NETWORKWORLD

This story appeared in *Network World* Magazine

How to solve Windows 7 crashes in minutes

By Dirk A. D. Smith, Network World
April 18, 2011 12:03 AM ET

Everything is perfect; you've upgraded to Windows 7. It's fully patched, all drivers are updated, security is tight, maybe you even have new hardware...yet the old Blue Screen of Death (BSOD) taunts you from your new high definition-screen.

The good news is that you can quickly solve the problem in most cases by using the Windows debugger tool. It's simple and free.

Back in the Window XP era (2005), we wrote a tutorial on solving Windows crashes (How to solve Windows system crashes in minutes). This is an updated version that will make you the master of system crash resolution in your home or office.

Is crash resolution different for different versions of Windows?

The same approach to resolve system crashes applies to the many variants of Windows, says Andre Vachon, principal development lead at Microsoft. "The latest releases of Microsoft Windows use the same operating system kernel, the same primary interfaces, drivers work on both server and client, and the debugger uses the same debug files. Further, we used the same code base and source tree to compile both 32- and 64-bit versions."

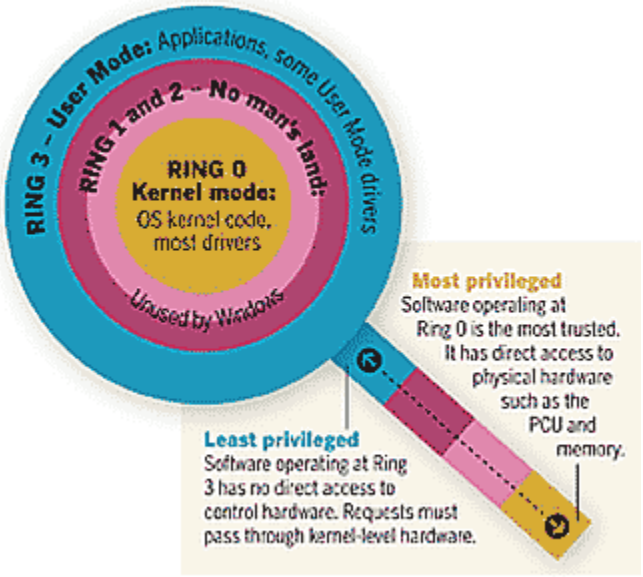
With that in mind and for simplicity I will refer to Windows 7. However, not only will the information apply to other current releases, much of it will apply to legacy versions back to Windows 2000.

Why Windows 7 crashes

Windows became more stable as it matured. And, while the operating system has gone from 16-bit to 32-bit and now 64-bit, the features have become more extravagant, and the footprint much larger - it is actually harder to bring down.

Still, it does fall over. However, the reasons for such system failures have not changed from the XP days.

Windows takes advantage of a protection mechanism that lets multiple applications run at the same time without stepping all over each other. Known now as User Mode and Kernel Mode, it was originally known as the Ring Protection scheme.



Kernel Mode

Kernel Mode (Ring 0) software has complete and unfettered access to the hardware. Software operating here is normally the most trusted because it can execute any instruction and reference any address in the system. Crashes in Kernel Mode are complete system failures requiring a reboot. This is where you find the operating system kernel code and most drivers.

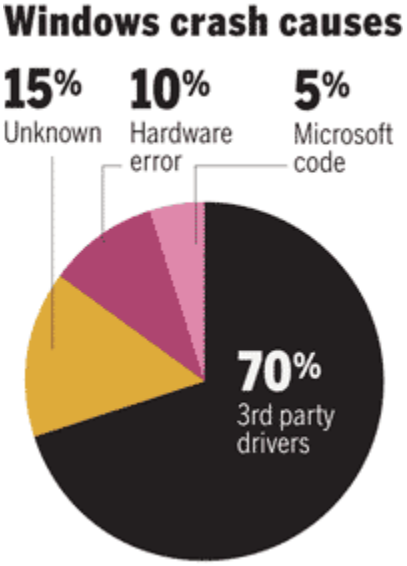
User Mode

User Mode (Ring 3) software cannot directly access the hardware or reference any address freely. It must pass instructions - perhaps more accurately requests - through calls to APIs. This feature enables protection for the overall operation of the system, regardless of whether an application makes an erroneous call or accesses an inappropriate address. Crashes in User Mode are generally recoverable, requiring a restart of the application but not the entire system. This is where you find most of the code running on your computer ranging from Word to Solitaire and some drivers.

So with much of the software running in User Mode these days, there is simply less opportunity for applications to corrupt system-level software and, for that matter, each other. However, kernel-mode software is not protected from other kernel-mode software. For example, if a video driver erroneously accesses a portion of memory assigned to another program (or memory not marked as accessible to drivers) Windows will stop the entire system. This is known as a Bug Check and the familiar Blue Screen of Death is displayed.

