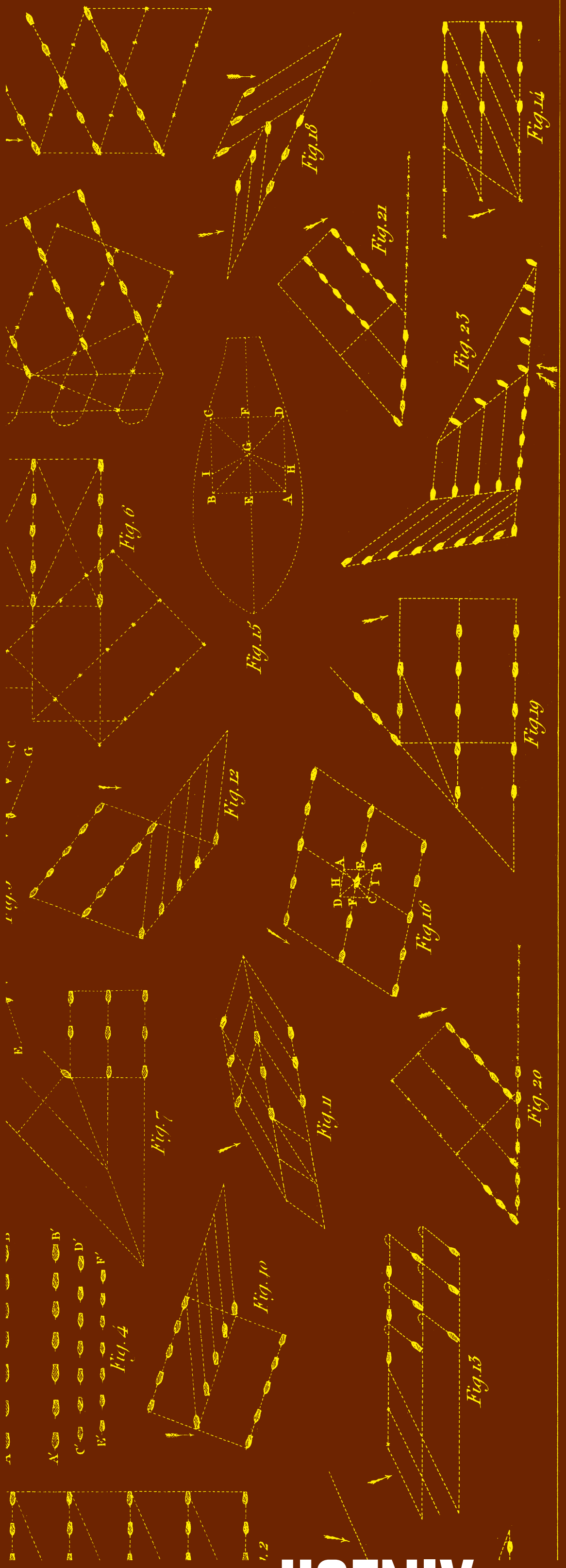
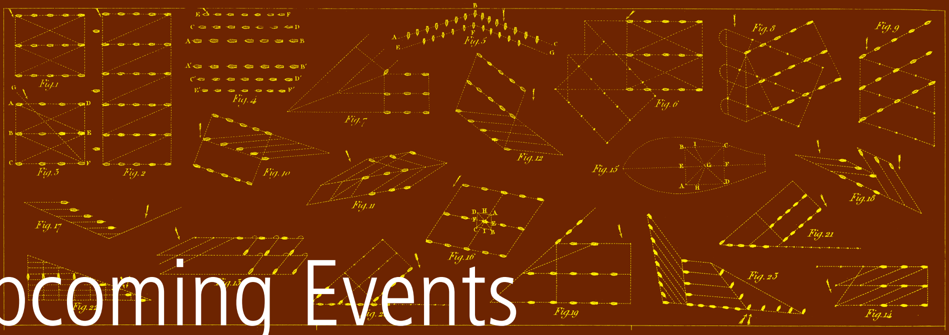
FEBRUARY 2007 VOLUME 32 NUMBER 1

;login:



USENIX
The Advanced Computing Systems Association

USENIX Upcoming Events



First Workshop on Hot Topics in Understanding Botnets (HotBots '07)

Co-located with NSDI '07

APRIL 10, 2007, Cambridge, MA, USA

http://www.usenix.org/hotbots07

Paper submissions due: February 26, 2007

Second Workshop on Tackling Computer Systems Problems with Machine Learning Techniques (SysML07)

Co-located with NSDI '07

APRIL 10, 2007, Cambridge, MA, USA

http://www.cs.duke.edu/nicl/sysml07

Third International Workshop on Networking Meets Databases (NetDB '07)

Co-located with NSDI '07

Sponsored by USENIX in cooperation with ACM SIGCOMM

APRIL 10, 2007, Cambridge, MA, USA

http://www.usenix.org/netdb07

4th USENIX Symposium on Networked Systems Design and Implementation (NSDI '07)

Sponsored by USENIX in cooperation with ACM SIGCOMM and ACM SIGOPS

APRIL 11–13, 2007, Cambridge, MA, USA

http://www.usenix.org/nsdi07

11th Workshop on Hot Topics in Operating Systems (HotOS XI)

Sponsored by USENIX in cooperation with the IEEE Technical Committee on Operating Systems (TCOS)

MAY 7–9, 2007, San Diego, CA, USA

http://www.usenix.org/hotos07

5th ACM/USENIX International Conference on Mobile Computing Systems, Applications, and Services (MobiSys 2007)

Jointly sponsored by USENIX and ACM SIGMOBILE, in cooperation with ACM SIGOPS

JUNE 11–15, 2007, Puerto Rico

http://www.sigmobile.org/mobisys/2007/

Workshop on Experimental Computer Science (ECS '07)

Sponsored by ACM SIGARCH and ACM SIGOPS in cooperation with USENIX, ACM SIGCOMM, and ACM SIGMETRICS

JUNE 13–14, 2007, San Diego, CA, USA

http://www.expcs.org/

Paper submissions due: February 9, 2007

Third International ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments (VEE '07)

Sponsored by ACM SIGPLAN and ACM SIGOPS in cooperation with USENIX

JUNE 13–15, 2007, San Diego, CA, USA

http://vee07.cs.ucsb.edu

Paper submissions due: February 5, 2007

2007 USENIX Annual Technical Conference

JUNE 17–22, 2007, Santa Clara, CA, USA

http://www.usenix.org/usenix07

Third Workshop on Hot Topics in System Dependability (HotDep '07)

Co-sponsored by USENIX

JUNE 26, 2007, Edinburgh, UK

http://hotdep.org/2007

Paper submissions due: February 15, 2007

2007 USENIX/ACCURATE Electronic Voting Technology Workshop (EVT '07)

Co-located with Security '07

AUGUST 6, 2007, Boston, MA, USA

http://www.usenix.org/evt07

Paper submissions due: April 22, 2007

16th USENIX Security Symposium

AUGUST 6–10, 2007, Boston, MA, USA

http://www.usenix.org/sec07

Paper submissions due: February 1, 2007

2007 Linux Kernel Developers Summit

SEPTEMBER 4–6, 2007, Cambridge, U.K.

21st Large Installation System Administration Conference (LISA '07)

Sponsored by USENIX and SAGE

NOVEMBER 11–16, 2007, Dallas, TX

For a complete list of all USENIX & USENIX co-sponsored events, see http://www.usenix.org/events.

contents

;login:

VOL. 32, #1, FEBRUARY 2007

EDITOR
Rik Farrow
rik@usenix.org

MANAGING EDITOR
Jane-Ellen Long
jel@usenix.org

COPY EDITOR
David Couzens
proofshop@usenix.org

PRODUCTION
Lisa Camp de Avalos
Casey Henderson

TYPESETTER
Star Type
startype@comcast.net

USENIX ASSOCIATION
2560 Ninth Street,
Suite 215, Berkeley,
California 94710
Phone: (510) 528-8649
FAX: (510) 548-5738

http://www.usenix.org
http://www.sage.org

;login: is the official magazine of the USENIX Association.

;login: (ISSN 1044-6397) is published bi-monthly by the USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.

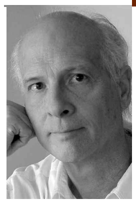
$85 of each member's annual dues is for an annual subscription to *;login:*. Subscriptions for nonmembers are $120 per year.

Periodicals postage paid at Berkeley, CA, and additional offices.

POSTMASTER: Send address changes to *;login:*, USENIX Association, 2560 Ninth Street, Suite 215, Berkeley, CA 94710.

RIK FARROW

musings



rik@usenix.org

AS TECHNOLOGISTS, WE ARE STUCK in a terrible double bind. We need to build upon the successes of the past—those who ignore history surely suffer from their ignorance. But there comes a time when past paradigms must be supplanted by new ideas, or we stagnate. This conundrum gets magnified by the effect of experience; that is, the more expert we become at a particular technology, the more value it has for us. To abandon what we know well, even if it no longer serves us, is like killing the goose that lays the golden eggs.

Strange thoughts indeed, but if you have read my columns before, I doubt that you are at all surprised. And it was new ideas, not retreads, that brought me to this point. I was fortunate enough to attend the WORLDS, OSDI, and HotDep workshops and symposium in Seattle (November 2006), where I got flooded with new ideas. Just as Washington State was being deluged with record rainfall, I felt like a privileged student at a fast-paced series of advanced seminars.

I discovered that OSDI and its ACM-sponsored companion, SOSP, are considered the most prestigious of operating systems conferences. One young attendee told me that getting a paper accepted at either conference could assure your future. I asked more seasoned veterans what they thought of this notion, and I got thoughtful responses that mostly agreed with this. I certainly came away impressed and uplifted (who cared if it was pouring rain outside?).

I would describe the view from ten thousand feet of OSDI '06 as papers presented on file system enhancements, performance improvements as well as measurement techniques, methods for improving reliability of operating systems, and security. You can read the summaries to get the complete picture, along with the papers themselves online. I want to constrain myself to those papers that most excited, entertained, or disturbed me.

Code Defense

Feng Zhou described SafeDrive as a method for guarding against failed device drivers through language extensions. By adding annotations to device drivers, processing and compiling those drivers, then using them with a modified Linux kernel, crashes in those drivers will be avoided, even safely recovered from. This is a pretty amazing idea, different from previous papers that included microkernel designs, separate hardware protection domains, and other techniques for software fault isolation (including SFI and XFI). I found myself wondering how good this solution was, as it still relied on the programmer doing the right thing (annotating the code properly in all the required places), with the programmer being the weak link already.

Still, SafeDrive does come closer to the solution of a real problem. Device drivers are notoriously difficult to write and debug. And the word in security circles is that wireless device drivers are looking like tempting targets, as a successful attack here bypasses all current defenses—except, of course, for solutions such as SafeDrive or XFI.

Úlfar Erlingsson of Microsoft Research presented the paper on XFI, a system that creates safe extensions by binary rewriting of Windows x86 Portable Executables. The inline guards provide runtime checks before calling other functions or making computed jumps, thus guarding against executing code at unexpected locations because of bugs or attacks. Úlfar's chosen example was one of my very favorites, a bug in a JPEG decoder that allows code of an attacker's choice to be executed. In a live demo, the unsafe version crashed the browser, while the XFI-protected version of the library returned, preventing a browser crash (and a potentially exploited system).

Charlie Reis presented BrowserShield, a different approach to protecting Internet Explorer. In research sponsored by Microsoft, Charlie explained that many browser vulnerabilities could be defended against by rewriting scripts before they could be interpreted by a browser. In the sample implementation, scripts would be partially rewritten using the Microsoft firewall (ISA) as a proxy, along with a bit of JavaScript running within the browser itself. When used in conjunction with patches and anti-virus software, this approach prevented successful attacks related to 19 critical vulnerabilities found during 2005 in IE.

