH rag.00609FC0
0060B66A . 64:A1 0000000(MOV EAX,DWORD PTR FS:[0]
0060B670 . 50 PUSH EAX
0060B671 . 64:8925 00000(MOV DWORD PTR FS:[0],ESP
0060B678 . 83EC 58 SUB ESP,58
0060B67B . 53 PUSH EBX
0060B67C . 56 PUSH ESI
0060B67D . 57 PUSH EDI
0060B67E . 8965 E8 MOV DWORD PTR SS:[EBP-18],ESP
0060B681 . FF15 C0A26200 CALL DWORD PTR DS:[<&KERNEL32.GetVersio kernel32.GetVe
0060B687 . 33D2 XOR EDX,EDX
0060B689 . 8AD4 MOV DL,AH
0060B68B . 8915 6C406E00 MOV DWORD PTR DS:[6E406C],EDX
0060B691 . 8BC8 MOV ECX,EAX
0060B693 . 81E1 FF000000 AND ECX,FF
0060B699 . 890D 68406E00 MOV DWORD PTR DS:[6E4068],ECX
0060B69F . C1E1 08 SHL ECX,8
0060B6A2 . 03CA ADD ECX,EDX
0060B6A4 . 890D 64406E00 MOV DWORD PTR DS:[6E4064],ECX
0060B6AA . C1E8 10 SHR EAX,10
0060B6AD . A3 60406E00 MOV DWORD PTR DS:[6E4060],EAX
0060B6B2 . 6A 01 PUSH 1
0060B6B4 . E8 D3500000 CALL rag.0061078C
0060B6B9 . 59 POP ECX
0060B6BA . 85C0 TEST EAX,EAX
0060B6BC .v75 08 JNZ SHORT rag.0060B6C6
0060B6BE . 6A 1C PUSH 1C
0060B6C0 . E8 C3000000 CALL rag.0060B788
0060B6C5 . 59 POP ECX
0060B6C6 > E8 61460000 CALL rag.0060FD2C
0060B6CB . 85C0 TEST EAX,EAX
0060B6CD .v75 08 JNZ SHORT rag.0060B6D7
0060B6CF . 6A 10 PUSH 10
0060B6D1 . E8 B2000000 CALL rag.0060B788
0060B6D6 . 59 POP ECX
0060B6D7 > 33F6 XOR ESI,ESI
0060B6D9 . 8975 FC MOV DWORD PTR SS:[EBP-4],ESI
0060B6DC . E8 51640000 CALL rag.00611B32
0060B6E1 . FF15 BCA26200 CALL DWORD PTR DS:[<&KERNEL32.GetComman GetCommandLine
0060B6E7 . A3 44656E00 MOV DWORD PTR DS:[6E6544],EAX
0060B6EC . E8 7D950000 CALL rag.00614C6E
0060B6F1 . A3 B03F6E00 MOV DWORD PTR DS:[6E3FB0],EAX
0060B6F6 . E8 26930000 CALL rag.00614A21
0060B6FB . E8 68920000 CALL rag.00614968
0060B700 . E8 17330000 CALL rag.0060EA1C
0060B705 . 8975 D0 MOV DWORD PTR SS:[EBP-30],ESI
0060B708 . 8D45 A4 LEA EAX,DWORD PTR SS:[EBP-5C] pStartupinfo
0060B70B . 50 PUSH EAX GetStartupInfo
