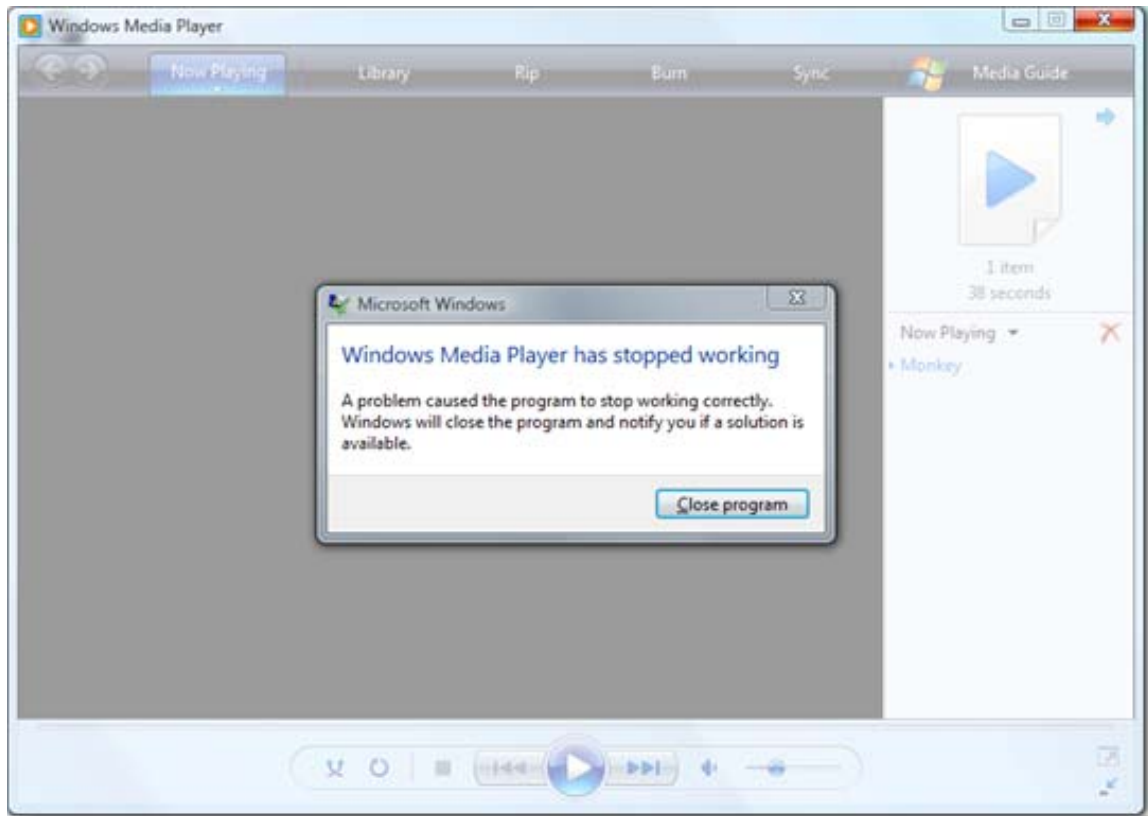


The Case of the Random IE and WMP Crashes

Mark Russinovich
(From Mark Russinovich Blog)

When I experienced a crash in Internet Explorer (IE) on my home 64-bit gaming system one day, I chalked it up to random third-party plug-in memory corruption. I moved on, but a few days later had another crash in IE. Then, Windows Media Player (WMP) started crashing every third or fourth time I used it:

