# **Debugging Windows Applications with IDA WinDbg Plugin**

Copyright 2011 Hex-Rays SA

## Quick overview:

The Windbg debugger plugin is an IDA Pro debugger plugin that uses Microsoft's debugging engine (dbgeng) that is used by Windbg, Cdb or Kd.

To get started, you need to install the latest Debugging Tools from Microsoft website: <u>https://msdn.microsoft.com/en-us/windows/hardware/hh852365</u>

or from the Windows SDK / DDK package.

Please make sure you should install the **x86** version of the debugging tools which is used by both IDA Pro and IDA Pro 64. The x64 version will <u>NOT</u> work.

After installing the debugging tools, make sure you select « Debugger / Switch Debugger » and select the WinDbg debugger.

Also make sure you specify the correct settings in the "Debugger specific options" dialog:

Tonfiguration	×
Debugging mode:	
C Kernel mode debugging	
C Non Invasive user-mode process attach	
C Kernel mode debugging with reconnect an	d initial break
Output flags:	
Normal output	
Fror output	
🖵 Warnings	
🗂 Additional output	
Prompt output	
Register dump before prompt	
$\square$ Warnings specific to extension operation	
Debug output from the target	
🗖 Debug input expected by the target	
🗖 Symbol messages	
O <u>K</u> Cancel	]

- User mode: Select this mode for user mode application debugging (default mode)
- Kernel mode: Select this mode to attach to a live kernel.
- Non Invasive debugging: Select this mode to attach to a process <u>non-invasively</u>
- **Output flags:** These flags tell the debugging engine which kind of output messages to display and which to omit
- Kernel mode debugging with reconnect and initial break: Select this option when debugging a kernel and when the connection string contains 'reconnect'. This option will assure that the debugger breaks as soon as possible after a reconnect.

To make these settings permanent, please edit the IDA\cfg\dbg\_windbg.cfg file.

**To specify the debugging tools folde**r you may add to the PATH environment variable the location of Windbg.exe or edit %IDA%\cfg\ida.cfg and change the value of the DBGTOOLS key.

After the debugger is properly configured, edit the process options and leave the connection string value empty because we intend to debug a local user-mode application.

Debug app	lication setup: windbg	×
<u>Application</u>	c:\temp\ar.exe	▼
Input file	c:\temp\ar.exe	<b>.</b>
<u>D</u> irectory	c:\temp	<b>•</b>
<u>P</u> arameters		•
<u>C</u> onnection	string	
☐ <u>S</u> ave f	network settings as default           OK         Cancel         Help	

Now hit F9 to start debugging:

Martin C:\Temp\ar.exe
<u>File Edit J</u> ump Searc <u>h</u> <u>V</u> iew Debugger <u>O</u> ptions <u>W</u> indows Help
📔 🕨 🕅 🖬 Windbg debugger 🔄 🛛 🛱 🛱 🛱 🏹 💭 🛱 🖺 🖺 📲 🗶 🛛 🚥 ൽ 🝻 🛛 🔛 👫 名
🗙 🔷 Debugger 🗙 🕅 Structures 🗙 En Enums
IDA View-EIP
*.text:004010DF db 0
.text:004010E0
tout. OR ALGER . Attributes. library function poreturn
tout 00001000 , ALLEIDULES. IIDEATY FUNCTION NOTECOTI
.Lext:004010E0 public start
EDXLTEXT:004010E0 Start proc near
EIP Lext:004010E0 jmp short loc_4010F2
.text:004010E0 ;
* .text:004010E2 db 66h, 62h, 3Ah
text:004010E5 aCHook db 'C++HOOK'
.text:004010EC :
* text:004010EC non
* text:004010ED imp pear ntr upk 80E180

The Windbg plugin is very similar to IDA Pro's Win32 debugger plugin, nonetheless by using the former, one can benefit from the command line facilities and the extensions that ship with the debugging tools.