0060B70C . FF15 B8A26200 CALL DWORD PTR DS:[<&KERNEL32.GetStartu
0060B712 . E8 F9910000 CALL rag.00614910
0060B717 . 8945 9C MOV DWORD PTR SS:[EBP-64],EAX
0060B71A . F645 D0 01 TEST BYTE PTR SS:[EBP-30],1
0060B71E .v74 06 JE SHORT rag.0060B726
0060B720 . 0FB745 D4 MOVZX EAX,WORD PTR SS:[EBP-2C]
EBP=0012FFF0
rag.<ModuleEntryPoint>
Registers (FPU)
EAX 00000000
ECX 0012FFB0
EDX 7FFE0304
EBX 7FFDF000
ESP 0012FFC4
EBP 0012FFF0
ESI 00360034 granny2.00360034
EDI 77FB3E88 UNICODE "C:\WINDOWS\System32\IMM32.DLL"
EIP 0060B65B rag.<ModuleEntryPoint>
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 1 FS 0038 32bit 7FFDE000(FFF)
T 0 GS 0000 NULL
D 0
O 0
EFL 00000286 (NO,NB,NE,A,S,PE,L,LE)
ST0 empty +UNORM 17B2 77F41778 77F417E6
ST1 empty -UNORM DC10 00000000 001487C0
ST2 empty -UNORM E164 00000000 00000000
ST3 empty -UNORM 8149 00000000 000000C8
ST4 empty -UNORM DD50 77D57092 10106A10
ST5 empty +UNORM 6D98 77D15843 0000002C
ST6 empty 1.0000000000000000000
ST7 empty 1.0000000000000000000
3 2 1 0 E S P U O Z D I
FST 4000 Cond 1 0 0 0 Err 0 0 0 0 0 0 0 0 (EQ)
FCW 027F Prec NEAR,53 Mask 1 1 1 1 1 1
Address Hex dump ASCII
0064B000 00 00 00 00 90 25 40 00 ....É%@.
0064B008 60 27 40 00 C0 3C 40 00 `'@.└<@.
0064B010 E0 3C 40 00 80 5C 40 00 ó<@.Ç\@.
0064B018 A0 60 40 00 00 6F 40 00 á`@..o@.
0064B020 B0 80 40 00 10 8E 40 00 ▒Ç@.▶Ä@.
0064B028 A0 9C 40 00 40 BA 40 00 á£@.@‖@.
0064B030 60 BA 40 00 90 BA 40 00 `‖@.É‖@.
0064B038 C0 BA 40 00 50 C1 40 00 └‖@.P┴@.
0012FFC4 77E614C7 RETURN to kernel32.77E614C7
0012FFC8 77FB3E88 UNICODE "C:\WINDOWS\System32\IMM32.DLL"
0012FFCC 00360034 granny2.00360034
0012FFD0 7FFDF000
0012FFD4 EE630CF0
0012FFD8 0012FFC8
0012FFDC 8053C88F
0012FFE0 FFFFFFFF
0012FFE4 77E74809 kernel32.77E74809
0012FFE8 77E71210 kernel32.77E71210
Analysing rag: 5853 heuristical procedures, 1423 calls to known functions
Paused

