← Back to Posts

17 Jul

Windows Kernel Debugging & Exploitation Part1 – Setting up the lab

By voidsec

Recently I was thrilled with the opportunity to build a PoC for ms-14-066 vulnerability aka “winshock” (CVE-2014-6321). While that will be material for another blog post, in order to debug the vulnerability, I had to set up a lab with windows kernel mode debugging enabled. So, without any further ado, here my setup and the steps used in order to enable **Windows Kernel Debug**.

Setup

Host system: Windows 10 with VMware Workstation 15.1.0 (build-13591040)

Guest systems:

- Windows 7 x86 ultimate sp1 (debugger)
- Windows 7 x86 ultimate sp1 (debuggee, using UART as debugging medium)

> Debuggee: a process or application upon which a debugger acts; the process that is being debugged.

Setting up the VMs

Install, as usual, one Windows 7 x86 in a newly created VM (also install VMware Tools).

Now that OS in the first machine has been installed, make a clone of it (Right Click on VM -> Manage -> Clone) I’ve used the “Full clone” option but it should also works fine with a “Linked Clone”.

Now you should have two identical machines.

Setting up the Serial Port

VMware machines already have a serial port (used for the printer), since we do not need it, we will remove the hardware from the configuration. In order to do so:

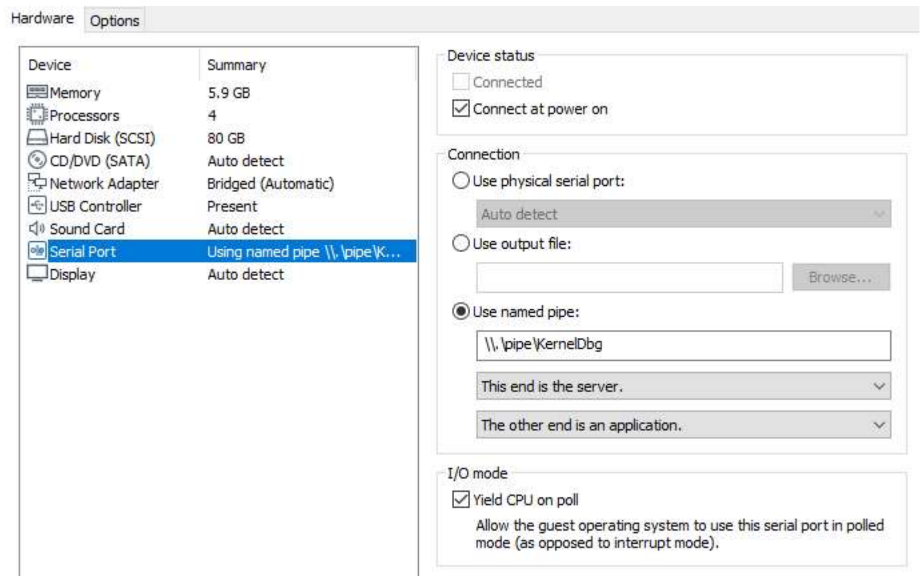
- Right click on VM -> Open VM directory
- Select the corresponding VMX file and open it with notepad
- Remove the lines beginning with serial0

Serial Port Configuration – Debugger

Paste the following lines inside the VMX debugger configuration file:

```
serial0.fileType = "pipe"
serial0.yieldOnMsrRead = "TRUE"
serial0.fileName = "\\.\pipe\KernelDbg"
serial0.present = "TRUE"
serial0.tryNoRxLoss = "TRUE"
```

Save and close the file. At the end of this procedure you should have the following hardware in the Debugger VM configuration.



Device	Summary
Memory	5.9 GB
Processors	4
Hard Disk (SCSI)	80 GB
CD/DVD (SATA)	Auto detect
Network Adapter	Bridged (Automatic)
USB Controller	Present
Sound Card	Auto detect
Serial Port	Using named pipe \\.\pipe\K...
Display	Auto detect

Device status
- [ ] Connected
- [x] Connect at power on

Connection
- ○ Use physical serial port: Auto detect
- ○ Use output file:
- ● Use named pipe: \\.\pipe\KernelDbg
  - This end is the server.
  - The other end is an application.