I want you to consider the implications of these fine research papers. We can't write secure or even defect-free software. Having accepted this fact, our finest scientists are designing a patchwork of systems that may make it possible to run buggy and dangerous systems as safely as possible. I know personally how difficult it is to write software , and I decided early on not to expose myself to the embarrassments suffered, because people continue to find security holes in software that is open source, over 20 years old, and written by programmers who are much better at it than I ever was. There is a temptation to blame programming languages, but we have yet to develop a safe programming language. After all, that is just another programming project (or meta-programming project), with all the complexities that that brings. Perhaps there is yet another way . . .

Charlie Reis also presented a WiP about building a browser where each site gets encapsulated by running within its own process. The Konqueror browser actually facilitates this work, according to Charlie. Divide-and-conquer has been a strategy that has worked well for some secure servers, such as Postfix and DJBDNS, and I personally feel that isolation of code and the use of least privilege is critical in any future solution. In the three papers just described, the approaches are to notice that software has been either exploited or simply run amuck, or to filter out attacks after they are known but not yet patched (BrowserShield).

At HotOS in 2005, I learned of an operating system called Asbestos. In Asbestos, all data gets tainted with labels as it flows through the system, and tainted data cannot escape via the network once it is mixed with data that has a different taint. During OSDI '06 Nickolai Zeldovich described HiStar, the successor to Asbestos. Like Asbestos, information flows get tainted. But HiStar goes beyond Asbestos in that everything gets labeled with categories, and these categories control how information can flow through the system. Nickolai used the example of running ClamAV, an open-source anti-virus system that must have read permission on all of an owner's files. HiStar can safely read all files because it prevents ClamAV from leaking any information read in any file.

HiStar approaches, and may have reached, what I'd like to see in a new, secure operating system. In HiStar, there is no root, nor is there a complex policy definition (as in SELinux); it is a system designed from the ground up to provide robust isolation. Combined with programming techniques, such as having a browser thread for each site visited, HiStar just might be the OS I have been dreaming of. It will take time to tell, plus additional time for me and others to understand this completely different OS with a Linux API.

Reduction and Configuration Management

Although log compression and configuration management might not seem related, there was an amazing paper by Chad Verbowski (also of Microsoft Research) and others that does unite these two disparate topics. Flight Data Recorder (FDR) (say, haven't I heard of another similarly named software project?) has the goal of capturing configuration and file changes from Microsoft systems and will be shipped with Windows Vista. Using a time window of only 6 ms, FDR captures all changes to system configuration–related registry entries and files, saves the log locally, then cleverly compresses it, without losing any interesting data, before uploading the compressed logs to a server. The goal was to capture data from thousands of servers while using less than 1% of network bandwidth, with a less than 20 MB/day logfile per system that can be analyzed in 3 seconds.

Sounds unbelievable, but FDR manages to compress each event into an average of 0.7 of a byte. The motivation for this clever work was the discovery that 33% of system outages were related to configuration changes, so tracking those changes was key to system reliability.

Speaking of system reliability, the final talk had to do with an interesting sensor network, one you might have heard about if you read science news. Geoff Werner-Allen explained some of the problems he and his co-authors had in monitoring Reventador, an active volcano located in an Ecudorian jungle. The current monitoring scheme relies on a barely luggable device powered by multiple car batteries. The team built small sensors powered by flashlight batteries, complete with a small seismometer and a microphone that captures subsonics typical of the rumblings of Reventador. The sensor communicated via a mesh to a single wireless uplink and then to a base station located in a hotel several kilometers away.

The sensors worked well. But in the hotel, the electric generators only ran about three hours a day, not enough to charge the batteries on the laptop used as the base station. Even with the occasional loss of the base station, the researchers were able to collect useful data. Geoff explained that time synchronization of the sensors had worked great in the lab but not so well in the field, but they were able to correct for time differences using some clever analysis.

If you think this is a cool project, well, so do I. There were some real downsides to the onsite research, though. One of the sensors shut down unexpectedly, and the cause turned out to be a chunk of rock that had smashed its antenna. Keep in mind that these researchers had to hike out and place, and later recover, the sensors on a very active volcano. Another issue had to do with insurgent groups that would block the only road leading back to "civilization." When this occurred, the supply truck would be blocked as well, and there would be no beer. Ah, the pains of doing field research.

After this talk, attendees for the most part stayed in the conference room. There were clusters of people gathered around the final three speakers. After fifteen minutes, USENIX staff had to urge people to move elsewhere so that the room could be reconfigured for HotDep '06.

The Lineup

We begin this issue with an article by Simson Garfinkel. Simson has been studying recycled hard drives and needed a way to store massive amounts of data and then analyze that data. While searching for the perfect hardware solution, he decided to try Amazon's Simple Storage Service (S3) and Elastic Computing Cloud (EC2). His article reports his experiences using these systems, their security, availability, and suitability for various uses. I really liked learning about EC2 and S3, as these services are a real hint of the distributed services we will see much more of in the near future. Simson also compares the Amazon services to other grid computing services. Like Simson, you might find that these services provide an alternative to buying and building your own grid computing cluster for a research project, but they are not quite ready for commercial use yet.

In the next article, Jorrit Herder and other project members provide an update to MINIX 3 that focuses on failure resilience. Herder writes a perfect companion article to the OSDI summaries, in that this paper looks at some of the same issues, such as device driver reliability, using a microkernel designed to run everything except a few core services in isolated tasks in userspace. I asked Jorrit why their work hadn't appeared at OSDI, and he said that this particular research wasn't far enough along at the OSDI paper submissions deadline.

Steven Hand and members of Xensource explain the issues involved in virtualization. I had become curious about what Intel and AMD were doing in the newer CPUs to provide hardware support for virtualization. I knew that a VM ideally sees an environment that appears exactly the same as a native, bare-metal environment, but at the same time the VM monitor must capture all direct accesses to the underlying hardware. There is also the messiness that occurs when an OS within a VM believes it controls the page maps, but in reality the VM page maps are just another level of abstraction on top of the monitor's page maps. Also, the Intel IA-32 architecture was not designed with VMs in mind, so there are instructions that behave differently when executed outside of ring 0 (in nonprivileged mode), causing more problems for designers of VMs. I hope this article will instruct you in what hardware manufacturers have done to help support the growing use and improve the performance of VMs.

In the Sysadmin section, Mark Burgess completes his cycle of articles about configuration management. Mark continues his exploration of how configuration management should be done by moving on from the representation of configuration information to talking about style of management. Should there be centralized, authoritative control, or something much more adaptive modeled on the economics of trade? As always, Mark provides a deeply thoughtful and very well written article.

Next, Leigh Griffin and John Ronan have written a guide to getting Xen servers up and running. Hewing to the operating system theme, which veered off into the world of VMs without much assistance, these two men decided that the world needed an easy-to-follow beginners' guide, and they set about writing it, then sharing it with *;login:* readers.

Robert Marmorstein and Phil Kearns have written about the tool they have been building that analyzes firewall policies, based on Linux netfilter. They start by building a series of tests to determine whether a firewall configuration actually worked as expected. From this experience they discovered that it made more sense to analyze firewall rulesets and then convert these rules into classes of systems that have similar policies. This technique can reveal unpleasant surprises in your firewall configurations, but ones that you will want to discover yourself, instead of having someone else do it for you.

David Blank-Edelman plays along with the operating system theme by explaining fork and how to work with multiple-thread Perl scripts. As David explains, this is a much deeper topic than can be handled in a column, but with his usual aplomb he provides us with working examples and pointers to cool modules that make using multiple threads a bit easier.

Robert Haskins decided to take a look at the various projects and products that support DHCP. As always, Robert writes from the viewpoint of an ISP, giving us a different perspective on a service that we generally configure and forget (until there is a change or a problem). Heison Chak joins in the VM subtheme, discussing how he runs different versions of Asterix within Xen virtual machines, and Robert Ferrell entertains us with his own views on operating systems.

In the Book Reviews, Elizabeth Zwicky, our official book reviewer, leads off with a long look at *Mastering Regular Expressions*. As Elizabeth writes, this is a topic that we all should learn more about, and she tells you just why this is important and what this book can do for you. Elizabeth next tackles a couple of management-level books, then has strong words about the final book on her list this issue. Next, Paul Armstrong discusses a good book he has read about IPv6. Sam Stover provides us with another in-depth look at a security book, and I follow on with two reviews of my own, including the newest in the Sysadmin Handbook series.

In Standards, some members of the C standard committee invite you to comment on changes related to support for threads. Finally, we have summaries for WORLDS, OSDI, and HotDep. We also received summaries of the Grace Hopper conference and workshop focusing on women in computing, and we end on that needed note.

Stuckness

I began my column by alluding to the tendency to stick with what is well known. Voyaging out beyond the frontier is risky, upsetting, and disturbing, because it suggests that perhaps what we have spent years learning is not the best solution. Pioneers have always had it rough. When we consider that Galileo was ordered to stand trial for heresy for his book suggesting that the earth revolves around the sun, our own trials pale. We do not face a literal burning at the stake for suggesting new ideas. We might be roasted for going up against the status quo, but that is only ego bruising, and at worst harmful to one's career.

I don't want to suggest (quoting Firesign Theater) that "Everything you know is wrong." Hardly. I do want to encourage you to keep on the lookout for new ideas, software, operating systems, and techniques that might very well solve problems in security, system administration, programming, and configuration management that so plague us today.

SIMSON GARFINKEL

commodity grid computing with Amazon's S3 and EC2

Simson L. Garfinkel is an Associate Professor at the Naval Postgraduate School and a Fellow at the Center for Research on Computation and Society at Harvard University. He is also a consulting scientist at Basis Technology Corp., which develops software for extracting meaningful intelligence from unstructured text, and a founder of Sandstorm Enterprises, a computer security firm that develops advanced computer forensic tools used by businesses and governments to audit their systems.

simsong@acm.org

AMAZON.COM RECENTLY INTRODUCED two new storage and computing services that might fundamentally change the way that we provision equipment for both e-commerce and high-performance computing. I learned about these services a few days after I had started looking for quotes to purchase a multiblade server with 10–20 TB of storage to further my own research in computer forensics. Rather than moving ahead with that purchase, I decided to evaluate whether or not I could use Amazon's offering for this real-world problem. My conclusion is that Amazon's offering is good enough for my research and will probably save me tens of thousands of dollars, but the system isn't yet ready for hosting serious e-commerce customers.

Amazon's Simple Storage Service (S3) and Elastic Compute Cloud (EC2) break up Amazon's awesome computer infrastructure into tiny little pieces that the company can incrementally rent out to any individual or business that needs e-commerce or high-performance computing infrastructure. Like Google and Yahoo!, Amazon has built computing clusters around the world, each with tens of thousands of computer systems. The theory behind these services is that economies of scale allow Amazon to run and rent out these services to businesses at a cheaper price than businesses can provide the services to themselves.

Amazon's Simple Storage System

Amazon announced S3 back in March 2006. The service allows anyone with a credit card to store information on Amazon's redundant and replicated storage network. Storage costs are 15 cents per gigabyte per month; data transfer is 20 cents per gigabyte. You may store an unlimited amount of information, and there is no setup fee.

The best way to think about S3 is as a globally available distributed hash table with high-level access control. You store data as a series of name/value pairs. Names look just like UNIX filenames; the value can be any serialized object between 0 and 5 GB. You can also store up to 4K of metadata with each object.

