

Debugging Win32 Code

Troubleshooting Beneath the Abstractions

Kent Forschmeidt
Development Lead
Windows NT Debugger Team
Microsoft Corporation

Agenda

- ◆ **Windows NT debugging tools**
 - Debuggers, related tools
- ◆ **Debugging environment**
 - Checked/free, symbols, dump files
- ◆ **Basic debugging commands**
- ◆ **Common debugging problems**
 - Meta problems
 - Memory corruption
 - Call stacks
- ◆ **Call stack demos**

Windows NT Debugging Tools

Debugger Types

- ◆ **High-level language debugging**
 - Scripting, interpreted, bytecode, etc.
- ◆ **Win32 user-mode debugging**
 - Win32 apps/services/DLLs, COM objects
 - (focus of this presentation)
- ◆ **Windows NT kernel-mode debugging**
 - Windows NT drivers
 - system crashes (blue screen)

Win32 Debugger Options

- ◆ WinDbg: user-mode, kernel-mode, GUI
- ◆ NTSD/CDB: user-mode, console
- ◆ KD: kernel-mode, console
- ◆ Visual C++ IDE: user-mode, GUI
- ◆ Compuware NuMega SoftICE: user-mode, kernel-mode, GUI, x86-only, 1-system
- ◆ Other third party IDE: mostly user-mode

KD

- ◆ Windows NT kernel-mode debugger
- ◆ i386KD for x86, alphaKD for Alpha
- ◆ Use with matching NT version
- ◆ Remoteable via Remote.exe

NTSD/CDB

- ◆ Win32 user-mode debugger
- ◆ CDB is NTSD without a creating a new window
- ◆ Use with matching NT version
- ◆ Remoteable via Remote.exe

Windbg

- ◆ Microsoft Windows debugger
- ◆ Win32 user-mode and Windows NT kernel debugger
- ◆ GUI, with source-level debugging
- ◆ Works on Windows NT 5.0 and 4.0
- ◆ User and kernel crash dumps
- ◆ Two kinds of remoteability

Related Tools

◆ Link

- `link -dump {-headers, -FPO, -disasm}`
- `link -dump -headers my.exe | findstr "check time"`

◆ PdbDump

- Available in Beta 3
- Examine the debugging information
- `pdbdump mydll.pdb dbi mydll.dll ...`

Related Tools

◆ Dumpchk

- Validate a crash dump file
- Display basic information
- Now supports kernel and user dumps

◆ cl

- cl -E to preprocess source
- cl /Fa file to make source/asm listing

◆ nmake

- Use nmake rules to generate helpful files

Related Tools

◆ Remote.exe

- Works with stdio console apps
- Remote shell - DO try this at home:

Start a remote shell on “MyComputer”:

```
c:> remote /s cmd remoteshell
```

Connect to it from elsewhere:

```
c:> remote /c MyComputer remoteshell
```

Debugging And Build Environment

Installing The Debugger

- ◆ **Install Windbg from SDK, DDK, MSDN or Web:**
 - <http://msdn.microsoft.com/developer/sdk/windbg.htm>
 - <http://www.microsoft.com/hwdev/ddk/ddknt.htm>
 - <http://www.microsoft.com/hwdev/ntdebugging.htm>
- ◆ **NEW: Debugger wizard: DbgWiz.exe**
 - Aid for configuring a debugging session
- ◆ **Updated: Windbg beta 2 online help**

Checked And Free Builds

- ◆ **Windows NT retail (free) build**
- ◆ **Windows NT checked (debug) build**
 - **Included on MSDN with DDK**
- ◆ **Service packs from SP3 forward available in retail and checked builds**
- ◆ **Hotfixes free build only**

Windows NT Symbol Files

- ◆ **Windows NT symbols on OS CD**
 - Located in \support\debug subdirectory
- ◆ **Symbols included with service packs and hotfixes**
- ◆ **On FTP server:**
 - Symbols are separate from executables and often not downloaded
 - <ftp://ftp.microsoft.com/bussys/winnt/winnt-public/fixes>
- ◆ **Debug wizard**
 - Helps find various symbol files