Crash causes by the numbers

While the numbers vary, they do not vary much. When combining data reported from several sources including my own 20 years dealing with crash prevention and resolution, a trend becomes clear; about 70% of Windows system crashes are caused by third party drivers operating in Kernel Mode, 15% is unknown, 10% is from faulty hardware (more than half from bad memory) and only about 5% from faulty Microsoft code.



An important point that is not well known is that most crashes are repeat crashes. This is so because most admins are not able to resolve system crashes immediately. As a result those crashes tend, unfortunately, to occur again...and again. More often than not, these events recur over weeks and in many cases over months before being resolved. By using the information in this article to solve crashes when they first occur, you will prevent many subsequent crashes.

Getting Started: System Requirements

To prepare to solve Windows 7 system crashes using WinDbg you will need a PC with the following:

• 32-bit or 64-bit Windows 7/Vista/XP or Windows Server 2008/2003

• Approximately 25MB of hard disk space (this does not include storage for dump files or for symbol files)

• Live Internet connection

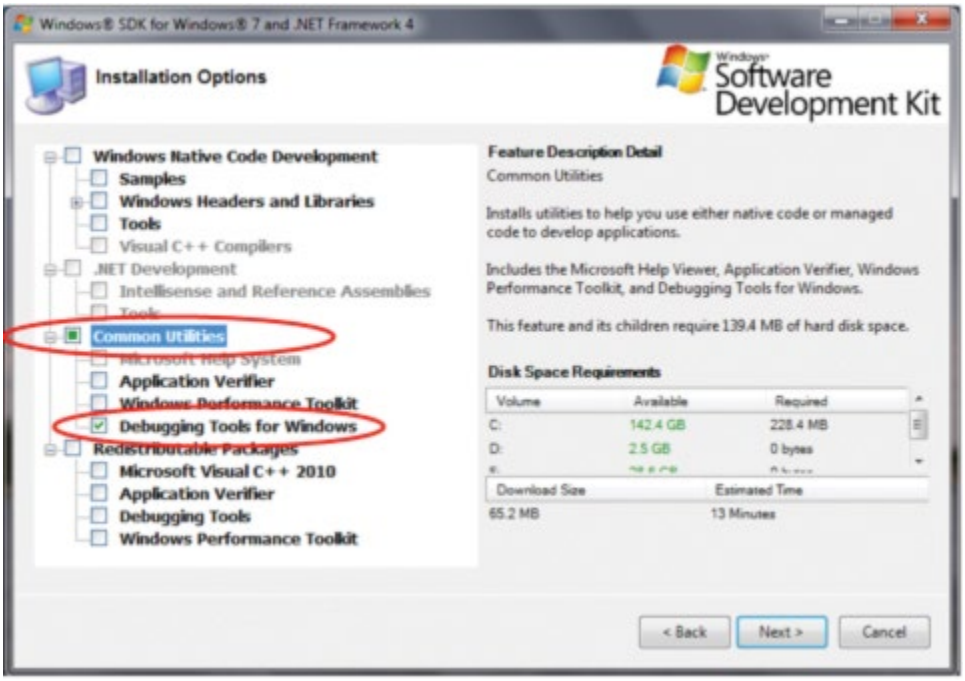
• Microsoft Internet Explorer 5.0 or later

• The latest version of WinDbg comes as an option in the Windows SDK. The SDK download file is called winsdk_web.exe, is 498KB in size, and can be downloaded for free. (Note that after installing the debugger you can delete the large download file thus freeing up lots of space.)

• A memory dump (the page file must be on C: for Windows to save the memory dump file)

Install WinDbg

After downloading the Windows SDK and running the Setup wizard, select the **Debugging Tools for Windows** option under **Common Utilities**.



Configure Startup and Recovery

This is annoying. Someone made it very non-intuitive to locate the dialogue box needed to check that your system is set to take the appropriate actions during a BugCheck, including whether to automatically restart and what size dump files to save.

Find the Startup and Recovery dialog box:

1. Select the **Start** button at the bottom left of your screen

2. Select **Control Panel**

3. Select **System and Security**

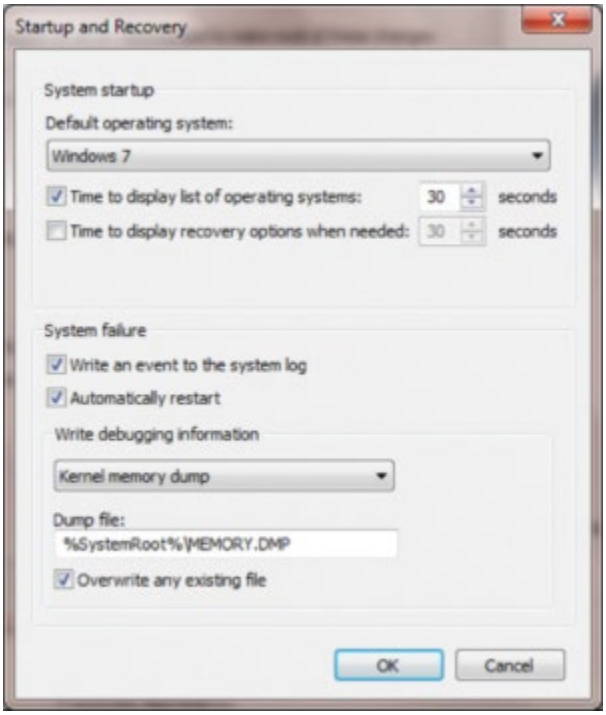
4. From the options in the right column, select **System**