All objects in Amazon's S3 must fit in the same global namespace. The namespace consists of a "bucket name" and an "object name." Bucket names are available from Amazon on a first-come, first-serve basis. You can't have "foo" because it's already been taken, but you could probably get a bucket name with your last name, and certainly with your last name followed by a random seven-digit number. Bucket names are reasonably secure: You can list the names of the buckets that your account has created, but you can't list the buckets belonging to other people. You can only have 100 buckets per account, so don't go crazy with them.

Access control is based on these buckets. You can make your bucket readable by up to 100 Amazon Web Service Account IDs and read/write for up to another 100 IDs. You can also make a bucket world readable, although this is a lot like handing the world a blank check, since downloads do cost 20 cents per gigabyte. Near the end of this article we'll see how this cost compares with existing hosting services.

You send data to S3 using a relatively straightforward SOAP-based API or with raw HTTP "PUT" commands (a technique that has taken on the name "REST," short for Representational State Transfer). Data can be retrieved using SOAP, HTTP, or BitTorrent. In the case of BitTorrent, the S3 system operates as both a tracker and the initial seed. There is also a program called JungleDisk that lets you treat storage on S3 as if it were a remote file system; JungleDisk runs on Linux, Mac OS, and Windows.

After delving into the needless complexity of SOAP, I gave up and decided to use the pure HTTP/REST API. I'm using S3 to store images of hard drives that I have acquired or developed during the course of my research in computer forensics. With REST, I can store raw data without having to first base-64 encode it. I also found it much easier to code up a simple S3 REST implementation than to deal with all of the overhead required for the SOAP client.

S3 Performance and Security

I tested S3's performance with the REST API from networks at MIT, Harvard, and my house. Both universities have ridiculously fast connections to multiple Internet carriers. Despite this speed, both my upload and download speeds averaged between 1 and 2 MB per second, depending on the time of day. I saw similar performance from my house, where I have a 30 megabit per second Verizon FiOS connection. Based on the feedback in the Amazon developer forums, these performance figures are at the upper end of what others are seeing. One developer in Germany reported seeing between 10 and 100 kilobytes per second, depending on the time of day. Although this speed is simply not fast enough for doing serious computation, it is good enough for backups and for using S3 to deliver Web objects. Clearly, though, performance is an area that needs work for all S3 users.

Security is another important part of any storage system. Amazon's S3 has impressive support for privacy, integrity, and short-term availability. The long-term availability of the service is unknown, since it ultimately depends upon Amazon's internal level of commitment. Surprisingly, the weakest part of S3 is the service's authentication architecture. I'll discuss each of these issues next.

Data privacy is accomplished through the use of encryption and access control. If you want your data to be encrypted, encryption must be done before the data is sent to S3. You can protect names of the objects and other metadata by communicating with Amazon's Web servers using SSL with HTTPS on port 443. In my testing with a 2-GHz Intel Core Duo MacBook on MIT's network, downloading data from S3 over SSL took roughly 10% longer than downloading the same data over raw HTTP. This minor overhead demonstrates that the computational cost of encrypting data these days is really minor compared to other costs; nonetheless, encryption still isn't free.

Integrity for stored data is accomplished with an end-to-end check using the MD5 cryptographic hash as a checksum. When an object is stored to S3, Amazon's system computes the MD5 of that object and returns that hash with its response. My S3 implementation compares Amazon's computed hash with a hash that I computed locally. If the two don't match, my implementation resends the data. Although my implementation will send objects of any size, my code never sends objects larger than 16 MB.

Short-term availability is a reflection of Amazon's connectivity, the load on its servers and network fabric, and even the reliability of its code. In my testing I found that somewhere between 0.1% and 1% of all PUTs had to be retried because the PUT did not complete successfully. Normally PUTs succeeded on the second retry, but sometimes I needed to retry three or four times. An Amazon employee posting in one of the developer forums recommended implementing an exponential back-off for failed writes [1], but my implementation just retries as soon as it receives an error. After writing more than a terabyte to S3, I never experienced a failure that required more than four retries.

Long-term availability is a bigger question, unfortunately. Once the data is actually stored at S3, it's Amazon's responsibility to ensure that it remains available for as long as the customer pays the bills. Amazon claims that the data is stored on multiple hard drives in multiple data centers. Unfortunately, Amazon doesn't back up this claim with any sort of Service Level Agreement (SLA). There is also no backup or recovery service in the event that you accidentally delete some important data. As a result, it's important to maintain a backup of any important data stored inside S3.

The *authentication strategy* of Amazon Web Services (AWS) looks quite robust at first. Unfortunately, it's really a steel bunker built on a foundation of quicksand.

AWS supports a simple authentication strategy based on the SHA1-HMAC algorithm. Every AWS account has an Access Key ID and a Secret Access Key. The Access Key ID is a 20-character string that's used to uniquely identify your account; the Secret Access Key is a 41-character string that's used to digitally sign SOAP and REST requests. To sign a request, you simply compute the HMAC of the request parameters using the Secret Access Key as the key for the HMAC. This HMAC is sent along with the request. Amazon's servers, which know your Secret Access Key, compute the same HMAC. If the two HMACs match, then the request is authorized. Requests include a timestamp to prevent replay attacks.

The HMAC approach is fast, efficient, and pretty secure. The underlying weakness is that the credentials are downloaded from the AWS Web site. This means that anyone who knows your Amazon username and password can download your Secret Access Key. Since Amazon allows the password to be reset if you can't remember it, by simply clicking on a link that's sent to the account's registered email address, anyone who has control of your email system can effectively delete all of the information you have stored in S3. Amazon will have to rethink this authentication architecture before organizations can trust it with mission-critical information.

The other real problem with S3 is the cost structure: Currently it costs nearly as much to upload and download a piece of information as it costs to store that same data for three months. Although this may be a dramatic demonstration that the cost of storage is dropping much faster than the cost of bandwidth, these bandwidth charges make S3 simply unaffordable for many projects. Unfortunately, Amazon's pricing made the S3 service completely unusable for me until the company introduced its second grid-computing offering—a high-performance computing utility that let me move my computation close to my data.

The Elastic Compute Cloud

Amazon's Elastic Compute Cloud (EC2) makes S3's pricing strategy far easier to manage by eliminating the bandwidth charges for moving data between storage and computation.

As its name implies, EC2 lets you rent time on a "cloud" of computers. These computers are all the equivalent of 1.7-GHz Xenon servers with 1.25 GB of RAM and 160 GB of local disk. The cost of these machines is 10 cents per CPU per hour. As with S3, it costs 20 cents per gigabyte to move data between the rest of the Internet and EC2. However, there is no charge to move between EC2 and S3. According to Amazon, each virtual machine has 250 megabits per second of bandwidth, although how that translates to speed between EC2 and S3 depends upon a variety of factors.

The "machines" that Amazon delivers with EC2 are actually virtual machines, each running on top of the Xen platform. You create a virtual machine by storing a disk image inside S3 using special tools that Amazon provides and then running a Java program that instantiates the virtual machine. A second Java program lets you monitor the progress of the machine's creation; when it is ready, the script displays the computer's hostname. Obviously, the image that you instantiated should have an account that lets you log into the machine.

Because EC2 is based on Xen, it should support any Linux distribution as well as NetBSD, FreeBSD, Plan 9, and other operating systems. In practice, though, EC2 is largely based on the RedHat Fedora Core operating system, although there are instructions on the Internet for using it with Ubuntu distributions. I found this disappointing, because FreeBSD and Darwin, followed by Ubuntu, are my preferred operating systems.

Amazon makes no promises about the reliability of the EC2 computers: Each machine can crash at any moment, and they are not backed up. In my experience these machines don't crash, but, remember, computers do fail. If you want reliable storage, you can run two or more EC2 machines as a cluster. A better approach, though, is to have the EC2 machines store information in S3, which is sold as a reliable, replicated service.

What's really neat about EC2 is that you can build a small system and expand it as it becomes more popular, by simply bringing up more virtual computers. In fact, you could even bring up virtual machines on Thursdays and Fridays, if those are your busy days, and shut those machines down during the rest of the week.

The EC2 security model is similar to that of S3, except that commands are signed with an X.509 private key. Unfortunately, you download your private key from the AWS Web site, so the security still fundamentally depends on the AWS username and password. That private key can be used to start up machines, shut them down, and configure the "firewall" that protects your virtual machines on the EC2 infrastructure. The firewall allows you to control which IP addresses and ports on the Internet can reach which of your virtual machines. By default all ports are closed; you'll probably want to open the firewall to allow port 22 (ssh) through, at the least. Machines that run Web servers should probably have port 80 opened. And, of course, you'll probably want to configure the firewall so that your virtual machines can communicate with each other, at least on some ports.

Amazon had an early security problem with EC2: The company was neglecting to wipe the computer's virtual disk drives before switching them from one customer to another. That problem has since been corrected.

s3_glue: A C++ implementation of the S3 REST API

As I already mentioned, I've been using S3 and EC2 for my research in computer forensics. As part of my research I've created an open source system for imaging hard drives and storing the results in highly compressed but random-access disk images [2]. This October I added support for S3 to the library so that images could reside on the local computer or on Amazon S3.

Amazon provides code samples for S3 in C#, Java, JavaScript, Perl, Python, and Ruby. Although these examples are instructive, my disk-imaging system is written in C++ for performance reasons. To make the code usable for others I separated out the basic S3 implementation from the code that is specific to my forensics library. The implementation can be downloaded from http://www.simson.net/s3/. It uses libcurl [3] for HTTP.

Recall that S3 objects are all given object names and that these objects are in turn placed into buckets. The REST API turns object and bucket names into URLs of the form http://s3.amazonws.com/bucket-name/object-name. Data is downloaded with an HTTP GET and uploaded with an HTTP PUT. There is also provision for setting up a virtual host (e.g., http://bucket.s3 .amazonws.com/object-name), which makes it somewhat easier to have S3 directly serve Web content to browsers.

S3 requests are authenticated through additional terms that are added to the query section of the URL. The "Signature=" term includes the HMAC of the requests headers represented in a canonical form and the user's AWS Secret Access Key. The "Expires=" term allows you to specify when the query will expire. Finally, the "AWSAccessKeyId=" term specifies the requestor. Remember, authentication isn't needed for buckets that are world-readable or world-writable.

HTTP 1.1 allows a client to request a range of bytes; S3 implements this part of the protocol, allowing you to request a few bytes of a very large object. S3 limits an object overall to 5 GB, although a bug in Amazon's load balancers means that objects are effectively limited to 2 GB in size. I store disk images larger than 16 MB as multiple pages, each of which is 16 MB in length before being compressed, so the 2GB limitation wasn't a problem for me.

My S3 implementation provides simple and efficient C++ functions for listing all buckets that belong to a user, making a new bucket, deleting a bucket, selectively listing the contents of a bucket, getting an object, saving an object, and removing an object. The code supports arbitrary name/value pairs for metadata on an object. This metadata needs to be stored with an object but can be independently retrieved.

S3 is pretty powerful as far as it goes, but there is a lot of functionality missing. There is no way to rename an object, for example. There is no way to search—you can't even search for objects of a particular length or that have a particular metadata field in their headers. In this way, S3 is a lot like Berkeley DB or the Python "dictionary" data structures: You can store data, get it back, and iterate. Anything else is up to you. Because objects can be listed and retrieved in lexical sort order, I expect that many applications will encode a lot of information inside the file name. That's what I did.

In addition to my S3 implementation, I've also created a command-line utility called "s3." This program is mostly for testing the S3 library and maintenance of the S3 system. It implements UNIX-like commands for listing the contents of a bucket, copying the contents of an object to standard output, deleting an object, deleting a set of objects, and managing buckets. This program is also available from my Web site.

