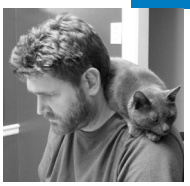
DAVE JOSEPHSEN

iVoyeur: top 5 2008



David Josephsen is the author of *Building a Monitoring Infrastructure with Nagios* (Prentice Hall PTR, 2007) and is Senior Systems Engineer at DBG, Inc., where he maintains a gaggle of geographically dispersed server farms. He won LISA '04's Best Paper award for his co-authored work on spam mitigation, and he donates his spare time to the SourceMage GNU Linux Project.

dave-usenix@skeptech.org

HAVE YOU EVER BEEN SO BUSY THAT the phrase "I've been busy" rings hollow and cliche in your ears? In the past year, desperate to position ourselves to scale at the pace our tiny company is growing, my cohort and I have ripped out, redesigned, and replaced our entire production infrastructure. Our routers went from IOS to JunOS to BSD, and our balancers from Alteons to Mod_proxy_balancer to balancer clusters using Carp to those using ClusterIP. Static routes over IPSEC VPNs have been replaced with OSPF routes over SSL VPNs. We fought through PCI certifications, designed IP and DNS standards, and implemented BGP-based datacenter fail-over systems. Anyway . . . I've been busy.

So busy, in fact, that for the first time since I discovered USENIX, I didn't make it to a single conference this year, USENIX or otherwise. January finds me shell-shocked, wondering where the time went, and curious as to what great work I missed out on in 2008. Fed up with being in the dark, I immediately declared a paper-reading weekend and set about going through conference proceedings to see what went on outside the cave I've been trapped in. I wasn't disappointed; as usual, the papers track of the various USENIX Cons in 2008 put me to shame. I might as well be a janitor. Been busy? Well, if you've been missing out on some great work, let me help you out with this list of my top 5 favorite monitoring-related papers of 2008.

We'll start with number 5, a paper called "Error Log Processing for Accurate Failure Prediction" [1]. This paper is from the new USENIX workshop on the analysis of system logs. With apologies to the authors of this paper whose intent was to explore failure prediction, this paper caught my attention because of the several clever log preprocessing techniques they used on free-form system logs to make them more machine-readable. These included removing data such as numbers from the logs, using Levenshtein distance to categorize and group similar individual events, and making use of a technique called tupling to identify multiple event entries that correspond to the same actual error. The log preparation sections in the preamble of this paper are of immediate practical use to folks who have a lot of logs and are looking for some quick ways to get a handle on them.

Speaking of logs, number 4, a tool paper called "Picviz: Finding a Needle in a Haystack" [2], applies a visualization technique called parallel coordinate plots to system logs. If you've read Greg Conti's excellent book *Security Data Visualization*, then you've seen parallel coordinate plots used to great effect to plot port scans. These graphs take any number of variables and assign them columns in the vertical plane. A single element of data made up of those variables is then represented as a line in the horizontal plane. Figure 1 is an example parallel coordinate plot of SSH logins shamelessly stolen from the Picviz Web site.

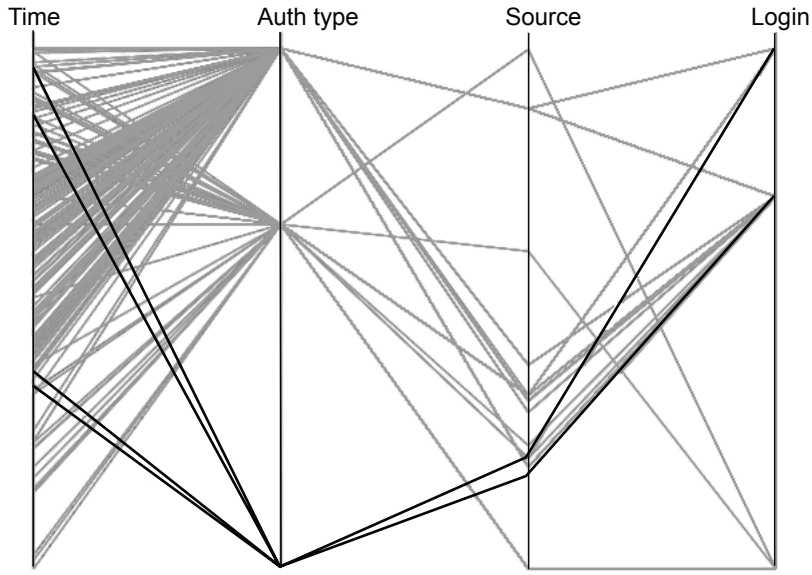


FIGURE 1: THIS PARALLEL COORDINATE PLOT MAKES IT EASY TO SPOT "SUSPICIOUS" SSH CONNECTIONS THAT ORIGINATE FROM MULTIPLE SOURCES BUT USE THE SAME USER ID.

