Debugging the Windows Kernel with VMWare and IDA WinDbg Plugin

Table of Contents

Last updated on September 02, 2020 — v0.1

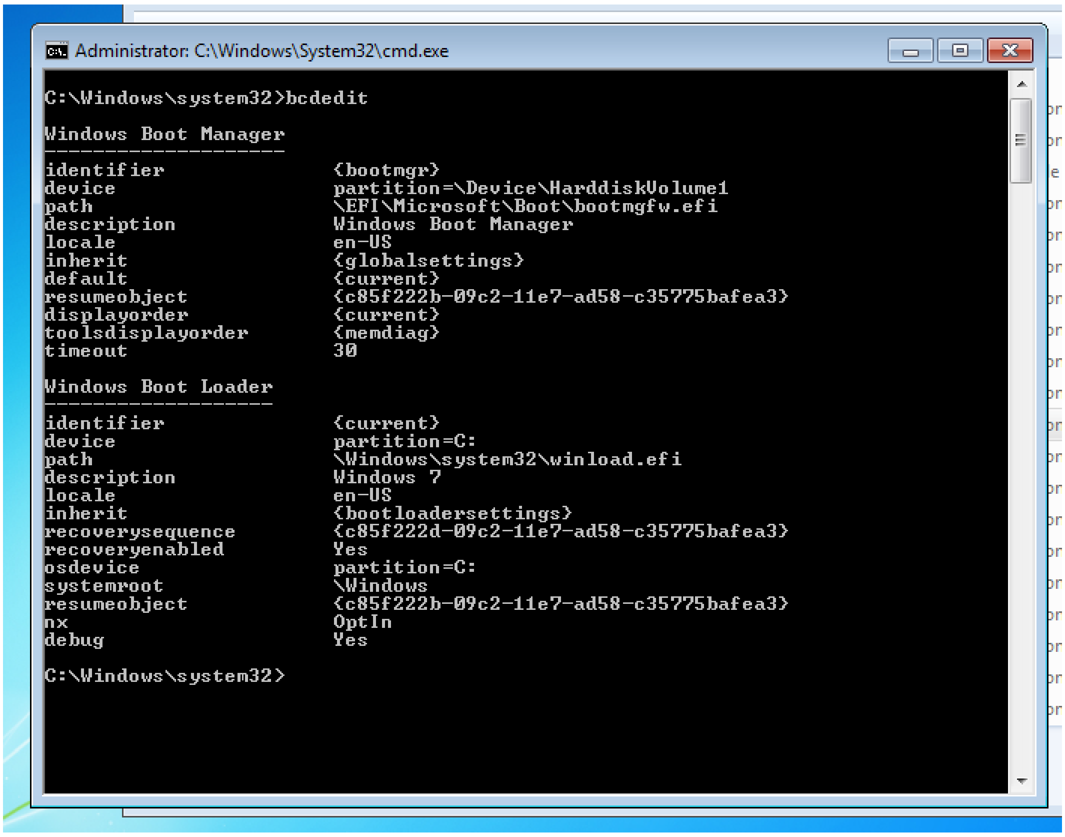
We will now demonstrate how to debug the kernel through a virtual machine.

In this example we will be using VMware Workstation 15 Player and Windows 7.

It is highly recommended to read the article Windows driver debugging with WinDbg and VMWare

1. Configuring the virtual machine

Run the VM and use the `bcedit` command[1] to configure the boot menu as stated in the article.