5. From the left column select **Advanced system settings** to display the **System Properties** box

6. In the System Properties box select the **Advanced** tab

7. In the Startup and Recovery area select the **Settings** button

See the Startup and Recovery dialog box below:



Ensure Startup and Recovery settings are correct

Under **System failure**

1. Check **Write an event to the system log**

2. Check **Automatically restart**

3. Select **Kernel memory dump**

4. Ensure dump file to be written to **%SystemRoot%\MEMORY.DMP**

5. Check **Overwrite any existing file** to save hard drive space

Note that this will mean that your system will save both a kernel dump file and a minidump file. However, while you will have a minidump for every event, only the last kernel dump will be saved.

Configure WinDbg

Launching the debugger: To launch WinDbg select the following:

```
Start | All Programs | Debugging Tools for Windows| WinDbg
```

If you are going to use it with any frequency, simplify launching the program by pinning it to the Startup menu or send a shortcut to the desktop.

What's the big deal about symbols?

Before you jump in to save the day by finding the miscreant module in a dump file you have to be sure the debugger is ready. Most importantly you have to be sure it will locate the symbol files for the precise version of the operating system that you are troubleshooting.

Symbol tables are a byproduct of compilation. When a program is compiled, the source code is translated from a high-level language into machine code. At the same time, the compiler creates a symbol file with a list of identifiers, their locations in the program, and their attributes. Some identifiers are global and local variables, and function calls. A program doesn't require this information to execute. Therefore, it can be taken out and stored in another file, reducing the size of the final executable.

Smaller executables take up less disk space and load into memory faster than large ones. But there is a flip side: When a program causes a problem, the operating system knows only the hex address at which the problem occurred. You need something more than that to determine which program was using that memory space and what it was trying to do. Windows symbol tables hold the answer and having access to symbols specific to your system's memory is like putting place names on a map. Conversely, analyzing a dump file with the wrong symbol tables would be like finding your way through San Francisco with a map of Boston.

Configure WinDbg to locate symbols

There are an amazing number of symbol table files for Windows. This is so because every build of the operating system, even one-off variants, results in a new file. Fortunately, WinDbg can handle it for you but you must configure it with the correct search path. To do this, launch WinDbg and select the following:

```
File | Symbol file path
```

Then enter the following path: (Make sure that your firewall allows access to msdl.microsoft.com)

```
srv*c:\cache*http://msdl.microsoft.com/download/symbols
```

Note that the address between the asterisks is where you want the symbols stored for future reference. For example, I store the symbols in a folder called symbols at the root of my c: drive, thus:

```
srv*c:\symbols*http://msdl.microsoft.com/download/symbols
```

When opening a memory dump, WinDbg will look at the executable files (.exe, .dll, etc.) and extract version information. It then creates a request to the symbol server at Microsoft, which includes this version information and locates the precise symbol tables to draw information from. It won't download all symbols for the specific operating system you are troubleshooting; it will download what it needs. Alternatively, you can opt to download and store the complete symbol file from Microsoft. This, however, will run from about 600MB to near 800MB for each version of the operating system you analyze. In contrast WinDbg downloaded less than 100MB to analyze several versions of the operating system on my test machine. Even with the low cost of hard drives these days, the space savings is significant.

About dump files

A memory dump file is a snapshot of what the system had in memory when it crashed. Though perhaps the least attractive and correspondingly least intuitive thing you are likely ever to look at, it is your best friend when the operating system falls over. Windows creates three different sizes of memory dumps; minidumps, kernel dumps, and full dumps.

1. Small or minidump

Windows 7 minidumps are 256K-bytes, which is tiny by any standard, however they have grown from the Windows 2000/XP days when they were only 64K. One of the reasons they are so small is that they do not contain any of the binary or executable files that were in memory at the time of the failure. However, those files are critically important for subsequent analysis by the debugger. As long as you are debugging on the machine that created the dump file WinDbg can find them in the System Root folders (unless the binaries were changed by a system update after the dump file was created). Alternatively the debugger should be able to locate them through SymServ. Properly configured, Windows 7 creates and saves a minidump for every crash event as well as a kernel dump (described below).

2. Kernel dump

Kernel dumps are roughly equal in size to the RAM occupied by the Windows 7's kernel. On my notebook a kernel dump runs about 344MB and compressed it is just over 100MB. One advantage to a kernel dump is that it contains the binaries. As a default I would always have the system save the latest kernel dump. Remember that while saving it, the system will also save a minidump.

