

Under the Covers with Xperf

Michael Morales

(Reprinted From WindowsItPro Magazine)

Executive Summary

Xperf, a versatile Microsoft Windows event-tracing tool, provides a stackwalking feature that you can use to determine which system functions are consuming the most processor time. You can use this information to uncover the source of Windows system performance problems, such as might occur after you update an application or make a configuration change. Michael Morales shows you how to run an Xperf stackwalking trace for a specific time period, then view and interpret the trace data to pinpoint resource-consumption issues that may be impeding your system's performance.

One of the more powerful features of Xperf is the stackwalking feature, which lets you capture and view the functions executed during a specific time period and determine the most processor-intensive operations. The stackwalking feature lets you see not only the where all the processor time is being spent but also the path of execution for a process that led up to the expensive operation. Xperf can also help uncover the impact of configuration changes to a system by collecting aggregate data and parsing it for information about performance or patterns of resource consumption. In this article, I'll walk you through step-by-step how to properly capture a stackwalking trace and how to interpret the data.

When Stackwalking Can Be Helpful

Here's an example of a scenario where Xperf stackwalking can be useful. Following a recent update to a web application, you notice that system performance deteriorates. Using Xperf and the stackwalking feature, you find out that processor is spending time trying to draw a border around a JPEG file on a website. As you examine the function calls within Xperf, you discover that each JPEG file lookup results in hundreds of registry lookups, which is where all the processor time is being spent. When you give this information to the application vendor or in-house developers, they quickly realize that all the registry lookups are unnecessary and that the information could be held in memory where access times are much faster and less processor-intensive.

Capturing a Stackwalking Trace

Here are the tasks you'll need to perform to use Xperf to capture a stackwalking trace.

Set system variables for symbol loading. On the target system, the best way to ensure that your symbols are properly configured is by setting the following two system variables. You can set these two variables via the command line; however, my recommendation is to set these variables using the Control Panel's System applet or by right-clicking My Computer and selecting Properties, clicking the Advanced tab, and clicking Environment Variables.

The first system variable you need to set is this one:

```
_NT_SYMBOL_PATH =  
srv*c:\symbols*http://msdl.microsoft.com/download/symbols*c:\APP1
```

If you're a developer, then you may want to add paths to your own application's symbols or to another vendor's symbols. Simply add an additional asterisk (*) to separate each path.

The second system variable to set is this:

```
_NT_SYMCACHE_PATH = c:\symbols
```

Setting this variable tells Xperf to strip each symbol of the unnecessary information (i.e., structure definitions) and create a smaller-sized symbol file. The smaller file increases the speed of symbol loading and improves the overall Xperf experience when you're viewing stacks.

Under the Covers with Xperf

Michael Morales

(Reprinted From WindowsItPro Magazine)

Specify the stackwalk flag. There are several ways you can use the stackwalking flag in Xperf. Following is a typical command line that turns on stackwalking for kernel events and can be used to help diagnose high-CPU-usage issues:

```
xperf -on latency -stackwalk profile
```

To find a more comprehensive list of stackwalking events, use this command:

```
xperf -help stackwalk
```

Turn on symbol loading. After the trace has been collected and you're viewing the trace file, you'll need to turn on symbol loading from within the Xperf viewer, as Figure 1 shows.