Symbol Files

- ◆ **COFF, PDB, CV, DBG, SYM...**
 - Simplifying and standardizing
- ◆ **PDB files:**
 - Latest WinDbg supports PDB 6.0
- ◆ **Use the default location**
 - Trust the build tools, don't be creative
 - `splitsym -s, rebase -x, -u`
 - `symbols\{exe,dll,...}`
 - `\support\debug\expandsym.cmd`

Instrumenting Your Code

- ◆ **Debug versions**
 - **OutputDebugString, Assert**
 - **Boundary and data integrity checks**
- ◆ **Best instrumentation can be left in**
 - **Graceful recovery**
 - **Log useful information**
- ◆ **Debug retail code!**
 - **Test what the customer uses**
 - **Debug code may act differently**

Instrumenting Retail Code

- ◆ Using debugger-specific code
 - Use IsDebuggerPresent
 - ASSERT, OutputDebugString, DebugBreak
 - New DbgHelp in Beta 3
 - Debug-specific part of ImageHlp
 - Redistributable
 - H, LIB, DLL in SDK
 - Stack walking, SymAPI, etc.
 - Other options for MFC code (see VC6)

Instrumenting Your Code

- ◆ **Trust the optimizer**
 - **Use more:**
 - **Variables**
 - **Intermediate results**
 - **Lines of code**
 - **Don't reuse variables**
 - **Don't hide variables in scope**
- ◆ **Read “Writing Solid Code” by Steve McGuire**

Exception Handling

- ◆ **Try/Except blocks**
 - **Body, filter, handler**
 - **When an exception is raised, filter gets control**
 - **Exception information passed to filter**
 - **Filter determines next action**
 - **Resume execution**
 - **Execute handler**
 - **Continue search for handler**

Exception Handling

- ◆ **Unhandled exception**
 - **UnhandledExceptionHandler** gets it
 - **Invoke handler registered by app**
 - **SetUnhandledExceptionHandler**
 - **Show popup**
 - **Invoke debugger**
 - **Exit**

Exception Handling

- ◆ **Debugger first chance**
- ◆ **Frame based handler**
 - **Application-supplied handler**
 - **UnhandledExceptionHandler**
 - **Check for presence of debugger**
 - **If it isn't there, try to start it up**
 - **Pass exception to debugger**
- ◆ **Debugger second chance**

Dump Files

- ◆ Post mortem debugging with dump files
- ◆ Kernel-mode dumps
- ◆ User-mode dumps

Kernel-mode Crash Dumps

- ◆ OS crash
 - Memory is saved to dump file
 - %systemroot%\memory.dmp
- ◆ Requirements
 - Supported storage driver (HCL)
 - Enough free disk space for dump
- ◆ New for NT 5.0:
 - Summary dump
- ◆ UI in Control Panel's System Recovery
 - (next slide)

Kernel-mode Crash Dump



Control panel



System

The screenshot shows the 'System Properties' dialog box with the 'System Startup' and 'Recovery' tabs selected. The 'System Startup' section shows the system name 'Windows NT Workstation Version 4.00' and a startup delay of 30 seconds. The 'Recovery' section shows the settings for what to do when a STOP error occurs.