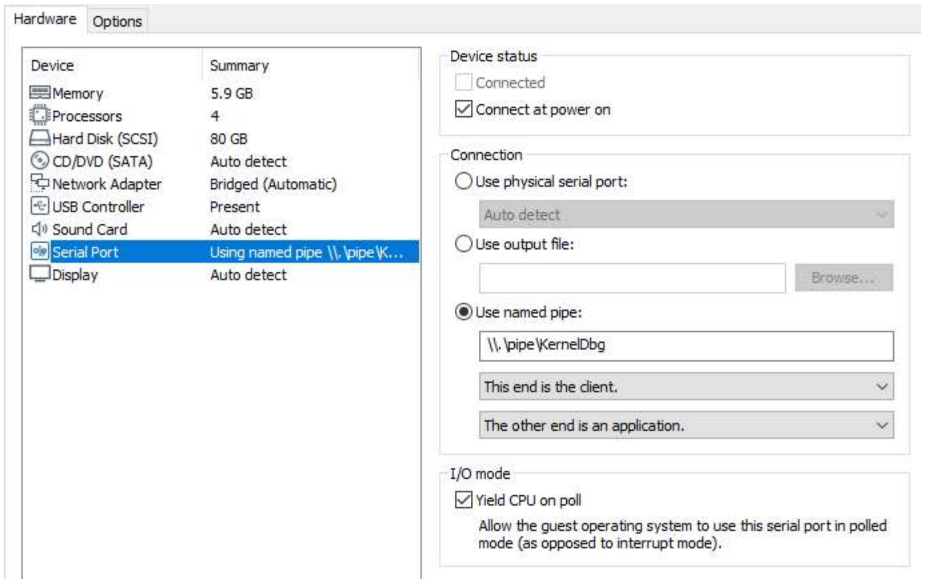
I/O mode
- [x] Yield CPU on poll
  Allow the guest operating system to use this serial port in polled mode (as opposed to interrupt mode).

Serial Port Configuration – Debuggee

Paste the following lines inside the VMX debuggee configuration file:

```
serial0.present = "TRUE"
serial0.fileType = "pipe"
serial0.yieldOnMsrRead = "TRUE"
serial0.fileName = \\.\pipe\KernelDbg
serial0.pipe.endPoint = "client"
serial0.tryNoRxLoss = "TRUE"
```

Save and close the file. At the end of this procedure you should have the following hardware in the Debuggee VM configuration.



Device	Summary
Memory	5.9 GB
Processors	4
Hard Disk (SCSI)	80 GB
CD/DVD (SATA)	Auto detect
Network Adapter	Bridged (Automatic)
USB Controller	Present
Sound Card	Auto detect
Serial Port	Using named pipe \\.\pipe\K...
Display	Auto detect

Device status
- [ ] Connected
- [x] Connect at power on

Connection
- ○ Use physical serial port: Auto detect
- ○ Use output file:
- ● Use named pipe: \\.\pipe\KernelDbg
  - This end is the client.
  - The other end is an application.

I/O mode
- [x] Yield CPU on poll
  Allow the guest operating system to use this serial port in polled mode (as opposed to interrupt mode).

Do NOT forget to select the Yield CPU on poll check box, as the kernel in the target virtual machine uses the virtual serial port in polled mode, not interrupt mode.

Setting up the Serial Port in Windows

Fire up both your machines and check that the serial port is connected using the named pipe: `\\.\pipe\KernelDbg`

Setting up COM port

Once Windows is up and running log into the machines as Administrator:

1. Start Bar Button
2. Right click on **Computer**.
3. Click on **Manage.**
4. **Computer Management** should open.

5. Click on the **Device Manager** button.
6. Expand the **Ports (COM & LPT)** tree item.
7. Right click on the COM port and choose **Properties**.
8. Click on the **Port Settings** tab.
9. Choose Bits per second **Baud Rate: 115200**.
10. Choose **8 data bits.**
11. **Parity: None.**
12. **Stop bits: 1.**
13. **Flow control: None**.
14. Click **Ok** and close all the dialogs.

Setting Up the Debugger Environment

Installing WinDbg is trivial but is also something that will waste our time (Windows SDK, Updates etc); since its installation can be automatized, I will recommend to install WinDbg via Chocolatey. From an elevated PowerShell launch the following command:

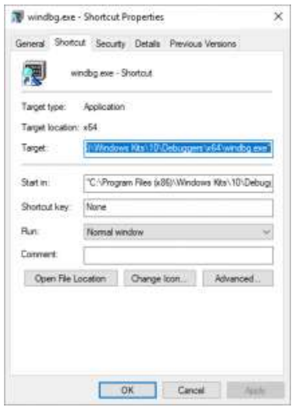
- `Set-ExecutionPolicy Bypass -Scope Process -Force; iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))`
- `choco install -y --force windbg`

It will take a bit; go grab a coffee 😉

Setting up symbols on Debugger

Microsoft provides redacted symbols for their software releases. This includes kernel components, to leverage this useful information, we'll need to setup WinDbg so it can access these resources.