For example, one can type "!chain" to see the registered Windbg extensions:

🗒 Output window

```
WINDBG>!chain
Extension DLL search Path:
    c:\dbg\WINXP;c:\dbg\winext;c:\dbg\winext\arcade;c:\dbg\pri;c:\dbg;c:\dbg\winext\arcade;Z:`
Extension DLL chain:
    dbghelp: image 6.11.0001.404, API 6.1.6, built Thu Feb 26 02:55:30 2009
        [path: c:\dbg\dbghelp.dll]
    ext: image 6.11.0001.404, API 1.0.0, built Thu Feb 26 02:55:30 2009
        [path: c:\dbg\winext\ext.dll]
    wow64exts: image 6.1.7015.0, API 1.0.0, built Thu Feb 26 02:55:24 2009
        [path: c:\dbg\WINXP\wow64exts.dll]
    exts: image 6.11.0001.404, API 1.0.0, built Thu Feb 26 02:55:24 2009
        [path: c:\dbg\WINXP\exts.dll]
    uext: image 6.11.0001.404, API 1.0.0, built Thu Feb 26 02:55:26 2009
        [path: c:\dbg\WINXP\exts.dll]
    uext: image 6.1.7015.0, API 1.0.0, built Thu Feb 26 02:55:26 2009
        [path: c:\dbg\WINXP\exts.dll]
    uext: image 6.1.7015.0, API 1.0.0, built Thu Feb 26 02:55:26 2009
        [path: c:\dbg\WINXP\exts.dll]
    uext: image 6.1.7015.0, API 1.0.0, built Thu Feb 26 02:55:26 2009
        [path: c:\dbg\WINXP\exts.dll]
    MINDBG
]
```

"!gle" is another command to get the last error value of a given Win32 API call.

🗮 Output window	_ 🗆 🗵
WINDBG>!gle	
LastStatusValue: (NTSTATUS) 0 - STATUS_WAIT_0	•
<u> </u>	
WINDBG	

Another benefit of using the Windbg debugger plugin is the use of symbolic information.

Normally, if the debugging symbols path is not set, then the module window will only show the exported names. For example kernel32.dll displays 1359 names:

Module: kernel32.dll		
<u>E</u> dit <u>S</u> earch		
Name	Address	▲
D GetStartupInfoA	76060DF0	
D CreateProcessW	7606102D	
D CreateProcessA	76061062	
D Sleep	760610EF	
D GetTickCount	760610FC	
D ReleaseMutex	7606110E	
D WaitForSingleObject	76061126	
D WaitForSingleObjectEx	76061141	
D SetLastError	76061199	
D GetLastError	760611B0	
D TIsGetValue	760611C0	
D GetCurrentProcessId	760611D8	
D SleepEx	760611EA	
D GetProcAddress	76061202	
D GetModuleHandleA	76061225	
D FIsGetValue	7606123D	
D ProcessIdToSessionId	7606124A	<u> </u>
Line 9 of 1359		11.

Let us configure a symbol source by adding this environment variable before running IDA:

#### set \_NT\_SYMBOL\_PATH=srv\*C:\Temp\pdb\*http://msdl.microsoft.com/download/symbols

It is also possible to set the symbol path directly while debugging:

🗮 Output window
WINDBG>.sympath srv*d:\Temp\pdb*http://msdl.microsoft.com/download/symbols Symbol search path is: srv*d:\Temp\pdb*http://msdl.microsoft.com/download/symbols Expanded Symbol search path is: srv*d:\temp\pdb*http://msdl.microsoft.com/download/symbols WINDBG>.reload /f Reloading current modules
WINDBG

and then typing ".reload /f" to reload the symbols.

Now we try again and notice that more symbol names are retrieved from kernel32.dll:

Module: kernel32.dll		
<u>E</u> dit <u>S</u> earch		
Name	Address	▲
C_Pmap_advpack	760A64D0	
D c_Pmap_advapi32	760A64D8	
D c_Omap_activeds	760A64E0	
<b>D</b> c_Pmap_actionqueue	760A64E8	
D c_Omap_aclui	760A64F0	
C_PmapEntries_xmllite	760A64F8	
C_OmapEntries_winspool	760A97B8	
C_PmapEntries_winspool	760A97C8	
D c_PmapEntries_wer	760ACB60	
C_PmapEntries_version	760AF780	_
C_PmapEntries_usp10	760B05D0	
C_PmapEntries_user32	760B0A80	
D c_PmapEntries_tapi32	760B2AE0	
C_PmapEntries_synceng	760B2C38	
D c_PmapEntries_sqmapi	760B3C20	
C_OmapEntries_shlwapi	760B5150	
C_OmapEntries_shell32	760B52E0	
C_PmapEntries_shell32	760B5768	
C_PmapEntries_query	760BAD 98	
D c_PmapEntries_psapi	760BAE60	•
Line 1 of 5818		11.

Now we have 5818 symbols instead!