Crunching the Numbers

In my research I have been running programs that take literally a month to run on a workstation with a terabyte hard drive. With Amazon's EC2 and S3 I can split the task up and run it on 30 virtual computers over the course of a day, for roughly $72. Or I can run it on 60 virtual computers for 12 hours, again for $72. This simple example demonstrates the big advantage of renting time on another organization's grid over building your own. Unless you have enough work to occupy your grid 100% of the time, every hour that a computer isn't working is an hour that you paid for but received nothing in return.

There are other alternatives to Amazon's offerings. Dreamhost, an ISP that I use for some of my personal work, just dramatically lowered the cost of its Web-hosting plans. For just $9.95/month you can have 200 GB of storage (automatically increasingly by 1 GB each week) and 2 TB a month of bandwidth. Amazon would charge $30 for the same storage but a whopping $400 for that much bandwidth. Unfortunately, Dreamhost had significant reliability problems this past summer.

Pair.com, a premium Web-hosting company at the other end of the cost/performance spectrum, charges $9.95/month for a basic Web-hosting account with 500 MB of disk storage and 40 GB per month of bandwidth. Amazon would charge 7.5 cents for the storage and $8 for the same bandwidth. Pair.com will rent you a dedicated 2.8-GHz Celeron computer with 512 MB of RAM and an 80-GB hard drive with 600 GB per month of traffic for $249/month. Amazon's EC2 machines are faster, have twice the RAM and twice the local disk, and cost just $72/month, although that 600 GB per month of bandwidth will cost you another $120. On the downside, Pair will provide 24/7/365 server monitoring and support, whereas the Amazon servers can crash and there is no support other than what's available in the developer forums. But Pair won't let you bring up 50 machines after lunch and then shut them down when you go home for dinner.

Whereas Amazon's EC2 is an automated provisioning system for virtual machines, another approach is being pursued by 3Tera, a small company in Aliso Viejo, California. 3Tera has developed an operating system for grid computing that allows a single application to be deployed across multiple machines in an automated fashion. As of this writing 3Tera has licensed its technology to UtilityServe, which will run AppLogic-based applications for between 75 and 99 cents per RAM-GB hour; bandwidth is $1.49 to $1.99

per GB; the company includes between 100 and 4000 GB of storage in its base packages, and it sells additional storage for $99 per 50 GB.

Conclusions

S3 and EC2 are obviously both young and immature services: They are tantalizing in what they promise, but Amazon needs to address the issues of authentication, availability, and long-term stability before businesses should seriously rely on this offering. I wouldn't trust my business to S3 or EC2 without a signed contract in place that clearly outlined Amazon's obligations and my recourse against Amazon if those obligations were not met.

At the same time, I think that S3 and EC2 are a taste of the kinds of computer utility services that will be available in the not-so-distant future. High-quality storage, computation, and bandwidth will be available at commodity prices. With any luck other companies will reimplement the server side of Amazon's APIs, making it possible to move a service easily from one provider to another. With these kinds of services, I can spend my time using a computer utility, rather than building one from blade servers and RAID boxes. I can then devote my time to worrying about algorithms instead of rack space, electricity bills, and cooling.

Because it is running so many computers, Amazon can run them a lot cheaper than I can. Assuming that the company can make good on its implicit availability and bandwidth commitments, this is going to be a very compelling offering.

REFERENCES

[1] http://developer.amazonwebservices.com/connect/thread.jspa?messageID=46813.

[2] http://www.afflib.org/.

[3] http://curl.haxx.se/.

JORRIT N. HERDER, HERBERT BOS, BEN GRAS, PHILIP HOMBURG, AND ANDREW S. TANENBAUM

roadmap to a failure-resilient operating system

Jorrit Herder holds an M.Sc. degree in Computer Science (cum laude) from the Vrije Universiteit in Amsterdam and is currently a Ph.D. student there. His research focuses on operating system reliability and security, and he is closely involved in the design and implementation of MINIX 3.

jnherder@cs.vu.nl

Herbert Bos obtained his M.Sc. from the University of Twente in the Netherlands and his Ph.D. from the Cambridge University Computer Laboratory (UK). He is currently an assistant professor at the Vrije Universiteit in Amsterdam with a keen research interest in operating systems, high-speed networks, and security.

herbertb@cs.vu.nl

Ben Gras has an M.Sc. in computer science from the Vrije Universiteit in Amsterdam and has previously worked as sysadmin and programmer. He is now employed by the VU in the Computer Systems Section as a programmer working on the MINIX 3 project.

beng@cs.vu.nl

Philip Homburg received a Ph.D. from the Vrije Universiteit in the field of wide-area distributed systems. Before joining this project, he experimented with virtual memory, networking, and X Windows in Minix-vmd and worked on advanced file systems in the Logical Disk project.

philip@cs.vu.nl

Andrew S. Tanenbaum is a professor of computer science at the Vrije Universiteit in Amsterdam. He has written 16 books and 125 papers and is a Fellow of both the ACM and the IEEE. He firmly believes that we need to radically change the structure of operating systems to make them more reliable and secure and that MINIX 3 is a small step in this direction.

ast@cs.vu.nl

IN RECENT YEARS, DEPENDABILITY AND security have become prime concerns for computer users. Nevertheless, commodity operating systems, such as Windows and Linux, fail to deliver a dependable and secure computing platform. The lack of proper fault isolation in the monolithic kernel of commodity systems means that a local failure can easily spread and corrupt other components. A single bug, say, a buffer overrun in a network driver, can overwrite crucial data structures, causing a subsequent, but unrelated, action to trigger a fatal exception. Recovery is usually not possible except by rebooting the computer.

While software is buggy by nature, device drivers are known to be especially failure-prone [1, 2]. It is irrelevant whether the failures are due to hardware glitches, improper device documentation, the arcane kernel programming environment, lack of quality control, limited testing, or code immaturity. The crucial point is that pieces of untrusted, third-party code, such as drivers and other extensions, run inside the kernel and can potentially take down the entire system. This property is inherent to the monolithic design used in commodity operating systems, and it cannot be solved through mere programming effort.

Our approach to dependability is to cope with imperfection and counter the more fundamental problem that driver failures threaten to take down the entire operating system. In particular, we have enhanced the MINIX 3 operating system with fault-resilience techniques to improve operating system dependability. We accept the fact that software is not perfect and probably never will be, and anticipate failures in device drivers and other critical operating system components. Our system is designed to withstand such failures and can often repair itself in a manner that is transparent to applications and without user intervention.

MINIX 3 has been under development for the past two years and is becoming increasingly mature. We have already reported on MINIX 3's multiserver architecture [3] and mechanisms to deal with dead device drivers [4]. In this article we loosely summarize what we have done to make MINIX 3 resilient against failures, where we stand now, and what is left for future work. An overview of the highlights of MINIX 3's development and a tentative roadmap for future work are given in Figure 1. As the figure shows, we hope to release a thoroughly tested fault-resilient version of MINIX 3 early next year.

The remainder of this article is organized as follows. We start out with a short introduction to the recent history of MINIX 3. Then we give an overview of the fault-resilience mechanisms we have implemented thus far and perform a brief reality check. In the end, we discuss our current and future work that will eventually lead to the release of a fault-resilient version of MINIX 3 and then conclude.

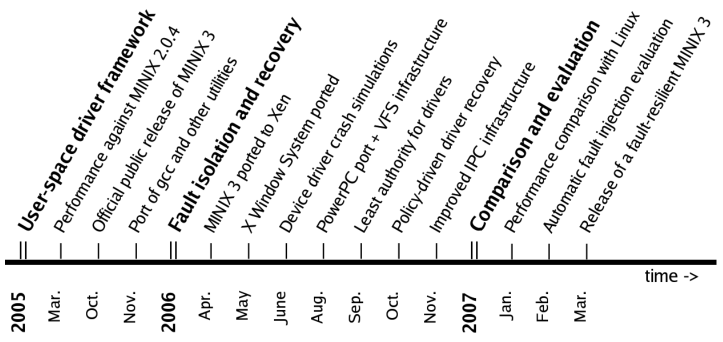


FIGURE 1: OVERVIEW OF THE HIGHLIGHTS OF MINIX 3'S DEVELOPMENT AND TENTATIVE ROADMAP FOR FUTURE WORK.

The Recent History of MINIX 3

As a base for MINIX 3 we used MINIX 2, which already ran some servers in user space but still had in-kernel device drivers. Starting in late 2003, we removed the drivers from the kernel, developed a user-space device driver framework, and officially released MINIX 3 in October 2005. Since then, the system has been downloaded over 100,000 times and a small but growing user community has formed to support MINIX 3. The official Web site (www.minix3.org) and newsgroup (comp.os.minix) are frequented by many enthusiasts who want to participate in our quest for a secure and dependable operating system.

The architecture of MINIX 3 is shown in Figure 2. All servers and drivers run as independent user-mode processes—each encapsulated in a private address space protected by the MMU hardware—on top of a tiny microkernel of under 4000 lines of executable code. The bottom half of the microkernel is responsible for programming the CPU and MMU, interrupt handling, and IPC. The in-kernel clock and system task provide an interface to kernel services, such as I/O and alarms, for the user-mode parts of the operating system. The most common servers provide file system services and process management functionality. A special server, called the *reincarnation server*, manages all servers and drivers and constantly monitors the system's well-being. With this design as a stable base we were able to achieve failure resilience, as we discuss below.

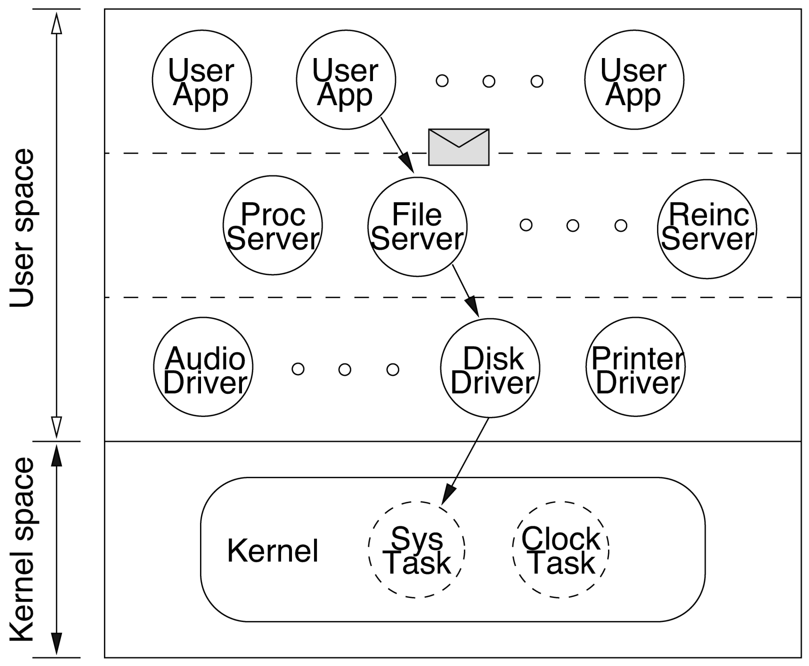


FIGURE 2: ARCHITECTURE OF MINIX 3. THE OPERATING SYSTEM IS COMPARTMENTALIZED IN USER SPACE, BUT COMPONENTS CAN INTERACT WITH EACH OTHER BY PASSING MESSAGES. THE POSIX READ() CALL, FOR EXAMPLE, IS TRANSFORMED INTO A REQUEST TO THE FILE SERVER, WHICH ASKS THE DISK DRIVER TO READ THE BLOCK FROM DISK, WHICH, IN TURN, ASKS THE IN-KERNEL SYSTEM TASK TO PERFORM PRIVILEGED OPERATIONS, SUCH AS WRITING TO THE DISK'S I/O PORTS.