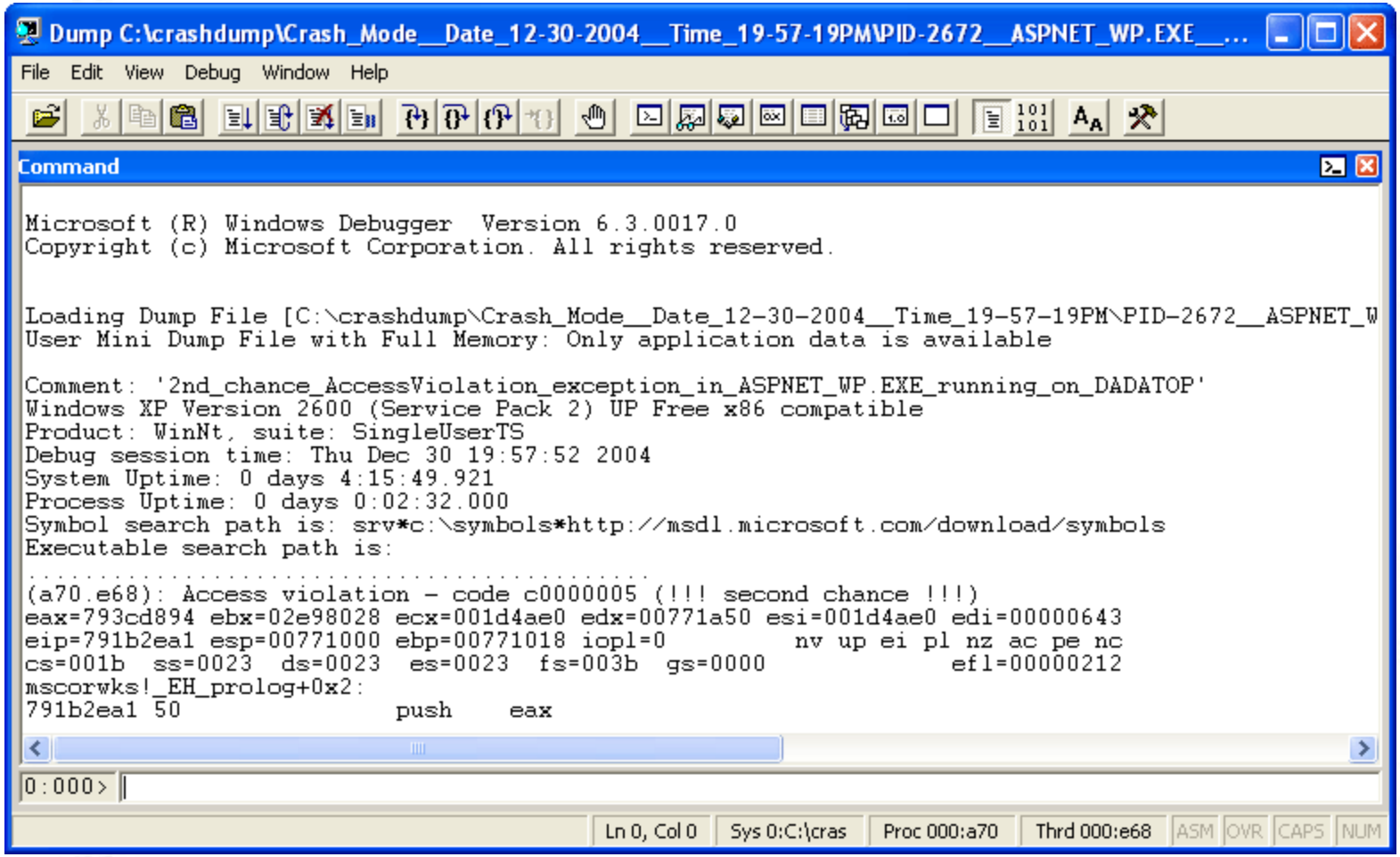
Immunity Debugger



SoftICE


EAX=823C6030 EBX=00000000 ECX=00000000 EDX=68F90001 ESI=E1256073
EDI=8234B3B8 EBP=ED43FC90 ESP=ED43FC2C EIP=BC00B794 o d I S z A p c
CS=0008 DS=0023 SS=0010 ES=0023 FS=0030 GS=0000 SS:ED43FC98=823C6030
[EBP+C] +struct _UNICODE_STRING * RegistryPath = 0x8237D000 <{...}>
[EBP+8] +struct _DRIVER_OBJECT * DriverObject = 0x823C6030 <{...}>
[EBP-4] void stdcall p (void) = 0x00000346 <#001B:00000346>
[EBP-8] unsigned long InitializerCount = 0x8
[EBP-34] +class KRegistryKey Key98 = {...}
[EBP-60] -class KRegistryKey NTKey =
unsigned long m_CreateDisposition = 0x0
+struct _OBJECT_ATTRIBUTES m_ObjectAttributes = {...}
irpdispatchtable +long proc (class KIrp) ::() array [28] = {0xBBFFF740,0xBB
names -char * array [29] =
+char *[0] = 0xBC0055B0 <"IRP_MJ_CREATE">
+char *[1] = 0xBC0055C0 <"IRP_MJ_CREATE_NAMED_PIPE">
+char *[2] = 0xBC0055DC <"IRP_MJ_CLOSE">
+char *[3] = 0xBC0055EC <"IRP_MJ_READ">
'string' c198 byte PROT (1)
0008:8010C698 5C 00 52 00 45 00 47 00-49 00 53 00 54 00 52 00 \.R.E.G.I.S.T.R.
0008:8010C6A8 59 00 5C 00 4D 00 41 00-43 00 48 00 49 00 4E 00 Y.\.M.A.C.H.I.N.
0008:8010C6B8 45 00 5C 00 53 00 59 00-53 00 54 00 45 00 4D 00 E.\.S.Y.S.T.E.M.
0008:8010C6C8 5C 00 43 00 55 00 52 00-52 00 45 00 4E 00 54 00 \.C.U.R.R.E.N.T.
_PsLoadedModuleList dword PROT (2)
0023:8016CCF0 827D2248 8234E0A8 00000000 00000000 H"}...4.....
0023:8016CD00 8016DC00 8016D7A0 00000000 00000000 ....
0023:8016CD10 00000000 00000000 00000000 00000000 ....
0023:8016CD20 00000000 00000000 00000000 00000000 ....
BoundsChecker::BchkdInfo word PROT (3)
0010:BC008680 0000 0000 0000 0000 0000 0000 0000 0000 ....
0010:BC008690 0000 0000 0000 0000 0000 0000 0000 0000 ....
0010:BC0086A0 0000 0000 0000 0000 0000 0000 0000 0000 ....
0010:BC0086B0 0000 0000 0000 0000 0000 0000 0000 0000 ....
Attr TID KTEB UTEB State Proc(Id)
001C 827A7620 00000000 Wait System(08)
NP 0020 827A73A0 00000000 Wait System(08)