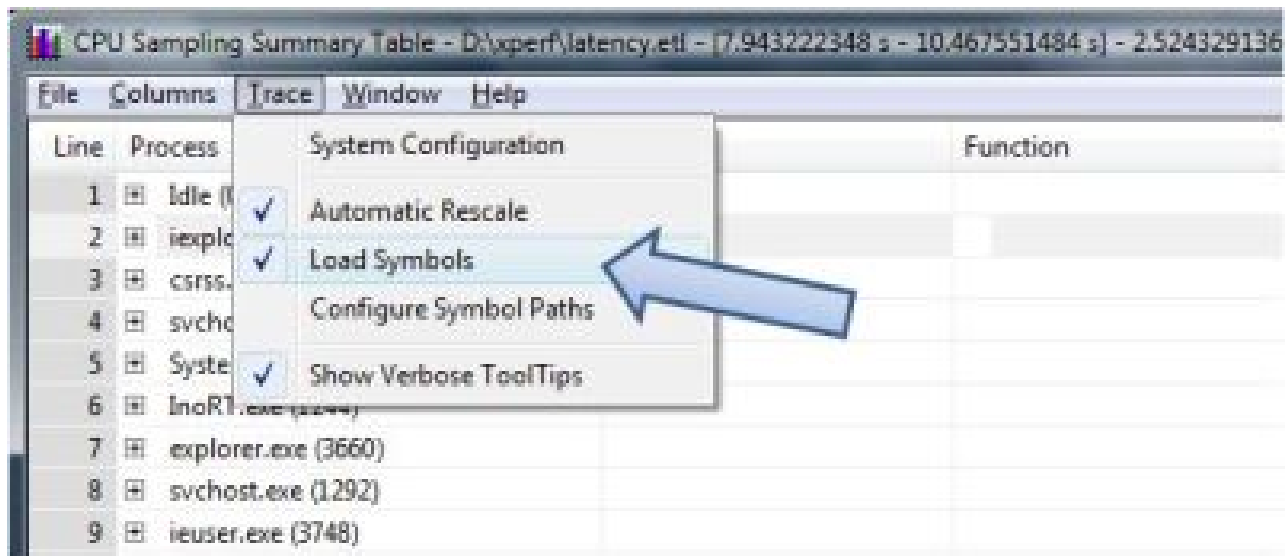


Figure 1
Loading Symbols from within Xperf

Those are the three main action items required to utilize the stackwalking functionality within Xperf, which leads us to a question: Is there any value in enabling the stackwalking feature if you don't have access to symbols? Answer: Yes, but it depends. Without symbols, you'll be able to see only the module within the process responsible for consuming the CPU. However, in some cases that could be enough information for you to make progress in resolving an issue, such as high CPU usage.

Capturing and Interpreting a Trace

Now let's walk through the complete set of steps you'll need to perform to first capture a stackwalking trace, then view and interpret the trace data.

Step 1: Issue the Xperf command. Following is a typical command line you can use to capture the information you'll need to diagnose a high-CPU-usage issue by using the stackwalking functionality. Once you've set up symbols (as I just described) and installed Xperf, you don't have to reboot or restart any other service. You're ready to run the following command:

```
xperf -on latency -stackwalk profile
```

Under the Covers with Xperf

Michael Morales

(Reprinted From WindowsItPro Magazine)

This command says, "turn on tracing for the kernel events that the latency group represents, and turn on the stackwalking profiler for this session." You won't be able to view any stack traces without issuing the -stackwalk profile flag.

Step 2: View and then stop the trace. The CPU spike needs to occur sometime while tracing has been enabled. To view and stop the trace, enter the following command:

```
xperf -d perf.etl
```

By default ETW places the information into the kernel.etl. By issuing the -d command you are saying, "merge the kernel.etl with the name of the etl file you specified (e.g., perf.etl).

Step 3: Open the perf.etl file. Now all that's left to do is open the perf.etl file inside the Performance Analyzer Viewer (i.e., Xperf), by issuing the following command:

```
xperf perf.etl
```

Figure 2 shows a complete end-to-end string of command-line entries.

