

Troubleshooting Windows Application Crashes Or Hangs

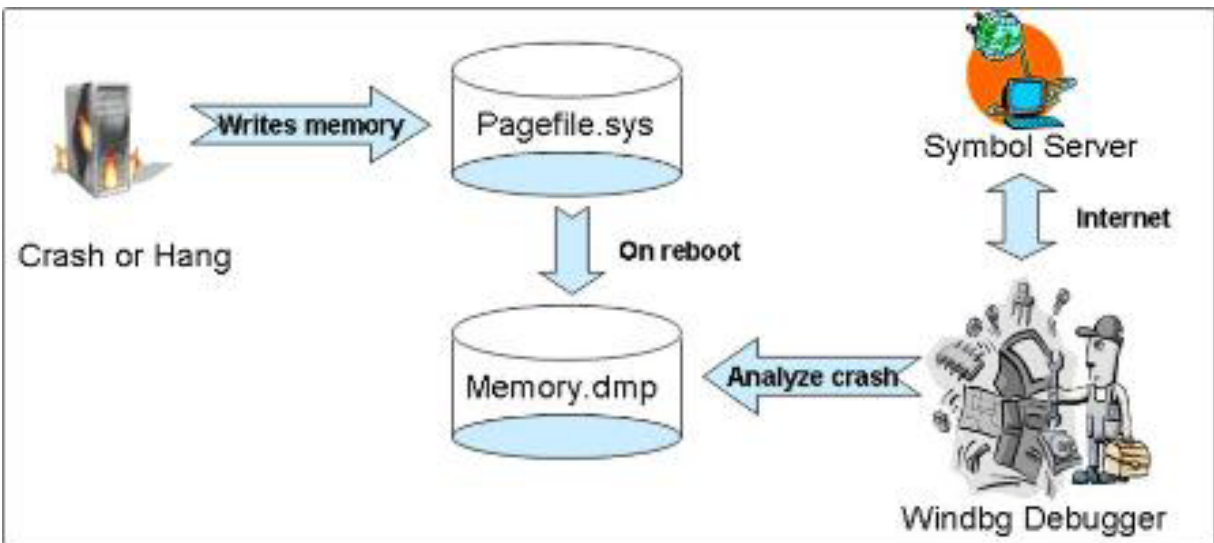
Bruce Mackenzie-Low

Crash, boom, bang! Your Windows server just experienced a Blue Screen of Death (BSOD) and your helpdesk is being flooded with calls. The server is rebooting, but this is the fourth crash you've encountered this week and users are becoming unruly. To top it off, you now face spending hours on the phone, being passed around the world, with each vendor pointing to the other as the culprit.

It's time to take matters into your own hands. With a basic knowledge of crash dump analysis, and a few simple commands, you can determine which driver is involved. Then, by intelligently searching the Internet you can potentially locate a hotfix or workaround to resolve the crashes.

This three-part series will cover the tools and steps you'll need to tackle some of the toughest Windows server outages.

To begin with, the diagram in Figure 1 provides an overview of what happens when a crash occurs. As you can see, when the server crashes it writes the contents of physical memory (RAM) to the pagefile on the system partition. On reboot, the pagefile is written to the memory.dmp file, which also resides on the system partition. Finally, after the server reboots, you can then use the Windows kernel debugger (WinDbg) with Microsoft's symbol server to analyze the crash.



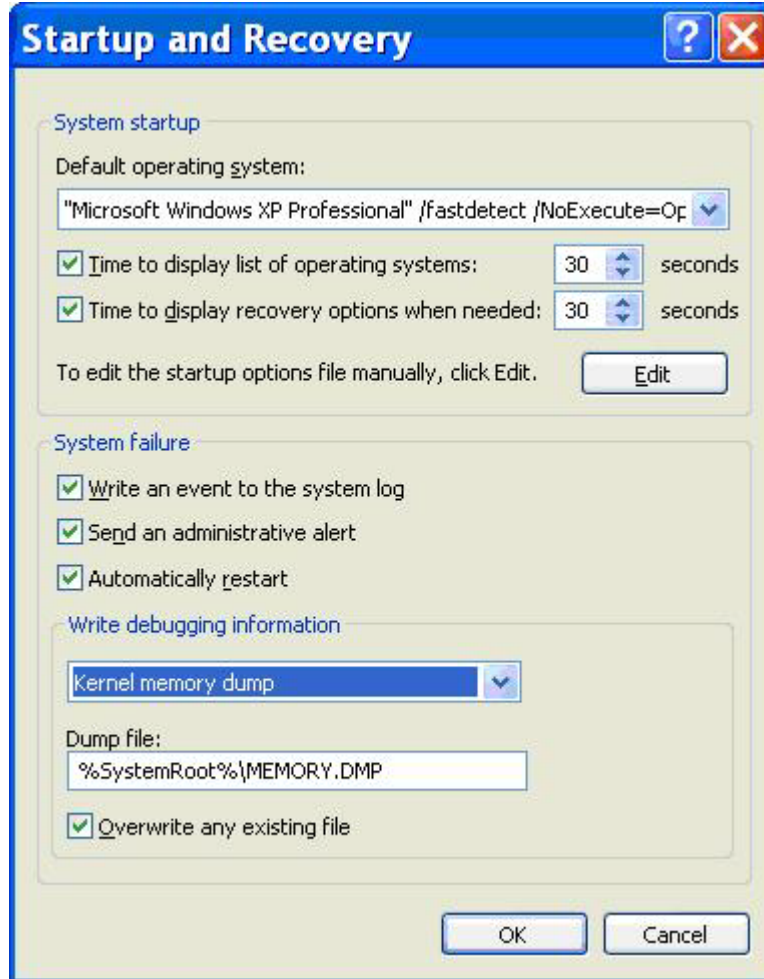
Three main areas need to be addressed to facilitate your crash dump analysis. First, the server must be configured to generate a crash when an unexpected condition or exception occurs. Next, you need to download the Windows debugger from Microsoft and set up the symbol server path. Finally, use the debugger to analyze the crash with a few simple commands. Now, let's take a closer look at each area.