NP 0024 827A6020 00000000 Wait System(08)
*S 0028 827A6DA0 00000000 Running System(08)
002C 827A6B20 00000000 Wait System(08)
kdriver.cpp PROT32
00074:Comments
00075: This routine is an part of the Driver::Works framework. It conforms
00076: to the system requirements for a driver's initial entry point. The
00077: driver writer implements member DriverEntry in the class derived from
00078: KDriver, and that member gets called (eventually) from here.
00079:*/
00080:{

WinDbg


Dump C:\crashdump\Crash_Mode__Date_12-30-2004__Time_19-57-19PM\PID-2672__ASPNET_WP.EXE__...
File Edit View Debug Window Help
Command
Microsoft (R) Windows Debugger Version 6.3.0017.0
Copyright (c) Microsoft Corporation. All rights reserved.
Loading Dump File [C:\crashdump\Crash_Mode__Date_12-30-2004__Time_19-57-19PM\PID-2672__ASPNET_W
User Mini Dump File with Full Memory: Only application data is available
Comment: '2nd_chance_AccessViolation_exception_in_ASPNET_WP.EXE_running_on_DADATOP'
Windows XP Version 2600 (Service Pack 2) UP Free x86 compatible
Product: WinNt, suite: SingleUserTS
Debug session time: Thu Dec 30 19:57:52 2004
System Uptime: 0 days 4:15:49.921
Process Uptime: 0 days 0:02:32.000
Symbol search path is: srv*c:\symbols*http://msdl.microsoft.com/download/symbols
Executable search path is:
(a70.e68): Access violation - code c0000005 (!!! second chance !!!)
eax=793cd894 ebx=02e98028 ecx=001d4ae0 edx=00771a50 esi=001d4ae0 edi=00000643
eip=791b2ea1 esp=00771000 ebp=00771018 iopl=0 nv up ei pl nz ac pe nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000212
mscorwks!_EH_prolog+0x2:
791b2ea1 50 push eax
0:000>
Ln 0, Col 0
Sys 0:C:\cras
Proc 000:a70
Thrd 000:e68
ASM
OVR
CAPS
NUM

Standardfunktionalität

- Programmzustand untersuchen
  - Register, Speicher, Stack, ...
- Programmzustand verändern
- Haltepunkte setzen
- Einzelschrittausführung
- Bekannte Datenstrukturen einsehen
- Exception-Handler-Kette einsehen
- ...