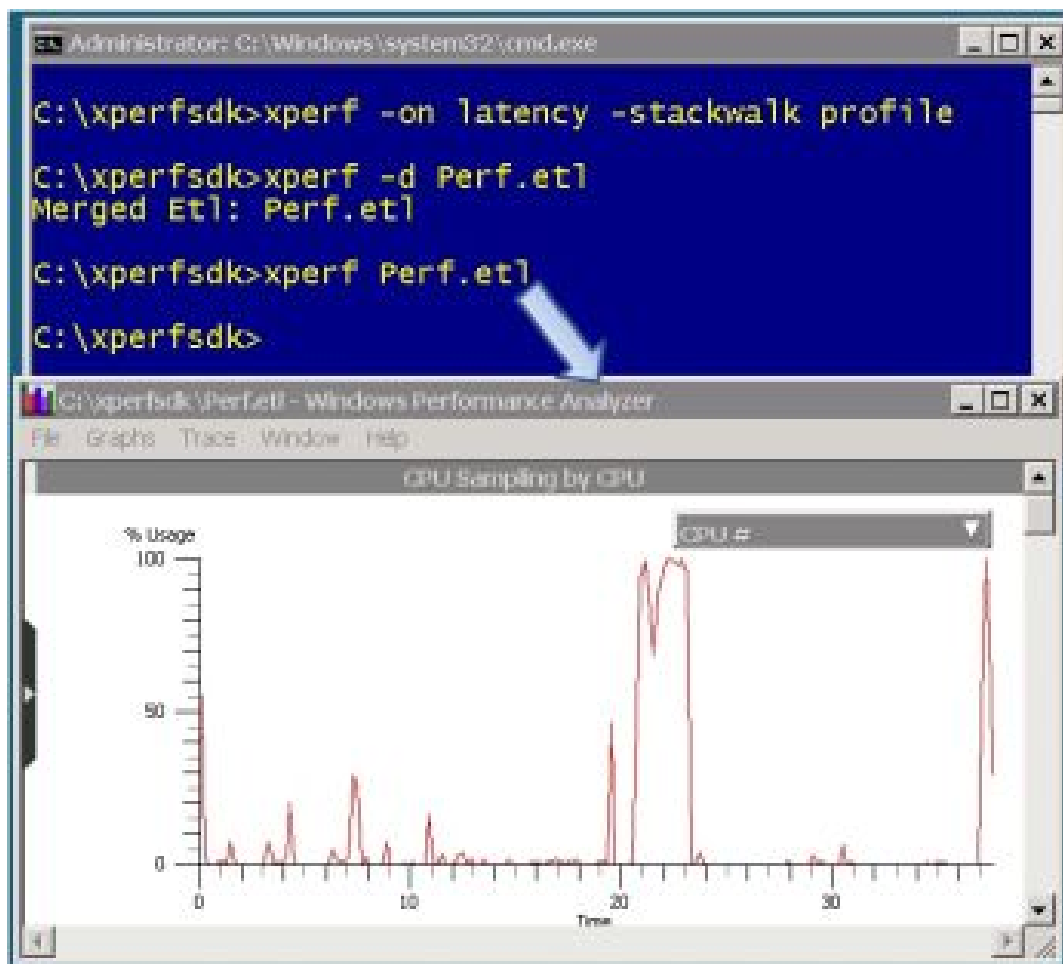


Figure 2
Xperf Trace Commands

Under the Covers with Xperf

Michael Morales

(Reprinted From WindowsItPro Magazine)

The Xperf viewer (aka Windows Performance Analyzer) will display a list of charts and graphs that you can view to start parsing through the data. As Figure 3 shows, once the viewer is opened, you can select the area of concern.