System Properties [?] [X]

General Performance Environment
Startup/Shutdown Hardware Profiles User Profiles

System Startup

"Windows NT Workstation Version 4.00"

30

Recovery

When a STOP error occurs, do the following:


- Write an event to the system log
- Send an administrative alert
- Write debugging information to:
%SystemRoot%MEMORY.DMP
- Overwrite an existing file
- Automatically reboot

OK Cancel Apply

User-mode Crash Dumps

- ◆ Get crash info from the field
- ◆ App or server crashes
 - Process state is saved to dump file
 - %systemroot%\user.dmp
- ◆ Dr. Watson - DrWtsn32.exe
 - creates user dump file
 - terminates process
- ◆ New for NT 5.0:
 - Dump a running (non-crashed) app

Dr. Watson

 **Dr. Watson for Windows NT** _ □ ↶

Log File Path:

Crash Dump:

Wave File:

Number of Instructions:

Number of Errors To Save:

Options

- Dump Symbol Table
- Dump All Thread Context
- Append To Existing Log File
- Visual Notification
- Sound Notification
- Create Crash Dump File

Application Errors

Verifying Dump Files

- ◆ **DumpChk utility**
 - Detects incomplete/corrupt dump file
 - Summarizes dump
- ◆ **New for NT 5.0:**
 - Supports user-mode dumps, as well as kernel-mode
 - Available in Beta 2, PDC release
- ◆ **Example (next slide)**

DumpChk - Verify

```
Command Prompt

C:\debug>dumpchk -q u:\*****\heaper\memory.dmp

Filename . . . . . .u:\v-redj\heaper\memory.dmp
Signature. . . . . .PAGE
ValidDump. . . . . .DUMP
MajorVersion. . . . . .free system
MinorVersion. . . . . .1381
DirectoryTablebase. . . . .0x00030000
PfnDataBase . . . . . .0xffbae000
PsLoadedModuleList. . . . .0x80147f30
PsActiveProcessHead . . . . .0x80147e28
MachineImageType . . . . .i386
NumberProcessors . . . . .1
BugCheckCode. . . . . .0x0000000a
BugCheckParameter1. . . . .0xe17b7b68
BugCheckParameter2. . . . .0x00000002
BugCheckParameter3. . . . .0x00000001
BugCheckParameter4. . . . .0x00000000

ExceptionCode . . . . .0x80000003
ExceptionFlags . . . . .0x00000001
ExceptionAddress . . . . .0x8011cef0

NumberOfRuns. . . . . .0x3
NumberOfPages . . . . .0x1f5d
Run #1
  BasePage . . . . . .0x1
  PageCount . . . . .0x9d
Run #2
  BasePage . . . . . .0x100
  PageCount . . . . .0xec0
Run #3
  BasePage . . . . . .0x1000
  PageCount . . . . .0x1000

*****
*****--> Validating the integrity of the PsLoadModuleList
*****
*****--> Performing a quick check <^C to end>
*****
*****--> Validating all physical addresses
*****
*****--> Validating all virtual addresses
*****
*****--> This dump file is good!
*****
```

Basic Debugging Commands

Displaying/Modifying Data

- ◆ **dd: dump DWORDS**
- ◆ **dw: dump WORDS**
- ◆ **db: dump BYTES**
- ◆ **da: dump ASCII**
- ◆ **du: dump Unicode (strings)**
- ◆ **d: repeat last dX command**
- ◆ **eX: edit data**
- ◆ **rXXX: display or modify registers**

Execution And Stepping

- ◆ **u: Unassemble**
 - u MyFunction
- ◆ **p: Program step**
 - p <count>
- ◆ **t: Trace into**
- ◆ **g: Go**
 - go until: g MyFunction

Breakpoints

- ◆ **Execution breakpoint**
bp MyFunction
- ◆ **Memory breakpoints - x86 only**
 - **Break on Read**
ba r4 0x45678
 - **Break on Write**
ba w2 0x10010

Breakpoints

- ◆ **Breakpoint with commands**
BP MyFunction /C “dd esp+4 l1;g”
- ◆ **Breakpoint by line number**
BP {,file.c,my.exe}@125
- ◆ **Pass count - only hit on the nth pass**
BP AnotherFunction /P 25