Configuring the Dump

To configure your server to generate a crash, use the Control Panel | System applet | Advanced tab | Startup | Recovery settings shown in Figure 2. You can choose from three types of memory dump files: small, kernel or complete. By default, Windows will produce a small, "mini-dump" file when the server crashes. This may sometimes contain enough debugging information, but typically a kernel memory dump file is required. In rare circumstances, it may be necessary to configure a complete memory dump to capture the required debugging information. Please see Microsoft KB article 254649 for additional information on configuring memory dump files.

Troubleshooting Windows Application Crashes Or Hangs

Bruce Mackenzie-Low



Installing the Windows Debugger

The next step is to install the Windows kernel debugger tool, which can be downloaded for free from Microsoft. There are three versions of the debugger (x86, x64 and IA64), depending on the architecture of the server where you plan to analyze the crash. Once WinDbg is installed, you must establish the symbol path to translate memory locations into meaningful references to functions or variables used by Windows. The typical symbol path used is:

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

Analyzing the Crash

Now that you have configured the server to generate a memory dump and installed the debugger with the correct symbol path, you are ready to analyze a crash. There are two ways to start up the debugger: from the program group "Debugging Tools for Windows" or from the DOS prompt with the WinDbg command. From within the debugger, use the File pull-down menu to "Open crash dump..." and point the debugger to your dump file.

When the dump file loads, you will notice the debugger's screen is divided into two regions: the output pane that occupies the majority of the window and the command prompt at the bottom. The first command to use is:

Troubleshooting Windows Application Crashes Or Hangs

Bruce Mackenzie-Low

```
!analyze -v
```

This command will perform a preliminary analysis of the dump and provide you with a best guess as to which driver caused the crash. The first thing the command shows you is the bug check type (also known as a stop code) and the arguments. The bug check type is very important and should be included with your query when you search the Internet for possible causes and fixes. As we see in the following example, WinDbg displays the bug check type as an LM_SERVER_INTERNAL_ERROR (stop code 54). In this case, if you searched the Microsoft website for LM_SERVER_INTERNAL_ERROR, you would find the known issue and hotfix documented in Microsoft KB 912947. Even the first argument matches the KB article.

```
3: kd> !analyze -v *****
*                               Bugcheck Analysis                               *
*****

LM_SERVER_INTERNAL_ERROR (54)
Arguments:
Arg1: 00361595
Arg2: e8aab501
Arg3: 00000000
Arg4: 00000000
```

The !analyze -v command goes on to list which driver caused the crash. In our example, WinDbg accurately calls out the srv.sys driver that caused the crash:

```
Probably caused by: srv.sys (srv!SrvVerifyDeviceStackSize+78 )
```

Several other useful commands provide more information about the crash, including:

- !thread – lists the currently executing thread
- kv – displays the stack trace indicating which drivers and functions were called
- !m t n – displays the list of installed drivers and their dates

Finally, you should be aware that the Windows debugger's online help is excellent. In particular, you can look up the stop code for the crash and use the online help to recommend how to troubleshoot the issue. To find the list of stop codes, go to the Help pull-down menu and select Contents | Debugging Techniques | Bug Checks (Blue Screens) | Bug Check Code Reference. Then scan down the list to locate your stop code.

Many people think debugging a crash is better left for those with Ph.D.'s, but with a basic understanding and a few simple commands, anyone can get a leg up on identifying what is contributing to or causing a server crash. It is likely that someone else out there has already experienced the same crash, so a thorough Internet search will probably lead to potential workarounds or patches for the issue.

Troubleshooting Windows Print Spooler Crashes

With the vast variety of printers and drivers on the market today, it's a daunting task to determine which one caused your print spooler to crash or hang. Hundreds of users can be affected by a single rogue print driver that seldom leaves any clues as to the cause. This article will tackle how you can determine which print driver caused your spooler to crash.