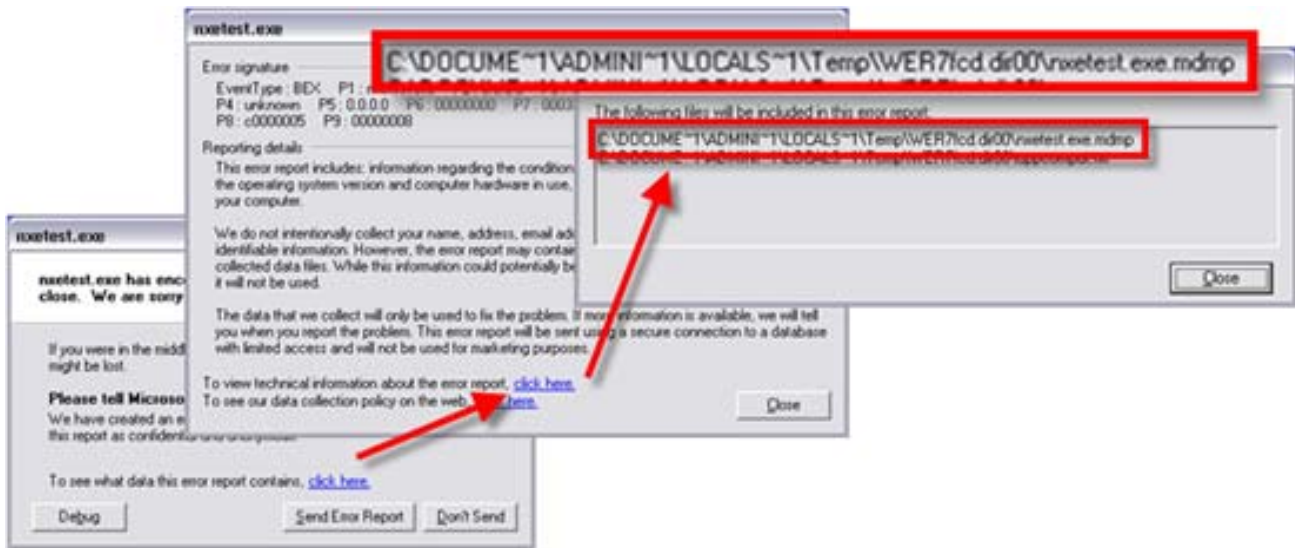


Crashes in different programs seemed to point at a more fundamental problem. I had over-clocked the CPU, so I speculated that the rash of crashes were a side-effect of CPU overheating and reluctantly dialed back the clock multiplier to the factory specification. To my dismay, however, the crashes continued. My next theory was that I had bad RAM, but the Windows Vista Memory Diagnostic failed to identify any problems.

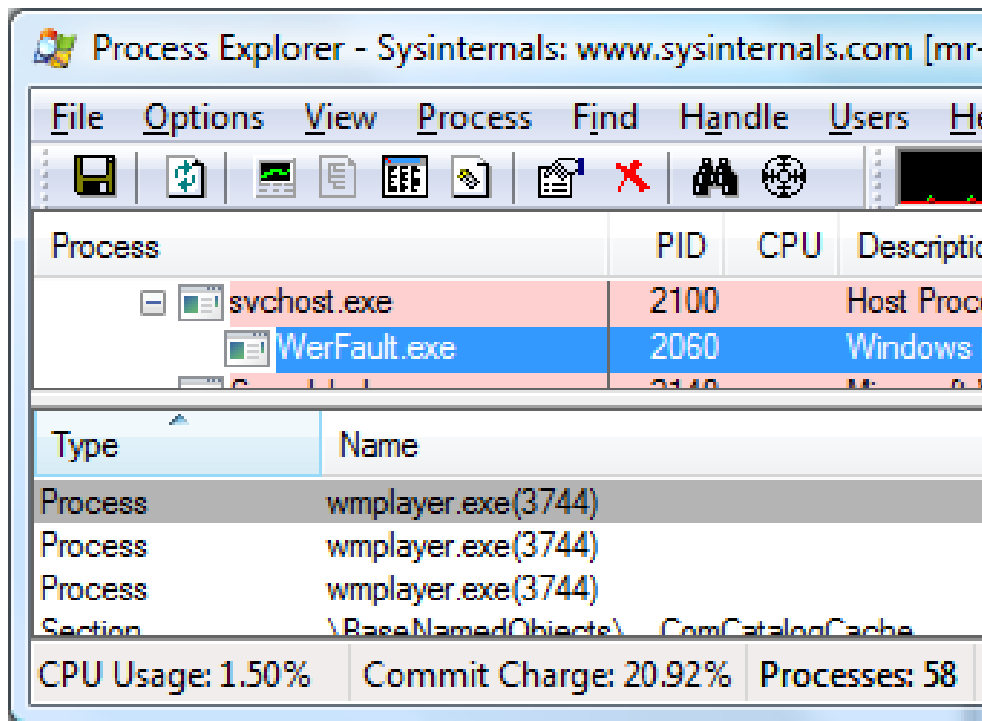
Hardware problems seemingly cleared, my next move was to look at the process crash dumps to see if they held any clues. But first I had to find a crash dump to look at. Windows XP's Application Error Reporting process always generates a dump before showing you the application crash dialog, and you can find the location of the dump by clicking to see the report details and then viewing the report's technical information:

The Case of the Random IE and WMP Crashes

Mark Russinovich
(From Mark Russinovich Blog)



Windows Vista's corresponding dialog doesn't offer a way to get at a report's technical information and it doesn't generate a dump unless Microsoft's Windows Error Reporting (WER) servers request it, which they only do for crashes reported in high volumes. Fortunately, WerFault, the process that presents the dialog, keeps the crashed process around until you press the Close Program button, which offers an opportunity to attach to the process with a debugger and examine it. You can see WerFault's handle to a crashed Windows Media Player process in Process Explorer:

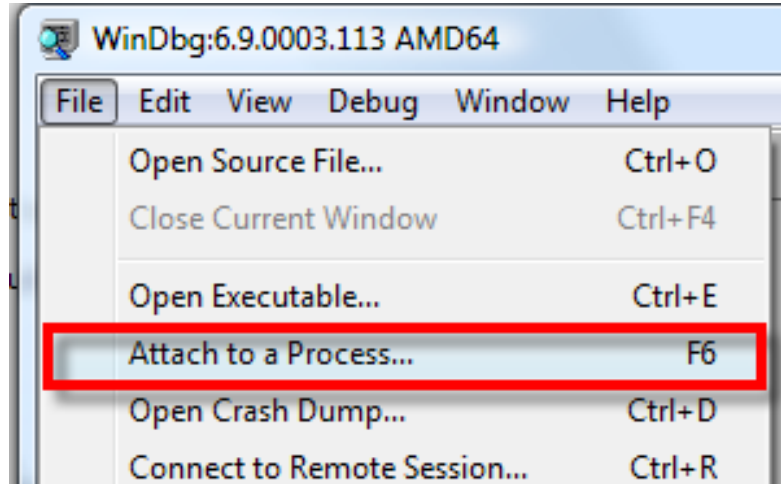


The next time I had a crash, I launched WinDbg, the Windows Debugger from the Debugging Tools for Windows package that's available for free download from Microsoft. After making sure that I had

The Case of the Random IE and WMP Crashes

Mark Russinovich
(From Mark Russinovich Blog)

the symbol configuration set to point at the Microsoft public symbol server (e.g. `srv*c:\symbols*http://msdl.microsoft.com/download/symbols`) in the Symbol File Path dialog, I went to the File menu and selected the "Attach to a Process..." menu entry:



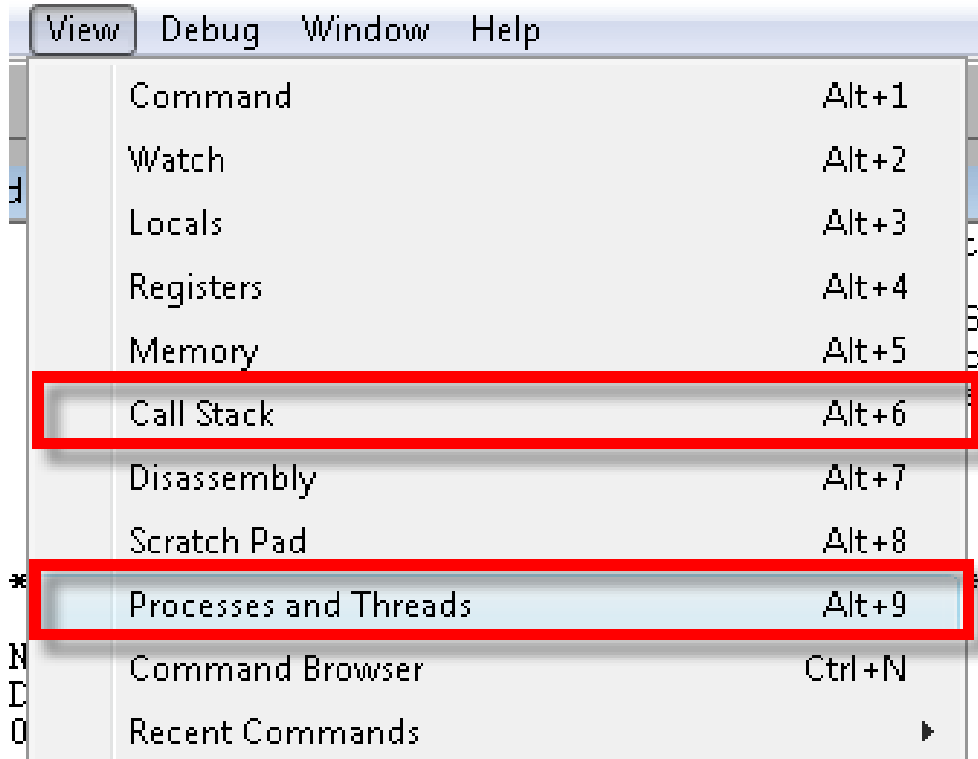
That opens the WinDbg process selection dialog, which I scrolled through to find the crashed process. When I selected the process, WinDbg opened it and presented the same interface it does when it loads a crash dump, except that when you load a crash dump, you can execute the `!analyze` debugger command that uses heuristics to try and pinpoint the cause of the crash; when you perform a debugger attach, an analysis will just tell you what you already know, that you attached with a debugger:

```
PROCESS_NAME: wmpplayer.exe
ERROR_CODE: (NTSTATUS) 0x80000003 - (EXCEPTION) Breakpoint A breakpoint has been reached.
NTGLOBALFLAG: 0
APPLICATION_VERIFIER_FLAGS: 0
FAULTING_THREAD: 00000da4
STACK_TEXT:
0588fbf0 77afd8d0 05889cc7 00000000 00000000 ntdll!DbgBreakPoint
0588fc20 76d619f1 00000000 0588fc6c 77b0d109 ntdll!DbgUiRemoteBreakin+0x3c
0588fc2c 77b0d109 00000000 05889c8b 00000000 kernel32!BaseThreadInitThunk+0xe
0588fc6c 00000000 77afd894 00000000 00000000 ntdll!_RtlUserThreadStart+0x23
STACK_COMMAND: ~~~[0x00000DA4]s ; kb ; ~43s ; .ecxr ; kb
PRIMARY_PROBLEM_CLASS: STATUS_BREAKPOINT
BUGCHECK_STR: APPLICATION_FAULT_STATUS_BREAKPOINT
FOLLOWUP_IP:
ntdll!DbgBreakPoint+0
77aa0004 cc int 3
SYMBOL_STACK_INDEX: 0
SYMBOL_NAME: ntdll!DbgBreakPoint+0
FOLLOWUP_NAME: MachineOwner
MODULE_NAME: ntdll
IMAGE_NAME: ntdll.dll
DEBUG_FLR_IMAGE_TIMESTAMP: 4549bdf8
BUCKET_ID: MANUAL_BREAKIN
FAILURE_BUCKET_ID: STATUS_BREAKPOINT_80000003_ntdll.dll!DbgBreakPoint
Followup: MachineOwner
```