Breakpoint Commands

- ◆ **BL: list breakpoints**
- ◆ **BD: disable breakpoint(s)**
 - **Disable breakpoints 1 and 2:**
 - **bd 1 2**
- ◆ **BE: enable breakpoint(s)**
 - **Enable breakpoints 2,3,4,5:**
 - **be 2-5**
- ◆ **BC: clear breakpoint(s)**
 - **Clear all breakpoints:**
 - **bc ***

Handy Commands

- ◆ **.Logopen**
 - **Open debugger log file**
- ◆ **.Logappend**
 - **Append to log file**
- ◆ **.Logclose**
 - **Close logfile**

Handy Commands 2

- ◆ **.Unload**
 - Releases symbol file
- ◆ **!Reload**
 - Reloads all symbols
- ◆ **!Sympath**
 - Changes symbol search path
- ◆ **.Open <filename>**
 - Substitute for file.open dialog, useful for debugging shell code

More Handy Commands

- ◆ **.attach <pid>**
 - Also f6
- ◆ **.start <command line>**
- ◆ **.list**
 - Show combined asm/source listing
- ◆ **Shift-F5**
 - Kills process, unloads everything
 - Also Debug.Stop Debugging

Handy Commands Again

- ◆ **remote <name>**
 - Works with `remote.exe`
- ◆ **SX**
 - **sxe <exception #>**
 - “enable” stop on exception
 - **sxd <exception #>**
 - “disable” to silently continue
 - **sxn <exception #>**
 - “notify” continue with message

Enough Handy Commands

- ◆ **.opt**
 - Lots of settings, mostly in GUI
 - MasmEval
 - Select C or asm expressions
 - CaseSensitive
 - For all alphabetic comparisons
 - EPStep
 - Step to entry, not main
- ◆ **HELP command**
 - Lots of info

Custom Extension DLLs

- ◆ Write your own debugger extension
 - WinDbg user- and kernel-mode
 - *KD kernel-mode
 - NTSD user-mode
- ◆ Samples in SDK and DDK
- ◆ More info in DDK docs
- ◆ Note: expect interface to widen to 64 bits in 5.0

Common Debugging Problems

Meta Problems

Problems caused by the debugger

Symbols Problems

- ◆ Do the symbol files match the images?
 - Read the debugger's error messages
 - Rebase, bind will change checksum
 - rebase -u
 - bind -s
 - NTDLL, KERNEL32 can give false error - fixed in ntsd, beta 3 windbg.
 - Load the right ones
 - !sympath "x:\symbols;y:\z\symbols"
 - !reload

Checking Your Symbols

- ◆ Use the linker to look at image and DBG

```
Link -dump -headers my.dll
```

```
Link -dump -headers my.dbg
```

- ◆ PdbDump (beta 3)

```
c:> link -dump -headers windbg.exe | findstr  
Format
```

```
{...} Format: NB10, 361c55e3, 1,  
windbg.pdb
```

```
c:> pdbdump windbg.pdb hdr | findstr sig  
sig = 0x361c55e3
```

Optimizers Eat Information

- ◆ **Source lines confused**
 - Line does not exist
 - Nearby lines overlap
 - Lines execute out of order
- ◆ **Locals confused**
 - Not there at all
 - Memory reused
 - Register use

Optimizer Havoc

- ◆ **Tricky stack usage**
 - **Makes stack analysis harder**
 - **Different usage at different call sites**
- ◆ **Inline or coalesced functions**
 - **A function might not exist on its own**
 - **Identical functions removed by linker**
 - **Multiple symbols, one function**

The Debugger Breaks It!