Troubleshooting Windows Application Crashes Or Hangs

Bruce Mackenzie-Low

Overview

The process of troubleshooting a print spooler crash is very similar to troubleshooting a system crash, as discussed in part one of this series. A print spooler, however, may not generate a crash dump on its own, so a tool called ADPlus is used to capture the memory dump. ADPlus is a VB script that can be downloaded for free from Microsoft as part of the Debugging Tools for Windows. Once you install the debugging tools, you will find ADPlus.vbs in the following folder:

Program Files\Debugging Tools for Windows

ADPlus can be used in two modes depending on whether your print spooler is hanging or crashing. In hang mode, ADPlus forces a process dump on an application, or in this case, a print spooler. The dump contains all of the threads associated with the process in addition to the various DLLs and print drivers involved. A few simple debugger commands allow you to determine which printer is being accessed by the spooler and its corresponding driver.

In crash mode, ADPlus will monitor a process and capture its memory dump when it experiences an unhandled condition. The main difference between the two modes is that crash mode must be established prior to the process terminating, whereas hang mode is used at the moment the process locks up. In either mode, only the process you are debugging is affected; the rest of the processes and the operating system continue without downtime.

Once a process dump is captured, you can then use the Windows Debugger (Windbg) to analyze the failure. As discussed in part one, the debugger can also be downloaded for free from Microsoft as part of the Debugging Tools for Windows.

In the following sections, we'll take a closer look at the steps required to capture a spooler dump, determine which print driver is the culprit and ultimately repair the problem.

Crash Mode

As mentioned above, ADPlus crash mode captures a process memory dump when your print spooler is intermittently terminating. Crash mode must be established prior to the problem that is causing the print spooler failure. The very first time you use ADPlus you must establish cscript as the default script interpreter. To accomplish this, open a command prompt and change your default to the Debugging Tools for Windows folder. Then execute the ADPlus.vbs script without any options:

```
C:\Program Files\Debugging Tools for Windows > ADPlus.vbs
```

You only need to perform this step once; you are then ready to use ADPlus to capture a spooler crash. Here we see the ADPlus syntax used to set up crash mode detection on the print spooler process:

```
Adplus -crash -pn spoolsv.exe
```

This command will attach the console debugger (cdb.exe) to the print spooler process and minimize the window. Once an unexpected condition is encountered, the debugger will produce a process memory dump and terminate the process. By default, the dump is written to a subfolder in the Debugging Tools for Windows folder. You can then use the Windows Kernel Debugger to analyze the resulting dump file.

Troubleshooting Windows Application Crashes Or Hangs

Bruce Mackenzie-Low

Hang Mode

In hang mode, use ADPlus to force a process memory dump when a print spooler either stops responding or becomes 100% compute-bound. This is evident when users complain that their jobs aren't printing even though the spooler process still exists. After forcing the process memory dump, ADPlus hang mode will resume the process instead of terminating it like in crash mode. Here we see the ADPlus syntax used to force a process crash with hang mode:

```
Adplus -hang -pn spoolsv.exe  
Analyzing the dump
```

Once the process dump file has been obtained, use the Windbg tool to analyze the print spooler failure. After installing Windbg, the first step to using the tool is to establish the debugger's symbol path to point to the Microsoft Symbol Server. Next, open the crash dump file with Windbg using the File pull-down menu, Open Crash Dump..., and then issue the command:

```
!analyze -v
```

This command will perform a preliminary analysis of the dump and provide a best guess as to what caused the failure. The kv command will display the stack trace showing you which drivers or DLLs are involved. A stack trace is read from the bottom up so the top of the stack is the most recently executed function. In the following example, we see a stack trace illustrating a spooler failure caused by the ABCdriver:

```
0:036> kv  
ChildEBP RetAddr  Args to Child  
  
0139e2b8 01cb8e03 0016daf0 018f0000 018f0006 ABCdriver!DrvUpgradePrinter+0x329b3  ←failing routine  
0139e4e8 01cb94e7 0016daf0 0139e534 01cb8bbc ABCdriver!DrvUpgradePrinter+0x32a93  
0139e4f4 01cb8bbc 01c831e8 01c8274c 0139e514 ABCdriver!DrvUpgradePrinter+0x33177  
0139e534 01cad37e 01c831e8 01c8274c 0139e514 ABCdriver!DrvUpgradePrinter+0x3284c  
0139e738 01c84fc3 028a1b00 01903330 0139e74c ABCdriver!DrvUpgradePrinter+0x277a7  
0139e88c 01c8557b 028b9a74 00000000 0139e910 ABCdriver!DrvDevicePropertySheets+0x913  
0139e8a0 01c85610 0139e910 0139e8e8 0139e910 ABCdriver!DrvDevicePropertySheets+0xecb  
0139e8dc 73076633 00000000 0139e910 00000000 ABCdriver!DrvDocumentPropertySheets+0x40  
0139e8fc 730765b2 00000000 0139e910 0205e148 winspool!DocumentPropertySheets+0x13a  
0139e934 10008a14 00000000 028b9a74 0205e148 winspool!DocumentPropertiesWNative+0xe6  
0139e958 10008878 77e6ba30 013fda40 0139ee08 XYZdriver!InitializePrintMonitor+0x4b14  
0139e994 10004342 00c85e78 000000cb 00000001 XYZdriver!InitializePrintMonitor+0x4978  
0139e9a8 761460a6 020713b0 00c85e78 000000cb XYZdriver!InitializePrintMonitor+0x442  
0139ee7c 00000000 014051e0 0000031f 00000000 localspl!PrintingDirectlyToPort+0x200
```

