CHAD VERBOWSKI

the secret lives of computers exposed

FLIGHT DATA RECORDER FOR WINDOWS

Chad Verbowski is the cofounder of the Cybersecurity and Systems Management research group, where he focuses on reducing complexity and improving the security, reliability, and efficiency of software and integrating the results into the next generation of Microsoft products. Before joining Microsoft Chad worked extensively in the systems management problem space, leading the development of flagship management products at MFS Datanet, Cisco Systems, and Manage.com.

chadv@microsoft.com

WE'VE ALL HAD THE GIDDY EXPERIENCE of setting up a new system and being impressed by our newly acquired performance and capability. Inevitably though, as time goes on, our new system has less time for doing our bidding and assumes a life of its own—hard drives grind for no apparent reason, it is achingly slow or stalls altogether despite available resources, or applications and devices no longer work as they once did. Are these the result of unwanted users or software wooing my system—or did I do something to disrupt the delicate fabric of state stored within? With the ever-increasing spare time gleaned from waiting on my nearly new system, I pondered these issues and decided to put together a plan to spy on the secret life of my computer. What you are about to read may not be the information you need to be the life of your next party, but it will help you win back the attention of your computer.

A New Surveillance Gadget—Flight Data Recorder

To solve this mystery I first need surveillance equipment that is capable of monitoring my multicore system. As a model, I considered the airline industry's success at understanding crashes by analyzing the data contained in the black-box flight data recorders that are now standard on every flight. Designing and building a flight data recorder that tracks *which* process interacted with *what* piece of state as *whom* and *when* proved to be a daunting task. There were three core challenges to overcome:

- Overhead—The first challenge of building such a device for a computer is to accurately monitor the 28 million daily interactions that software running on my computer has with the roughly 200,000 files and 100,000 settings that it contains. Furthermore, to avoid contaminating the results, we must ensure that this equipment was undetectable by the running software and did not impact the resources and availability of the system.
- Volume of Information—The second challenge is to contain the fire hose of information collected without dropping any. Ideally we want to be able to audit up to many thou-

sands of systems and be able to correlate the information across time and across systems to develop an understanding of what they are doing. Tracking 28 million daily events at 250 bytes per event requires 7 GB of space, which is simply too much to handle. Naively, we may apply our favorite byte compressor, such as GZIP; however, we would find that this would only reduce the volume by approximately 90%, still leaving us with 700 MB daily to deal with. We will either quickly run out of storage space or cripple our system from the I/O requirements of writing these humongous logs to disk.

- Analysis of the Results—The third challenge is to be able to make sense of all this data. If we monitored 5 machines for a week, at 28 million interactions per day we would have 1 billion events to contend with. Traditionally we would attempt to cram them into our favorite database and apply SQL queries to discover the golden nuggets of truth. However, this approach is woefully slow and does not scale. The first problem involves the overhead in converting the data from their archived and compressed form into something that can be bulk-inserted into a database. Then there is the time taken to actually insert it. We found that, even with the stars aligned, our enterprise-class database server could spike to inserting events at a rate of 10,000 events per second. This means the insertion of our 1 billion events would take more than 27 hours. Even if we took the time to prepare and insert this data, we would still need several hours to index the tables, and countless hours to run our queries. Clearly, databases are a significant bottleneck that will force us to limit the number of machine days we can analyze or to filter or condense the data before analysis.

Faced with these challenges, many folks have run screaming from the room, thinking there is more likelihood of success to be had working on their perpetual motion machines. However, I remain unfazed—possibly because I am slow on the uptake, but just maybe because I have a key insight. Traditionally, the three challenges were attacked individually, but perhaps we can drastically improve our results if we optimize across all of them.

Based on this insight, Flight Data Recorder (FDR) collects events with virtually no system impact, achieves 350:1 compression (*0.7 bytes per event*), and analyzes a machine day of events in 3 seconds (*10 million events per second*) without a database. How is this possible, you ask? It turns out that computers tend to do highly repetitive tasks, which means that our event logs (along with nearly all other logs from Web servers, mail servers, and application traces) consist of highly repetitive activities. This is a comforting fact, because if they were truly doing 28 million distinct things every day it would be impossible for us to manage them. If we normalize the events as we receive them, rather than storing them in our log as a flat sequential list, we find that normalization removes the redundancy and provides us with a 35:1 reduction in log size. By maintaining the event logs in the normalized form, we make it faster and easier to analyze the log files directly, which saves us the overhead of putting them into a database.