MINIX 3 currently runs over 400 standard UNIX applications, including the X Window system, two C compilers, language processors, several shells, many editors, a complete TCP/IP stack that supports BSD sockets, a virtual file system infrastructure, and all the standard shell, file, text manipulation, and other UNIX utilities. The POSIX-compliant interface offered by MINIX 3 facilitates porting of common Linux and BSD applications. For example, porting an application to our system is often simply a matter of recompilation.

Performance measurements on a 2.2-GHz Athlon show that the overhead of our system is 5–10% compared to the base system (MINIX 2) with in-kernel device drivers [3]. User-mode Fast Ethernet runs at full speed, and our user-mode disk drivers show an average overhead of about 8% to perform disk I/O compared to in-kernel disk drivers. Since neither MINIX 2 nor MINIX 3 has been tuned for performance, we expect to find a somewhat higher overhead when the system is compared to Linux or FreeBSD. Nevertheless, MINIX 3 feels fast and responsive. For example, the boot time, as measured between exiting the multiboot monitor and getting the login prompt, is less than 5 seconds. At that point, a POSIX-compliant operating system is ready to use.

Crash simulation experiments show that our system can withstand failures and gracefully recover by restarting the driver rather than rebooting the entire computer [3]. For example, in one experiment, we used wget to retrieve a 512-MB file from the Internet while repeatedly killing the Ethernet driver every 4 seconds. The network transfer successfully completed in all cases, with a performance degradation of just 8%. Although these experiments prove the viability of our approach, manually killing a driver to simulate a crash is not representative for many device driver failures. Therefore, we recently started to experiment with automatic fault injection, with promising results, as discussed next.

Achieving Fault Resilience

The key principles we used to make MINIX 3 failure resilient are fault isolation, defect detection, and run-time recovery. Fault isolation is required to prevent problems from spreading and limit the damage bugs can do. When a bug is properly caged it becomes easier to pinpoint the defect, and recovery may be possible. In the following we briefly discuss how we realized each principle in MINIX 3. As an aside, this model may have consequences for the accountability of software vendors [5], but in this article, we focus on the technical aspects of our design.

FAULT ISOLATION

Although we have fully compartmentalized the operating system in user space, as illustrated in Figure 2, isolation cannot be achieved by means of address-space separation alone. This is because servers and drivers need potentially dangerous mechanisms to communicate and share data in order to make the system work. Instead of granting such powers to all processes, we have carefully reduced the privileges of each according to the Principle Of Least Authority (POLA). Each device driver, for example, is loaded with a protection file that precisely lists its resources, including device memory, I/O ports and IRQ lines, and IPC capabilities. The reincarnation server ensures that the restriction policy is in place before the newly started driver gets to run.

Memory protection is realized by combining MMU and kernel protection. The MMU ensures that a process cannot directly access another process's memory. However, to prevent memory corruption in processes that need to share data, processes can grant access to precisely specified memory areas by sending a capability that is checked by the kernel when data is read or written. It has to be noted that DMA is still a potential danger, but this is a hardware problem and not a limitation of our system. Fortunately, I/O MMUs are becoming more common, and when we have the proper hardware we will solidify our defenses.

DEFECT DETECTION

The reincarnation server is the central component that guards all servers and drivers in the system. During system initialization the reincarnation server adopts all processes in the boot image as its children; servers and drivers that are started on the fly also become its children. Therefore, in line with the POSIX model, the reincarnation server will be notified by the process manager when a system process exits. Based on the exit status retrieved from the process manager, three cases can be distinguished: a process exit or panic, a CPU or MMU exception, or a user signal. Each of these cases is considered as a separate defect class.

In addition, the reincarnation server has three other ways to monitor the system for anomalies. When a driver's protection file specifies so, the reincarnation server periodically pings the driver and expects it to reply with a heartbeat message. Not responding is considered a defect and initiates the recovery procedure. Furthermore, the reincarnation server acts as an arbiter in case of problems. For example, the network server can request replacement of an Ethernet driver that does not adhere to the multiserver protocol. Finally, the user can instruct the reincarnation server to dynamically update the system. In this way, when a bug or other vulnerability is found, the defective component can be replaced on the fly as soon as a patch is available.

RECOVERY PROCEDURE

When a server or driver is started, it can be associated with a (generic) shell script that governs its recovery procedure. When a defect has been detected, the reincarnation server looks up the malfunctioning process's recovery script from its internal tables and runs it. All relevant parameters, such as the component that failed, defect class, and failure count, are passed along so that the script can decide what to do. The simplest policy may log the error and shut down the malfunctioning component, but in many cases it is possible to replace it with a fresh copy. A sample policy script that uses a binary exponential backoff protocol in restarting failed components is shown in Figure 3.

```
component=$1            # args from reinc. server
reason=$2               # dynamic update = 6
repetition=$3           # current failure count

if [ ! $reason -eq 6 ]
then
    sleep $((1 << ($repetition - 1)))
fi
service restart $component
```

FIGURE 3: RECOVERY SCRIPT THAT USES A BINARY EXPONENTIAL BACKOFF PROTOCOL IN RESTARTING A FAILED COMPONENT TO PREVENT BOGGING DOWN THE SYSTEM IN CASE OF REPEATED FAILURES, UNLESS THE USER EXPLICITLY REQUESTED A DYNAMIC UPDATE [4].

Once a component has been restarted it needs to be reintegrated into the system. First, the reincarnation server updates the corresponding name server entry in the *data store*, which uses a publish/subscribe mechanism to inform dependent components about the new system configuration. For example, the file server will be notified when a disk driver is restarted and its new IPC endpoint is published in the data store. At this point, the file server can reinitialize its own tables and can request the driver to reinitialize itself. If the restarted component lost state during its crash, it can, in principle, retrieve a backup made by the crashed component from the data store. In our current prototype implementation, however, all drivers are stateless or can be reinitialized from the server level. Recovery of stateful components is not used by our prototype implementation, but the mechanisms required to do so are in place.

Reality Check

Although our system has been designed to recover from failures in both servers and drivers, there are limits to what we can do. Since the core operating system servers maintain a lot of state, recovery is currently not supported. For example, the process server keeps track of process IDs, child-parent relationships, alarms, and more. Although a crash does not take down the entire system, all user programs will be seriously hampered. Nevertheless, our approach deals with an important class of problems, since 70% of the operating system typically consists of driver code, with reported error rates 3–7 times higher than those of ordinary code [2].

The assumption underlying our recovery procedure is that failures are transient and can be repaired by replacing malfunctioning components. For example, rare timing causing an exception, software aging from memory leaks, and the like may bring down a component, but in many cases a restart will cure the problem. Moreover, our design not only helps when disaster strikes but also opens the possibility for ante-mortem updates. At any point in time the user can update the system by requesting the reincarnation server to replace a component under suspicion with a new one. This feature helps system administrators to keep the system in good shape without system downtime. It may also be useful in embedded systems that need to automatically replace components when new versions are available.

Work for the Near Future

The general fault-resilience mechanisms presented here are currently implemented in MINIX 3, but more work needs to be done, as shown in Figure 1. In particular, a better performance assessment and a more thorough evaluation of MINIX 3's ability to recover from failures are needed. We have already studied the performance of MINIX 3 compared to MINIX 2 and have concluded that the transformation of in-kernel drivers into user-space drivers resulted in a performance overhead of about 5–10%. We are currently investigating how MINIX 3 compares to other UNIX-like operating systems such as Linux and FreeBSD. Preliminary results show that the overhead is somewhat higher, but we do not have the precise numbers yet. However, because MINIX 3 is not optimized for performance—in contrast to the other systems—it will be hard to tell to what extent the overhead is due to MINIX 3's multiserver design or to the differences in, for example, compiler quality, memory management algorithms, and file system implementation.

In addition, we are working on a better evaluation of MINIX 3's ability to survive failures in critical operating system components and transparently repair the system. Our current focus has been to reincarnate dead device drivers, but recovery from failures in stateful components has our interest as well. Furthermore, we have mostly tested the system's failure resilience by manually killing components, but this approach is not representative for failures that are caused by, say, programming bugs. Therefore, we recently ported the fault injection tool used by Nooks [6] to MINIX 3, which allows us to inject more representative faults by mutating the driver binaries. This method already proved its value, as we discovered a small number of bugs in the core components, which we fixed. More importantly, the results thus far indicate that our system is indeed capable of surviving and recovering from common failures.

Summary and Conclusion

In this article, we briefly described the recent history of our work on MINIX 3 and we showed how the modular design of MINIX 3 can be exploited to achieve failure resilience within the operating system. A timeline with the highlights of MINIX 3 thus far and a tentative roadmap for future work was presented. Although more development and testing is needed, the principles discussed here show that it is possible to improve operating system dependability by revisiting design choices that were made decades ago. All in all, we believe that MINIX 3 has serious potential to claim a niche in the operating system market—for example, on moderately powerful embedded systems where security and dependability are at stake, such as mobile phones, set-top boxes, and medical appliances.

REFERENCES

[1] T.J. Ostrand and E.J. Weyuker, "The Distribution of Faults in a Large Industrial Software System," *Proc. 2002 ACM SIGSOFT Int. Symp. on Software Testing and Analysis*, ACM, pp. 55–64, 2002.

[2] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. Engler, "An Empirical Study of Operating System Errors," *Proc. 18th ACM Symp. on Operating System Principles*, pp. 73–88, 2001.

[3] J.N. Herder, H. Bos, B. Gras, P. Homburg, and A.S. Tanenbaum, "Reorganizing UNIX for Reliability," *Proc. 11th Asia-Pacific Computer Systems Architecture Conference*, pp. 81–94, 2006.

[4] J.N. Herder, H. Bos, B. Gras, P. Homburg, and A.S. Tanenbaum, "Who's Afraid of Dead Device Drivers," Technical Report IR-CS-D29, Vrije Universiteit, Amsterdam, 2006.

[5] A.R. Yumerefendi and J.S. Chase, "The Role of Accountability in Dependable Distributed Systems," *Proc. 1st Workshop on Hot Topics in System Dependability*, 2005.

[6] M.M. Swift, M. Annamalai, B.N. Bershad, and H.M. Levy, "Recovering Device Drivers," *Proc. 6th Symp. on Operating System Design and Implementation*, pp. 1–15, 2004.

STEVEN HAND, ANDREW WARFIELD, AND KEIR FRASER

hardware virtualization with Xen

Steven Hand is a Senior Lecturer at the University of Cambridge and a founder of XenSource, the leading open source virtualization company. His interests span the areas of operating systems, networks, and security.

steven.hand@cl.cam.ac.uk

Andrew Warfield completed his Ph.D. at the University of Cambridge in May 2006. He now works as the lead storage architect for XenSource, and is also an Adjunct Professor in the Computer Science Department at the University of British Columbia. Andrew currently lives in Vancouver, Canada.

andrew.warfield@cl.cam.ac.uk

Keir Fraser is an EPSRC academic fellow and lecturer at the University of Cambridge and a founder of XenSource. He completed his Ph.D. in 2004 and now manages the Xen project.

keir.fraser@cl.cam.ac.uk

XEN IS A VIRTUAL MACHINE MONITOR (VMM) that we've been developing at the University of Cambridge for the past several years. As a VMM, Xen allows a single physical computer to be divided up into a number of smaller virtual computers, each running its own operating system and applications. Xen is free, but is also available as part of a number of commercial offerings.