Administrator: C:\Windows\System32\cmd.exe

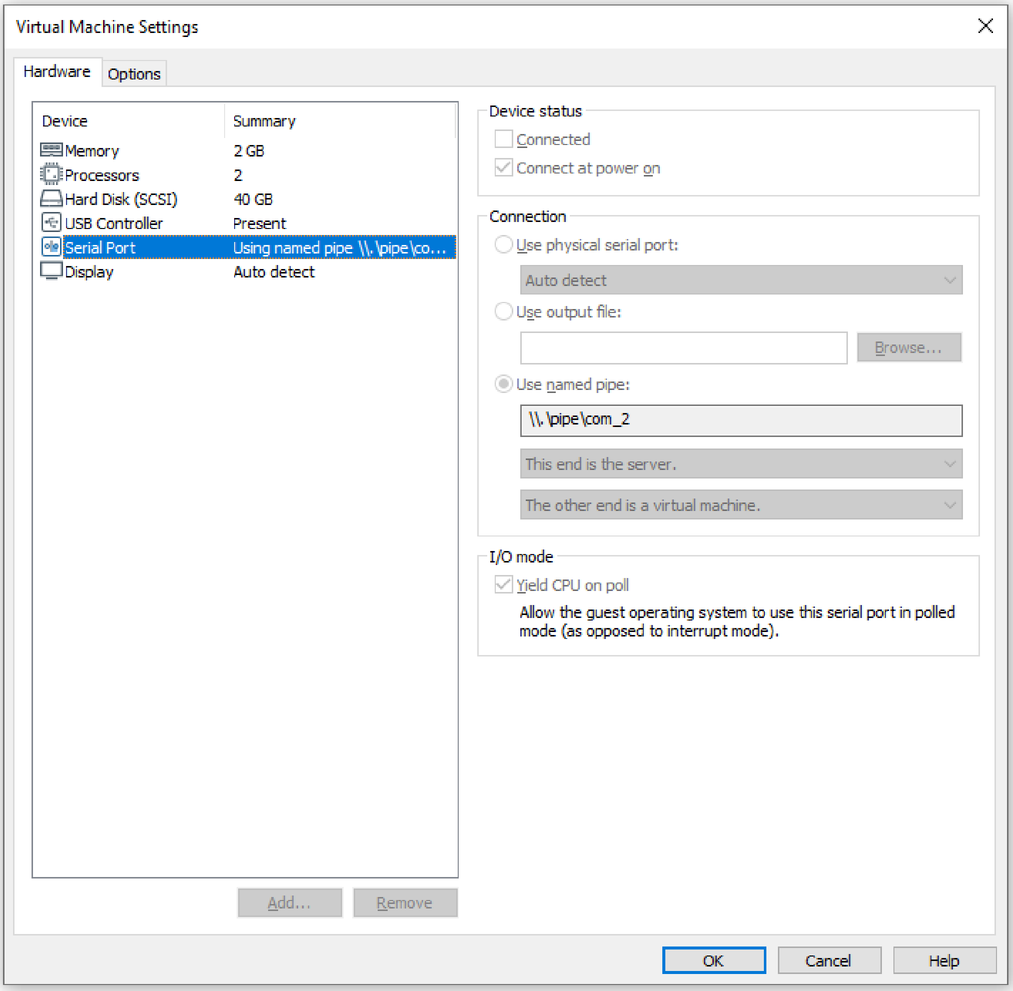
```
C:\Windows\system32>bcdedit

Windows Boot Manager
--------------------
identifier              {bootmgr}
device                  partition=\Device\HarddiskVolume1
path                    \EFI\Microsoft\Boot\bootmgfw.efi
description             Windows Boot Manager
locale                  en-US
inherit                 {globalsettings}
default                 {current}
resumeobject            {c85f222b-09c2-11e7-ad58-c35775bafea3}
displayorder            {current}
toolsdisplayorder       {memdiag}
timeout                 30

Windows Boot Loader
-------------------
identifier              {current}
device                  partition=C:
path                    \Windows\system32\winload.efi
description             Windows 7
locale                  en-US
inherit                 {bootloadersettings}
recoverysequence        {c85f222d-09c2-11e7-ad58-c35775bafea3}
recoveryenabled         Yes
osdevice                partition=C:
systemroot              \Windows
resumeobject            {c85f222b-09c2-11e7-ad58-c35775bafea3}
nx                      OptIn
debug                   Yes

C:\Windows\system32>
```

Edit the VM hardware settings and add a new serial port with option `use named pipe`:



Device	Summary
Memory	2 GB
Processors	2
Hard Disk (SCSI)	40 GB
USB Controller	Present
Serial Port	Using named pipe \\.\pipe\co...
Display	Auto detect

Device status

☐ Connected

☑ Connect at power on

Connection

○ Use physical serial port:

Auto detect

○ Use output file:

Browse...

● Use named pipe:

\\.\pipe\com_2

This end is the server.

The other end is a virtual machine.

I/O mode

☑ Yield CPU on poll

Allow the guest operating system to use this serial port in polled mode (as opposed to interrupt mode).

Restart the VM to debug. At the boot prompt, select the menu item containing `[debugger enabled]` from the boot menu.

