Writing Debugger Extensions

Microsoft CPR Summit, 2007

Presented by:

T. Roy

CodeMachine Inc.

www.codemachine.com
Speaker Introduction

- **T.Roy**
  - Masters in Computer Engineering
  - 20 years experience in system software development
  - 10 years international teaching experience
  - Specialization in Windows Driver Development and Debugging
  - Founder of CodeMachine

- **CodeMachine Inc.**
  - Consulting and Training Company
  - Based in Palo Alto, CA, USA
  - Custom Driver Development and Debugging Services
  - Corporate on-site training in Windows Internals, Networking, Device Drivers and Debugging
  - [http://www.codemachine.com](http://www.codemachine.com)
CodeMachine Courses

- Internals Track
  - Windows User Mode Internals
  - Windows Kernel Mode Internals
- Debugging Track
  - Windows Basic Debugging
  - Windows User Mode Debugging
  - Windows Kernel Mode Debugging
- Development Track
  - Windows Network Drivers
  - Windows Kernel Software Drivers
  - Windows Kernel Filter Drivers
  - Windows Driver Model (WDM)
  - Windows Driver Framework (KMDF)
Agenda

- Debugger Extension Environment
- Debugger Extension Interface
- Debugger Extension Implementation
Debugger Extension DLLs

- Any task performed in the debugger that is repetitive can be automated by debugger extension DLLs
- Extension DLLs read data from the target system or process, parse the data and display it in an easily readable format

Extension DLL Architecture
- Win32/Win64 DLLs that run in the debugger’s process context
- Implements special entry points as required by debugger engine
- Uses the debugger engine API and symbol handler API
- Commands supported by extension DLLs are implemented as DLL exports

Extension DLL programming considerations
- Debugger Extensions run “in-proc”
  - Debugger uses exception handler around extensions to recover from AVs
- Perform all target data access through the debugger engine APIs
  - Cannot use standard Win32 APIs to access data on the debug target since this does not work on crash dumps
- Must handle Ctrl-Break to return control to debugger engine
  - Debugger cannot stop extension code while it is executing
Build Environment

- Software Requirements
  - Windows Driver Kit
    - Required for build tools, compiler, linker etc
  - Debugger Package
    - Required for header files, libraries and sample code
    - Perform “custom” installation of debugger and select SDK
  - Debugger Extension documentation is included in debugger help file

- Building Extension DLLs
  - Open a WDK build window (command prompt)
  - In the build window set the following environment variables
    - DBGSDK_INC_PATH=C:\WinDBG\sdk\inc
    - DBGSDK_LIB_PATH=C:\WinDBG\sdk\lib
  - Change to the directory containing the ‘sources’ file for the debugger extension DLL
  - Type ‘build –cW’ at the command prompt
  - Copy the .dll file to %WinDBG%\winext directory
  - Type ‘!DllName.command’ at the debugger prompt to run commands from the extension DLL
  - To unload an extension DLL use the command “.unload DLLName”
Debugger Extension Interface

```
DECLARE_API(command)
{
  .
  .
}
```

```
DbgEng.dll
Debugger Engine

"!extdll.command"

WinDBG.exe/KD.exe/
CDB.exe/NTSD.exe

DllInit()
WinDBGExtensionDllInit()

ExtensionApiVersion()
CheckVersion()

command()

GetFieldOffset()
GetFieldValue()

GetExpressionEx()
ReadMemory()

extdll.dll
```
### Debugger Extension - Template Code

**Header Files**
```
#include <windows.h>
#include <wdbgexts.h>
#include <ntverp.h>
```

**Declare Globals**
```
EXT_API_VERSION ApiVersion = {{VER_PRODUCTVERSION_W >> 8),
                          (VER_PRODUCTVERSION_W & 0xff),
                          EXT_API_VERSION_NUMBER64, 0};
WINDBG_EXTENSION_APIS ExtensionApis;
```