Locate WinDbg shortcut -> right click on the shortcut -> select "Properties" and in the "Shortcut" tab you should see something similar:



windbg.exe - Shortcut

Target type: Application
Target location: x64
Target: |\Windows Kits\10\Debuggers\x64\windbg.exe"
Start in: "C:\Program Files (x86)\Windows Kits\10\Debug
Shortcut key: None
Run: Normal window
Comment:

Select the "Target" text field and append the following string:
`-y "srv*c:\symbols*https://msdl.microsoft.com/download/symbols"`

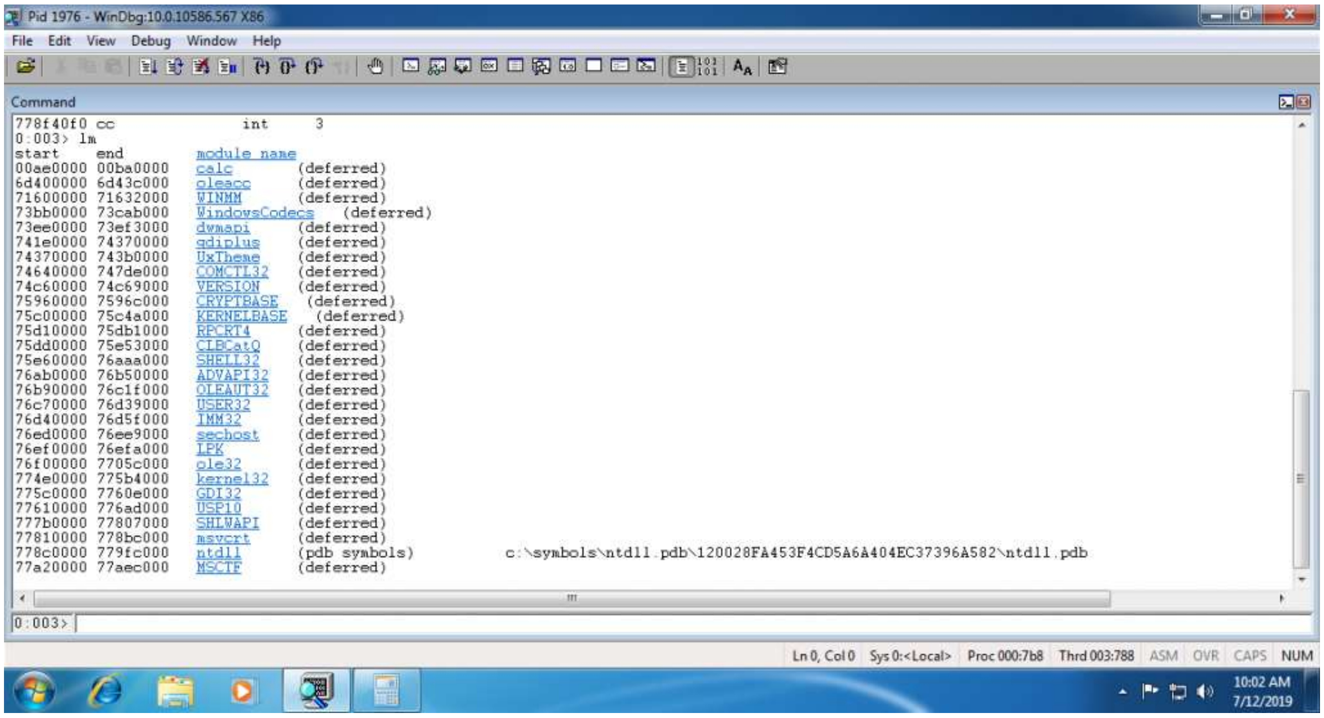
This will download all available symbols from the Microsoft Symbol Server to your local directory at `c:\symbols`. If you prefer to place your downloaded symbols somewhere else choose another local path instead.

Save your shortcut by clicking 'OK'.

Let's test WinDbg to ensure that everything is working fine:

1. Run the shortcut and a copy of the pre-installed application Calc.exe
2. Select "File" -> "Attach to a Process..." (or hit F6) in WinDbg. Scroll down all the way and select "calc.exe" from your process list. Then hit 'OK'.