3. Complete or full dump

A full memory dump is about equal to the amount of installed RAM. With many systems having multiple GBs, this can quickly become a storage issue, especially if you are having more than the occasional crash. Normally I do not advise saving a full memory dump because they take so much space and are generally unneeded. However, Microsoft's Vachon advises that "if you are trying to debug a very complex problem, such as an RPC issue between multiple services in the box and you want to see what the services are doing in User Mode, the full memory dump can be very helpful." Therefore, stick to the kernel dump but be prepared to switch the setting to generate a full dump on occasion.

What if you do not have a memory dump to work with?

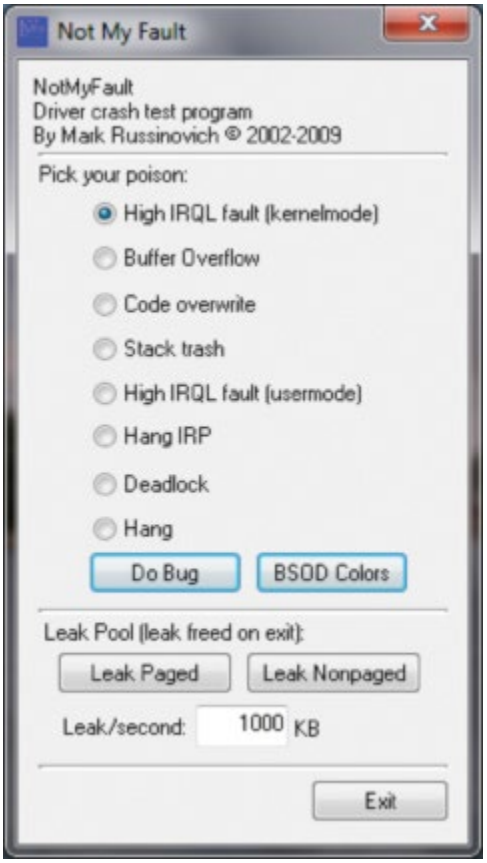
If you do not have a memory dump to look at, do not worry, you can make it crash! The simplest way (without having to change Registry settings) is to run a cool tool called NotMyFault (thank you Mark Russinovich and the team at SysInternals.) It provides a selection of options to load a misbehaving driver (which requires administrative privileges).

But remember...it WILL CREATE A SYSTEM CRASH! So prepare your system and be sure to let anyone who needs access to the system to log off for a few minutes. Save any files that contain information you might otherwise lose and close applications. If you have configured your system as described above, it should work fine. The machine should go down, reboot, and you will have both a minidump as well as a kernel dump to look at. I've used it plenty of times and had no problems.

Download NotMyFault and force a system crash

1. Download the **NotMyFault** tool from the following Microsoft Web site and extract the files to a folder:
http://download.sysinternals.com/Files/Notmyfault.zip

2. Right-click on **NotMyFault.exe** or at the Command Prompt type **NotMyFault.** If you get the message "You don't have permission to open this file" then try again but when right-clicking select **"Run as Administrator".**



3. From the menu select **"High IRQL fault (kernelmode)"** and the **Do Bug** button. This will generate a memory dump file and a "Stop D1" error.

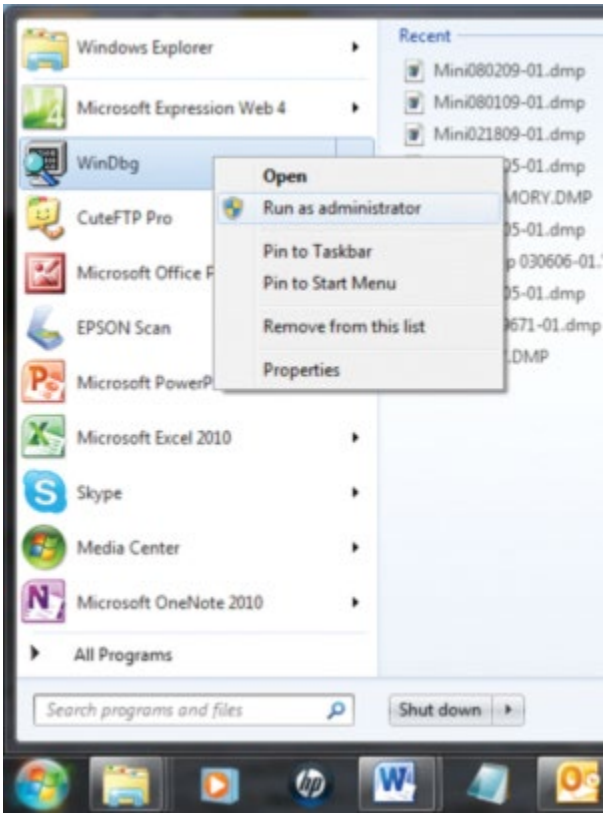
4. Sit back...your system will be back in momentarily and you will have both a minidump and kernel dump to view.