2. Configuring Windbg debugger plugin

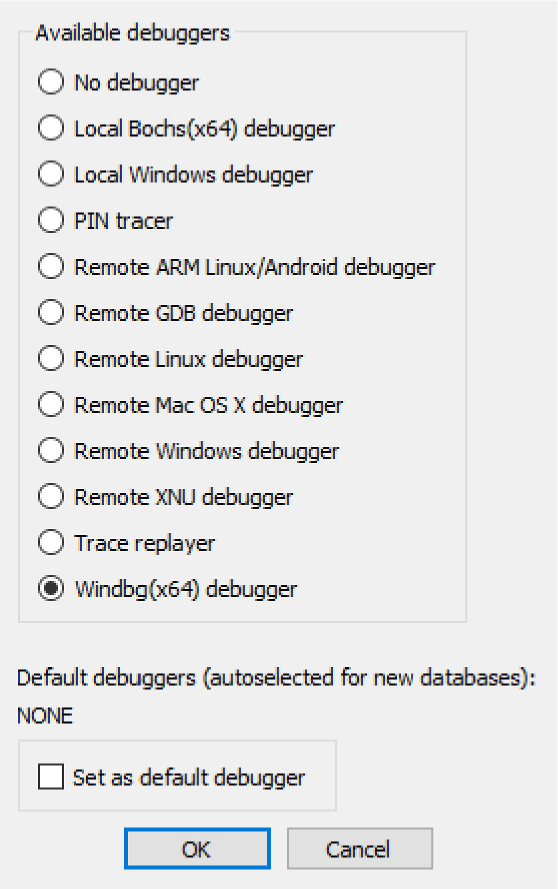
The connection string `com:port=\\.\pipe\com_2,baud=115200,pipe,reconnect` for Windbg plugin should refer to the named pipe we set up in the previous steps.

2.1. Starting the debugger step by step

Start IDA Pro with an empty database:[2]

```
> ida64 -t sample.i64
```

Select the Windbg debugger using "Debugger > Select debugger":



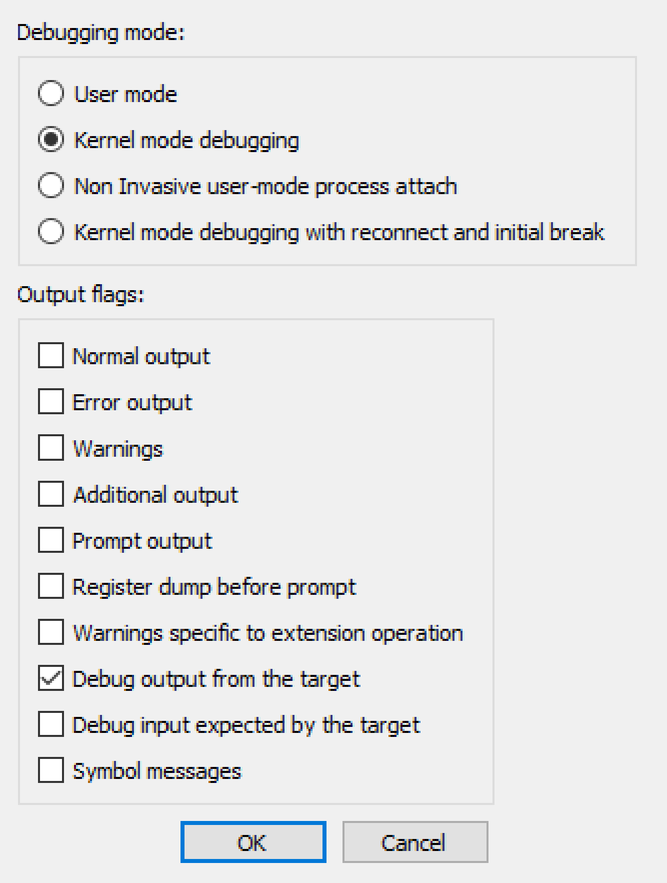
Available debuggers

- ○ No debugger
- ○ Local Bochs(x64) debugger
- ○ Local Windows debugger
- ○ PIN tracer
- ○ Remote ARM Linux/Android debugger
- ○ Remote GDB debugger
- ○ Remote Linux debugger
- ○ Remote Mac OS X debugger
- ○ Remote Windows debugger
- ○ Remote XNU debugger
- ○ Trace replayer
- ● Windbg(x64) debugger