If we GZIP our normalized files, we can squeeze an additional 10:1 compression, providing us with 350:1 overall. However, having GZIP files gives us the unsavory task of decompressing them before we can analyze the normalized tables for our query. Ideally, we want to decompress only the sections of our normalized tables on which we need to perform our analysis. Pondering this problem, we were motivated by traditional operating system page table design. Our solution was to overlay our normalized

tables atop 64k pages that are individually GZIP'd. This enables us to retain our 350:1 compression property, yet provides us with the ability to traverse the tables and decompress only the sections we need for analysis.

With our new FDR gadget we can easily monitor all file, registry, process, and module load activity in about 20 MB per day. Best of all, a single collection server can easily process in real-time the logs from 5000 systems and archive those logs on its available local disk for six months. With our surveillance tool in place, we are ready to begin our investigation of what computers do all day.

What Computers Do All Day—An Investigative Report

In my quest to understand the secret life of my computer I found that many people are often unwillingly forced into solving very similar problems in the course of their daily lives. At one large Internet company, one-third of outages were found to be caused by human error, and three-quarters of the time taken to resolve the issue was spent by administrators scouring the systems to identify what changes needed to be made. Similarly, a large software support organization found that their engineers were able to identify the root cause of only a scant 3% of the calls they received.

Before investigating my own computer's sordid life, I wanted to understand the state of what ought to be well-managed and well-maintained systems. To understand this I monitored hundreds of MSN production servers across multiple different properties. My goal was to learn how and when changes were being made to these systems, and to learn what software was running. Surely machines in this highly controlled environment would closely reflect the intentions of their masters? However, as you'll see in the following, we found some of them sneaking off to the back of the server room for a virtual cigarette.

THE LOCKDOWN PACT

Although rare, there are periods when system administrators like to kick back and enjoy an uninterrupted dinner with their families. The last thing they need is a problem with one of the pesky attention-seeking servers. To avoid problems, administrators form a secret pact they call *lockdown*, during which they all agree not to make changes to the servers for a specific period of time. The theory is that if no changes are made, no problems will happen and they can all try to enjoy their time outside the hum of the temperature-controlled data center. Using FDR, I monitored these servers for over a year to check the resolve of administrators by verifying that no changes were actually made during lockdown periods. What I found was quite surprising: Each of the five properties had at least one lockdown violation during one of the eight lockdown periods. Two properties had violations in every lockdown period. We're not talking about someone logging in to check the server logs; these are modifications to core Line-Of-Business (LOB) and OS applications. In fact, looking across all the hundreds of machines we monitored, we found that most machines have at least one daily change that impacts LOB or OS applications.

ALL RIGHT, WHO BROKE IT?

One of my favorite examples of a troubled computer is summarized in an

email from one system administrator to all other system administrators in that organization. It reads, "Whoever is changing the page-file setting on these computers please stop—you are taking down our site!" What I like about it is the way it shows how even if we know what the root cause of our problem is, we are powerless to understand how it happened and therefore powerless stop it from recurring in the future. What makes finding the culprit of these changes so difficult is the latency between when the page-file setting is changed and when the symptom of the change shows up (a crash from exhausting physical memory). It turns out that the setting does not take effect until the system is rebooted, which can be a long time after the change was made. Once FDR was installed on these systems, we found that this page-file setting is actually modified quite frequently—tens of servers are affected every two to three months. The FDR logs show that this modification is made by a remote Registry call, likely from a rogue administrative script. Armed with this intelligence report, administrators can now quickly undo the change if it happens again, and most important, they have the critical information required to keep this problem from recurring.