Another useful command is !peb, which allows you to see all of the drivers and DLLs associated with the print spooler process. The command displays the process environment block as we see in the following example. Much of the output has been omitted [...] as it goes on for several pages:

Troubleshooting Windows Application Crashes Or Hangs

Bruce Mackenzie-Low

```
0:036> !peb
PEB at 7ffd4000
  InheritedAddressSpace: No
  ReadImageFileExecOptions: No
  BeingDebugged: Yes
  ImageBaseAddress: 01000000
[...]
```

	Base	TimeStamp		Module
	1000000	42435e25	Mar 24 18:41:09 2005	C:\WINDOWS\system32\spoolsv.exe
	7c800000	424377d2	Mar 24 20:30:42 2005	C:\WINDOWS\system32\ntdll.dll
[...]				
	73070000	424377d1	Mar 24 20:30:41 2005	C:\WINDOWS\system32\winspool.drv
	10000000	45460890	Oct 30 08:13:36 2006	C:\WINDOWS\system32\xyz.dll
	1150000	4616295d	Apr 06 05:05:01 2007	C:\WINDOWS\system32\spool\PRTPROCS\W32X86\abc.DLL
	1190000	45d1d08e	Feb 13 07:51:58 2007	C:\WINDOWS\system32\spool\PRTPROCS\W32X86\abc2.DLL
	74030000	424377f3	Mar 24 20:31:15 2005	C:\WINDOWS\system32\win32spl.dll

```
[...]
```

```
WindowTitle: 'C:\WINDOWS\system32\spoolsv.exe'
ImageFile: 'C:\WINDOWS\system32\spoolsv.exe'
CommandLine: 'C:\WINDOWS\system32\spoolsv.exe'
DllPath: C:\WINDOWS\system32;C:\WINDOWS\system32;C:\WINDOWS;. ;C:\Program Files\Windows...
Environment: 00010000
  ALLUSERSPROFILE=C:\Documents and Settings\All Users
  ClusterLog=C:\WINDOWS\Cluster\cluster.log
  CommonProgramFiles=C:\Program Files\Common Files
  COMPUTERNAME=PrintServer
[...]
```

Finally, to determine the printer and job that is being accessed at the time of the failure, use the !teb command. That will display the thread environment block that provides the stack base and limit. You can then display the stack contents with the dc command to reveal the printer that is causing the problem. You will have to scroll through several pages of output, but you will eventually recognize the printer, job and port number in ASCII text to the right:

```
0:036> !teb
TEB at 7ffa4000
  ExceptionList: 0139e28c
  StackBase: 013a0000 ← base of stack
  StackLimit: 01398000 ← top of stack
[...]
```

```
0:036> dc 01398000 013a0000
01398000 00000000 00000000 00000000 00000000 .....
01398010 00000000 00000000 00000000 00000000 .....
[...]
```

0139f300	00000001	00520050	004e0049	00450054	...P.R.I.N.T.E.	← Printer name
0139f310	0044004f	00310052	0000002c	006f004a	R.1.....J.o.	
0139f320	00200062	00300032	00000033	00340030	b. .2.0.3...0.4.	
0139f330	0020002c	006f0050	00740072	00000000	,. .P.o.r.t.....	

Troubleshooting Windows Application Crashes Or Hangs

Bruce Mackenzie-Low

In this case, the printer name is PRINTER1, the job number is 203, and the port number is 04. The stack contents also contain the associated driver name if you look closely. Once you know the printer and the driver, you can contact the appropriate vendor to determine if an updated driver is available that resolves your issue.

As you can see, troubleshooting a print spooler failure is straightforward once you become familiar with the tools. Starting with ADPlus to capture the dump, then using Windbg to analyze it, and finally leveraging the Web to intelligently search for similar crash footprints will lead you to your solution. Taking matters into your own hands will save you time, money and keep your users happy.

Simple Tools and Techniques for Finding Windows Memory Leaks

As we continue our series on tackling the toughest Windows server outages, the time has come to explore the different tools and techniques used to track down Windows memory leaks.

As you may know, memory leaks are caused by poorly written applications or drivers that allocate memory and then subsequently fail to de-allocate all of it. After time, this can lead to the depletion of system memory pools (paged or non-paged) causing the server to eventually hang.