Windows Debugger

- Meistens mit Hilfe der Windows Debug API implementiert
  - Funktionalität in kernel32.dll, psapi.dll und dbghelp.dll
- Wichtige Funktionen
  - DebugActiveProcess: Prozess debuggen
  - WaitForDebugEvent: Wartet auf Debugger-Events
  - DebugBreakProcess: Anhalten des Prozesses der vom Debugger kontrolliert wird.

Laufenden Prozess debuggen

- Benutzer muss SeDebugPrivilege besitzen
- OpenProcess
- DebugActiveProcess
- WaitForDebugEvent-Schleife
- DebugSetProcessKillOnExit (Windows XP+)
- DebugActiveProcessStop (Windows XP+)

Neuen Prozess debuggen

- Benutzer muss SeDebugPrivilege besitzen
- CreateProcess mit Debug-Flags
- WaitForDebugEvent-Schleife
- DebugSetProcessKillOnExit (Windows XP+)
- DebugActiveProcessStop (Windows XP+)

Die Debugger-Schleife

```
DEBUG_EVENT event;

for (;;)
{
    WaitForDebugEvent(&event, INFINITE);

    switch(event.dwDebugEventCode)
    {
        // Debug Events werden hier verarbeitet
    }

    ContinueDebugEvent( … );
}
```

Mögliche Debug Events

- EXCEPTION_DEBUG_EVENT
- CREATE_THREAD_DEBUG_EVENT
- CREATE_PROCESS_DEBUG_EVENT
- EXIT_THREAD_DEBUG_EVENT
- EXIT_PROCESS_DEBUG_EVENT
- LOAD_DLL_DEBUG_EVENT
- UNLOAD_DLL_DEBUG_EVENT
- OUTPUT_DEBUG_STRING_EVENT
- RIP_EVENT