It is also possible to use the "x" command to quickly search for symbols:

🗮 Output window	
<pre>WINDBG&gt;x *!*continue* 75da406f RPCRT4!LRPC_CASSOCIATION::ContinueAlpcConnectAfterSidNormalize = <no information="" type=""> 760604bc kernel32!_impDbgUiContinue = <no information="" type=""> 7608677 kernel32!ContinueDebugEvent = <no information="" type=""> 76104176 kernel32!DbgUiContinue = <no information="" type=""> 77c5754d ntdl!RtlAddVectoredContinueHandler = <no information="" type=""> 77ca576a ntdl!RtlRemoveVectoredContinueHandler = <no information="" type=""> 77c1fe90 ntdl!RtlRemoveVectoredContinueHandler = <no information="" type=""> 77c1fe90 ntdl!NtContinue = <no information="" type=""> 77c1fe90 ntdl!NtContinue = <no information="" type=""> 77c1fe90 ntdl!RtlContinue = <no information="" type=""> 77c5fb3d ntdl!RtlContinue = <no information="" type=""> 77c5fb3d ntdl!RtlCollvectoredContinueHandlers = <no information="" type=""> 77c20920 ntdl!NtDebugContinue = <no information="" type=""> 77c20920 ntdl!NtDebugContinue = <no information="" type=""> 77c20920 ntdl!NtDebugContinue = <no information="" type=""></no></no></no></no></no></no></no></no></no></no></no></no></no></no></no></pre>	
WINDBG x *!*continue*	

(Looking for any symbol in any module that contains the word "continue")

## Debugging a remote process:

We have seen how to debug a local user mode program, now let us see how to debug a remote process. First let us assume that "pcA" is the target machine (where we will run the debugger server and the debugged program) and "pcB" is the machine where IDA Pro and the debugging tools are installed.

To start a remote process:

- On "pcA", type:
  - dbgsrv -t tcp:port=5000

(change the port number as needed)

- On "pcB", setup IDA Pro and Windbg debugger plugin:
  - "Application/Input file": these should contain a path to the debuggee residing in "pcA"
  - Connection string: tcp:port=5000,server=pcA

Now run the program and debug it remotely.

To attach to a remote process, use the same steps to setup "pcA" and use the same connection string when attaching to the process.

More about connection strings and different protocols (other than TCP/IP) can be found in "debugger.chm" in the debugging tools folder.

## Debugging the kernel with VMWare:

We will now demonstrate how to debug the kernel through a virtual machine.

In this example we will be using VMWare 6.5 and Windows XP SP3.

#### Configuring the virtual machine:

Run the VM and then edit "c:\boot.ini" file and add one more entry (see in bold): [operating systems] multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP Professional" /noexecute=optin /fastdetect multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Local debug" /noexecute=optin /fastdetect /debug /debugport=com1 /baudrate=115200

For MS Windows Vista please see: http://msdn.microsoft.com/en-us/library/ms791527.aspxp

Actually the last line is just a copy of the first line but we added the "/debug" switch and some configuration values.

Now shutdown the virtual machine and edit its hardware settings and add a new serial port with option "use named pipes":

Add Hardware Wizard	x
<b>Specify Socket</b> Which socket should this serial port connect to?	
Named pipe       Named pipe       \\.\pipe\com_1       This end is the server.       The other end is a virtual machine.	
⊂Device status ⊂ Connect at power <u>o</u> n	
< <u>B</u> ack Finish Cancel	

Press "Finish" and start the VM. At the boot prompt, select "Local debug" from the boot menu:



Configuring Windbg debugger plugin:

Now run IDA Pro and select Debugger / Attach / Windbg



Then configure it to use "Kernel mode" debugging and use the following connection string: com:port=\\.\pipe\com\_1,baud=115200,pipe

It is possible to use the 'reconnect' keyword in the connection string: com:port=\\.\pipe\com\_1,baud=115200,pipe,reconnect

Also make	sure the a	opropriate	option is	selected	from the	e debugger	specific of	options.
		r r	- F				- <b>r</b>	· F · · · · ·

Debug application setup: windbg	Debugger setup       X         Events       Stop on debugging start         Stop on process entry point       Stop on thread start/exit         Stop on library load/unload       Stop on debugging message
OK     Cancel     Help       Configuration     X       Debugging tools folder     c:\dbg\	Log Segment modifications Thread start/exit Library load/unload Breakpoint Debugging message
<ul> <li>User mode</li> <li>Kernel mode debugging</li> <li>Non Invasive user-mode process attach</li> <li>Kernel mode debugging with reconnect and initial break</li> </ul>	Options  Reconstruct the stack  Show debugger breakpoint instructions  Autoload PDB files  Set as just-in-time debugger <u>3</u>
Output flags:	Edit exceptions     Reload exceptions     Set specific options       OK     Cancel     Help

Please note that the connection string (in step 1) refers to the named pipe we set up in the previous steps.