Xen was designed from day one to get every last ounce of performance out of commodity x86 machines. The past year has seen chip vendors such as Intel and AMD launch next-generation processors that provide hardware assistance for virtualization. In this article, we provide a background to Xen, show how these new hardware features can be used to provide high-performance virtualization even for proprietary or legacy operating systems, and look toward the future of hardware virtualization.

Xen: Virtualization for the Masses

System virtualization technology has been around for over four decades. Pioneered by IBM with VM/370, system virtualization allows you to divide a single powerful computer into a number of smaller, less powerful computers called virtual machines. Each virtual machine runs its own operating system and applications and is strongly isolated from other virtual machines. This provides enhanced flexibility, management, and security.

For many years, virtualization was limited to "big iron" machines. However, the increasing power and prevalence of commodity off-the-shelf (COTS) systems have made virtualization an attractive technology for regular x86 boxes. Virtual machine monitors (VMMs) such as Xen and VMware provide system virtualization for COTS systems and are now in use on hundreds of thousands of machines worldwide.

There is, however, a problem: The Intel IA-32 architecture was not designed with virtualization in mind, and so certain instructions which should trap when executed with insufficient privilege simply behave differently, and various privileged states are visible even to user-mode software. This means that traditional system virtualization approaches are insufficient. Instead, new techniques are needed to make x86 VMMs a reality.

THE PROBLEM WITH IA-32

In a classic 1974 paper, Popek and Goldberg describe the basic principles for system virtualization. In particular, they identify three requirements for something to be considered a VMM:

- Equivalence: Software running in a virtual machine should behave exactly as it would on a "real" machine (barring timing effects).
- Performance: The vast majority of machine instructions executed when running within a virtual machine should be executed "natively" on the real hardware, and without intervention from the VMM.
- Resource control: The VMM must be in complete control of the hardware resources.

These requirements typically lead to a "trap and emulate" approach in which the VMM runs hosted operating systems in user mode. Most of the time, the software runs exactly as it would on a real machine, but if the operating system (OS) attempts to perform a privileged operation, a hardware trap will occur. Since the VMM executes in supervisor mode, it can catch this hardware exception, inspect the state of the OS that caused it, and emulate the behavior that would have occurred on real hardware. The VMM can then resume the virtual machine, allowing execution to continue.

This approach will satisfy Popek/Goldberg requirements as long as the processor is guaranteed to trap whenever any privileged operation is attempted in user mode. Unfortunately, the original IA-32 architecture does not guarantee this: Various instructions which should trap don't, and in some cases they simply have different semantics than they would have on a real machine. In addition, certain kinds of privileged machine state (such as page tables and segment descriptor tables) reside in memory and hence are visible to user-mode software.

SOLVING THE PROBLEM

There are two main ways that we can work around these problems with the IA-32 architecture: binary rewriting and paravirtualization. In the former approach, the VMM dynamically scans the memory of the guest OS looking for problematic instructions, and rewrites any it finds with alternative instruction sequences. This approach is costly and fragile, especially as the x86 uses variable-length instructions, but it can be made to work in most cases.

The latter approach, used by Xen, modifies the operating system source code to make it aware that it is running on top of a VMM. The resulting enlightened operating system can run extremely efficiently in a virtual machine environment: Typically an overhead of just 1% is observed. It can also work in cooperation with the VMM to provide advanced features such as CPU, memory, and device hotplug, or even live migration, seamlessly relocating a running virtual machine from one physical node to another.

At the time of writing, there are enlightened versions of modern Linux, BSD, and Solaris operating systems that run efficiently on Xen. Furthermore, Microsoft has announced that it is working on an enlightened version of its forthcoming "Longhorn" operating system.

Nonetheless, there is a large existing base of legacy operating systems that cannot use the paravirtualized technique. To support these, we need to look to the processor vendors and their recently introduced hardware support for virtualization.

Hardware Virtualization

To work around the problems with the original IA-32 architecture, both major processor vendors have recently introduced hardware extensions. Intel's technology is called VT-x, or VT for short, and ships in most recent processors including the Xeon 51xx series, the Xeon 71xx series, and the Core Duo and Core 2 Duo processors. The equivalent AMD technology, called AMD-V, ships in recent (stepping F2) Opteron and AMD64 processors. (Although there are some important differences between VT-x and AMD-V, this article will avoid them in the interests of simplicity. More technical details on both technologies are available from the references given at the end of the article.)

These hardware virtualization (HV) technologies both operate by making the processor aware of multiple virtual machine contexts (VMCs). A VMC is analogous to a process control block (PCB) in an operating system: a copy of the state required to resume or schedule that virtual machine. The VMC holds a strict superset of the contents of a PCB, however; for example, in addition to the values of general purpose and floating-point registers and flags, the VMC will contain the values of the processor control registers (such as cr0, cr4, and cr8). The VMC will also include the values of certain model-specific registers (MSRs) such as CSTAR and EFER, as well as an expanded version of each segment selector.

Hence with hardware virtualization technology, the VMM acts somewhat like a traditional operating system, but scheduling virtual machines instead of processes. The HV extensions include instructions to launch and/or resume a given VMC, which causes the hardware to load the relevant processor state and continue execution. The new execution environment includes its own privilege levels ("rings" in IA-32 terminology), so the operating system kernel can operate in what it believes is supervisor mode and runs its own applications in what it believes is user mode. The new execution environment can also operate in a completely independent processor mode; for example, the VMM can run in 64-bit mode, one VM in 32-bit paged mode, and another VM in 16-bit real mode.

The act of launching and/or resuming a VMC is sometimes called entering a virtual machine and, as previously mentioned, can be seen as analogous to scheduling a process in an operating system. However, things are different when we consider the opposite case: exiting a virtual machine. Whereas in an operating system a process will usually only be descheduled as a result of an interrupt or system call, a VMM wishes to intercept execution in a much wider range of situations. Examples include instructions that manipulate processor interrupt state, interactions with the TLB, instructions that access or update control registers or MSRs, and attempts to put the processor into a halt state.

To allow maximum flexibility, hardware virtualization allows the VMM to select precisely which events it wants to intercept. The selected events will cause a vmexit, effectively a trap from the running virtual machine into the VMM. Since the set of allowable events includes all privileged x86 instructions, this allows implementation of the classic trap-and-emulate scheme, and hence it enables efficient virtualization of nonparavirtualized operating systems.

XEN: HARDWARE VIRTUAL MACHINES

Xen uses the hardware virtualization technologies described above to enable support for legacy or proprietary operating systems. It uses the trap-and-emulate approach to deal with privileged instructions, which enables efficient virtualization without the overhead or fragility of binary rewriting.

However, existing hardware virtualization support only provides part of the solution required to enable the execution of hardware virtual machines. In particular we can consider a modern COTS system as comprising three main components:

- The processor
- The memory subsystem
- The I/O subsystem

Current VT-x and AMD-V technologies help with processor virtualization, but they do not deal with memory or I/O. Software support within Xen is required to complete the picture.

VIRTUALIZING MEMORY

Most operating systems expect a contiguous range of physical memory (RAM) starting from address 0x0. When running on top of a VMM, however, many operating systems are run concurrently, and will be allocated varying amounts of physical memory from the overall pool. One job for the VMM then is to translate between physical addresses as seen by an individual virtual machine (“guest physical addresses” or just “physical addresses” for short) and the actual physical addresses as seen by the real hardware (“machine addresses”).

There are two interesting cases to consider depending on which mode the virtual machine is executing in: (1) real mode or protected mode or (2) paged mode. In the former case, addresses generated by the virtual machine are physical addresses (albeit modified by segment translation); in the latter case the virtual machine generates virtual addresses, which it expects to be translated via the processor’s paging mechanism. Xen handles both of these cases by the same means: shadow page tables.

The basic idea is simple: The guest creates and manages its own page tables, which translate from virtual to guest physical addresses. When the guest wishes to use an address space for the first time, it will update its cr3 register to point to the root page table. Using hardware virtualization, this causes a vmexit, which allows Xen to create a shadow copy of the root page table. Unlike the guest version, the version used by Xen translates from virtual addresses directly to machine addresses, and so it can be used by the “real” (hardware) MMU.

For space efficiency, it is not necessary to make shadow copies of every part of the current page table; instead, copies can be made on demand as the operating system (or its hosted processes) access various parts of the virtual address space. In addition, Xen must be able to track any updates made by the guest to its page tables and reflect the appropriate changes in the shadow copies. For these reasons, Xen ensures that guest page table pages are always mapped read-only. As a consequence, any page table modification attempted by the guest will result in a fault into the VMM. Xen can then intercept the access and maintain coherence between guest and shadow page tables.

The current implementation (in Xen 3.0.3) has been designed for high performance and includes a number of optimizations above and beyond the scheme just described. It also incorporates support for other modes of operation, which may be used for the live migration of virtual machines. Interested readers can learn more from the references given at the end of the article.

VIRTUALIZING I/O

The final part of the picture entails dealing with the I/O subsystem. This includes simple platform devices (such as timers and interrupt controllers), disk drives, video cards, USB controllers, and network interface cards.

COTS systems expect to access such devices either via I/O instructions (direct or memory mapped) or via memory-mapped PCI bus addresses. As with page table updates, Xen intercepts any such accesses and emulates the behavior of device hardware. For platform devices, this is relatively straightforward since they perform no actual I/O per se. Other devices are more complex and may require the ability to send packets on a real network interface card or read data from a real storage device.

Xen supports these I/O devices by instantiating a device model process for each virtual machine. This emulates the behavior of the rest of the platform hardware, which can be configured to include the desired number and type of network interface cards, IDE controllers, graphics cards, and USB controllers. These virtual devices handle any accesses made by device drivers running in the virtual machine, mirroring the state transitions that would be made by an equivalent piece of hardware. They also interact with a—potentially virtualized—instance of that hardware: For example, a disk device can be represented as a sparse file.

Providing an emulated platform allows operating systems to run without requiring that they are at all aware of virtualization. However, emulation can be rather slow, particularly for devices such as network interface cards, which can require many (emulated) bus cycles to, for example, transmit a packet.

Hence Xen also provides the ability to load new, virtualization-aware device drivers into the operating system after it has been installed. These paravirtualized drivers understand the underlying VMM, and hence they can more directly interact with the virtual hardware. In the case of networking, this can increase performance by an order of magnitude.

Next Steps in Hardware Virtualization

We've seen how Xen uses existing hardware virtualization of the processor to efficiently and robustly run unmodified operating systems, augmenting this with software support for memory virtualization (shadow page tables) and I/O virtualization (device model).

Looking ahead, we envision a number of further hardware enhancements: hardware support for memory virtualization, platform virtualization, and device virtualization.

NESTED/EXTENDED PAGE TABLES

Even though shadow page tables can be implemented efficiently, they still require a number of transitions between the VMM and the guest. These transitions can cost hundreds or even thousands of cycles, and so it is desirable to keep their number to an absolute minimum. To this end, both Intel and AMD have recently announced hardware support for virtualizing the MMU. Intel's scheme is called extended page tables (EPT); AMD's is called nested page tables (NPT).

Both operate by adding an extra level of translation; in essence, a new page table (called the EPT or NPT, respectively) is introduced to translate between (guest) physical and machine addresses. There is one of these per virtual machine, since all address spaces within that virtual machine share the same physical to machine mapping. In addition, the hardware is now explicitly aware of the guest page tables.

Consequently, on a TLB miss, the hardware can walk the guest page tables directly, using the additional EPT or NPT to translate the physical addresses contained within page table entries. On completion of the walk, the TLB is updated with the resulting virtual to machine mapping and execution continues.

Note that even though this extra level of indirection does involve more lookups, it does not require any vmexits, hence improving overall efficiency. The hardware can also cache intermediate translation results to further improve performance.

