Setting Up Your Windows Kernel Debugging Environment

for WinDbg over network connection

# Background Information

When debugging a program using a debugging tool, such as *Microsoft Visual Studio* (Windows) or *gdb* (Linux), interrupting program execution through breakpoints and stepping through to executing single lines of source code at a time is occurring in usermode. The debugged program is typically executed as a child process of the debugging tool to be able to manipulate program execution and the user interacts with the debugging tool to control the debugged program.

But can the same be done with debugging kernel objects? Suppose a system is set up to debug itself and there is an active breakpoint that will be hit when any new process is created. Once a new process is about to be created and the breakpoint is hit, how will the user interact with the system now? Until the system receives a command to continue execution, it is essentially left hanging. The system cannot respond to keyboard or mouse input since keyboard events have to be handled by the unresponsive kernel.[[1]](#footnote-0)

A basic workaround for small kernel patches was to write simple quick-to-execute print statements. However, to gain the ability to quickly read from/write to memory and navigate around program execution would require for the debugging tool to be hosted by a separate system, hence this kernel debugging setup document.

# Requirements

* Two networked Windows computers, ideally on same LAN
  + Does not necessarily have to be physical machines;   
    one or both can be a virtual machine.

One computer is to be designated the *debug host* (will be running the debugger)   
and the other will be the *debug target* (will be debugged)

* On the *debug host*, WinDbg has been installed

WinDbg is a part of Debugging Tools for Windows, which is contained within the Windows 10 SDK[[2]](#footnote-1). If you only want Debugging Tools for Windows, be sure to have only that selected during installation.

## Recommended

* On the *debug host*, Internet connectivity to download and use Microsoft symbols to aid in debugging Windows system components

Symbols are often in the forme of .pdb files, which contain information such as location of public functions and names/data types of global structs. This is especially useful for setting breakpoints.

# Instructions

1. Obtain and note the IP address of the debug host.
   * If both machines are on the same LAN, you may use command `ipconfig` to retrieve the machine’s given IP address.  
     Otherwise, you will have to determine the ‘public IP address’ of the debug host.  
     -

Note: If you are expecting to have multiple kernel debugging sessions in the future, it is advised that you set the debug host’s IP address statically. This will lessen needed configuration in the future since you will not need to look up and/or modify IP-related settings.

1. Verify the debug host and target are able to communicate with each other (for instance, try using the `ping` command).
2. On the *debug target*, run the following in an *administrative* command prompt to configure the machine to establish a debugging session:

SET HostIP=192.168.0.2  
SET Port=12321  
bcdedit /debug on  
bcdedit /dbgsettings net hostip:%HostIP% port:%Port%

HostIP is the IP address of the *debug host* determined earlier.  
The selected Port should be unused on the *debug host*.

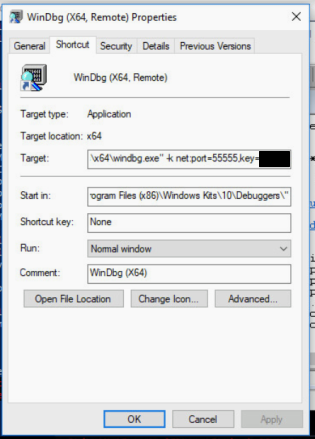
After running the above lines, Windows will display a long automatically-generated key that should be noted for later *debug host* setup.

While it is recommended to use the automatically-generated key, you may specify your own easier-to-type key. It should be in the forme of four alphanumeric strings of maximum length 13, delimited by a period (‘.’).  
Example keys:

* 1.2.3.4
* Nota.very.secure.key1
* 9re10qw7ua7rly4g.pd6h814evoivhbr5.ndc22fau93i8rbtb.108orsja5n1fnc3u

To use your custom key, replace the second ‘bcdedit’ command above with the following:  
bcdedit /dbgsettings net hostip:%HostIP% port:%Port% key:%Key%  
And, of course, replace %Key% with your custom key.

After running the above, your system will begin attempting to contact and authenticate with the supplied IP address immediately after restart.

1. On the *debug host*, start listening for a kernel debug connection:
   1. Create a shortcut to *windbg.exe* on your Desktop.  
      Assuming you have installed *Debugging Tools for Windows* to its default location, you would create a shortcut to either
      * “%ProgramFiles(x86)%\Windows Kits\10\Debuggers\x86\windbg.exe”   
        for debugging a 32-bit *debug target*, or
      * “%ProgramFiles(x86)%\Windows Kits\10\Debuggers\x64\windbg.exe”   
        for debugging a 64-bit *debug target*
   2. View the file properties of the created shortcut by right-clicking it and selecting ‘Properties’.
   3. To the ‘Target’ field, append (without quotes):  
      “-k net:port=PORT,key=KEY”

Command-line arguments explanation:

* -k: Start a kernel debugging session
* net: Debug session to take place over TCP/IP
* port: Port to listen on for incoming debug session connections.
  + Should match port specified on the *debug target*.
* key: Encryption key for use in network debugging.
  + Should match key specified on the *debug target*.
* Other command-line arguments can be found in WinDbg’s help topics under ‘Debugger Reference > Command-Line Options’.
  1. Save the dialog box settings by clicking ‘OK’ or ‘Apply’ and run the shortcut.
  2. *Debug host* is now listening for incoming debug session connections…

1. Restart *debug target*.   
   If configured properly, *debug host* should start generating output stating it is “connected to target”
2. Load debugging symbols:  
   Send a ‘Break’ command and enter `.symfix; .reload`.  
   This will reset the (probably blank) symbol path to point to the Microsoft symbol store. To view the current symbol path, use command `.sympath`.  
   These two commands will have to be entered in every new debugging session.

Your debugging environment is now set up. If any disconnects occur or the debug target and host fall out of sync, restarting WinDbg from the shortcut should typically reconnect the debugging session.

# See Also

[Common WinDbg Commands (Thematically Grouped)]  
<http://windbg.info/download/doc/pdf/WinDbg_cmds.pdf>

# Technical Considerations

* When a set breakpoint is hit, the debug target will essentially appear unresponsive to anything monitoring the debug target (e.g. hypervisors).   
  Ensure the monitor will not automatically restart the debug target while execution is paused.
* The snapshotting feature of most virtualisation software will revert virtual hardware configuration (namely, network adapters), which will affect the (also reverted) Windows networking configuration. Do not forget to check these settings when troubleshooting a debugging session failing to connect after a reverted snapshot.

1. WinDbg *does* support local kernel debugging (via the -kl flag), but limitations apply. [↑](#footnote-ref-0)
2. The Windows 10 SDK is available here: <https://go.microsoft.com/fwlink/p?LinkID=271979> [↑](#footnote-ref-1)