Long before a Windows server hangs though, there are typically other symptoms of a memory leak. The main things to watch out for are entries in the system event log from the server service (SRV component). In particular, be on the lookout for:

Event ID 2019: The server was unable to allocate from the system nonpaged pool because the pool was empty

or

Event ID 2020: The server was unable to allocate from the system paged pool because the pool was empty

These two events are indicative of a Windows memory leak and need to be investigated immediately. Other signs of a memory leak include excessive pagefile utilization and diminishing available memory.

Perfmon

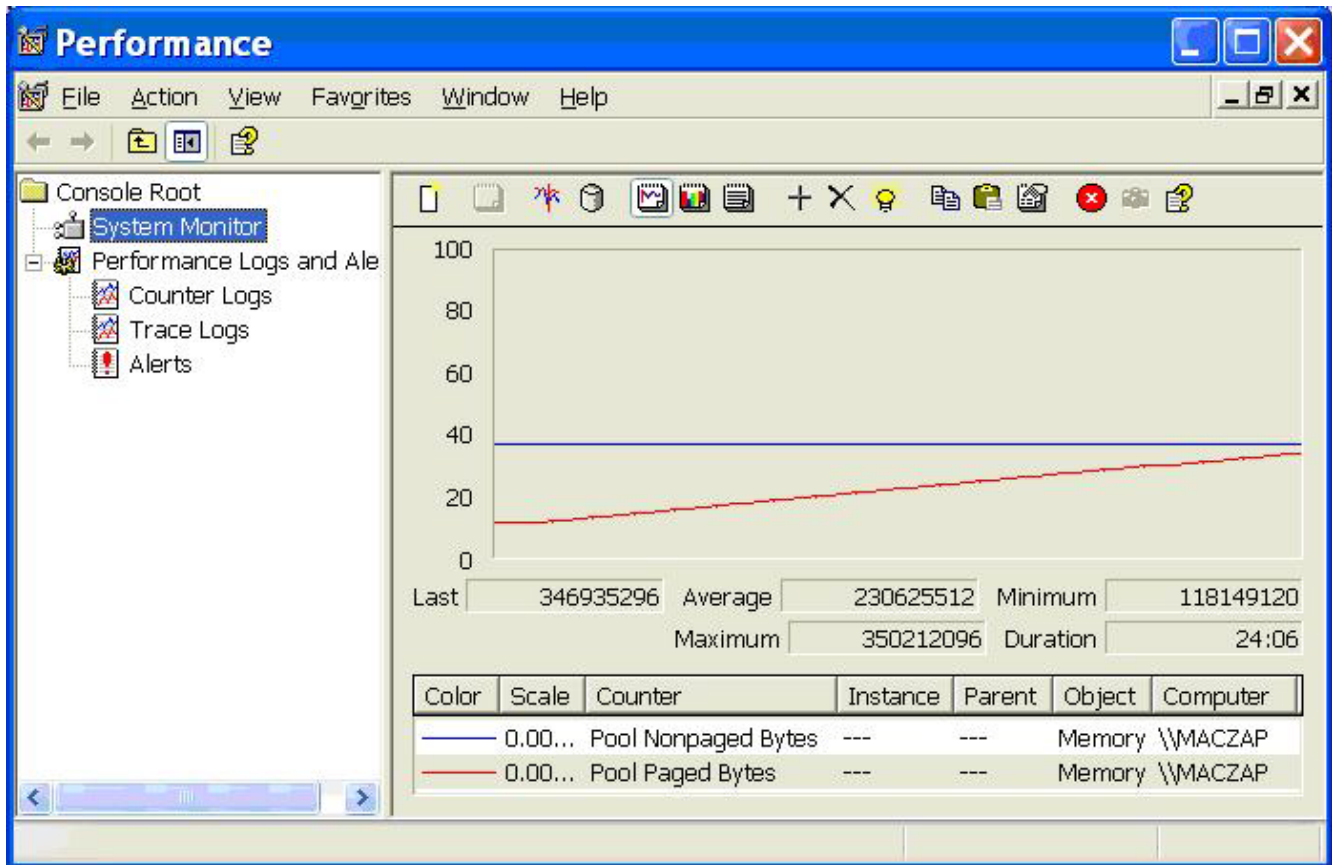
The first tool typically used to diagnose memory leaks is Perfmon, a graphical tool built into Windows. By collecting performance metrics on the appropriate counters, you can determine whether the memory leak is being caused by a user process (application) or a kernel mode driver. The performance metrics can be collected in the background with the counters being written to a log file. The log file can subsequently be read by Perfmon or the Performance Analysis of Logs (PAL) from CodePlex. Microsoft KB article 811237 explains how to setup Perfmon to log performance counters. There is also a free tool called PerfWiz from Microsoft which provides a wizard to help setup Perfmon logging.