VIRTUALIZING THE PLATFORM

Help is also coming for platform virtualization. First in line are extensions from AMD and Intel that allow enhanced protection from DMA-capable devices.

In today's COTS systems, devices are not subject to any translation or protection checks when they access memory. This means that a malicious or buggy device driver can program a device to read or write any piece of memory in the system, bypassing the VMM and any installed security policy.

Currently shipping AMD-V chips include support for device exclusion vectors (DEVs), which addresses this risk. A DEV is a bitmap with 1 bit for every 4K page of physical (host) memory. Any attempted memory access by a device first causes a lookup (based on the device and bus ids) to a protection domain; this is then used to select an appropriate DEV, and the target address is checked against the appropriate bit. If the bit is set, the access is disallowed. This can be used by a VMM to protect itself and any other key data (such as security policies) from rogue DMA accesses.

Similarly, Intel has announced VT-d, a forthcoming technology aimed at providing enhanced support for platform virtualization. VT-d is also a northbridge-based approach that interposes on device accesses, and it also maps devices to protection domains. However, VT-d takes a more generalized IOMMU approach: In particular, device-issued DMA addresses are no longer "physical" addresses but are instead translated through a hardware table. This allows protection as well as arbitrary remapping of the bus address space. VT-d support is expected to ship in 2007.

VIRTUALIZING DEVICES

Finally, there is work on making I/O devices themselves virtualization-aware, to allow direct yet safe sharing between multiple virtual machines. This is particularly of interest for high-throughput, low-latency devices such as gigabit network interface cards and next-generation graphics cards.

Some of this work involves proposed extensions to PCIe being developed by the PCI-SIG. These extensions include address translation and the introduction of virtual functions within PCI devices. There is also ongoing development of "smart" I/O devices that provide translation, protection, and multiplexing between multiple clients. Early results indicate that bare-metal performance can be maintained without sacrificing safety.

Conclusion

As virtualization continues to grow as an important technique for managing modern systems, the software and hardware used to provide it are maturing at a dramatic rate: The original version of Xen stemmed from a research project at the University of Cambridge and allowed a specific handful of modified operating systems to be efficiently virtualized on uncooperative x86 hardware. Nearly four years later, Xen is a mature and robust VMM supporting paravirtualized OSes that are increasingly maintained by the OS developers themselves; Xen has further been incorporated as a core feature in the major Linux distributions, being directly included with their release kernels.

Chip makers have also embraced virtualization and have released hardware features to assist VMMs. Xen now includes support for both Intel's VT and AMD's V processor extensions, allowing unmodified legacy OSes to be efficiently and safely virtualized. As a result, Xen can now host nonparavirtualized OSes, such as Microsoft Windows, on modern hardware. Hardware will continue to evolve in support of virtualization in the immediate future, providing more direct support for both memory and I/O devices. We look forward to incorporating these features into Xen as they become available, as they promise to provide even greater performance and stability for the virtualization of COTS systems.

REFERENCES

The following resources are useful for finding out more about hardware virtualization and Xen:

"Intel Virtualization Technology," *Intel Technology Journal*: http://www.intel.com/technology/itj/2006/v10i3/index.htm.

AMD64 Architecture Programmers Manual, Volume 2: System Programming: http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/24593.pdf.

Xen downloads: http://xensource.com/download.

Symposium on Operating System Principles (SOSP) 2003 paper on Xen: http://www.cl.cam.ac.uk/netos/papers/2003-xensosp.pdf.

Shadow2 presentation at Fall 2006 Xen Summit: http://www.xensource.com/files/summit_3/XenSummit_Shadow2.pdf.

Xen Source Code Repository: http://xenbits.xensource.com.

MARK BURGESS

configuration management: models and myths

PART 4: THERE'S NO I/O WITHOUT U

Mark Burgess is professor of network and system administration at Oslo University College, Norway. He is the author of Cfengine and many books and research papers on system administration.

Mark.Burgess@iu.hio.no

TRADE AND COMMUNICATION HAVE been in partnership since symbiosis emerged from evolution's curiosity shop, as a trick for smuggling contraband into the sum of the parts. This partnership is central to management of systems. Communication pervades, from the question a user asks at the help desk, to the data received from a router, to traffic flow statistics, to the implementation of configuration operations performed by software—there is an exchange of messages about state, about intention, and about change. Computer management is also increasingly about trade. All this communication is not for whim or curiosity; it has a direct value to us in terms of time, money, or service. However you look at it, the world of networks is the world of commerce.

When two parties wag their tongues or dance their dances, they are both altered by the exchange. When more parties are involved, there is a cumulative effect that spreads out along dentritic paths, binding the squawking flock together and forming a *network*. The trails blazed by those conversational patterns fashion the resulting behavior of all parties in the network. This wave of influence is the essence of management, whether it spreads like a diplomatic envoy or like a forest fire.

In this piece, I hope to persuade you to reexamine your beliefs about centralized, authoritative management in computing (or elsewhere). I want to argue that, in our modern world (surfing its way on the rising wave of free-market economics), we need to rethink the tradition of hierarchical, centralized governance in guiding the behavior of systems. It's a familiar song: Delegation and decentralization are not only desirable but inevitable if we are to cope with the rate at which we have to (re)configure and repair systems in a vibrant and adaptive network, built on the economics foundation of trade and services.

From a Cat's Cradle, Like a Bat Out of Hell

This is the network age, an age in which webs of communication are accelerating our technological and economic development. What is a network?

We overload this most important word with a plethora of meanings. Even in the limited domain of computer science, its meaning is not clear. Sometimes we mean the cable that joins the computer to the wall; sometimes we mean the infrastructure that enables communication between computers, including the routing and the switching; sometimes we implicitly mean the protocols that are spoken over these channels of copper and air; and sometimes we mean the abstract collection of computers themselves that are connected by the infrastructure (a social network of interaction formed between human and computer).

A network is "simply" a device or construct that joins many things or places together. The first technological networks were roads and sewers, built many thousands of years ago. Even before that, humans formed tribes linking humans together in structures of about thirty. But we should not be fooled into thinking that networks are human creations. In nature, networks are everywhere: Crystals and molecules are held together by networks of interatomic bonds. The structures of these networks determine the large-scale properties of the substances they form. In biology it is not genes that generate the complexity of the living world, but rather the networks of interconnections formed from the proteins that the genes encode. Our bodies are ripe with networks, as Arab scholars discovered and drew in exquisite detail during the first millennium: networks for blood transport, nerve signals, immunity, etc. Biology is a testament to the successful cooperation of multiple communicating parts.

Face-Centered Squareness

But as humans, we are frightened by complexity, even as we embrace it. The structures we configure purposely into networks (i.e., their topologies) are simple-minded. As they grow beyond our designs, we fret like worried parents over the consequences of this growth and try to protect systems with firewalls and other barriers. Recall Alvin Toffler's comments about industrialization from the last episode of the series.

It is no coincidence that published maps of the Internet look like snowflakes or leaves (see the images famously generated by Bill Cheswick at AT&T). It is not that the Internet appears biological; the point is that these biostructures are themselves networks. This is what networks look like when they emerge in the natural world purely as a result of mutually beneficial interaction. When engineers *design* networks they look quite different—like stars and trees. Humans only build in this "organic" way when things are "out of control" (i.e., when they are no longer designed by an engineer), but they "grow" following an economic principle of development rather than a regulated one. Why not? If these structures are so successful in nature, why don't we build like this? Perhaps we are small-minded.

Humans love to build centralized and hierarchical structures, whether industries, governments, or armies. Possibly there is a social-anthropic reason for this (perhaps an expert, on reading this, will tell me the answer): There is evidence to show that people have evolved to work in groups of around thirty at the most. Once this size is exceeded, they tend to break up into subgroups that cluster around a new leader. Another reason might be cultural, though the structure of families in which a family clusters around a dominant male, as attested to by the unfailingly nauseating and predictable references to "my father" or "daddy, daddy" (seldom "my mother" or "my family") at every tearful moment in American TV and film.

But there is a fascinating phenomenon going on here. Even if the size of our attention is limited, there is nothing obviously programmed into us that says how these groups must be organized, or is there? Well, roll up! Be amazed by our human propensity (perhaps desire) to subordinate ourselves before an authority figure. We behave like a bureaucracy of sheep in uniform. I always think of the scene from Monty Python's *Life of Brian* in which Brian tells the crowd, "Don't listen to me—you have to think for yourselves," to which the crowd cheers in unison, "Yes, yes! We must think for ourselves!"

Why command hierarchies? There is no evidence to suppose that such a structure is better than another. It is somewhat "natural" perhaps, like a dividing river or a branching tree, but it is far from robust. The tree structure has a certain clarity to it, but also great fragility. A tree is all about not reproducing the same alternative twice. A branching tree is a clean, economical, and logical separation of concerns, but the same properties also make it a structure of minimum redundancy and therefore quite fragile and blooming with bottleneck inefficiency.

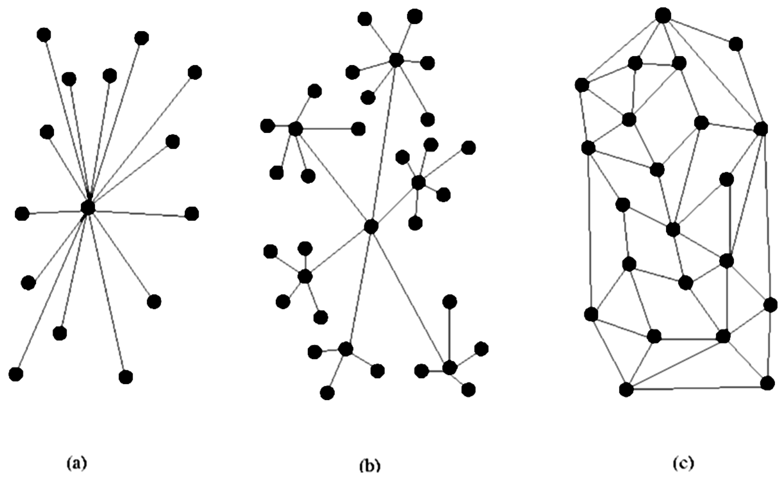


FIGURE 1. FROM CENTRALIZED STAR TOPOLOGY, TO HIERARCHICAL CENTRALIZATION, TO DECENTRALIZED MESH TOPOLOGY.

Create Like a God, Command Like a King, and Work Like a Slave

In terms of overhead, we can see why subordination (i.e., centralization) is appealing—if everyone follows a single master (Fig. 1a), then there are only $N - 1$ agreements instead of $N(N - 1)/2$ in a community of N agents. And yet public keys have shown us that we can form peer-to-peer collaborations with only N agreements and a little skill. What about consistency?

Every piece of knowledge must start from somewhere. That means there must be a source and a direction from which the information spreads. If there is only one source of information, then it must be consistent. Hence starlike topologies are perfect for local consistency. Q.E.D. If we move up a scale, then coordinating local communities according to a common policy can also be done from a single source; hence star hierarchies solve the problem (Fig. 1b). Tradition wins the day.

So, fine; centralization is sufficient, if we assume that the chief of the network can handle the burden—but is it necessary? The odds seem to be in favor of centralization. But is this good engineering? Let's look at this dipolar list:

- Centralization (single source) versus delegation
- Top down versus bottom up
- Hierarchical versus peer-to-peer
- Data normalization versus data-mining

What if we look at survivable networks, such as biological organisms? Biological "devices" evolved only to survive in a changing environment. How? Through redundant distributed networking—the opposite of centralization. Even though some few of our major organs are singular (e.g., the brain and the heart) there is still redundancy built in: We are amazed by stories of how people who have suffered brain injuries learn, for the most part, to reroute their brain functions as a result of the phenomenal inbuilt redundancy. The single points of failure (spinal cord, heart, etc.) are still our greatest weaknesses, and these things limit our growth.