ILLUMINATING UNWANTED APPLICATIONS

We would all expect server environments to be highly controlled: The only thing running should be prescribed software that has been rigorously tested and installed through a regulated process. Using the FDR logs collected from the hundreds of monitored production servers, I learned which processes were *actually* running. Without FDR it is difficult to determine what is actually running on a system, which is quite different from what is installed. It turns out that only 10% of the files and settings installed on a system are actually used; consequently, very little of what is installed or sitting on the hard drives is needed. Reviewing a summary of the running processes, we found several interesting facts. Fully 29% of servers were running unauthorized processes. These ranged from client applications such as media players and email clients to more serious applications such as auto-updating Java clients. Without FDR, who can tell from where the auto-updating clients are downloading (or uploading?) files and what applications they run? Most troubling were the eight processes that could not be identified by security experts. Based on their names, some of these could be benign in-house tools (mlconv.exe, monnow.exe, sitreremover .exe); however, others (e.g., lsacacheagent.exe) sound like potential tools for compromising security (since LSA typically refers to the Windows security system).

REMEMBERING TO LOCK THE DOOR

Few of us obsess overly on securing our home and possessions; we tend to content ourselves with a few commonsense tasks routinely followed to ensure our protection. We lock the doors on our car when we park it, and we lock the doors on our house when nobody's home. Although in the back of our mind we know that if someone is determined to get in they probably can, we don't want to make it easier for them. When it comes to servers, there are some similar best practices. One of them is to avoid leaving credentials or primary security tokens on systems. Primary tokens are created with credentials and can be used to hop to another system. The remote system receives secondary credentials, which cannot be used to hop

again. These are used by hackers who compromise a server to hop from one system to the next and spread throughout your network. Using the FDR logs, we found six services (daemons) running on many machines that were using hard-coded credentials, which could potentially be harvested by hackers. We also found that a third of the systems had screensavers running on them from when administrators had logged in remotely and left their sessions active. These remote sessions leave primary tokens on the system for hackers to harvest. By running FDR on these systems we can quickly identify these potential security problems and ensure we are not making it easier for undesirables to break in.

WHY IS MY SYSTEM SLOW?

When applications are running on our system we really have no clue as to whether the amount of resources they are consuming is reasonable or not. Should this monitoring agent be consuming 5% CPU, or 15%? We really don't know. From running FDR we not only see what processes are reading and writing on the system, but we also have the timestamps for each interaction. Using these timestamps, we can easily calculate how many operations (reads/writes) each process is doing per second. We can even tell if it is reading the same thing over and over and over again. In fact, by looking for these patterns we identified processes that were doing just that. An LOB application was reading the crypto settings 240 times per second, and a management agent was continuously reading the 10,000 service (daemon) settings in a tight loop. Without this information, a developer would usually be oblivious to these useless performance costs. Although these may seem insignificant on the surface, consider that losing 10% of your CPU across 100 servers is equivalent to buying 10 extra servers. Furthermore, if one application is unnecessarily dominating the system by reading settings, other applications will perform more slowly.

When the Cat's Away, the Mice Will Play . . .

Peering through our FDR microscope at the daily lives of our computers, we found many unexpected activities. They made us laugh, they made us cry, but in the end they provided us with knowledge that empowered us to improve our systems. For the past 20 years, systems management has been more of a "dark art" than a science or engineering discipline because we had to assume that *we did not know* what was really happening on our computer systems. Now, with FDR's always-on tracing, scalable data collection, and analysis, we believe that systems management in the next 20 years can assume that we *do know and can analyze* what is happening on every machine. We believe that this is a key step to removing the "dark arts" from systems management.

You too can leverage FDR technology for investigating your computer, by using some of the products that incorporate FDR technology. The first wave of products includes Windows Vista, which contains the drivers that expose the file, Registry, process, and module information through the Event Tracing for Windows (ETW) providers. There is also the Application Compatibility Toolkit v5.0, which contains the Update Impact Analyzer (UIA) and monitoring agents that use FDR technology for understanding Windows systems to enable you to better prepare for upgrades and patches.

REFERENCES

[1] C. Verbowski, E. Kıcıman, A. Kumar, B. Daniels, Y.-M. Wang, R. Roussev, S. Lu, J. Lee, "Analyzing Persistent State Interactions to Improve State Management," SIGMETRICS, Saint-Malo, France, 2006.

[2] C. Verbowski, E. Kıcıman, B. Daniels, S. Lu, J. Lee, Y.-M. Wang, and R. Roussev, "Flight Data Recorder: Monitoring Persistent-State Interactions to Improve Systems Management," OSDI, Seattle, WA, 2006.

[3] C. Verbowski, J. Lee, and Y.-M. Wang, "LiveOps: Systems Management as a Service," LISA '06, Washington, D.C., 2006.