**Initialization Callback**
```
VOID WinDbgExtensionDllInit( PWINDDBG_EXTENSION_APIS lpExtensionApis,
                          USHORT MajorVersion,  USHORT MinorVersion ) {
    ExtensionApis = *lpExtensionApis;
}
```

**Version Query Callback**
```
LPEXT_API_VERSION ExtensionApiVersion( VOID) {
    return &ApiVersion;
}
```

**Version Check Callback**
```
VOID CheckVersion ( VOID) {
}
```

**DLL Entry Point**
```
BOOLEAN DllInit ( HMODULE Module, DWORD Reason, DWORD Reserved ) {
    return TRUE;
}
```
Debugger Extension – Common Tasks

- Querying global variables

```c
if ( GetExpressionEx("nt\!NtBuildNumber", &dwBuildNumber, NULL ) != TRUE ) {
    dprintf("Error reading nt\!BuildNumber\n" );
}
```

- Reading field offsets of a structure

```c
getFieldValue ( IrpPointer, "_IRP", "Tail.Overlay.CurrentStackLocation", pIOSL );
```

- Reading field offsets of a structure

```c
if( GetFieldOffset ( "_EPROCESS", "ActiveThreadLink", &ActiveThreadLinkOffset ) ) {
    dprintf("Error reading _EPROCESS->ActiveThreadLink\n" );
}
```

- Reading Virtual Address Space

```c
if ( ReadMemory ( VirtualAddress, &Buffer, sizeof(Buffer), &BytesReturned) != TRUE ) {
    dprintf("Error reading memory @ %p\n", VirtualAddress );
}
```
Debugger Extension - Example

- Implement a debugger extension which displays a list of threads waiting on a dispatcher object
  - List of threads is maintained in a doubly linked list whose list head is in the dispatcher object
  - Extension DLL needs to traverse this doubly linked list and print information about each thread it finds

- Capability to parse doubly linked lists can be applied to many data structures e.g.
  - List of processes in system
  - List of threads in a process
  - List of IRPs per thread

- Debugger extension (myexts.dll) should support 2 commands
  - !help – will display the list of commands and parameters
  - !waitlist <object> – will display the list of threads waiting on the object

- Debugger extension should use the simple ‘C’ API provides in the header file sdk\inc\wdbgexts.h

- Debugger extension should run unmodified on 64-bit targets
Waiting Threads

- Dispatcher objects contain the DISPATCHER_HEADER structure
  - WaitListHead is the head of the list of threads waiting on an object
- Threads contain a set of KWAIT_BLOCK structures
  - WaitListEntry is used to queue the thread to the object it is waiting for
- `dt` command can walk this chain of waiting threads
  - `dt nt!_KEVENT <ObjectAddress> Header.WaitListHead.Flink`
  - `dt nt!_KWAIT_BLOCK <Flink> -l WaitListEntry.Flink Thread`
## Debugger Extension – File Manifest

### SOURCES

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TARGETNAME</td>
<td>myexts</td>
</tr>
<tr>
<td>TARGETPATH</td>
<td>obj</td>
</tr>
<tr>
<td>TARGETTYPE</td>
<td>DYNLINK</td>
</tr>
<tr>
<td>DLLENTRY</td>
<td>_DllMainCRTStartup</td>
</tr>
<tr>
<td>TARGETLIBS</td>
<td>$(SDK_LIB_PATH)\kernel32.lib</td>
</tr>
<tr>
<td>USE_MSVCRT</td>
<td>1</td>
</tr>
<tr>
<td>UMTPYE</td>
<td>windows</td>
</tr>
<tr>
<td>SOURCES</td>
<td>exts.cpp</td>
</tr>
</tbody>
</table>

### myexts.def

<table>
<thead>
<tr>
<th>EXPRPTS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>help</td>
<td></td>
</tr>
<tr>
<td>waitlist</td>
<td></td>
</tr>
<tr>
<td>CheckVersion</td>
<td></td>
</tr>
<tr>
<td>WinDbgExtensionDllInit</td>
<td></td>
</tr>
<tr>
<td>ExtensionApiVersion</td>
<td></td>
</tr>
</tbody>
</table>