Load a dump file

If you get the message "You don't have permission to open this file", re-launch WinDbg by right-clicking on it and selecting Run as administrator.

Once the debugger is running, select the menu option **File | Open crash dump** and point it to open the memory dump you want to analyze. When offered to **Save information for workspace** select **Yes** if you want it to remember where the dump file is.

WinDbg looks for the Windows symbol files for that precise build of Windows. It references the symbol file path, accesses microsoft.com, and displays the results.



NOTE: If the debugger seems busy, it is probably the first time a dump file for a specific machine has been opened, therefore, WinDbg is downloading symbols from SymServ. The next time a dump is opened for the same machine the debugger will likely seem much faster since the symbol files will be available locally.

A Command window will appear. This is where the crash analysis will be displayed. At the lower left will be a KD> prompt. To the right of the prompt is a single-line window where you will enter commands.

Possible error messages

If you get the message

```
*** ERROR: Symbol file could not be found. Defaulted to export
symbols for ntoskrnl.exe -
```

one of the following three things is usually wrong:

• Your path is incorrect; check to make sure there are no typos or other errors (such as a blank white space) in the symbol file path you entered earlier

• Your connection failed; check your Internet connection to make sure it is working properly

• Your firewall blocked access to the symbol files or the symbol files were damaged during retrieval

If your path and connection are solid, then it's likely that the problem is your firewall. If a firewall initially blocks WinDbg from downloading a symbol table, it can result in a corrupted symbol file. If unblocking the firewall and attempting to download the symbol file again does not work; the symbol file remains damaged. The quickest fix is to close WinDbg, delete the symbols folder (which you most likely set at c:\symbols), and unblock the firewall. Now, reopen WinDbg and a dump file. The debugger will recreate the folder and re-download the symbols.

If you see this message,

```
***** Kernel symbols are WRONG. Please fix symbols to do
analysis.
```

then WinDbg was unable to retrieve the proper symbols and it will resort to using the default symbol table. But as the warning suggests, it cannot produce accurate results. Remember that symbol tables are generated when programs are compiled, so there is a symbol table file for every Windows version, patch, hot fix, and so on. Go back up to the section above and ensure you have the right path set, the connection is good, and it is not blocked.

Look through WinDbg's output. You may see an error message similar to the following that indicates it could not locate information myfault.sys:

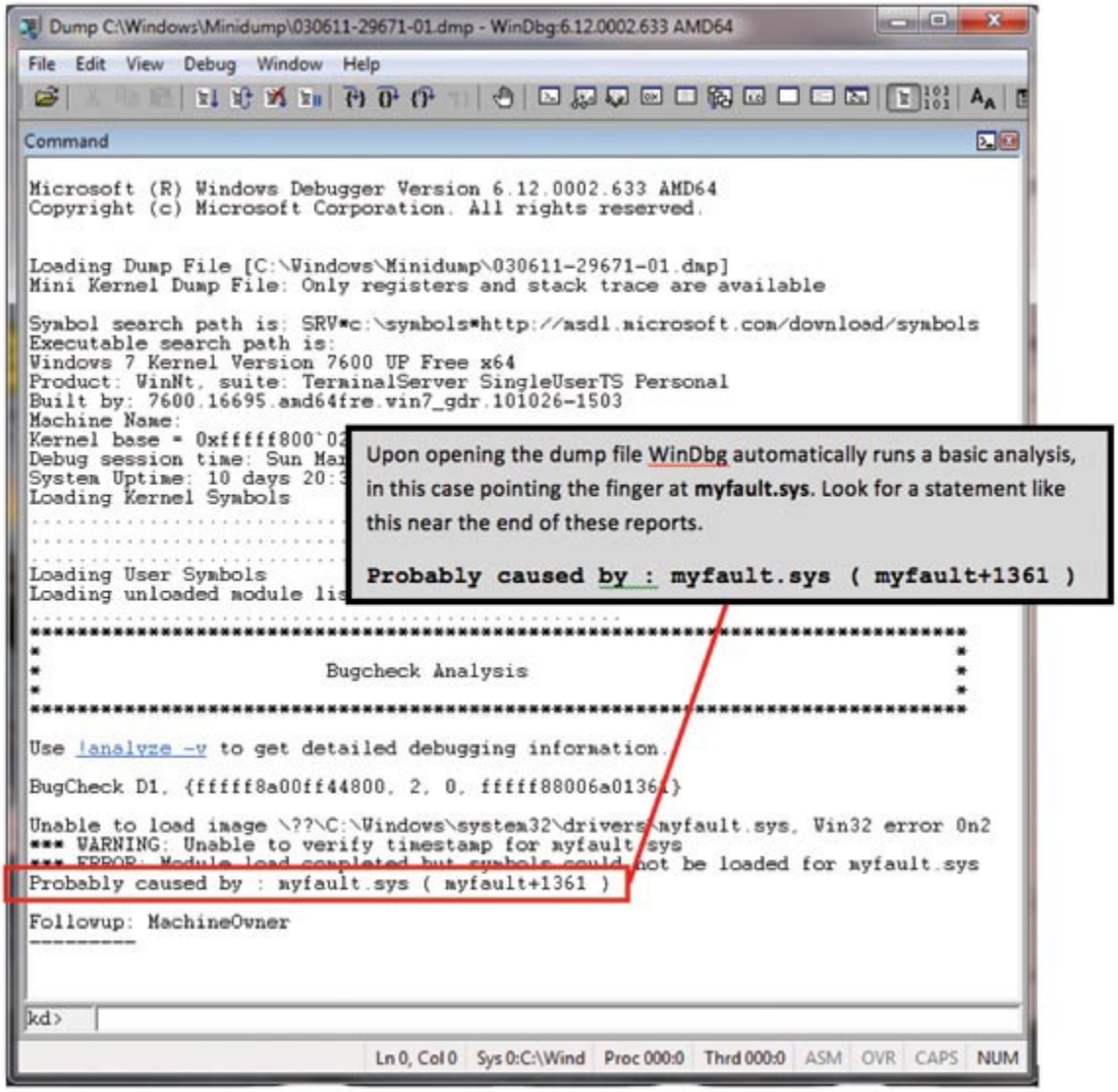
```
Unable to load image
\??\C:\Windows\system32\drivers\myfault.sys, Win32 error 0n2

*** WARNING: Unable to verify timestamp for myfault.sys

*** ERROR: Module load completed but symbols could not be loaded
for myfault.sys
```