- ◆ **Debug Heap Manager**
 - Memory allocations will be different
 - Attach to process to get regular heap
- ◆ **Timing will be slightly different**
 - Synchronization problems will change
- ◆ **Did startup environment change?**
 - Desktop shortcut environment may be different from command line
 - Startup directory might matter

App Problems

Whether the debugger is there or not

Heap/Memory Corruption

- ◆ **Many causes**
 - Buffer overruns
 - Random / uninitialized pointers
 - Reuse of stale or freed pointer
 - Secondary damage
- ◆ **Symptoms often indirect**
 - Delayed
 - Masked
 - Unrelated code

Solving Memory Corruption

- ◆ **Examine boundaries**
- ◆ **Look at data near broken stuff**
 - **Look for pointers**
 - **Reverse execute code**
- ◆ **Instrument when possible**
 - **Code or breakpoints**
- ◆ **Find source of problem**
- ◆ **Guess**

Call Stacks

Call Stacks

- ◆ **Where thread is, how it got there**
 - List of call frames
- ◆ **Finding arguments in stack**
 - Don't make assumptions
- ◆ **Why stacks get corrupted**
- ◆ **Reconstructing damaged stacks**

Broken Call Stacks

- ◆ **Stack's owner overwrote**
- ◆ **Pointer to stack passed to other thread**
- ◆ **Other random pointer**
- ◆ **Mismatched calling convention/prototype**
 - **Cast, GetProcAddress**
 - **Use of PVOID, PROC, FARPROC**
 - **Vector table**

Stack Trace Commands

- ◆ **k: Walk call stack**
- ◆ **kb: Show bits**
- ◆ **kv: Verbose - show frame info**

Useful Related Commands

- ◆ **DD: Dump dword**
- ◆ **U: Unassemble**
- ◆ **LN: List nearest symbol**
- ◆ **LM: List modules**

Alpha And Stacks

- ◆ Different calling convention
- ◆ No stack frame required
 - Most arguments passed in registers
 - Finding arguments requires disassembling function prologues
- ◆ Assembly language differences
 - RISC instruction set
 - Fixed instruction size
 - Aligned instructions

X86 Stack Frame Types

- ◆ **EBP: traditional linked list of frames**
- ◆ **FPO: frame pointer omitted**
 - EBP not used
 - EBP used as general purpose register
- ◆ **Ad hoc: assembly code**
 - With or without debug info