If you suspect a user mode application is leaking memory, you can use Perfmon to collect the Process object counters, Pool Paged Bytes and Pool Nonpaged Bytes for all instances. This will display whether any processes continue to allocate paged or non-paged pool, without subsequently de-allocating it. If you suspect a kernel mode driver is leaking memory, use Perfmon to collect the Memory object counters, Pool Nonpaged Bytes and Pool Paged Bytes.

Troubleshooting Windows Application Crashes Or Hangs

Bruce Mackenzie-Low

In the following example, Perfmon is being used to monitor performance counters for the memory object, namely paged and non-paged pool. By right-clicking each counter, you can adjust the scale to have both counters appear on the same graph. As you can see in Figure 1, the Pool Paged Bytes counter (red line) continues to grow without decreasing, meaning it is leaking memory. Looking at the minimum value for the paged pool counter, it appears it has gone from a value of 118 MB to a maximum value of over 350 MB.



So at this point in our example, we know we have a paged pool leak. We can then use Perfmon to examine the Process object for Pool Paged Bytes. If no processes show a corresponding increase in paged pool usage, we can conclude that a driver or kernel mode code is leaking memory.

Poolmon

To further isolate the memory leak, we need to determine which driver is allocating the memory. When drivers allocate memory, they insert a four-character tag into the memory pool data structure to identify which driver allocated it. By examining the various pool allocations, you can determine which drivers are responsible for allocating how much pool. To associate which tags correspond to certain drivers, see Microsoft KB article 298102. You could also install the Debugging Tools for Windows and check the following file:

`\\Program Files\\Debugging Tools for Windows\\Triage\\Pooltag.txt`

The Memory Pool Monitor utility (Poolmon) is a free tool from Microsoft that will watch pool allocations and display the results illustrating the corresponding drivers. In the following example, Poolmon is

Troubleshooting Windows Application Crashes Or Hangs

Bruce Mackenzie-Low

being used to track the leaking pool tag "Leak" at the top of the list. Poolmon shows the number of allocations, number of frees, the difference, and the number of bytes allocated. Poolmon will also show the name of the driver if it is setup properly.

Here we can see the tag "Leak" belongs to the Notmyfault.sys driver and has over 83 MB of paged pool allocated.

```
Memory: 2096624K Avail: 1347912K PageFlts: 384 InRam Krn1: 3560K P:280448K
Commit: 612732K Limit:8231732K Peak: 615868K Pool N:12284K P:300188K
System pool information
Tag Type Allocs Frees Diff Bytes Per alloc Mapped_Driver
Leak Paged 40435 < 185> 0 < 0> 40435 83134360 < 380360> 2056 [Notmyfault]
Save Paged 841408 < 0> 840771 < 0> 637 79784232 < 0> 125249 Unknown Driver
Wmit Paged 1622 < 0> 8 < 0> 1614 26772768 < 0> 16587 Unknown Driver
UIHT Paged 1 < 0> 0 < 0> 1 8392704 < 0> 8392704 Unknown Driver
GH05 Paged 22364 < 16> 21837 < 16> 527 4002936 < 0> 7247 Unknown Driver
CM35 Paged 87 < 0> 24 < 0> 63 3395584 < 0> 53898 Unknown Driver
Itfd Paged 4894 < 0> 4011 < 0> 883 1657136 < 0> 1876 Unknown Driver
MmSt Paged 6081 < 0> 4782 < 0> 1299 1529352 < 0> 1177 Unknown Driver
SACC Paged 250 < 0> 0 < 0> 250 1000968 < 0> 4035 Unknown Driver
Wdm Paged 311 < 0> 286 < 0> 25 730232 < 0> 29209 [cinemst2][ess][ichaud][LHIDUSB]
PTR3 Paged 4216 < 0> 1808 < 0> 2408 712768 < 0> 296 Unknown Driver
Ntff Paged 2809 < 0> 2022 < 0> 787 654784 < 0> 832 [ntfs]
CMUa Paged 284200 < 0> 272929 < 0> 11271 542944 < 0> 48 Unknown Driver
Gla1 Paged 352 < 0> 44 < 0> 308 492800 < 0> 1600 Unknown Driver
R300 Paged 28 < 0> 4 < 0> 24 429896 < 0> 17912 Unknown Driver
Ntff Paged 1175 < 0> 769 < 0> 406 383264 < 0> 944 Unknown Driver
Ohtb Paged 255 < 0> 102 < 0> 153 373680 < 0> 2442 Unknown Driver
Ggb Paged 162 < 0> 126 < 0> 36 350992 < 0> 9749 Unknown Driver
CMDa Paged 35925 < 0> 33568 < 0> 2357 349584 < 0> 148 Unknown Driver
SAU Paged 139535 < 0> 138997 < 0> 538 319952 < 0> 594 Unknown Driver
Geac Paged 311 < 0> 248 < 0> 63 309608 < 0> 4914 Unknown Driver
Gla5 Paged 1060 < 0> 275 < 3> 785 307720 < -1176> 392 Unknown Driver
CM01 Paged 1801 < 0> 1727 < 0> 74 303104 < 0> 4096 Unknown Driver
Ioke Paged 13301 < 18> 13108 < 18> 193 296128 < 0> 1534 Unknown Driver
NtFB Paged 146 < 0> 121 < 0> 25 277608 < 0> 11104 [ntfs]
CM16 Paged 69 < 0> 3 < 0> 66 270336 < 0> 4096 Unknown Driver
Key Paged 309473 < 70> 307263 < 70> 2210 265144 < 0> 119 Unknown Driver
CM25 Paged 1348 < 0> 1336 < 0> 12 196608 < 0> 16384 Unknown Driver
```