This means that the debugger was looking for information on myfault.sys. However, since it is like a third-party driver (OK, it is made by Microsoft but it is certainly not a regular Microsoft product) there are no symbols for it (Microsoft does not store all of the third-party drivers). You can ignore this error message. Vendors do not typically ship drivers with symbol files, and they aren't necessary to your work; you can pinpoint the problem driver without them.

When you have WinDbg open a dump file, it automatically runs a basic analysis. Without even giving the debugger any direct commands (other than to open a specific dump file) it has named a suspect as shown in the screen below.



Commands

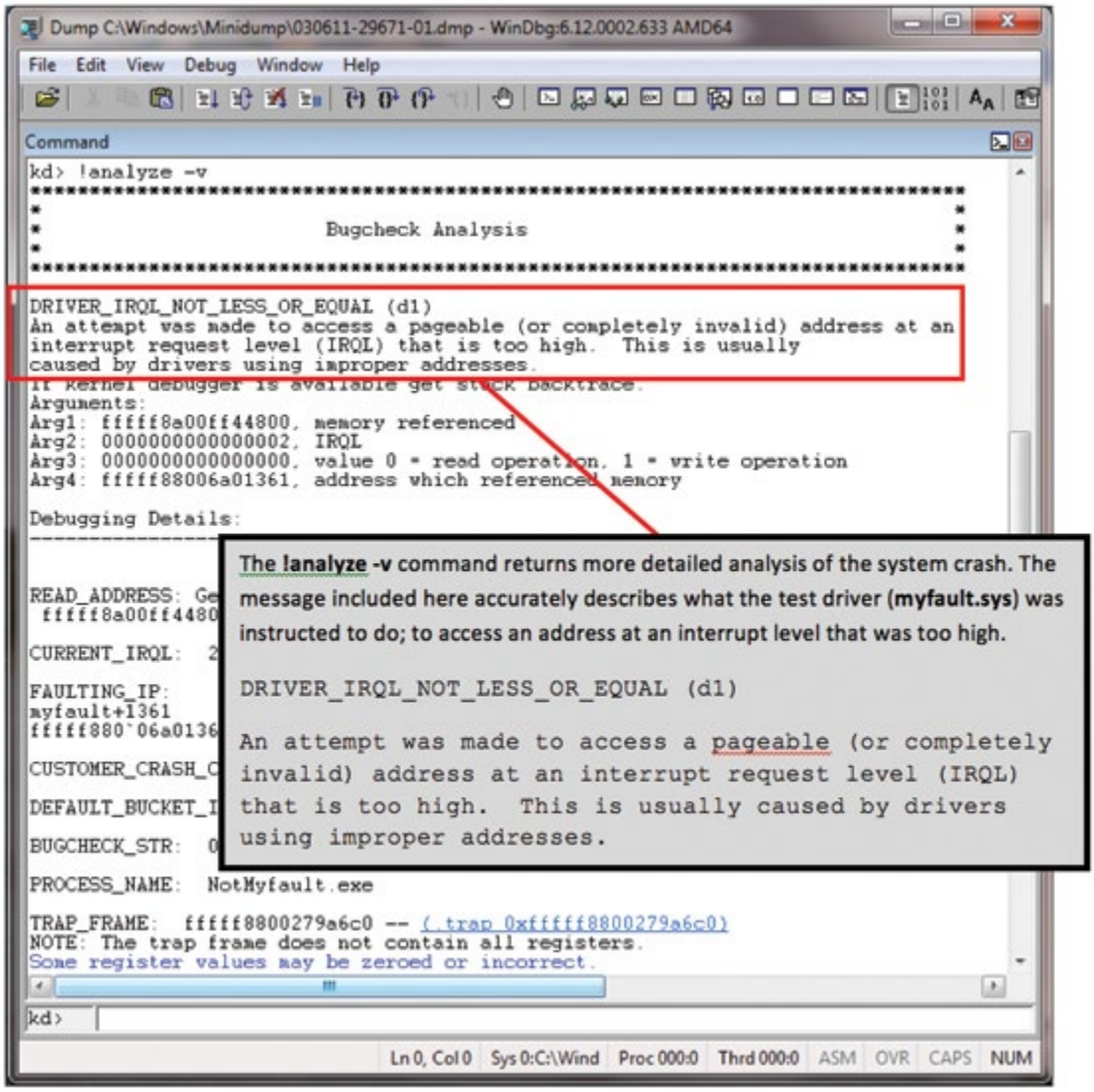
There are hundreds of commands to control WinDbg; it is a very capable tool. Fortunately...we only need one. To get fancy, we'll use two more, bringing the total to three. They are !analyze -v, lmv, and lmvm. If you want to sound like this is not the first time you've used a debugger, here is how you pronounce the first command: "bang analyze dash vee".