Let's try to load symbols for all the running modules (executable and DLLs). First, let's list what modules are currently loaded in our process by using the `lm` (**l**ist **m**odules) command in the text box immediately to the right of ">" (bottom left corner of the "Command" window):


Pid 1976 - WinDbg:10.0.10586.567 X86
File Edit View Debug Window Help
Command
Ln 0, Col 0 Sys 0:<Local> Proc 000:7b8 Thrd 003:788 ASM OVR CAPS NUM
10:02 AM
7/12/2019

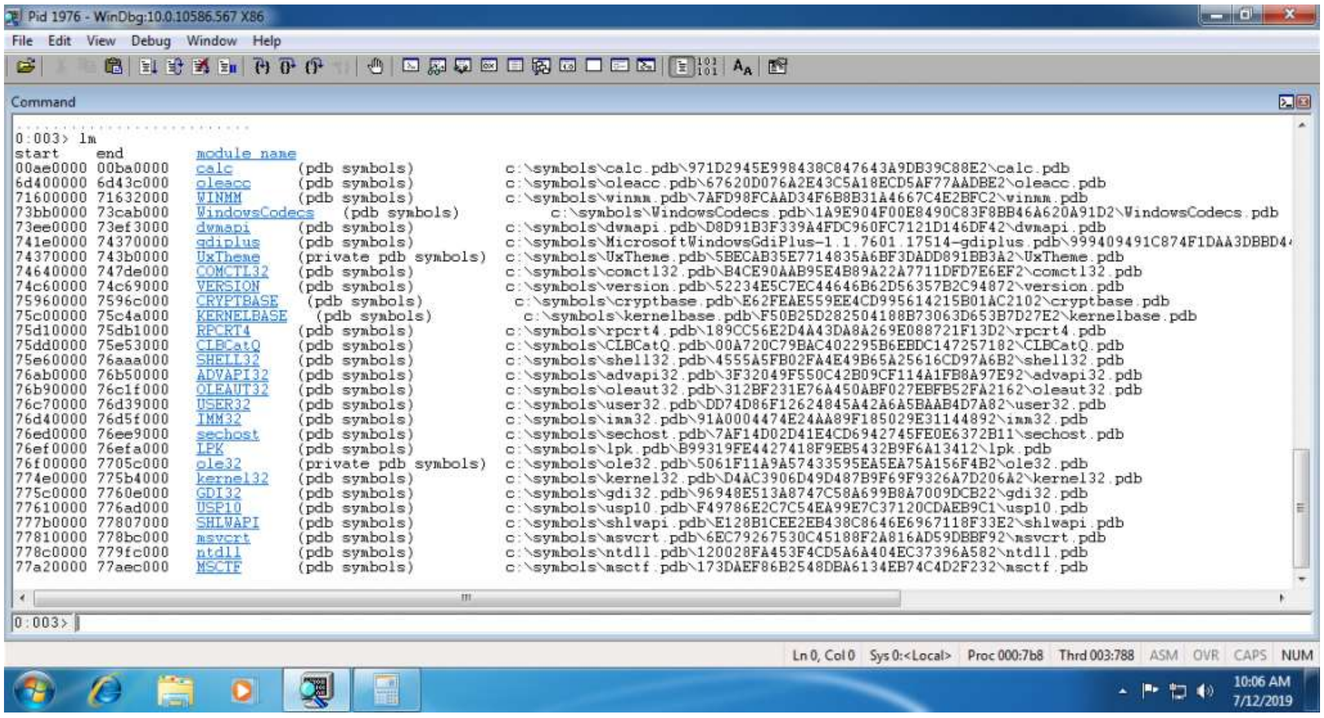
```
778f40f0 cc              int     3
0:003> lm
start    end        module name
00ae0000 00ba0000   calc       (deferred)
6d400000 6d43c000   oleacc     (deferred)
71600000 71632000   WINMM      (deferred)
73bb0000 73cab000   WindowsCodecs    (deferred)
73ee0000 73ef3000   dwmapi     (deferred)
741e0000 74370000   gdiplus    (deferred)
74370000 743b0000   UxTheme    (deferred)
74640000 747de000   COMCTL32   (deferred)
74c60000 74c69000   VERSION    (deferred)
75960000 7596c000   CRYPTBASE   (deferred)
75c00000 75c4a000   KERNELBASE   (deferred)
75d10000 75db1000   RPCRT4     (deferred)
75dd0000 75e53000   CLBCatQ    (deferred)
75e60000 76aaa000   SHELL32    (deferred)
76ab0000 76b50000   ADVAPI32   (deferred)
76b90000 76c1f000   OLEAUT32   (deferred)
76c70000 76d39000   USER32     (deferred)
76d40000 76d5f000   IMM32      (deferred)
76ed0000 76ee9000   sechost    (deferred)
76ef0000 76efa000   LPK        (deferred)
76f00000 7705c000   ole32      (deferred)
774e0000 775b4000   kernel32   (deferred)
775c0000 7760e000   GDI32      (deferred)
77610000 776ad000   USP10      (deferred)
777b0000 77807000   SHLWAPI    (deferred)
77810000 778bc000   msvcrt     (deferred)
778c0000 779fc000   ntdll      (pdb symbols)          c:\symbols\ntdll.pdb\120028FA453F4CD5A6A404EC37396A582\ntdll.pdb
77a20000 77aec000   MSCTF      (deferred)

0:003>
```

It should look something like this, anyway if your list looks different from mine, don't worry. Different versions of Windows and different versions of Calc will have different modules loaded.