Default debuggers (autoselected for new databases):
NONE

□ Set as default debugger

Then configure it to use “Kernel mode debugging” debugging in the “Debugger specific options” dialog:



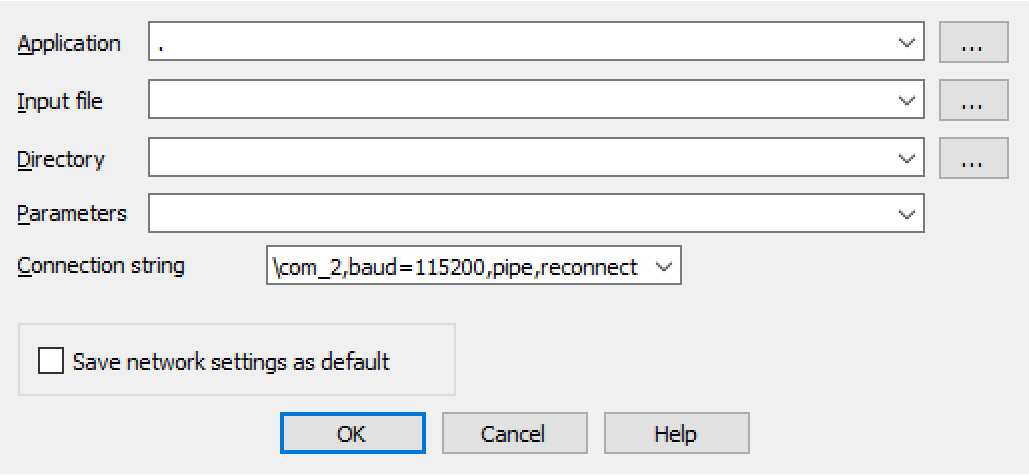
Debugging mode:

- ○ User mode
- ● Kernel mode debugging
- ○ Non Invasive user-mode process attach
- ○ Kernel mode debugging with reconnect and initial break

Output flags:

- □ Normal output
- □ Error output
- □ Warnings
- □ Additional output
- □ Prompt output
- □ Register dump before prompt
- □ Warnings specific to extension operation
- ☑ Debug output from the target
- □ Debug input expected by the target
- □ Symbol messages

After the debugger is properly configured, edit the process options and set the connection string:[3]



Application: .

Input file:

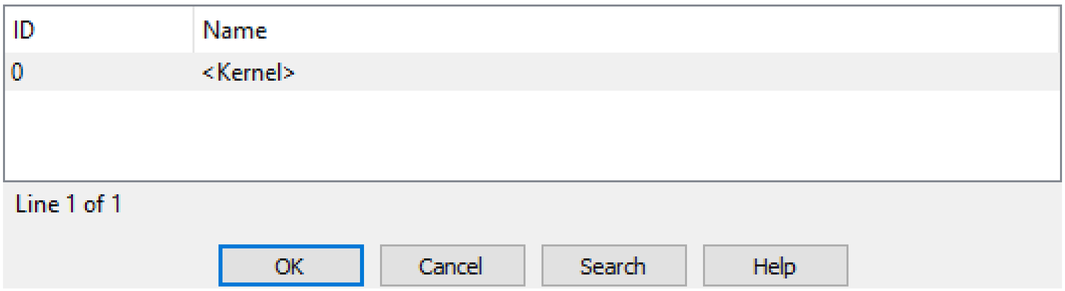
Directory:

Parameters:

Connection string: \com_2,baud=115200,pipe,reconnect

☐ Save network settings as default

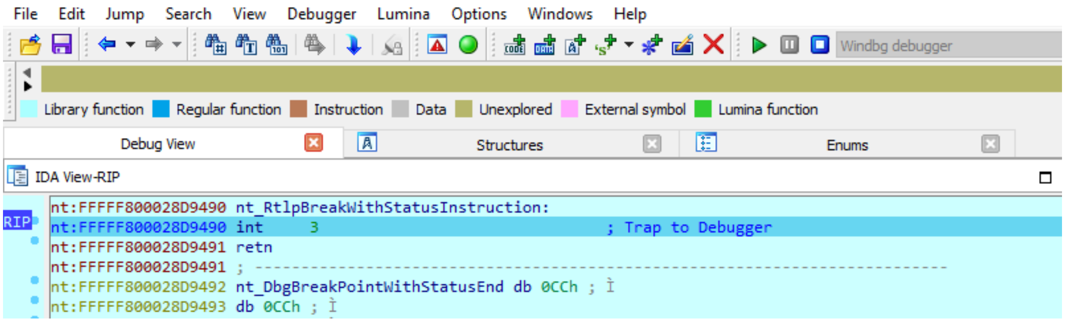
Finally, start debugging using "Debugger > Attach to process":