!analyze -v	Analyze in Verbose Mode	!analyze –v displays information describing the state of a system when it crashed, the fault encountered, and who is the primary suspect.
lmv	Loaded Module Verbose	lmv displays a list of drivers and their path, version and vendor information. It often includes a product description. Output from lmv can take a long time. Watch the bottom left of the WnDbg interface where you would normally expect to see the kd> prompt. When retrieving information it will display *BUSY*. Only when the kd> prompt returns can you use additional commands.
lmvm [Module Name]	Loaded Module Verbose Module Name	lmvm [module name] enables you to tell the debugger to retrieve information for only that specific module. For example: lmvm myfault.sys

!analyze -v

Type !analyze -v on the command line at the bottom of the Command window (note the space between the command and the "-v"). The "v" or verbose switch tells WinDbg that you want all the details. The explanation it gives is a combination of English and programmer-speak, but it is nonetheless a great start. In fact, in many cases you may not need to go any further. If you recognize the cause of the crash, you're probably done.

Here's an example for the analysis of our crash using the NotmyFault driver.



An important feature of the debugger's output using !analyze –v is the stack text. Whenever looking at a dump file always look at the far right end of the stack for any third party drivers. In this case we see myfault. Note that the chronologic sequence of events goes from the bottom to the top; as each new task is performed by the system it shows up at the top, pushing the previous actions down. In this rather short stack you can see that myfault was active, then a page fault occurred, and the system declared a BugCheck which is when the system stopped (Blue Screened). Note that some data was removed to fit this exhibit on a page as indicated by the "truncated" comments).

```
STACK_TEXT:
fffff880`0279a578 fffff800`02a8cca9 : […truncated…] nt!KeBugCheckEx
fffff880`0279a580 fffff800`02a8b920 : […truncated…] nt!KiBugCheckDispatch+0x69
fffff880`0279a6c0 fffff880`06a01361 : […truncated…] nt!KiPageFault+0x260
fffff880`0279a850 00000000`00000000 : […truncated…] myfault+0x1361
```

Analysis with lmv

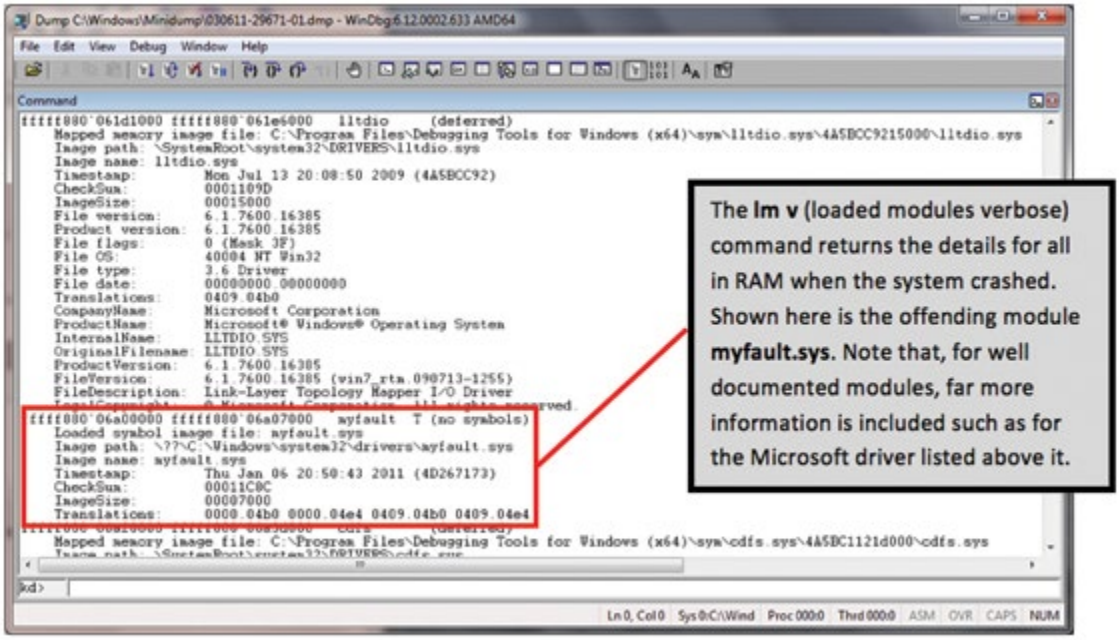
The next step is to confirm the suspect's existence and find any details about him. Typing lm in the command line displays the loaded modules; v instructs the debugger to output in verbose (detail) mode, showing all known details for the modules.