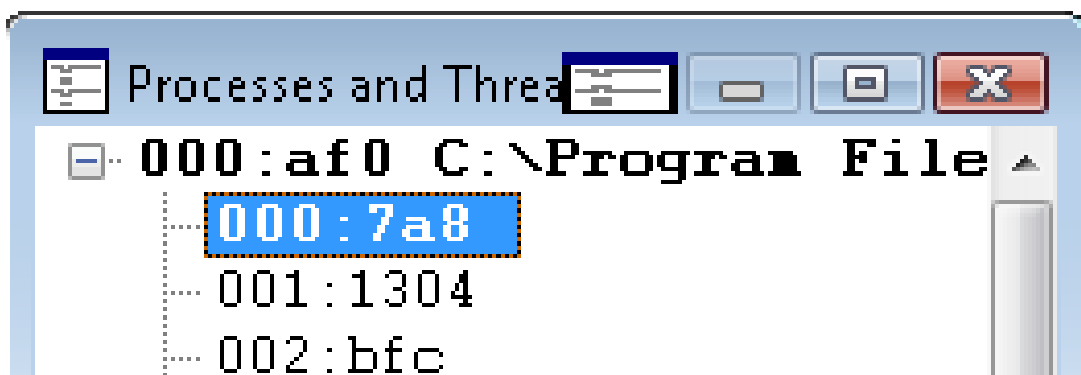
The Case of the Random IE and WMP Crashes

Mark Russinovich
(From Mark Russinovich Blog)

Looking for a potential cause of a crash when attached requires looking at the stack of each thread in the process, so I opened the Processes and Threads and Call Stack dialogs in the View menu:



I started examining threads by selecting the first entry in the threads dialog:



The WinDbg command window usually grays and says "Busy" as WinDbg pulls symbols from the symbol server, after which the call stack dialog populates with the function nesting of the selected thread at the time of the crash. I examined each thread's stack in turn, moving between threads by pressing the down arrow and then the enter key, hunting for a stack that had function names with the words "exception" or "fault" in them. Near the end of the list I came across this one:

The Case of the Random IE and WMP Crashes

Mark Russinovich
(From Mark Russinovich Blog)

```
Calls - Dump C:\Talks\demos\labs\crash\nvapphlp\wmp-crash.dmp - WinDbg:6.9.0003.113 X86
Raw args  Func info  Source  Addr  Headings  Nonvolatile regs  Frame nums  Source args  More  Less
ntdll!ZwWaitForSingleObject+0x15
ntdll!RtlReportExceptionEx+0x14b
ntdll!RtlReportException+0x3c
ntdll!RtlpTerminateFailureFilter+0x14
ntdll!RtlReportCriticalFailure+0x14
ntdll!_EH4_CallFilterFunc+0x12
ntdll!_except_handler4+0x8e
ntdll!ExecuteHandler2+0x26
ntdll!ExecuteHandler+0x24
ntdll!KiUserExceptionDispatcher+0x14
ntdll!RtlReportCriticalFailure+0x14
ntdll!RtlpReportHeapFailure+0x24
ntdll!RtlpLogHeapFailure+0x14
ntdll!RtlFreeHeap+0x60
kernel32!HeapFree+0x14
nvappfilter+0x6577 and information not available. Following frames may be wrong.
nvappfilter+0x4f7c
nvappfilter+0x3eae
nvappfilter+0x6577
vsz_32!closesocket+0x85
dnsapi!Socket_CloseEx+0x28
```

I noticed that the top of the list is full of functions with “Exception” in their names. Looking down the list (up the stack), I saw that a function in Nvappfilter called Kernel32.dll’s HeapFree function, leading to the crash. The exception in the heap’s free routines meant that either the caller passed a bogus heap address or that the heap was already corrupted when the function executed. If a Windows DLL had been the caller I would have suspected the latter, but in this case the caller was a third-party DLL, which I could tell by the fact that WinDbg couldn’t locate symbol information for it and hence didn’t know the names of the functions within it. I confirmed that by issuing the lm (list module) command to look at its version information:

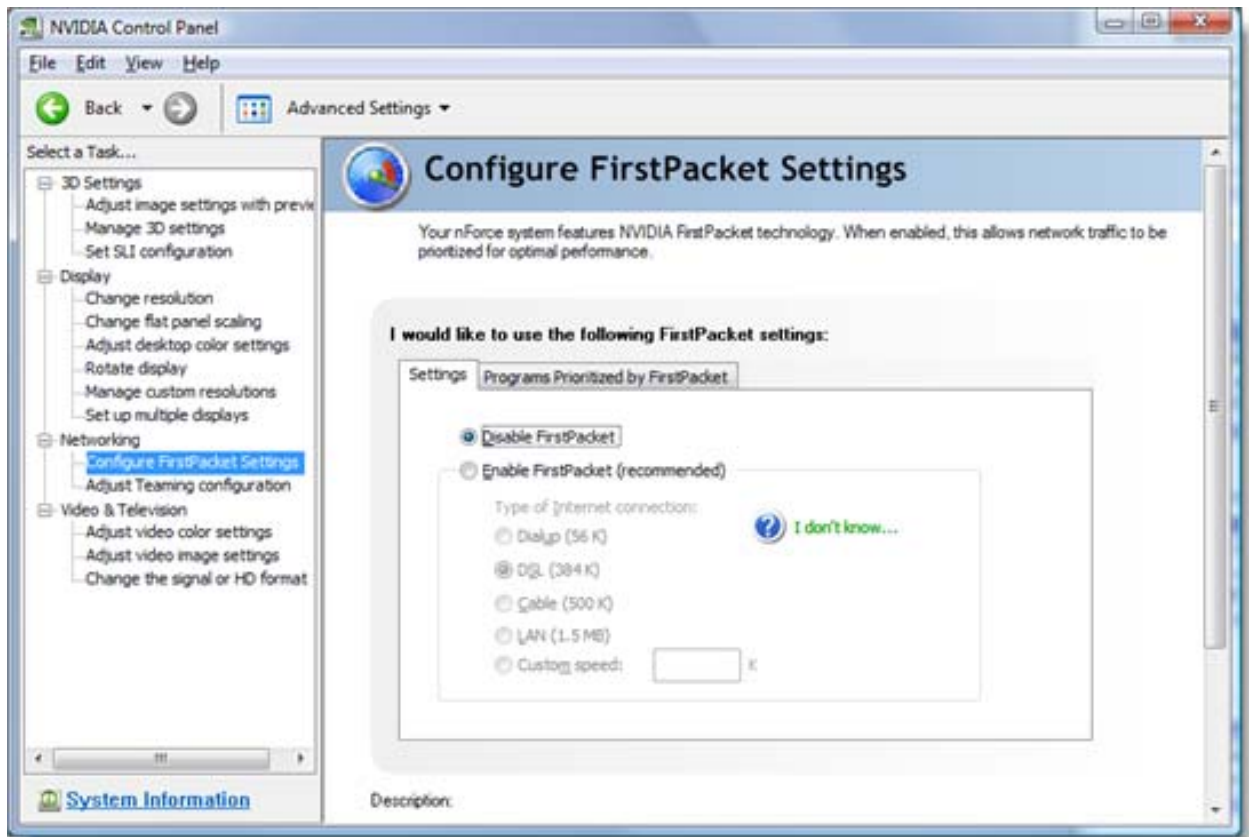
```
0:036> lm v mnvappfilter
start  end  module name
10000000 10029000 nvappfilter (export symbols) nvappfilter.dll
Loaded symbol image file: nvappfilter.dll
Image path: C:\Windows\System32\nvappfilter.dll
Image name: nvappfilter.dll
Timestamp: Mon Aug 20 15:10:19 2007 (46CA114B)
CheckSum: 00030FC8
ImageSize: 00029000
File version: 2.2.0.465
Product version: 2.2.0.465
File flags: 0 (Mask 3F)
File OS: 40004 NT Win32
File type: 2.0 Dll
File date: 00000000.00000000
Translations: 0409.04b0
CompanyName: NVIDIA
ProductName: NVIDIA Application Filter
InternalName: NVIDIA Application Filter
OriginalFilename: nvappfilter.dll
ProductVersion: 1.0.2.0
FileVersion: 1.0.2.0
PrivateBuild: 1.0.2.0
SpecialBuild: 1.0.2.0
FileDescription: NVIDIA IAM LSP
LegalCopyright: Copyright © 2004 NVIDIA Corporation
LegalTrademarks: Copyright © 2004 NVIDIA Corporation
Comments: Copyright © 2004 NVIDIA Corporation
```

The Case of the Random IE and WMP Crashes

Mark Russinovich
(From Mark Russinovich Blog)

Nvappfilter was now my primary suspect, but I didn't have direct evidence that it was responsible. I continued to use the system and followed the same debugging steps on the next several crashes. Whether it was IE, WMP or a game, the faulting stack was always the same, with Nvappfilter calling HeapFree. That's still not conclusive proof, but the anecdotal evidence was pretty compelling.