Next, let's force a symbol load of all modules within our process by executing: `.reload /f`.

> To access WinDbg manual, you can type the command `.hh` within the debugger (or select "Help" and then "Search" from the menu bar). Typing `.hh search_terms` automatically runs a search for the user supplied argument. `.hh .reload` documents the `.reload` command.

WinDbg will took few moments to load all symbol information. You can see the status of WinDbg in the bottom left corner. After WinDbg has loaded symbols, run the lm command again (If WinDbg stays "BUSY" for a long time, you can force it to stop its current task by pushing CTRL+Break on your keyboard or by selecting "Debug" and then Break" from the menu bar).


Pid 1976 - WinDbg:10.0.10586.567 X86
File Edit View Debug Window Help
Command
Ln 0, Col 0 Sys 0:<Local> Proc 000:7b8 Thrd 003:788 ASM OVR CAPS NUM
10:06 AM
7/12/2019

```
0:003> lm
start    end        module name
00ae0000 00ba0000   calc       (pdb symbols)          c:\symbols\calc.pdb\971D2945E998438C847643A9DB39C88E2\calc.pdb
6d400000 6d43c000   oleacc     (pdb symbols)          c:\symbols\oleacc.pdb\67620D076A2E43C5A18ECD5AF77AADBE2\oleacc.pdb
71600000 71632000   WINMM      (pdb symbols)          c:\symbols\winmm.pdb\7AFD98FCAAD34F6B8B31A4667C4E2BFC2\winmm.pdb
73bb0000 73cab000   WindowsCodecs    (pdb symbols)          c:\symbols\WindowsCodecs.pdb\1A9E904F00E8490C83F8BB46A620A91D2\WindowsCodecs.pdb
73ee0000 73ef3000   dwmapi     (pdb symbols)          c:\symbols\dwmapi.pdb\D8D91B3F339A4FDC960FC7121D146DF42\dwmapi.pdb
741e0000 74370000   gdiplus    (pdb symbols)          c:\symbols\MicrosoftWindowsGdiPlus-1.1.7601.17514-gdiplus.pdb\999409491C874F1DAA3DBBD4
74370000 743b0000   UxTheme    (private pdb symbols)  c:\symbols\UxTheme.pdb\5BECAB35E7714835A6BF3DADD891BB3A2\UxTheme.pdb
74640000 747de000   COMCTL32   (pdb symbols)          c:\symbols\comctl32.pdb\B4CE90AAB95E4B89A22A7711DFD7E6EF2\comctl32.pdb
74c60000 74c69000   VERSION    (pdb symbols)          c:\symbols\version.pdb\52234E5C7EC44646B62D56357B2C94872\version.pdb
75960000 7596c000   CRYPTBASE   (pdb symbols)          c:\symbols\cryptbase.pdb\E62FEAE559EE4CD995614215B01AC2102\cryptbase.pdb
75c00000 75c4a000   KERNELBASE   (pdb symbols)          c:\symbols\kernelbase.pdb\F50B25D282504188B73063D653B7D27E2\kernelbase.pdb
75d10000 75db1000   RPCRT4     (pdb symbols)          c:\symbols\rpcrt4.pdb\189CC56E2D4A43DA8A269E088721F13D2\rpcrt4.pdb
75dd0000 75e53000   CLBCatQ    (pdb symbols)          c:\symbols\CLBCatQ.pdb\00A720C79BAC402295B6EBDC147257182\CLBCatQ.pdb
75e60000 76aaa000   SHELL32    (pdb symbols)          c:\symbols\shell32.pdb\4555A5FB02FA4E49B65A25616CD97A6B2\shell32.pdb
76ab0000 76b50000   ADVAPI32   (pdb symbols)          c:\symbols\advapi32.pdb\3F32049F550C42B09CF114A1FB8A97E92\advapi32.pdb
76b90000 76c1f000   OLEAUT32   (pdb symbols)          c:\symbols\oleaut32.pdb\312BF231E76A450ABF027EBFB52FA2162\oleaut32.pdb
76c70000 76d39000   USER32     (pdb symbols)          c:\symbols\user32.pdb\DD74D86F12624845A42A6A5BAAB4D7A82\user32.pdb
76d40000 76d5f000   IMM32      (pdb symbols)          c:\symbols\imm32.pdb\91A0004474E24AA89F185029E31144892\imm32.pdb
76ed0000 76ee9000   sechost    (pdb symbols)          c:\symbols\sechost.pdb\7AF14D02D41E4CD6942745FE0E6372B11\sechost.pdb
76ef0000 76efa000   LPK        (pdb symbols)          c:\symbols\lpk.pdb\B99319FE4427418F9EB5432B9F6A13412\lpk.pdb
76f00000 7705c000   ole32      (private pdb symbols)  c:\symbols\ole32.pdb\5061F11A9A57433595EA5EA75A156F4B2\ole32.pdb
774e0000 775b4000   kernel32   (pdb symbols)          c:\symbols\kernel32.pdb\D4AC3906D49D487B9F69F9326A7D206A2\kernel32.pdb
775c0000 7760e000   GDI32      (pdb symbols)          c:\symbols\gdi32.pdb\96948E513A8747C58A699B8A7009DCB22\gdi32.pdb
77610000 776ad000   USP10      (pdb symbols)          c:\symbols\usp10.pdb\F49786E2C7C54EA99E7C37120CDAEB9C1\usp10.pdb
777b0000 77807000   SHLWAPI    (pdb symbols)          c:\symbols\shlwapi.pdb\E128B1CEE2EB438C8646E6967118F33E2\shlwapi.pdb
77810000 778bc000   msvcrt     (pdb symbols)          c:\symbols\msvcrt.pdb\6EC79267530C45188F2A816AD59DBBF92\msvcrt.pdb
778c0000 779fc000   ntdll      (pdb symbols)          c:\symbols\ntdll.pdb\120028FA453F4CD5A6A404EC37396A582\ntdll.pdb
77a20000 77aec000   MSCTF      (pdb symbols)          c:\symbols\msctf.pdb\173DAEF86B2548DBA6134EB74C4D2F232\msctf.pdb

0:003>
```