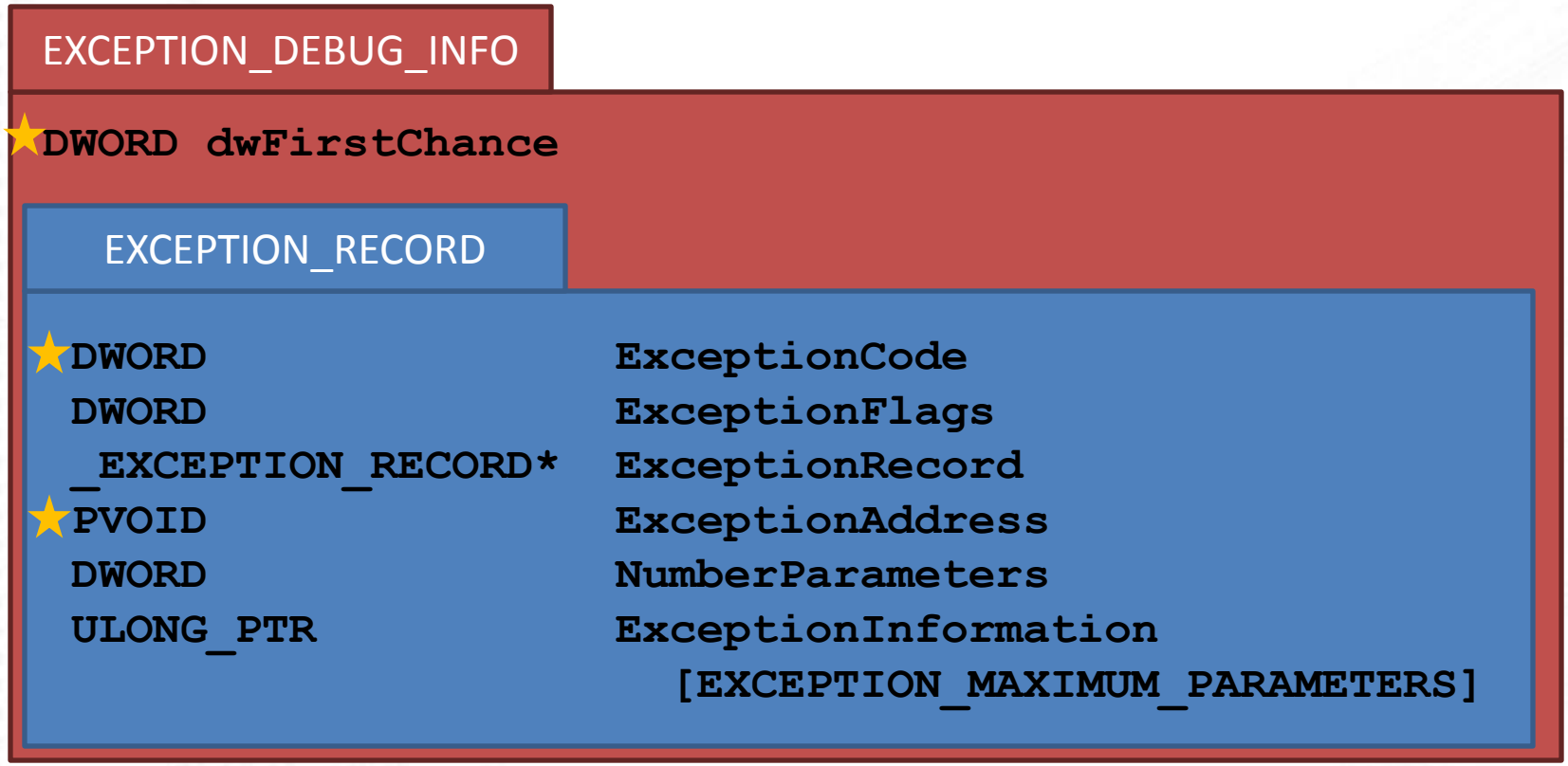
Verarbeitung des Programmzustands

- Registerwerte lesen
  - OpenThread + GetThreadContext
- Registerwerte schreiben
  - OpenThread + GetThreadContext + SetThreadContext
- Speicher auslesen
  - VirtualProtextEx + ReadProcessMemory + VirtualProtextEx
- Speicher schreiben
  - VirtualProtextEx + WriteProcessMemory + VirtualProtectEx

Exception Events

- EXCEPTION_DEBUG_EVENT
- Treten auf im Falle einer Ausnahmesituation
  - Access Violation
  - Divide By Zero
  - Privileged Instruction
  - …
- Treten auch auf, wenn ein Haltepunkt getroffen oder ein Einzelschritt ausgeführt wird

Behandlung von Exception Events



Behandlung von Exception Events

- Herausfinden der genauen Ausnahme
  - event.u.Exception.ExceptionRecord.ExceptionCode
- Reguläre Ausnahmen
  - First Pass: An Anwendung weitergeben
  - Second Pass: Fehlermeldung anzeigen
- Haltepunkt und Einzelschritt Exception
  - Verarbeiten im Debugger

Feature: Software-Haltepunkte

- Möglichkeit ein Programm an einer beliebigen Stelle anzuhalten
- Ermöglicht das Überspringen unwichtiger Programmpfade
- Originalcode wird durch Haltepunkt-Befehl 0xCC ersetzt