At that point I went to see if there were updates for Nvappfilter, but I wasn't sure what software package it was associated with. I entered its name in a Web search and discovered that it's part of the nVidia's FirstPacket feature that prioritizes game traffic and that's included in the nForce motherboard's software:

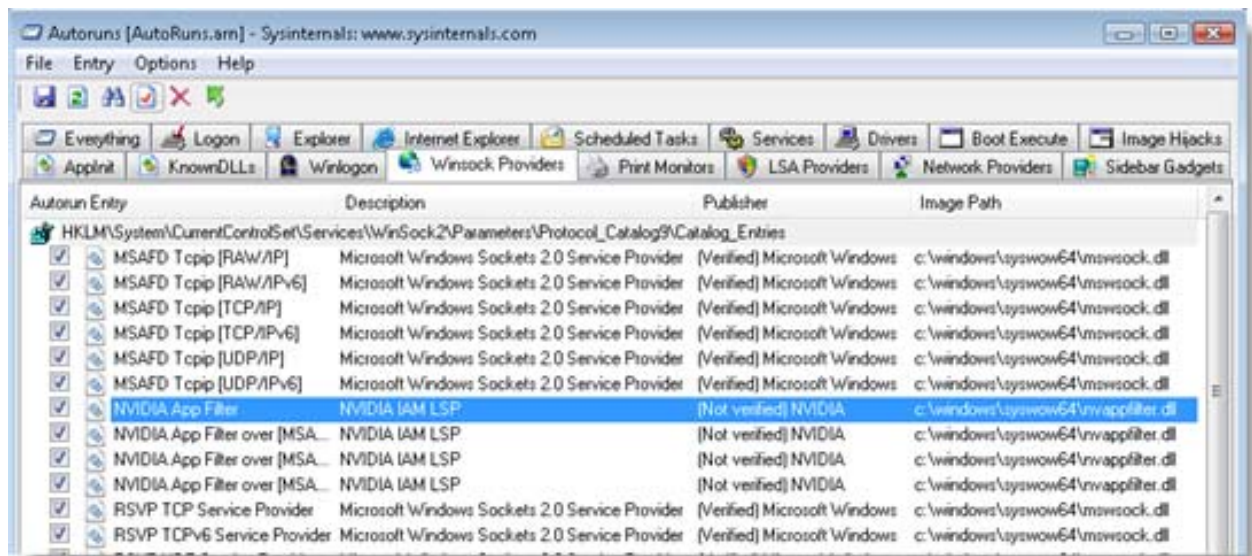


I went to nVidia's site and downloaded the most recent nForce driver package, but it failed to update Nvappfilter.dll and I continued to have the crashes.

The nVidia control panel offers no way that I could find to prevent Nvappfilter from loading, so my only recourse was to manually disable it. I wasn't using the FirstPacket feature, which I had previously been unaware of, so I wouldn't miss it, but first I had to figure out how it configured Windows to load it. For that I turned to Autoruns, where I found references to Nvappfilter's 32-bit and 64-bit versions in the Winsock Layered Service Provider (LSP) section:

The Case of the Random IE and WMP Crashes

Mark Russinovich
(From Mark Russinovich Blog)



I deleted all of Nvappfilter's entries, rebooted the system and have been crash-free since. While I was writing this post, I checked again for nForce software updates to see if Nvappfilter had been updated. The latest version doesn't look like it includes Nvappfilter or any other Winsock LSP, so assuming Nvappfilter was at fault, it's no longer an issue.

One other thing I've done since I investigated these crashes is take advantage of Vista SP1's "local dumps" functionality so that I'll automatically get a crash dump to investigate for any application crash I experience. If you create a key named HKLM\Software\Microsoft\Windows\Windows Error Reporting\LocalDumps, WerFault will always save a dump. Crashes go by default into %LOCALAPPDATA%\Crashdumps, but you can override that with a Registry value and also specify a limit on the number of crashes WerFault will keep.