As you can see, most modules now have a local symbol path listed to the right of their **module names**. It's very possible that there may be some modules that still do not have symbols loaded; these modules are most likely not distributed by Microsoft (e.g. 3rd party antivirus vendors).

For validation, go to the directory that you've setup for your local symbol cache, e.g. **C:\symbols**. If the folder contains data your setup is working.

Lazy Fix

If for some reason, the above steps failed, WinDbg may be able to fix automagically the problems for you if you issue `.symfix` and then `.reload /f`. In this case, WinDbg will alter your symbol path to the Microsoft Symbol Server. Your downloaded symbols will be stored (locally) in WinDbg's current working directory (`C:\Program Files (x86)\Windows Kits\10\Debuggers\x64`) or `C:\ProgramData\dbg`.

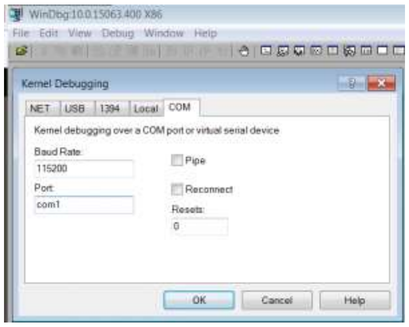
Kernel Debugging with WinDbg

Manually configure WinDbg each time for kernel debugging is boring. So, we can configure it in the shortcut. You can append the following line to the "Target" textbox:

```
-k com:port=com1,baud=115200
```

Hit 'OK' and you should be all set. Now when you run this shortcut of WinDbg, it will correctly configure your symbol path (without having to use environment variables) and will automatically start kernel debugging on COM1 port.

So, launch our WinDbg in kernel mode following the newly created shortcut or launch WinDbg, press CTRL+K and the following widow will pop-out:



Kernel debugging over a COM port or virtual serial device
Baud Rate: 115200
Port: com1
Pipe [ ]
Reconnect [ ]
Resets: 0

Set the COM tab as above and press "OK".

Setting Up the Debuggee Environment via Serial Port (or UART)

To enable Kernel Debugging, open cmd.exe as Administrator, and add another entry to the boot loader using bcdedit utility:

```
bcdedit /copy {current} /d "Windows 7 with kernel debug via COM"
```

Then enable debug mode on new entry UUID:

```
bcdedit /debug {UUID-RETURNED-BY-PRECEDENT-COMMAND} on
```

Note: graph parentheses are needed for the above command

Now instruct Windows serial communication as debugging medium and use the "fastest" baudrate (i.e 115200 symbols/sec). Since we'll only use serial debugging for this VM, we can use the `bcdedit /dbgsettings` global switch.

```
bcdedit /dbgsettings serial debugport:1 baudrate:115200
```