### makefile

```
!INCLUDE $(NTMAKEENV)\makefile.def
```

### myexts.cpp

```
... DECLARE_API ( help ) {
    dprintf (  
      " waitlist - Prints a list of threads waiting on an object\n"
      "    help    - Prints this help\n"
    );
}
...```

---

**Note:** The code snippets are extracted from the Debugger Extension – File Manifest section of the document. The code examples are given in a simplified form for demonstration purposes. Actual implementation details, such as paths and environment variables, might differ based on the specific system setup and requirements.
DECLARE_API( waitlist )
{
    ULONG WaitListHeadOffset, WaitListEntryOffset;
    ULONG64 Object;

    if ( ( Object = GetExpression(args) ) == 0 ) {
        dprintf("Usage: !waitlist <dispatcher-object> \n");
        return;
    }

    if ( GetFieldOffset("nt!_DISPATCHER_HEADER", "WaitListHead", &WaitListHeadOffset ) ) {
        return;
    }

    if ( GetFieldOffset("nt!_KWAIT_BLOCK", "WaitListEntry", &WaitListEntryOffset ) ) {
        return;
    }

    dprintf("Object %p WaitList: \n", Object);
    dprintf("%8s %6s %6s %8s %8s %8s %8s %3s\n",

    ParseLinkedList( Object + WaitListHeadOffset, EntryCallback, WaitListEntryOffset, NULL );
}
myexts.cpp – Double Link List Parsing

```c
BOOLEAN ParseLinkedList (  
    ULONG64 ListHead, PLIST_CALLBACK Function,  
    ULONG FieldOffset, PVOID Context ) {  

    LIST_ENTRY64 ListEntryHead, ListEntryCurrent;  
    ULONG64 ListCurrent;  

    ReadListEntry ( ListHead, &ListEntryHead );  

    for (ListCurrent = ListEntryHead.Flink ;  
         ListCurrent != ListHead ;  
         ListCurrent = ListEntryCurrent.Flink ) {  

        ReadListEntry ( ListCurrent, &ListEntryCurrent );  

        if ( ! Function ( ListCurrent - FieldOffset, Context ) ) {  
            return FALSE;  
        }  
        if ( CheckControlC () ) {  
            return FALSE;  
        }  
    }  
  
  return TRUE;
}
```
myexts.cpp – waitlist Callback

```c
BOOLEAN EntryCallback ( ULONG64 WaitBlock, PVOID Context )
{
    ULONG64 Thread, Pid, Tid;
    ULONG WaitTime, KernelApcDisable, SpecialApcDisable, CombinedApcDisable;
    UCHAR IdealProcessor;

    GetFieldValue(WaitBlock, "nt!_KWAIT_BLOCK", "Thread", Thread);
    GetFieldValue(Thread, "nt!_ETHREAD", "Cid.UniqueProcess", Pid);
    GetFieldValue(Thread, "nt!_ETHREAD", "Cid.UniqueThread", Tid);
    GetFieldValue(Thread, "nt!_ETHREAD", "Tcb.WaitTime", WaitTime);
    GetFieldValue(Thread, "nt!_ETHREAD", "Tcb.KernelApcDisable", KernelApcDisable);
    GetFieldValue(Thread, "nt!_ETHREAD", "Tcb.SpecialApcDisable", SpecialApcDisable);
    GetFieldValue(Thread, "nt!_ETHREAD", "Tcb.CombinedApcDisable", CombinedApcDisable);
    GetFieldValue(Thread, "nt!_ETHREAD", "Tcb.IdealProcessor", IdealProcessor);

    dprintf ( "%8p %6I64x %6I64x %8u %8s %8s %8s %3u\n",
             Thread, Pid, Tid, WaitTime,
             KernelApcDisable ? "DISABLED" : "ENABLED",
             SpecialApcDisable ? "DISABLED" : "ENABLED",
             CombinedApcDisable ? "DISABLED" : "ENABLED",
             IdealProcessor );

    return TRUE;
}
```