The Picviz paper has a lot of elements that I like in a good paper. They've identified and solved a problem for me; catching odd stuff going on in a large amount of log data is difficult, and Picviz makes it easier. Behind the paper is a GPL'd tool that I can get my hands on and play with right now. Too often, promising work never goes very far for lack of an available implementation. And, speaking of implementation, I really like Picviz's design, which mimics that of GraphViz. The authors have created a mark-up language for drawing parallel coordinate plots, which means I can write glue code to connect it to existing parsers and monitoring apps.

They didn't stop at providing a framework; they used their own mark-up language to write a GUI for lighter-weight, interactive, or real-time use. I'd use Picviz over something like Conti's Rumint [3] when I want something less purpose-specific and more flexible. I can graph whatever I want with Picviz whether my data originated from log files, PCAP dumps, or NetFlow logs. I can use it to bolt parallel coordinate plots onto existing dashboards, and it works equally well as a forensics or real-time IDS tool. Good work.

The writers of my third favorite monitoring-related paper in 2008 had a fascinating problem, that of being ignored. It seems that botnet attacks against Web applications have increasingly abandoned the traditional brute-force scanning mechanisms in favor of clever Google searches. This is bad news for folks trying to write honeypots, since unless your honeypot gets indexed in a way that piques the interest of the attacker, no traffic will arrive at your honeypot. So the problem becomes getting a single machine to respond to

search robots in a way that makes it attractive to whatever the botnets happen to be into that week.

Their rather ingenious solution, described in "To Catch a Predator: A Natural Language Approach for Eliciting Malicious Payloads" [4], was to use natural language processing techniques to generate dynamic responses to the indexing services based on real host interactions. The methodology they used to glean their training data was clever, as is their solution to a rather sticky problem. And although this work is probably not immediately useful to those of us outside botnet research, I think their methodology could really blow the roof off the honeynet, tar pit, and protocol misdirection scene. It seems like the kind of technique that would find unexpected practical application all over the place. Good work.

Number 2 on the list is a paper called "CloudAV: N-Version Antivirus in the Network Cloud" [5]. I've long been flummoxed by the concept of multi-version programming. I've read several papers arguing about the reliability it adds or doesn't and whether its core assumption of statistical independence is borne out or not. I have a pretty good mental picture of how it's supposed to work, but I've never been able to come up with a problem domain into which it seems to fit. The concept is, given some problem, you write *N* different software solutions and compare the results. Folks argue whether it's actually possible to come up with *N* independent ways to solve a single problem (in other words, there's an assumption the *N* solutions can be statistically independently derived, and this doesn't hold water). Folks also argue whether solving a problem *N* times gives you a better solution or not, or they wonder why you would want to do that work.

Well this paper solidified my thinking: malware detection is a great problem domain for multi-version programming. It's a problem, in fact, for which most of us have used *N*-version programming without ever realizing it. Every moderate-sized to large company I've ever worked for has used at least two different anti-virus programs from competing vendors, usually at least one at the mail gateway and another on the desktop. That viruses sometimes hit the desktop that aren't caught by the MX might suggest that competing anti-virus software is independent enough in practice to work, or maybe not. Perhaps the viruses hit the desktops from a different vector, or maybe there was a temporal problem with updates. In practice, it doesn't really matter. Having some heterogeneity there always seemed like a good idea.

I'll be honest: I *hate* the entire realm of malware detection. I've always put a large space between myself and those unfortunate enough to be tasked with managing the corporate/campus/whatever AV system. That software has been a buggy, intrusive, ill-thought-out nightmare for years. CloudAV, however, if the implementation does what the paper says it does, has single-handedly made AV not suck anymore (or at least reduced the suck factor by several orders of magnitude). They've taken the multi-version programming concept to the extreme, implementing ten anti-virus engines and two behavioral detection engines on a central Xen host running a VM for each engine. They then install lightweight clients on each host that trap file-creation system calls, blocking them while sending the new file to a broker in front of the scan engines. The broker, before sending the file back for scanning, records metadata about the file in a database and checks it against hashsums of already scanned files, preserving network bandwidth and creating a gold mine of forensic data.