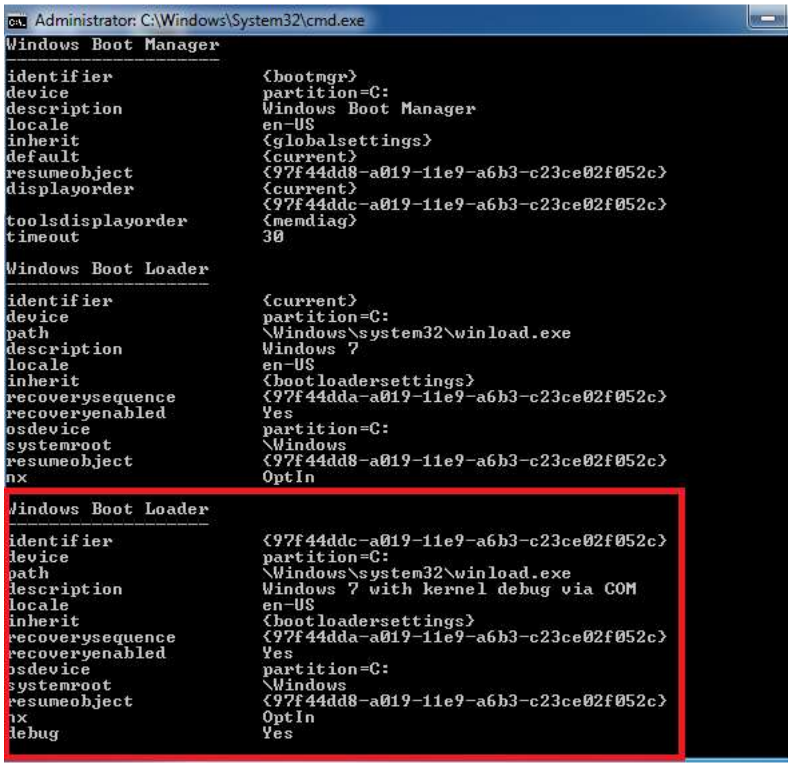
You can add the /noumex switch to the dbgsettings command to avoids user mode exceptions from causing the system to break into the kernel debugger. e.g. `bcdedit /dbgsettings serial debugport:1 baudrate:115200 /noumex`.

Note: if we wanted to set debug settings specific to one entry of the boot loader, we would've used `bcdedit /set` instead. e.g: `bcdedit /set {UUID-RETURNED-BY-PRECEDENT-COMMAND} debugtype serial`

Now validate that the settings have been successfully applied:

- `bcdedit /dbgsettings`
- `bcdedit`

You should see similar command prompt output to this:



```
Administrator: C:\Windows\System32\cmd.exe
Windows Boot Manager
--------------------
identifier              {bootmgr}
device                  partition=C:
description             Windows Boot Manager
locale                  en-US
inherit                 {globalsettings}
default                 {current}
resumeobject            {97f44dd8-a019-11e9-a6b3-c23ce02f052c}
displayorder            {current}
                        {97f44ddc-a019-11e9-a6b3-c23ce02f052c}
toolsdisplayorder       {memdiag}
timeout                 30

Windows Boot Loader
-------------------
identifier              {current}
device                  partition=C:
path                    \Windows\system32\winload.exe
description             Windows 7
locale                  en-US
inherit                 {bootloadersettings}
recoverysequence        {97f44dda-a019-11e9-a6b3-c23ce02f052c}
recoveryenabled         Yes
osdevice                partition=C:
systemroot              \Windows
resumeobject            {97f44dd8-a019-11e9-a6b3-c23ce02f052c}
nx                      OptIn

Windows Boot Loader
-------------------
identifier              {97f44ddc-a019-11e9-a6b3-c23ce02f052c}
device                  partition=C:
path                    \Windows\system32\winload.exe
description             Windows 7 with kernel debug via COM
locale                  en-US
inherit                 {bootloadersettings}
recoverysequence        {97f44dda-a019-11e9-a6b3-c23ce02f052c}
recoveryenabled         Yes
osdevice                partition=C:
systemroot              \Windows
resumeobject            {97f44dd8-a019-11e9-a6b3-c23ce02f052c}
nx                      OptIn
debug                   Yes
```

Note that debug parameter must be turned to Yes for the “Windows 7 kernel debug via COM” entry.

Now, from Start bar button or with Win+R key combination, launch “`msconfig`” in the dialog box. You will access certain Windows startup configurations. In the “Boot” tab, make sure to have the following configuration:

Now the next step is to restart the Debuggee VM and boot up in Kernel Debug mode:

ALWAYS make sure to have the Debugger VM with WinDbg started and listening for connection in kernel mode before firing up the Debuggee VM.