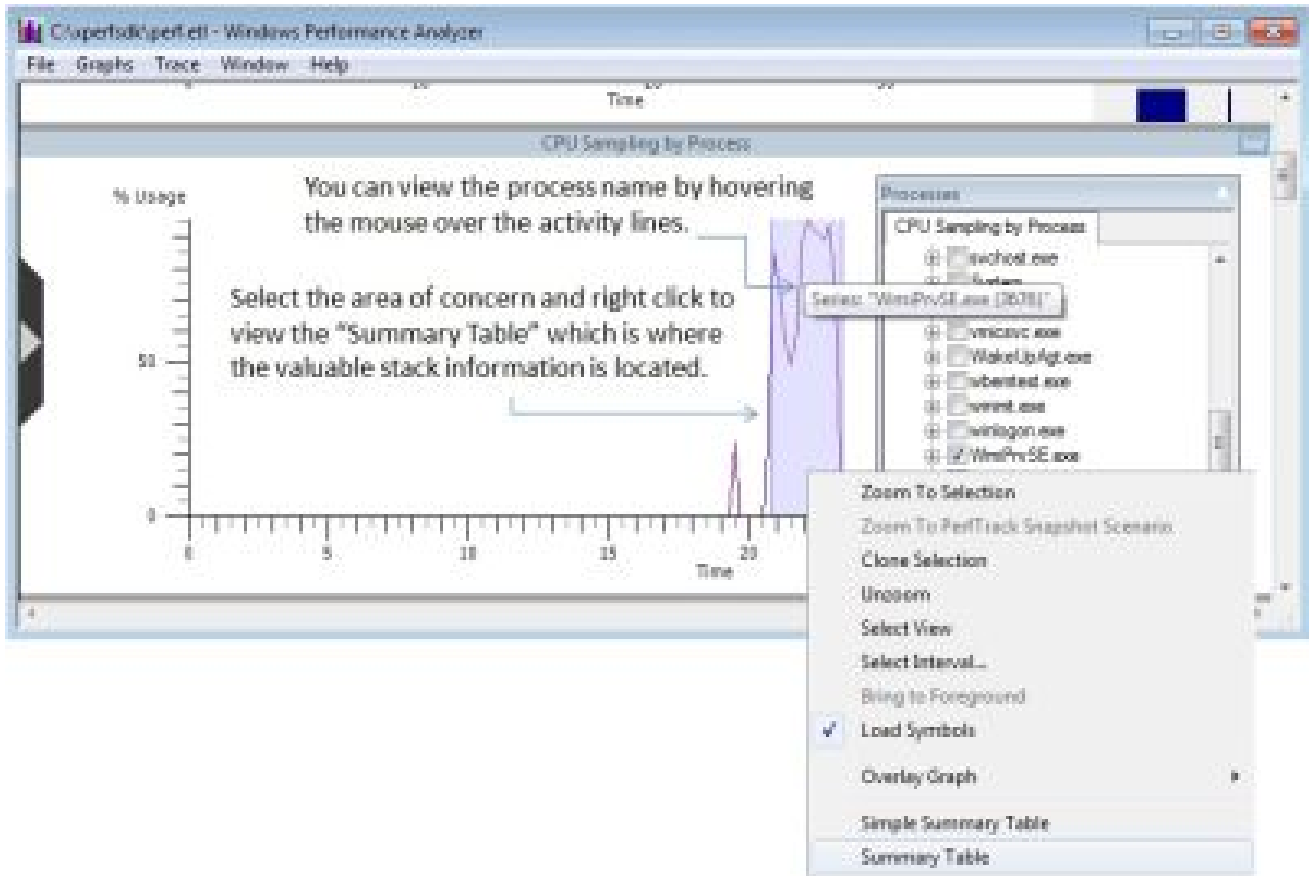


Figure 3
Viewing the Summary Table

To help make the graph less cluttered, you can uncheck all but the particular process you want to explore, as I've done. In this case, the process spiking the CPU is WMIPrvse.exe. Next, select the time of concern, right-clicked on the selected area, and choose Summary Table.

Once the Summary Table is open, click the chevron fly-out arrow at the far left. Make sure the Stack option is checked. This may take a few seconds as the symbols and stack load. Uncheck the Module and Function check boxes to make the view appear as it does in Figure 4.

Under the Covers with Xperf

Michael Morales

(Reprinted From WindowsItPro Magazine)