X86 EBP Stack Frame

Highest address

Args

Return address

Previous EBP

Saved registers

Local storage

Lowest address

X86 FPO Stack Frame

Highest address

Args

Return address

Saved registers

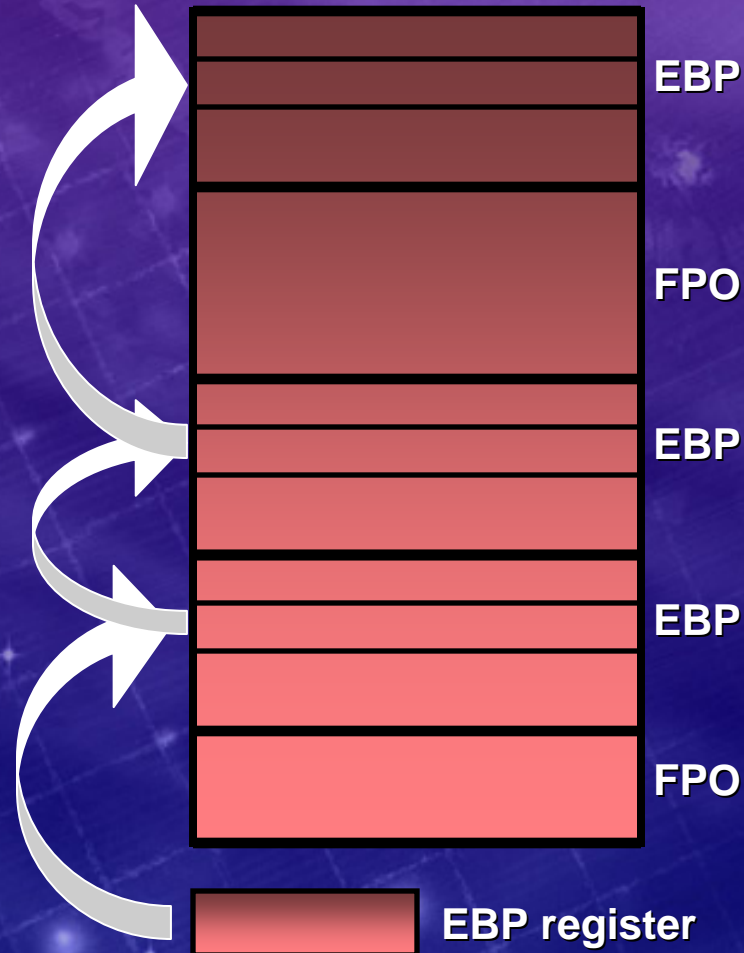
Local storage

Lowest address

EBP Chain Skips FPO Frames

Highest Address

Lowest Address



Demo One

Call stack basics

EBP Function

```
push ebp
mov ebp, esp
sub esp, xxx           ; end of prolog
...                   ; local reference
mov eax, [ebp + 14h] ; 4th arg
mov [ebp - 10h], ecx ; local variable
...
mov esp, ebp          ; start of epilog
pop ebp
ret 4                 ; discard 1 arg
```

Typical FPO Function

```
mov eax, [esp + 4]    ; 1st arg. other instructions may
push edi              ; appear between pushes & pops
mov ecx, [esp + 0Ch] ; 2nd arg - ESP has changed!
push esi
mov edx, [esp + 18h] ; 3rd arg
sub esp, 84h          ; allocate local storage
...
add esp, 84h          ; discard locals
pop esi               ; restore registers
pop edi
ret 0Ch               ; discard 3 args
```

Using The Compiler

- ◆ Microsoft Visual C++ switches
 - Enabling FPO
 - /Oy
 - #pragma optimize("y", on)
 - Disabling FPO
 - /Oy-
 - #pragma optimize("y", off)

FPO Is Everywhere...

- ◆ Turning it off in your code won't make it go away
- ◆ Windows NT OS DLLs use FPO
- ◆ Other Vendors use FPO

FPO Record

- ◆ Needed by debugger to walk stack
- ◆ Defined in Platform SDK's winnt.h
- ◆ Frame types:
 - FRAME_FPO, FRAME_NONFPO, FRAME_TRAP, FRAME_TSS
- ◆ FPO_DATA structure fields:
 - (next slide)

FPO_DATA Fields

dword ulOffStart:	1st byte of function code
dword cbProcSize:	# bytes in function
dword cdwLocals:	# bytes in locals/4
word cdwParams:	# bytes in params/4
word cbProlog:8:	# bytes in prolog
word cbRegs:3:	# regs saved
word fHasSEH:1:	TRUE if SEH in func
word fUseBP:1:	TRUE if EBP used
word cbFrame:2:	frame type

FPO Function Using EPB For General Register

push ebx ; save volatile regs

push edi

push esi

push ebp ; ebp should be last saved

...