The client can run in three modes: transparently allowing all user actions while sending files to the scan server; blocking user actions and warning the user if a suspicious file is encountered; or blocking user actions and ulti-

mately preventing them regardless of what the end user thinks. I'm not clear on how policy is enforced in the system (e.g., what prevents end users from killing the client locally, a question I expect was asked in the session), but IMO whatever problems it has are certainly more than made up for in what it delivers.

Right off the bat it provides a model the malware hasn't accounted for, so it goes a long way toward limiting the malware's ability to detect and attack the AV system itself. It eliminates the need to maintain definitions on end-user systems, as well as eliminating pretty much every other virus-software integration problem that so plagues the desktops of the heathen solitaire proletariat. It completely insulates you from management decisions regarding AV vendor licensing; you'll never have to rip out McAfee on 1000 workstations to install Sophos instead, for example. And as a bonus it gives you a powerful forensics tool. Have a data sensitivity standard? CloudAV can tell you every host that touched a given file and in what context. Crazy good work. The only two problems it has are that I can't download it now, and I can't buy it now.

Finally, my number 1 pick in 2008 is "Sysman: A Virtual File System for Managing Clusters" [6], although a more accurate title might be "Sysman, a Virtual File System for Managing Whatever You Have Plugged into the Wall." First off, I should warn you that I'm a bit of a sucker for filesystem interfaces. You might have guessed this if you've read my recent NagFS articles. I know XML is all the rage, and configuration management doesn't equate to remote control, but nothing beats /proc when it comes to management simplicity and leveraging existing skills. OpsWare understood this when they created its Global Shell (GS) before the company was purchased by HP. Long have I wished for an OpsWare GS that I could afford. We've even discussed rolling it ourselves several times in FUSE. The one year I don't get to go to LISA, along comes Sysman to scratch that itch. I really wish I had been at this session—I have a lot of questions.

Curiously, the Sysman authors don't seem to have heard of OpsWare Global Shell, but no matter, they appear to have gotten it right the first time. Sysman creates a filesystem containing directories that represent devices on the network. These devices can be detected and set up automatically and their features automatically become subdirectories inside Sysman. Upon detecting Linux servers, for example, Sysman will create a subdirectory representing the server and subdirectories inside the server directory corresponding to the server's proc and sys directories. Reading from /sysman/10.10.1.111/proc/cpuinfo will return the contents of the cpuinfo file in the proc directory on server 10.10.1.111. A "commands" file provides access to remote command execution: write to a server's commands file to send it a command and then read from its commands file to glean the last command's output.

Filesystem interfaces are great because you can bring all of your existing scripting skills to bear, and existing tools gain enormous amounts of power. Imagine how easy Nagios Event Handlers are to write given a Sysman filesystem, for example. Have a down Apache daemon? Just do:

```
echo '/etc/init.d/apache restart' > /sysman/apachehost/commands
```

There are considerations, of course, including security and stability, but you get my drift. If CloudAV has the potential to actually get me interested in reining in the malware problem, then Sysman has the potential to completely turn my world on its ear. A filesystem interface to my entire datacenter? Are you kidding me? It would be the most enabling thing to happen to me since I learned regular expressions. Simple, powerful, elegant: great

work. Now, where can I get it? There's an eight-year-old version on SourceForge that I doubt is very functional. What gives, guys? Been busy?

REFERENCES

[1] F. Salfner and S. Tschirpke, "Error Log Processing for Accurate Failure Prediction," WASL '08: http://www.usenix.org/events/wasl08/tech/full_papers/salfner/salfner_html.

[2] S. Tricaud, "Picviz: Finding a Needle in a Haystack," WASL '08: http://www.usenix.org/events/wasl08/tech/full_papers/tricaud/tricaud_html.

[3] Rumint, open source network and security visualization tool: http://www.rumint.org/.

[4] S. Small, J. Mason, F. Monrose, N. Provos, and A. Stubblefield, "To Catch a Predator: A Natural Language Approach for Eliciting Malicious Payloads," Security '08, http://www.usenix.org/events/sec08/tech/full_papers/small/small_html/index.html. [Editor's note: The authors of this paper also wrote a related article that appears in the December 2008 issue of *;login:*.]

[5] J. Oberheide, E. Cooke, and F. Janhanian, "CloudAV: N-Version Antivirus in the Network Cloud," Security '08: http://www.usenix.org/events/sec08/tech/full_papers/oberheide/oberheide_html/index.html.

[6] M. Banikazemi, D. Daly, and B. Abali, "Sysman: A Virtual File System for Managing Clusters," LISA '08: http://www.usenix.org/events/lisa08/tech/full_papers/banikazemi/banikazemi_html/index.html.