Behandlung von Haltepunkten

- Ermittlung der Haltepunkt-Adresse
  - Adresskorrektur durchführen
- Originalbefehl wiederherstellen
- Warten bis der Prozess fortgesetzt werden soll
- Einzelschritt ausführen
  - Achtung: Race Condition
  - Achtung: Was, wenn am nächsten Befehl auch ein BP ist?
- Haltepunkt wiederherstellen
- Prozess weiter ausführen

Feature: Hardware-Haltepunkte

- Funktionieren ganz anders
- Maximal 4 Haltepunkte pro CPU
- Adressen in den Registern DR0 bis DR3
- DR7 zur Konfiguration der Haltepunkte
- Vorteil: Modifizieren Originalcode nicht

Feature: Einzelschrittausführung

- Nur der nächste Befehl soll ausgeführt werden
  - Eventuell Verzweigung in Unterfunktion
- Minimale Änderung des Programmzustands
- Vielleicht das wichtigste Feature zum Programmverständnis
- Einzelschritt eines Threads wird mit dessen Trap-Flag ausgeführt

Behandlung von Einzelschritten

- Ermittlung der Einzelschrittaddresse
  - Keine Addresskorrektur notwendig
- Trap-Flag wird automatisch zurückgesetzt

Feature: Prozedurschritt

- Sprung zum nächsten Befehl
  - Ignorieren von Funktionsaufrufen
- Sehr nützlich um bekannte Funktionen zu überspringen

Behandlung von Prozedurschritt

- Prozedurschritt wird nie nativ unterstützt
- Stattdessen Simulation
  - Ist der aktuelle Befehl ein Funktionsaufruf?
  - Nein: Dann Einzelschritt
  - Ja: Dann Haltepunkt auf den nächsten Befehl und Programm normal ausführen

Feature: Tracemodus

- Ausführen eines Programms und Protokollierung des Programmzustands nach jedem Befehl
- Sehr nützlich zum Datensammeln für spätere statische Analyse

Behandlung von Tracemodus

- Tracemodus wird nie nativ unterstützt
- Stattdessen Simulation
  - Wiederholte Ausführung von Einzelschritten
  - Aufzeichnung von Registerwerten und Speicherzellen

Thread Events

- Treten in zwei Fällen auf:
  - Ein neuer Thread wird gestartet
  - Ein aktiver Thread wird beendet

Feature: Threads

- Begrenzte Nützlichkeit beim Reverse Engineering
- Wichtig bei Malware-Analyse
  - Einfacher Weg zum Finden von Command-Threads
- Ab und zu ist es nützlich Threads einschlafen zu lassen

Behandlung von Thread Creation

CREATE_THREAD_DEBUG_INFO

```
HANDLE                  hThread;
LPVOID                  lpThreadLocalBase;
LPTHREAD_START_ROUTINE  lpStartAddress;
```

Behandlung von Thread Creation

- CREATE_THREAD_DEBUG_EVENT
- Thread ID feststellen
- Thread Zustand feststellen
- Startadresse feststellen
- Weitergabe der Informationen an den Benutzer
  - Eventuell Programm anhalten

Behandlung von ExitThread

Behandlung von Thread Exit

- EXIT_THREAD_DEBUG_EVENT
- Thread ID feststellen
- Weitergabe der Informationen an den Benutzer
  - Eventuell Programm anhalten

Prozess Events

- Treten in vier Fällen auf:
  - Ein neuer Prozess wird gestartet
  - Der aktive Prozess wird beendet
  - Eine DLL wird in den aktiven Prozess geladen
  - Eine DLL wird aus dem aktiven Prozess entladen

Feature: Prozesse

- Begrenzte Nützlichkeit beim Reverse Engineering
- Wichtig bei der Malware-Analyse
  - Häufig starten Malwareprozesse neue Prozesse

Behandlung von Process Creation

CREATE_PROCESS_DEBUG_INFO	
★HANDLE	hFile
★HANDLE	hProcess
★HANDLE	hThread
LPVOID	lpBaseOfImage
DWORD	dwDebugInfoFileOffset
DWORD	nDebugInfoSize
LPVOID	lpThreadLocalBase
★LPTHREAD_START_ROUTINE	lpStartAddress
LPVOID	lpImageName
WORD	fUnicode