pop ebp

pop esi

pop edi

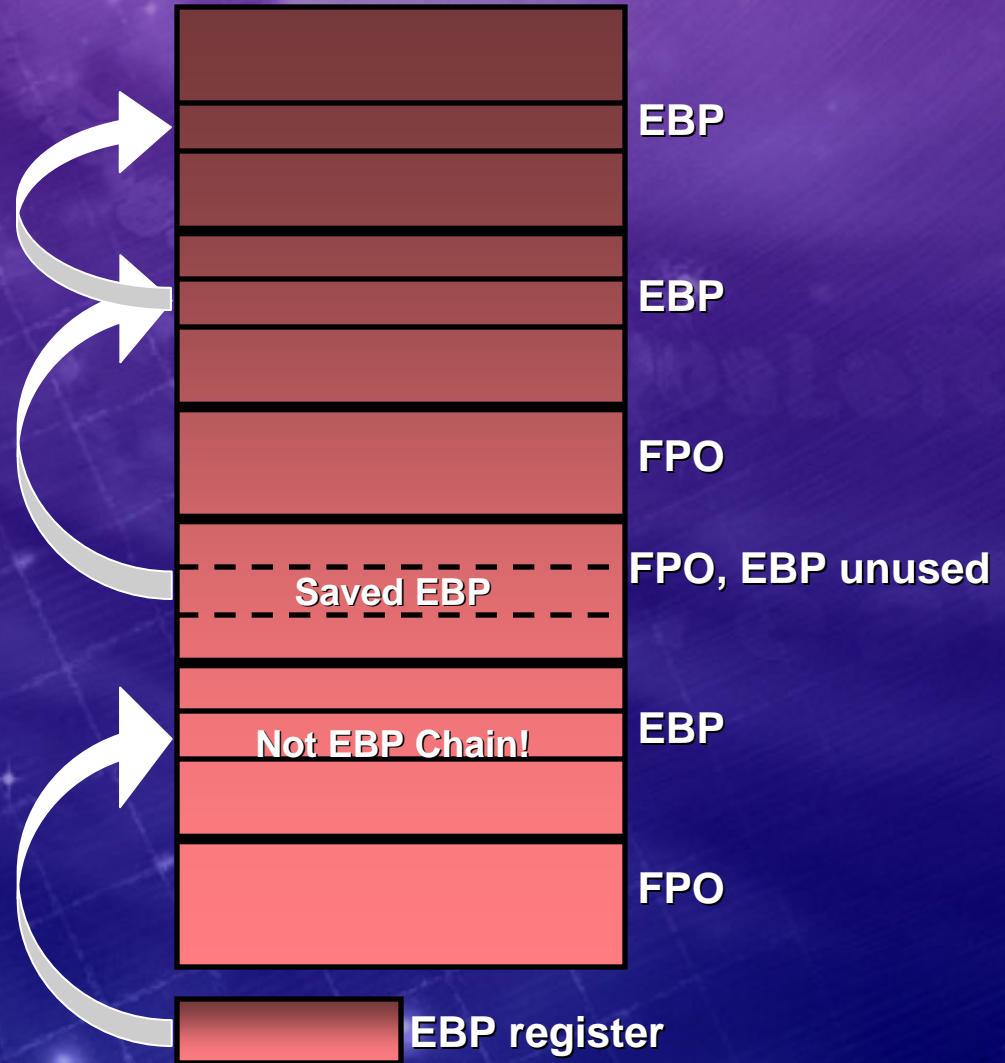
pop ebx

ret 8 ; discard 2 arguments

Mixed Call Stack

Highest address

Lowest address



Demo Two

Damaged stack (handout)

Stack Summary

- ◆ **Frame base always RA - 4**
- ◆ **kv: FPO [params, locals, regs]**
- ◆ **Stack usage can change during function**
- ◆ **Look for discontinuities, completeness**
 - **Does this make sense?**
- ◆ **.COD file, .list command**

FPO Function Using EPB For General Register

```
push ebx    ; save volatile regs
push edi
push esi
push ebp    ; ebp should be last saved
...        ; BUT SOMETIMES IS NOT!
pop ebp
pop esi
pop edi
pop ebx
ret         ; discard 2 arguments
```

Summary

- ◆ Overview of tools
- ◆ Debug environment
- ◆ Common debugging problems
- ◆ Understanding call stacks

Calls To Action

- ◆ **Use retail symbols and user-mode dumps to retain useful diagnostic info from customers in the field**
- ◆ **Understand assembly language**
- ◆ **Learn how call stacks work**
- ◆ **A debugger is an essential development tool, learn to use it well**
- ◆ **More info:**
 - **www.microsoft.com/hwdev/ntdebugging.htm**

Books

- ◆ **Writing Solid Code**
Steve Maguire
Microsoft Press
- ◆ **Advanced Windows**
(has a good chapter on SEH)
Jeffrey Richter
Microsoft Press