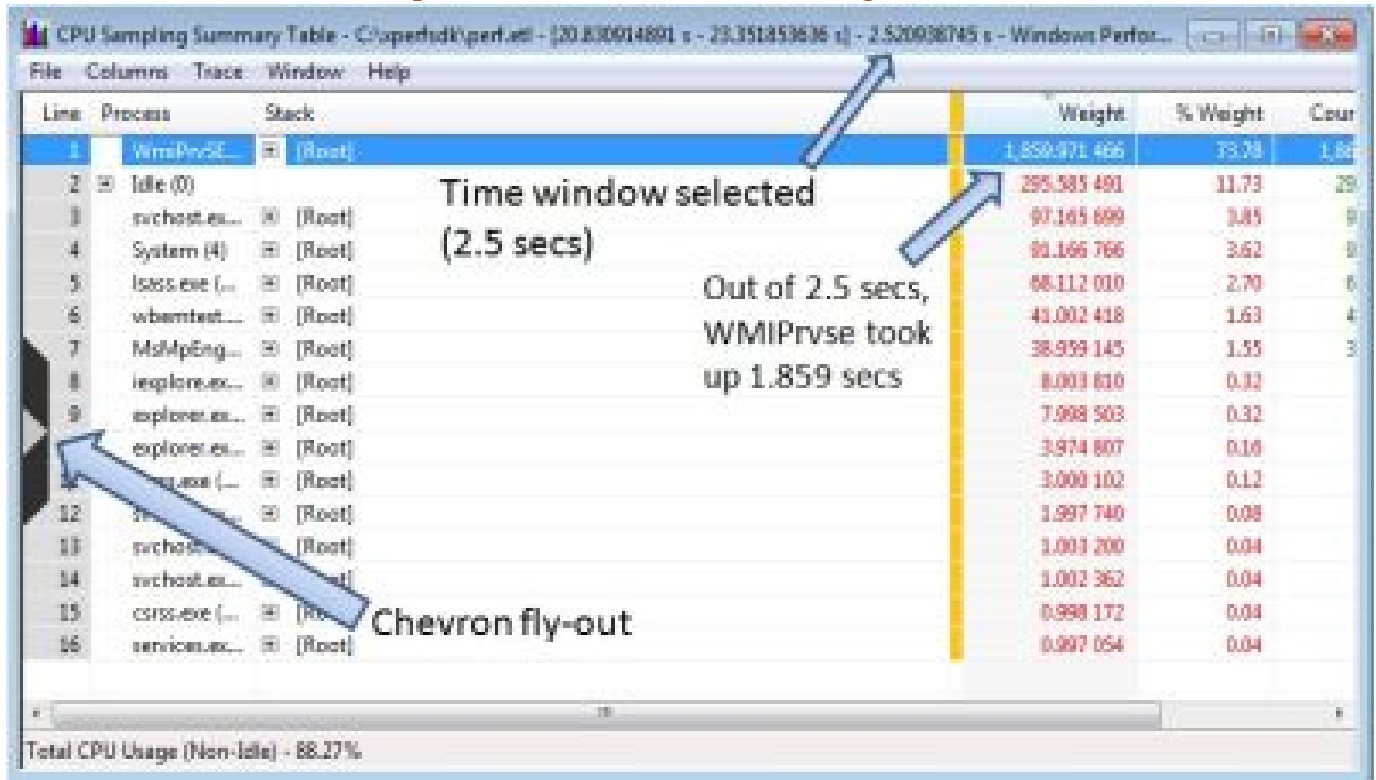


Figure 4
Summary Table Detail

Notice the yellow line in Figure 4. Columns to the left of the yellow line determine the sort order. So, since the Process column is the furthest left, with the Stack column next, basically this indicates that you want to sort CPU time by process, then by stack. You could also sort by process, then by module if you didn't have access to symbols and only wanted to see which module consumed the most CPU time.

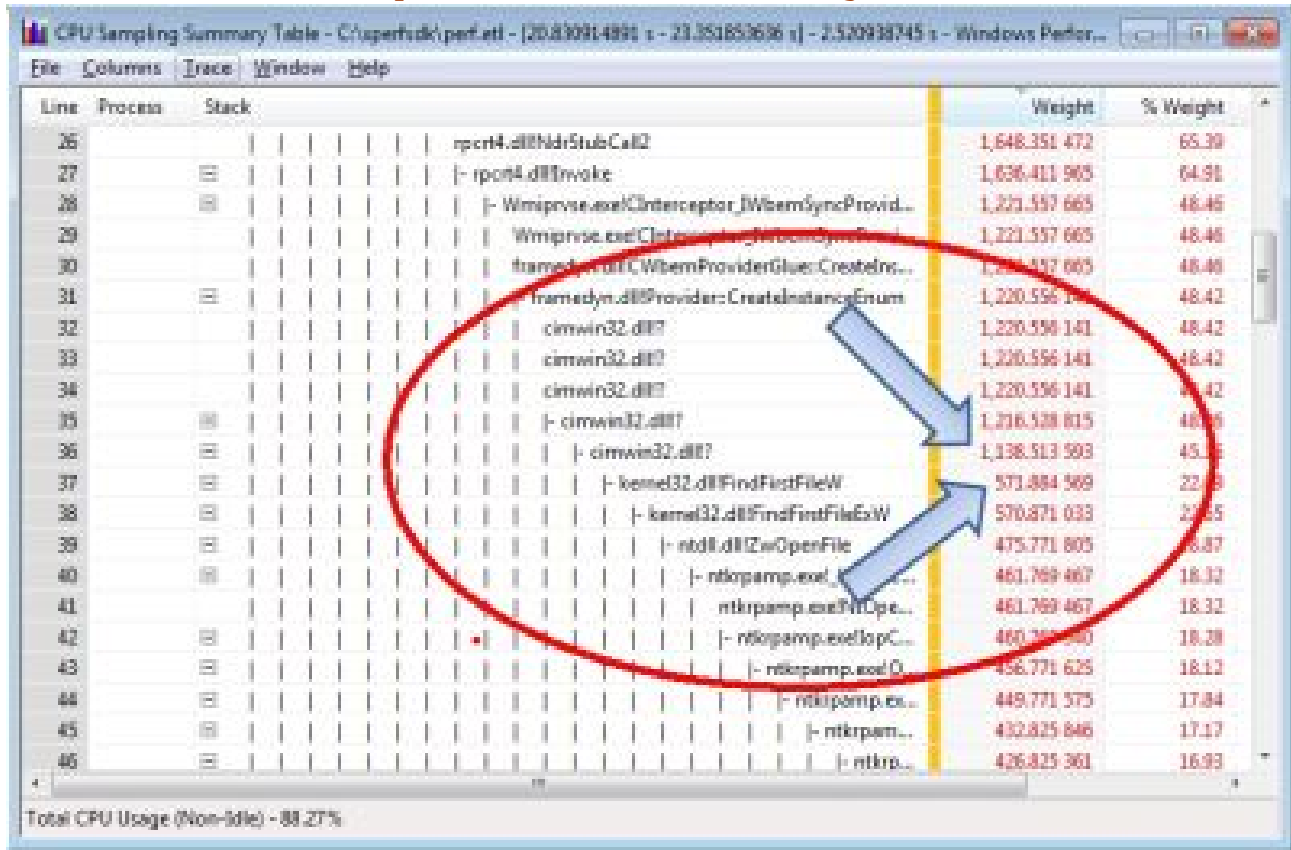
Here's the key to understanding this data. The time interval you selected appears in the title bar of the Summary Table. Thus, in Figure 4, we can see that our selected area encompassed 2.5 seconds. The highest-consuming process during this time period was WMIPrvse.exe, which we know because it has the topmost position in the graph. The numbers in the Weight column represent how many samples were collected during the selected time window. So, for example, we selected a time window of 2.5 seconds, and during that time we can expect to have approximately 2,500 events firing to collect event tracing information—the total of all the values in the Weight column. In this example, then, the CPU was in the context of the WMIPrvse.exe process 1,859.971 times, which represents a weight of 73.8 percent relative to all processes during that same time.