If everything worked up correctly, the following screen should show up in the Debugger VM in WinDbg:

We won't be able to interact with the virtual machine until we stop it running at an instruction. To do that, press "Ctrl+Pause" (or "Debug -> Break" menu). If we go to the virtual machine, we can now see that it is frozen, and we cannot move the mouse or obtain a response from the keyboard. is normal, given that the VM is stopped at an instruction. From the last line, we can see that execution of the instructions has been stopped by usi INT 3, which has been stopped in the kernel's memory. Below that is the console with which we can start to send commands to WinDbg.

Just to be sure that the symbols have been loaded correctly, run the following commands:

- `!sym noisy`
- `.reload /f`

Troubleshooting

Sometimes happens that while firing-up the Debuggee VM it stuck at the “Starting Windows” screen with WinDbg in the Debugger VM not receiving anything.

In this case I will suggest to close WinDbg and load it again, it immediately gets the connection from the Debuggee VM and Windows will start loading till the end.

You’re now debugging the Windows 7 x86 VM kernel! But as you’ll see, Serial Port debugging will drastically slow down all operations on the debuggee. Therefore, projects like VirtualKD came to life.

More Windows Debuggee Flavours

If the debuggee VM is a Windows 8 or later, there is faster debugging method (Network based), as detailed below.

When preparing the VM, make sure to add an extra Network Card as Host-Only, and linked to the same interface as the one specified on the host. **Important note**: if using VBox, in the “Advanced” section, select one of the Intel Pro card (preferably PRO/1000 MT Desktop). The reason for that is that Windows network kernel debugging does not work with all network controllers.

Boot the VM, and then open the “Device Manager” (Control Panel -> System -> Advanced system settings -> on the Hardware tab). Expand “Network adapters” and select the 2nd device’s properties menu. On the new window, the “Location” field will be required to assign this interface for debugging:

This indicates us the bus parameters we will need to provide bcdedit later on, with the format `<BusNumber>:<DeviceNumber>:<FunctionNumber>` (in this case 0.8.0).

Now open an administrator prompt and use bcdedit utility to create a new entry to the boot manager like we did on Windows 7 and enable the debug mode for it. But unlike Windows 7, now we have to setup the network properties:

```
bcdedit /dbgsettings net hostip:ip-of-debugger-vm port:50000 key:Kernel.Debugging.Is.Funbcdedit /set {dbgsettings} busparams <BusNumber>.<DeviceNumber>.<FunctionNumber>
```

Running the debugging session

On the debugger

Start the debugger VM first and prepare WinDBG for kernel-mode debugging (Ctrl-K) by selecting NET as debug vector and set the Port and Key adequately. WinDBG will then be waiting for new connection.

On the debuggee

Start the VM, when the boot loader menu shows up, select the one with the network kernel mode enabled.

The debugger will show some activity.

Windows XP

Since `bcdedit` does not exist on Windows XP in order to enable kernel debugging, you must alter the `boot.ini` file. The easiest way to do this is by clicking on Start and then Run (Start+R). Enter `C:\boot.ini` and hit 'OK'.

Append the string `/debug /debugport=COM1 /baudrate=115200` to the end of the first entry in the `[operating systems]` section.

Save the `boot.ini` via "File" and then "Save" from the menu bar, close the file and shutdown Windows.

Conclusion

Hopefully this article helped you setting up your Windows Kernel Debugging. If you encounter any issues you can reach me on Twitter or send me an email.

Stay tuned because in the next part we will dive into ms-14-066 vulnerability aka "winshock" (CVE-2014-6321) and I'll try to build a PoC for it.

RELATED **POSTS**

A Drone Tale

16/11/2018 | voidsec

During the previous months I've been a speaker for various international conferences: Hackinbo (Italy, 26-27 May), Sec-T (Stockholm, 13-14...

Read More →

WineHat 2015

16/11/2015 | voidsec

No Black Hats, No White Hats, just Wine Hats! Questo motto racchiude l'essenza di ciò che per me è stato il WineHat,...

Read More →

Perform a Nessus scan via port forwarding rules only

13/03/2020 | voidsec

Reportage HackInBo 2 Winter Edit

26/10/2015 | voidsec

Il 17 Ottobre a Bologn quinta edizione dell'H l'evento tutto italiano sicurezza informatica,

This post will be a bit different from the usual technical stuff, mostly because I was not able to...

Read More ➞

Read More ➞

Search...

RECENT POSTS

- Tabletopia: from XSS to RCE
- SLAE – Assignment #7: Custom Shellcode Crypter
- SLAE – Assignment #6: Polymorphic Shellcode
- SLAE – Assignment #5: Metasploit Shellcode Analysis
- SLAE – Assignment #4: Custom shellcode encoder

CATEGORIES

- Advisories (14)
- Blog (80)
- News (15)
- Reportage (12)
- SLAE x86 (7)