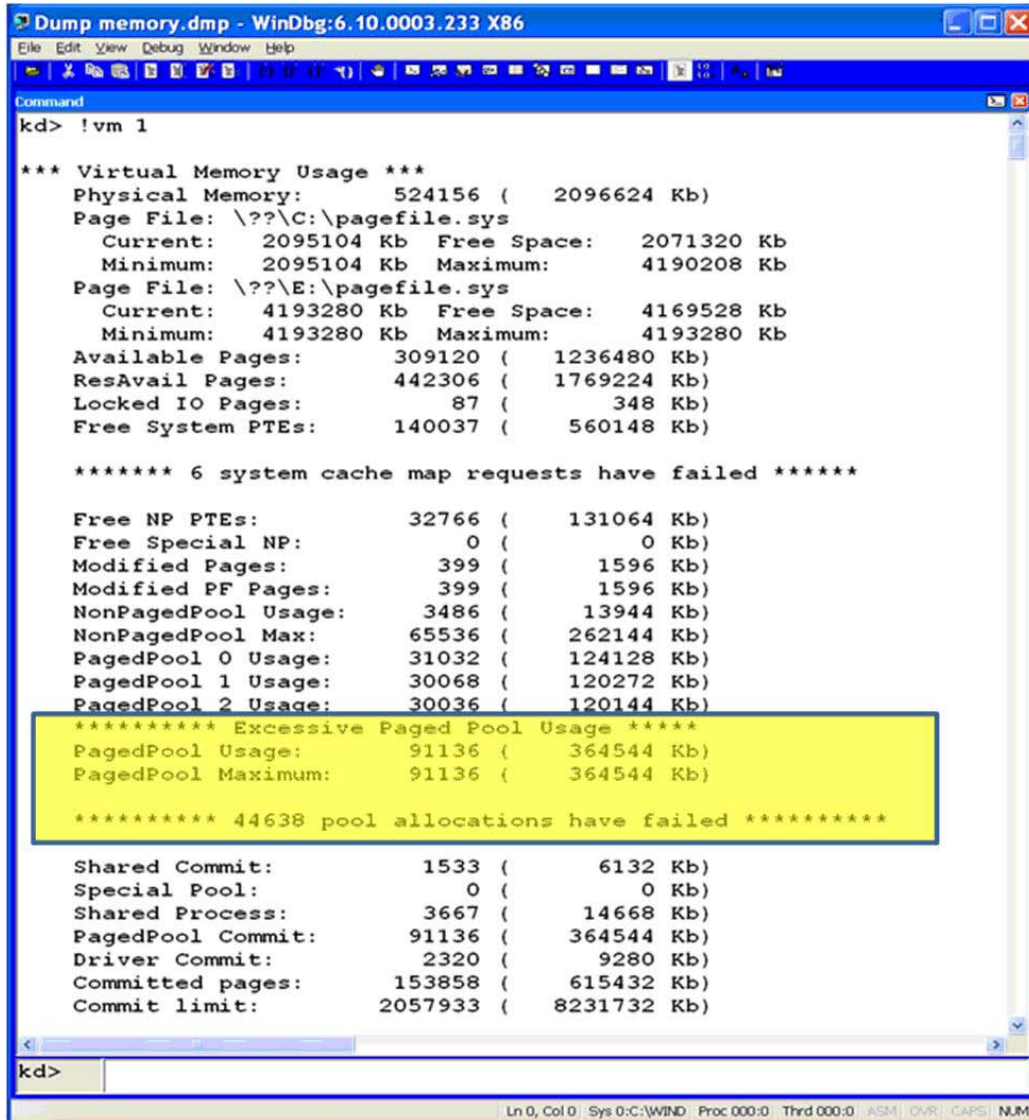
Windbg

If all else fails and your server locks up completely due to a memory leak, you can always force a crash dump and subsequently analyze it as discussed in my previous article on why Windows servers hang. The key things to look for when analyzing the crash with the Windows Kernel Debugger (Windbg) utility are the memory pool usage and which data structures are consuming the pool.

The first command to use in the debugger is !vm 1, as seen in the following example. This command will display the current virtual memory usage, in particular the non-paged and paged pool regions. The debugger will flag any excessive pool usage and any pool allocation failures as shown in Figure 3. The trick is to compare the usage with the maximum as highlighted in yellow below. If the usage is at or near the maximum, then the server hung because it ran out of pool.