Furthermore, by expanding the plus (+) signs, we can start to unroll the stack and find out what functions consumed the most CPU time. You can also interpret the numbers under the Weight column as 1ms per event, so 1,859 translates to approximately 1.8 seconds. As you start to unroll the stack, you're looking for big drop-offs in time as the areas you'll investigate. In our example, Figure 5 shows a drop-off from 1.14 seconds to .57 seconds.

Under the Covers with Xperf

Michael Morales

(Reprinted From WindowsItPro Magazine)



Line	Process	Stack	Weight	% Weight
26		rpcrt4.dllNdrStubCall2	1,648,351,472	65.39
27		- rpcrt4.dllInvoke	1,636,411,903	64.91
28		- Wmiiprse.exeKInterceptor_IWbemSyncProvid...	1,221,557,665	48.46
29		Wmiiprse.exeKInterceptor_IWbemSyncProvid...	1,221,557,665	48.46
30		framedyn.dllCWBemProviderGlue::CreateIn...	1,220,556,141	48.42
31		framedyn.dllProvider::CreateInstanceEnum	1,220,556,141	48.42
32		cimwin32.dll?	1,220,556,141	48.42
33		cimwin32.dll?	1,220,556,141	48.42
34		cimwin32.dll?	1,220,556,141	48.42
35		- cimwin32.dll?	1,218,528,815	48.36
36		- cimwin32.dll?	1,138,513,593	45.15
37		- kernel32.dllFindFirstFileW	571,884,569	22.39
38		- kernel32.dllFindFirstFileExW	570,871,033	22.35
39		- ntdll.dllZwOpenFile	475,771,805	18.67
40		- ntirpamp.exe...	461,769,467	18.32
41		ntirpamp.exeNtOpe...	461,769,467	18.32
42		- ntirpamp.exeIopC...	460,768,930	18.28
43		- ntirpamp.exeO...	458,771,625	18.12
44		- ntirpamp.es...	449,771,375	17.84
45		- ntirpam...	432,825,846	17.17
46		- ntirp...	426,825,361	16.93

Total CPU Usage (Non-Idle) - 88.27%

Figure 5
Finding The Time Drop-Off In The Summary Table

If I were a developer, I'd start to investigate the cimwin32.dll function calls into kernel32!FindFirstFileW. If I were an administrator, I'd immediately do two things:

- Search online for an update to cimwin32.dll.
- Provide this information to the application's vendor; doing so will significantly reduce the time required to resolve the issue if an update isn't available.

The Anti-Debugger

Sometimes debugging is the only way to get to the root cause of a system performance problem. However, by using Xperf, you can uncover an application's behavior by scanning the names of the function calls and also determine exactly where all the system time is being spent—all without knowing a single debugging command or opening a dump file or being an expert in Windows architecture (although it wouldn't hurt!). Next month, I'll expand on the Xperf topic by digging into the multiple ways you can use the tool to help diagnose some common performance issues.