ID	Name
0	<Kernel>

Line 1 of 1

IDA Pro may display a wait box "Refreshing module list" for some time. Then it will display something like this:



```
nt:FFFFF800028D9490 nt_RtlpBreakWithStatusInstruction:
nt:FFFFF800028D9490 int     3                                    ; Trap to Debugger
nt:FFFFF800028D9491 retn
nt:FFFFF800028D9491 ; ---------------------------------------------------------------------
nt:FFFFF800028D9492 nt_DbgBreakPointWithStatusEnd db 0CCh ; Ì
nt:FFFFF800028D9493 db 0CCh ; Ì
```

2.2. Starting the debugger using a command line option

The simplest way to start WinDbg Plugin is to run IDA Pro with the following option:

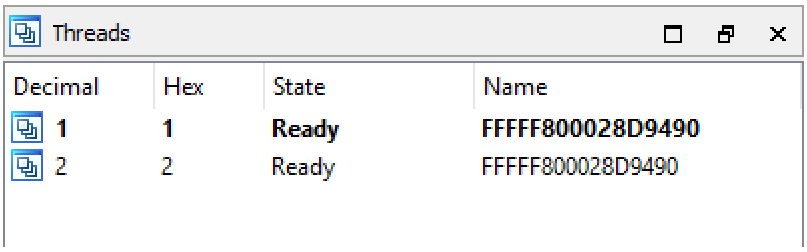
```
> ida64 -rwindbg{MODE=1}@com:port=\\.\pipe\com_2,baud=115200,pipe,reconnect+0 sample.i64
```

- `{MODE=1}` means "Kernel mode"
- `+0` means the "<Kernel>" process

3. Debugging

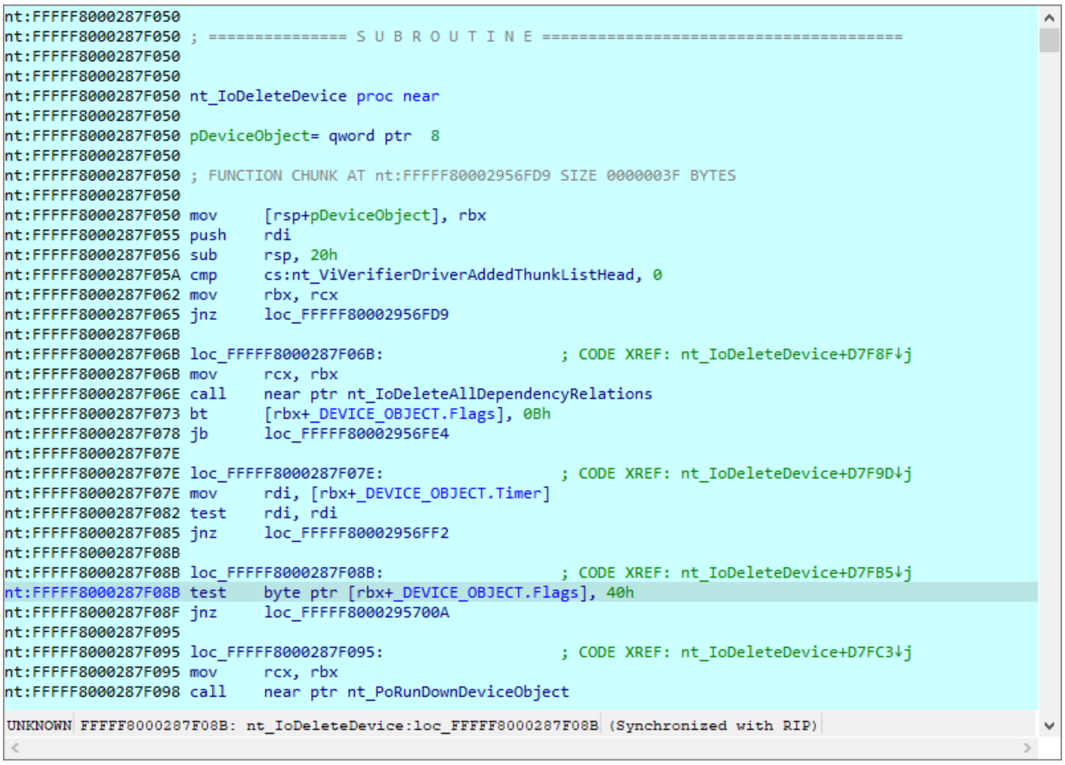
In kernel mode IDA Pro will display one entry in the threads window for each processor.