Troubleshooting Windows Application Crashes Or Hangs

Bruce Mackenzie-Low



The screenshot shows a WinDbg window titled "Dump memory.dmp - WinDbg:6.10.0003.233 X86". The command window displays the output of the '!vm 1' command, which shows virtual memory usage statistics. A yellow highlight box is drawn around the "Excessive Paged Pool Usage" section, which includes the text "***** Excessive Paged Pool Usage *****", "PagedPool Usage: 91136 (364544 Kb)", "PagedPool Maximum: 91136 (364544 Kb)", and "***** 44638 pool allocations have failed *****".

```
kd> !vm 1

*** Virtual Memory Usage ***
Physical Memory:      524156 ( 2096624 Kb)
Page File: \??\C:\pagefile.sys
Current: 2095104 Kb Free Space: 2071320 Kb
Minimum: 2095104 Kb Maximum: 4190208 Kb
Page File: \??\E:\pagefile.sys
Current: 4193280 Kb Free Space: 4169528 Kb
Minimum: 4193280 Kb Maximum: 4193280 Kb
Available Pages:      309120 ( 1236480 Kb)
ResAvail Pages:       442306 ( 1769224 Kb)
Locked IO Pages:      87 ( 348 Kb)
Free System PTEs:     140037 ( 560148 Kb)

***** 6 system cache map requests have failed *****

Free NP PTEs:         32766 ( 131064 Kb)
Free Special NP:      0 ( 0 Kb)
Modified Pages:       399 ( 1596 Kb)
Modified PF Pages:    399 ( 1596 Kb)
NonPagedPool Usage:   3486 ( 13944 Kb)
NonPagedPool Max:     65536 ( 262144 Kb)
PagedPool 0 Usage:    31032 ( 124128 Kb)
PagedPool 1 Usage:    30068 ( 120272 Kb)
PagedPool 2 Usage:    30036 ( 120144 Kb)

***** Excessive Paged Pool Usage *****
PagedPool Usage:      91136 ( 364544 Kb)
PagedPool Maximum:    91136 ( 364544 Kb)

***** 44638 pool allocations have failed *****

Shared Commit:        1533 ( 6132 Kb)
Special Pool:          0 ( 0 Kb)
Shared Process:        3667 ( 14668 Kb)
PagedPool Commit:     91136 ( 364544 Kb)
Driver Commit:        2320 ( 9280 Kb)
Committed pages:      153858 ( 615432 Kb)
Commit limit:         2057933 ( 8231732 Kb)

kd>
```

Finally, you can use the debugger to display the paged or non-paged pool data structures with the !poolused command. Various options on the command allow you to specify either paged or non-paged pool and sort the output. In the following example, the !poolused 5 command is used to display the paged pool data structures, sorted in descending order by usage. In Figure 4, you can see the pool structure with the tag "Leak" is consuming the most paged pool (over 115 MB) and is associated with the notmyfault.sys driver.

Troubleshooting Windows Application Crashes Or Hangs

Bruce Mackenzie-Low

```
kd> !poolused 5
Sorting by Paged Pool Consumed

Pool Used:
NonPaged          Paged
Tag  Allocs  Frees  Diff  Used  Allocs  Frees  Diff  Used
Leak  0        0      0      0    56242   0      56242 115633552 Sysinternals , Binary: notmyfault.sys
SaveE  8        4      4     224 1037906 1037269 637 79784232 UNKNOWN pooltag 'SaveE', please update pool
Wmit   8        0      8    32768 1621    8     1613 26707232 Wmi Trace
UlHT   0        0      0      0      1      0      1     8392704 Hash Table , Binary: http.sys
Gh05   0        0      0      0    14224 13765  459 3812968 GDITAG_HMGR_SPRITE_TYPE , Binary: win32k.sys
CM35   0        0      0      0      91     28     63 3395584 Internal Configuration manager allocations
MmSt   0        0      0      0    28867 27770 1097 1308824 Mm section object prototype ptes , Binary:
SrHK   0        0      0      0     6784   0     6784 1178552 Hash key , Binary: sr.sys
SACC   1        0      1     8192   250    0     250 1008968 UNKNOWN pooltag 'SACC', please update pool
Wdm    2344    1510   834 662416 284    260   24 730200 WDM
Ntff   22      0     22  4576 8367  7518 849 706368 FCB_DATA , Binary: ntfs.sys
SrHB   0        0      0      0    3110   0    3110 696640 Hash bucket , Binary: sr.sys
PTr3   0        0      0      0    3369 1213  2156 638176 UNKNOWN pooltag 'PTr3', please update pool

kd>
```

As you can see, using tools such as Perfmon, PerfWiz, PAL, Poolmon and WinDbg, you can monitor the memory leak, determine whether it is paged or non-paged memory, and discover what driver or application is responsible. After that, contacting the software vendor is usually the best option to see if they have an updated driver or image available that resolves the memory leak.