Behandlung von Process Creation

- CREATE_PROCESS_DEBUG_EVENT
- Ermittlung der geladenen Datei
  - Erstaunlich kompliziert (Google: GetFileNameFromHandle)
- Ermittlung weiterer Informationen
  - Prozess ID
  - Initialer Thread
  - Startaddresse
- Weitergabe der Informationen an den Benutzer
  - Eventuell Programm anhalten

Behandlung von Process Exit

Behandlung von Process Exit

- EXIT_PROCESS_DEBUG_EVENT
- Ermittlung des Rückgabewertes
- Beenden des Debuggers

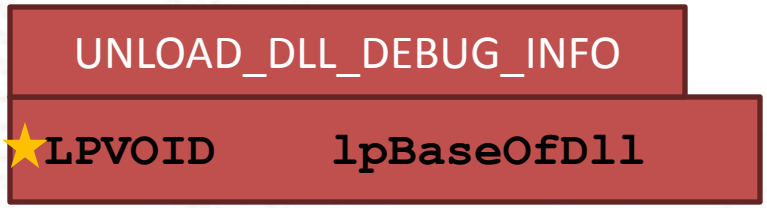
Behandlung von DLL Loading

LOAD_DLL_DEBUG_INFO	
★HANDLE	hFile
★LPVOID	lpBaseOfDll
DWORD	dwDebugInfoFileOffset
DWORD	nDebugInfoSize
LPVOID	lpImageName
WORD	fUnicode

Behandlung von DLL Loading

- LOAD_DLL_DEBUG_EVENT
- Ermittlung der geladenen Datei
  - Erstaunlich kompliziert (Google: GetFileNameFromHandle)
- Ermittlung weiterer Informationen
  - Ladeadresse der Datei im Prozess
  - Dateigröße
- Updaten der internen Prozessverwaltung
- Weitergabe der Informationen an den Benutzer
  - Eventuell Programm anhalten

Behandlung von DLL Unloading



Behandlung von DLL Unloading

- UNLOAD_DLL_DEBUG_EVENT
- Ermittlung der entladenen Datei
  - Erstaunlich einfach
- Updaten der internen Prozessverwaltung
- Weitergabe der Informationen an den Benutzer
  - Eventuell Programm anhalten

Anti-Debugging

- Malware und andere Programme wehren sich aktiv gegen Analyse
- Anti-Debugging Code ist nur eine von vielen Möglichkeiten
- Vorgehensweisen
  - Erkennen eines Debuggers
  - Verhindern der Funktion eines Debuggers
  - Erschwerung der Analyse

Erkennen eines Debuggers

- Offizieller Weg
  - IsDebuggerPresent
  - CheckRemoteDebuggerPresent
- Dutzende inoffizieller Wege je Debugger
- Erkennen von Veränderungen im Code durch Haltepunkten
- Erkennen von Timing-Veränderungen
- Suche nach RE-Tools im Allgemeinen

Verhindern der Funktionalität

- Verhindern, dass sich ein Debugger an einen Prozess klemmt
- Automatisches Entfernen von Haltepunkten
- Gezieltes Auslösen von bekannten Bugs in Debuggern

Erschwerung der Analyse

- Laufzeitpacker benutzen
- Codeobfuscation
- Versuchen den Disassembler zu stören
- Komplizierter Verlauf des Kontrollflusses, etwa über Exceptionhandler

Referenzen

- Iczelion‘s Win32Asm Tutorials
  - http://win32assembly.online.fr/tutorials.html
- MSDN Debugging API
  - http://msdn.microsoft.com/en-us/library/ms679303(VS.85).aspx
- Gray Hat Python von Justin Seitz
- Advanced Windows Debugging von Hario Herwardt und Daniel Pravat