Don't worry if, after running the command lmv, you see the message *BUSY*in the bottom left of WinDbg's interface. This is because it is gathering detailed information for modules loaded when the system failed and it may take a couple of minutes. When done you will see kd> back where BUSY was.

This is a lot of information. Locating the driver of interest can take a while, so simplify the process by selecting

```
Edit | Find
```

and enter the suspect driver, in this case **myfault**. The amount of information you see depends upon the driver vendor. Some vendors put little information in their files; others such as Microsoft tend to be thorough.



Analysis with lmvm

A great way to get right to a specific module is the lmvm command. In this case, enter lmvm myfault and the debugger will only return data specific to that module.

```
kd> lmvm myfault
start             end                 module name
fffff880`06a00000 fffff880`06a07000   myfault  T (no symbols)
    Loaded symbol image file: myfault.sys
    Image path: \??\C:\Windows\system32\drivers\myfault.sys
    Image name: myfault.sys
    Timestamp:        Thu Jan 06 20:50:43 2011 (4D267173)
    CheckSum:         00011C8C
    ImageSize:        00007000
    Translations:     0000.04b0 0000.04e4 0409.04b0 0409.04e4
```

After you find the vendor's name, go to its Web site and check for updates, knowledge base articles, and other supporting information. If such items do not exist or do not resolve the problem, contact them. They may ask you to send along the debugging information (it is easy to copy the output from the debugger into an e-mail message or Word document) or they may ask you to send them the memory dump (zip it up first, both to compress it and protect data integrity).

The other third

Fortunately, in about two out of three cases you'll know the cause as soon as you open a dump file. But sometimes the information it provides is misleading or insufficient. What do you do then?

Sometimes it is the hardware

If you have recurring crashes but no clear or consistent reason, it may be a memory problem. Download the free test tool, Memtest86. This simple diagnostic tool is quick and works great. Many people discount the possibility of a memory problem, because they account for such a small percentage of system crashes. However, they are often the cause that keeps you guessing the longest.

Is Windows the culprit?

Sorry...this is NOT likely! As surprising as it may seem, the operating system is rarely at fault. If ntoskrnl.exe (Windows core) or win32.sys (the driver that is most responsible for the "GUI" layer on Windows) is named as the culprit, and they often are, don't be too quick to accept it. It is far more likely that some errant third-party device driver called upon a Windows component to perform an operation and passed a bad instruction, such as telling it to write to non-existent memory. So, while the operating system certainly can err, exhaust all other possibilities before you blame Microsoft.

Wrong driver named

Often you will see an antivirus driver named as the cause. For instance, after using !analyze -v, the debugger reports a driver for your antivirus program at the line "IMAGE_NAME". This may well be the case, but bear in mind that such a driver can be named more often than it is guilty. Here's why: For antivirus code to work it must watch all file openings and closings. To accomplish this, the code sits at a low layer in the operating system and is constantly working. In fact, it is so busy it will often be on the stack of function calls that was active when the crash occurred, even if it did not cause it. Because any third-party driver on that stack immediately becomes suspect, it will often get named. From a mathematical standpoint it is easy to see how it will so often be on the stack whether it actually caused a problem or not.

Missing vendor information?

Some driver vendors don't take the time to include sufficient information with their modules. So if lmv doesn't help, try looking at the subdirectories on the image path (if there is one). Often one of them will be the vendor name or a contraction of it. Another option is to search Google. Type in the driver name and/or folder name. You'll probably find the vendor as well as others who have posted information regarding the driver.

Summary

Now that you have taken the time to prepare for the next BSOD, remember that in most cases you will be able to open the dump file and know the cause in less than one minute. To nail the cause of two out of three critical failures that fast and that easily is gratifying - especially to your users.

Smith is a freelance consultant and writer in IT. He can be reached at Dirk@LandfallResearch.com.

Read more about software in Network World's Software section.