For example a two processor configuration[4] yields:



Decimal	Hex	State	Name
1	**1**	**Ready**	**FFFFF800028D9490**
2	2	Ready	FFFFF800028D9490

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



```
nt:FFFFF8000287F050
nt:FFFFF8000287F050 ; =============== S U B R O U T I N E =======================================
nt:FFFFF8000287F050
nt:FFFFF8000287F050
nt:FFFFF8000287F050 nt_IoDeleteDevice proc near
nt:FFFFF8000287F050
nt:FFFFF8000287F050 pDeviceObject= qword ptr  8
nt:FFFFF8000287F050
nt:FFFFF8000287F050 ; FUNCTION CHUNK AT nt:FFFFF80002956FD9 SIZE 0000003F BYTES
nt:FFFFF8000287F050
nt:FFFFF8000287F050 mov     [rsp+pDeviceObject], rbx
nt:FFFFF8000287F055 push    rdi
nt:FFFFF8000287F056 sub     rsp, 20h
nt:FFFFF8000287F05A cmp     cs:nt_ViVerifierDriverAddedThunkListHead, 0
nt:FFFFF8000287F062 mov     rbx, rcx
nt:FFFFF8000287F065 jnz     loc_FFFFF80002956FD9
nt:FFFFF8000287F06B
nt:FFFFF8000287F06B loc_FFFFF8000287F06B:                   ; CODE XREF: nt_IoDeleteDevice+D7F8F↓j
nt:FFFFF8000287F06B mov     rcx, rbx
nt:FFFFF8000287F06E call    near ptr nt_IoDeleteAllDependencyRelations
nt:FFFFF8000287F073 bt      [rbx+_DEVICE_OBJECT.Flags], 0Bh
nt:FFFFF8000287F078 jb      loc_FFFFF80002956FE4
nt:FFFFF8000287F07E
nt:FFFFF8000287F07E loc_FFFFF8000287F07E:                   ; CODE XREF: nt_IoDeleteDevice+D7F9D↓j
nt:FFFFF8000287F07E mov     rdi, [rbx+_DEVICE_OBJECT.Timer]
nt:FFFFF8000287F082 test    rdi, rdi
nt:FFFFF8000287F085 jnz     loc_FFFFF80002956FF2
nt:FFFFF8000287F08B
nt:FFFFF8000287F08B loc_FFFFF8000287F08B:                   ; CODE XREF: nt_IoDeleteDevice+D7FB5↓j
nt:FFFFF8000287F08B test    byte ptr [rbx+_DEVICE_OBJECT.Flags], 40h
nt:FFFFF8000287F08F jnz     loc_FFFFF8000295700A
nt:FFFFF8000287F095
nt:FFFFF8000287F095 loc_FFFFF8000287F095:                   ; CODE XREF: nt_IoDeleteDevice+D7FC3↓j
nt:FFFFF8000287F095 mov     rcx, rbx
nt:FFFFF8000287F098 call    near ptr nt_PoRunDownDeviceObject
```

```
UNKNOWN FFFFF8000287F08B: nt_IoDeleteDevice:loc_FFFFF8000287F08B (Synchronized with RIP)
```

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 from process”, however to detach and keep the kernel suspended select “Debugger > Terminate Process”.

4. 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):
  - `kdsrv:server=@{tcp:port=6000,server=127.0.0.1},trans=@{com:port=\\.\pipe\com_3,baud=115200,pipe}`

[1] or edit the `c:\boot.ini` file for Windows XP

[2] you can use any name of your choice instead of `sample`

[3] specify any non-empty string for "Application" because IDA Pro does not allow an empty string in this field

[4] as was specified in the VM hardware settings