Finally, press OK to attach and start debugging.

n Choose p	rocess to attach to	
ID	Name	
0	<kernel></kernel>	
	<b>-</b>	
ОК	Cancel Help Sear	ch
Line 1 of 1		/

When IDA attaches successfully, it will display something like this:

	nt:8052A854 nt:8052A854	nt_RtlpBreakWithStat	usInstructio	n: ; Trap to Debugger
**	nt:8052A854	int		
•	nt:8052A855	retn	4	
	nt:8052A855			

If you do not see named labels then try checking your debugging symbols settings.

Note: In kernel mode IDA Pro will display one entry in the threads window for each processor.

For example a two processor configuration yields:

Device	Summary
I Memory	256 MB
Processors	2
VMWare configi	uration

🗐 Thread	🗒 Threads			- 🗆 ×	
Decimal	Hex	State			
圖1	1	Ready			
B) 2	2	Ready			

Threads in IDA

This screenshot shows how we are debugging the kernel and changing the disassembly listing (renaming stack variables, or using structure offsets):

```
nt:804F1808 ; int stdcall nt IoDeleteDevice(int pDeviceObject)
nt:804F1808 nt IoDeleteDevice proc near
nt:804F1808
nt:804F1808 pDeviceObject
                             = dword ptr 8
nt:804F1808
nt:804F1808
                             mov
                                      edi, edi
nt:804F180A
                             push
                                      ebp
nt:804F180B
                             mov
                                      ebp, esp
nt:804F180D
                                      nt_IopVerifierOn, 0
                             CMD
nt:804F1814
                             push
                                      esi
                     L
nt:804F1815
                             mov
                                      esi, [ebp+pDeviceObject]
nt:804F1818
                             jz 🛛
                                      short loc 804F1820
nt:804F181A
                             push
                                      esi
                                      near ptr nt IovDeleteDevice
nt:804F181B
                             call
nt:804F1820
                                                       ; CODE XREF: nt IoDeleteDevice+10<sup>†</sup>j
nt:804F1820 loc 804F1820:
                             test
                                      byte ptr [esi+(DEVICE_OBJECT.Flags+1)], 8
nt:804F1820
nt:804F1824
                             jz -
                                      short loc 804F182C
nt:804F1826
                             push
                                      esi
                                      near ptr nt_IoUnr_gisterShutdownNotification
nt:804F1827
                             call
nt:804F182C
                                                       ; CODE XREF: nt IoDeleteDevice+1C<sup>†</sup>j
nt:804F182C loc 804F182C:
nt:804F182C
                             push
                                      edi
                                      edi, [esi+DEVICE_OBJECT.Timer]
nt:804F182D
                             mov
nt:804F1830
                             test
                                      edi, edi
nt:804F1832
                                      short loc 804F1842
                             jz -
nt:804F1834
                             push
                                      edi
                                      near ptr nt IopRemoveTimerFromTimerList
nt:804F1835
                             call
nt:804F183A
                             push
                                      0
nt:804F183C
                             push
                                      edi
                                      near ptr nt ExFreePoolWithTag
nt:804F183D
                             call
nt:804F1842
```

At the end you can detach from the kernel and resume it or detach from the kernel and keep it suspended.

To detach and resume, simply select the "Debugger / Detach", however to detach and keep the kernel suspended select "Debugger / Terminate Process".

# Debugging the kernel through kdsrv.exe

In some cases, when debugging a 64bit kernel using a 1394 cable then 64bit drivers are needed, thus dbgeng (32bits) will not work. To workaround this problem we need to run the kernel debugger server from the x64 debugging tools folder and connect to it:

- Go to "Debugging Tools (x64)" installation
- Run kdsrv.exe (change the port number/transport appropriately):
   kdsrv -t tcp:port=6000
- Now run ida64 and specify the following connection string (change the transport value appropriately):
  - o kdsrv:server=@{tcp:port=6000,server=127.0.0.1},trans=@{com:port=\\.\pipe\com\_3,bau d=115200,pipe}