But surely all this redundancy and variation is much too expensive to maintain! I quote Toffler once again: "As technology becomes more sophisticated, the cost of introducing variations declines." Recall that the fear Toffler spoke of in industrialization was precisely that mass-production would lead to an inflexible lack of choice in a market—that you could have any color as long as it was black. Well, he also argued that this was nonsense once you have technology, because that is when you really can afford to make things cheaply.

So do we see any such technologies that diversify the playing field for configuration management? Indeed, we have various levels of automation tools, from SNMP-based Tivoli and Openview to policy-template configuration tools such as Cfengine, LCFG, Pikt, and the Web-based interfaces provided by a variety of operating systems. Although SNMP is now widely regarded as a failure (even by the IETF) for everything with the possible exception of monitoring, the template-based tools, albeit imperfectly, enable great variability in mass-produced environments. The largest Cfengine installations, for instance, run into the tens of thousands on a single site, some with large variations from laptops to supercomputers. Clearly variation management is no longer a real issue for technology. Let's see how it comes about.

Speak to Me in Many Voices, Make Them All Sound Like One

The first universal theory of symbolic communication was pioneered by Claude Shannon in the 1940s. He turned the idea of communication into a science and implicitly solved the issue of maintenance at the same time. His theory of communication over a noisy channel is one of the classics of electrical engineering (or information science; take your pick). It makes that important point that you cannot escape from the problem of signal noise in a system. All systems contain noise (uncontrolled variations), in some manner or form. If a message is communicated in a noisy environment it becomes unclear, rough and grainy; the chance of it being understood and obeyed is much smaller (as the pony said to the ventriloquist, "I'd love to talk to you, but I'm afraid I'm a little horse").

Symbolic (digital) communication was the basic technology that enabled electronic networking and computation. After this, the idea of queuing and packet switching brought us from the fast train-track communication of the telephone network to the automobile diversity of the Internet protocol.

Networks now relay communications between peers by encapsulating any kind of message with a single lingua franca of IP (more or less).

The messages might now all sound like a single language, but they carry more diversity than ever before. By deregulating the centralized structure of the telecoms and by deregulating the single source content using the open (emergent) standard of IP, diversity has grown into a commerce of communication with a tolerable level of variation. Remarkably, a plethora of standards has converged into one, just as kids who are left to dress without school uniforms converge on jeans and T-shirts and a small number of basic themes. The lesson here is that when you deregulate something, you might actually end up with greater uniformity than before, because people lose interest in fighting for supremacy—they become content to live and prosper in their own niches.

In previous issues, I talked about the structure of patterns in a configuration. A network too is a configuration, which can be laid out as a formal language. The hierarchical tree structure we are used to in a context-free language (e.g., XML) has no a priori superiority to that of a more random peer-to-peer structure. Hierarchies possess *relative computational simplicity*, meaning that they are cheap to parse or build by linear computation, but they are only marginally more expressive than regular grammars that correspond to peer relationships.

Why would we think that a context-free, military hierarchy was the best solution to management? Perhaps because the alternatives are currently too hard for us to fully understand. Grammars that use parentheses make it easy to put things in boxes, and this is a comfortable way of marking out territory and assigning responsibility. But, in practice, we do not even use deep hierarchies in the organization of network patterns, usually implementing only two levels: master and slave hosts. That depth of pattern grammar can easily be built as a regular language, but it is "faux," being more about limitation than structure. It requires only that each slave in the network promise to follow the instructions of a master, which in linguistic terms is just a prefix. This is perhaps a clue that what we really value is perceived *cheapness* rather than subordination. We just think that hierarchy must be cheaper than a less structured pattern, although the theory of languages says otherwise.

Selling Your Soul at the Crossroads

In the past few years, users and researchers alike have come to realize the economic limitations of hierarchical regulation. A hierarchy implies a set of bottlenecks and barriers, of permissions and subordinations, but users have been given the power to compute and, by George, they do! *Service-oriented computing* has arrived to stay. It is direct, it is valuable to individuals, and it is subordinate to no one.

Technology is no longer the plaything of governments and governing boards for strategic purposes; it lies in the hands of ordinary folk who simply want to trade. This desire to trade, to exchange information and services in mutually beneficial ways, is what has driven the Internet into a state of biological complexity. It is a new technological symbiosis that enables our society to move to a new level of cooperation that can handle groups of bigger than thirty.

But what of the cost of this ad hoc, symbiotic organization? Price is clearly a subjective point of view in this story, and the cost of management is somewhat dependent on your particular skills (as Toffler says, "As technology becomes more sophisticated . . ."). Today these currencies for management are in flux, and centralization is being displaced by peer services, through the Web, and through file-sharing software. Could it be that *planned structure* could be supplanted by an organically (economically) grown *emergent structure*?

Well, this might all sound like a dream from some 1970s biological material, but don't reject this thought without seeing the wood in the trees: Just because a network was not designed does not mean that it is less functional than one that is. Just because behavior emerges from the cradle of economic self-interest (symbiosis) does not mean that it is less predictable than a military operation. All life and society emerged this way—and we do very nicely, thank you.

So, in case you thought I had forgotten about configuration management in this daydream about networking, let's tie the floating pieces in our Article-Area-Network together, seeing how we can have the best of both worlds: predictability and freedom along with personal safety and opportunistic self-interest.

Autonomous Meditation: The State of Standing Still

Shannon's model of the noisy channel applies equally to computers talking to themselves. Self-interest begins with the correct functioning of the individual. What easier way to maintain system state than to have it chant that state over and over again until it works harmoniously?

The passage of time brings many influences to bear on systems that we do not have any control over. Developers of computer systems have made a frequent error in viewing *configuration management* only as *change management* (as in a transaction system such as a database). It is a bit like believing that the weather is really an air conditioner with a nice neat knob to switch it on and off.

Self-maintenance is communication if you see a computer system as being in a constant state of meditation, at each moment repeating a mantra that we can call its state. We would like it to chant a message that agrees with our policy for its state. By repeating the message one then reinforces it. This metaphor describes the idea of autonomous configuration management.

From the previous articles, we think of the state of the computer as some string of configuration-operational characteristics that forms an alphabet. This can be coded in any imaginable way (e.g., suppose it is "ABHEKSYGHETFDH . . . ," where A means something like "chmod 644 /etc/passwd," etc.)

After a while, corruption of the state message owing to run-time interactions, meddlesome users, and network connections (i.e., noise) could lead to this state message being garbled, changing some of the symbols into others. Such a change must be corrected by reiterating the actual policy (e.g., "ABCD" -> "ABXD" -> "ABCD"). Just like the message over a noisy channel, we have to correct these errors.

The convergent operations we mentioned in the last article deal with this problem nicely. In operational language, a single operator is a unit of one kind of instigator of change, which we write:

$$O\,q = q'$$

to mean an operation applied to a state q leads to a transition to a new state q′. But rather than thinking about a transition from one state to a new state, think of this, rather, as error correction. A “convergent” operator is a message that tells any state to transform into a policy compliant state:

C (Any state) -> (Policy state)

The policy state is said to be a fixed point of the policy operator since once you get there you stay there. So, in terms of this language, all we need to do is to repeat the entire policy over and over again, like a never-ending mantra, with a separate operator for each independent kind of change:

$C_1C_2C_3 \ldots C_n$ (Any state) -> (Policy state)

Alva Couch called this the Maelstrom property in his LISA paper from 2001. The importance of this property is that a computer can simply chant its policy message to itself, applying strings of these operational messages, and this will ensure that it is always in the error-corrected, policy-compliant state. This is not science fiction. The approach was developed originally and used for Cfengine, in a limited form, and something like it is now being used in NETCONF and some other configuration management technologies designed to replace SNMP.

Though I Speak with the Tongues of Convergent Fixed-Point Operators

There is, of course, a danger to emphasizing the role of communication, language, or error correction too much. In the network management communities people often get stuck by confusing basic ideas with the technologies that communicate them. It would be no surprise to us that our leaders failed to govern the country if they held the view that management were the same as SNMP. Ideas generalize implementations; they should not be limited by them.

In the case of SNMP there was a conscious decision made by the IETF to avoid complexity in the protocol. This regulation, in turn, led to an explosion of complexity in the data structures (MIBs). You cannot suppress noise by trying to pretend it is not there. What fixed-point semantics offers us (at least in the cases where it has been possible to implement such a thing) is the guarantee that a message repeated will be easily implemented, rather than being a Mission Impossible.

What I am proposing here is that it can be sufficient to manage device internals individually, while allowing networks to trade freely. The result will not necessarily be “out of control” in a bad way; it will regulate itself *autonomically*.

Five Farthings, Say the Bells of St. Martin’s

For managing the configurations of computers, networks are not essential. It is clearly possible to administer changes and repairs to a completely isolated, stand-alone computer, either manually or with the aid of automation, “one-on-one.” However, the network opened the door to both collaboration and interconnection, therefore making it possible to manage systems remotely. This is only true, however, if such collaboration has an economic justification.

We can disconnect any computer from a network and take over the management of the device, and no one can stop us; hence the idea that computers are “controlled” from outside is only a convenient fiction. They are controlled insofar as they want to be. It is an act of *voluntary cooperation* to allow oneself to be managed by an external authority. Just as we bow to authorities, network management researchers find this idea almost impossible to accept.

Messages of different types have different values for us. Just being associated with a service provider in a BGP peering agreement can be worth a lot of money—kudos to the peer that earns respect and hence the promise of future profit. Association is a social currency that is worth something—not necessarily money. We have to learn to recognize the different forms of currency in play in economic cooperation. The face of commerce is changing.

Why would anyone talk about trade if they could control everything and nail everything down. You only have to compare the state of dictatorships with democracies to answer that one. The authoritarian regime might argue, you need me for:

- Offloading and convenience
- Specialist knowledge
- Separation of concerns
- Mutual advantage

But all of these things can, in fact, be obtained from your neighbor and today these are used as the primary reasons for outsourcing to companies that are considered to be *subordinate,* not superordinate, authorities.

Don't we need a law-maker or some other kind of authority to govern? In contract law it is observed that the threat of litigation in an authoritarian regime is essentially insignificant in determining whether the terms of a contract are upheld or broken. The principal factor that determines whether or not people break the law is the potential loss in economic value to the participants in the contract. Hmmm. . . think about that one.

Haggling Over the Price

It is possible to describe policy patterns and structures graphically using a theory of *promises*. We have been developing this theory in Oslo for the past two years. It allows us to study these matters of economically motivated cooperation. Promise theory tells us that we do not have to abandon personal autonomy to have distributed cooperation.

The economics of system administration change. At the beginning of the 1990s, when I developed Cfengine, there was a particular need for competitive garbage collection in systems—simply to keep them alive. Disks had finite size and the memory of the system was limited. This shaped the functions that were built into the configuration engine. Today, this recycling task has (for the moment) been deemphasized, as we have wider margins in modern systems. Tools that are produced today place more emphasis on installation, as if our fossil reserves of storage were infinite, or they ignore the presence of unnecessary processes, as if the heat produced by these unnecessary computations will not cost us dearly in the long run (either from increased electricity bills or the melting of the polar ice caps). The only thing that will apparently convince us to be more careful today is our confused paranoia over "security" (whatever that means to us).

Today "over-provisioning" (over-provisionizationing) allows us to swagger richly through the data center, paying little attention to the waste. This too is purely a matter of economics. We currently have no incentive to try to improve, but the time will come again when this bountiful Cretaceous era of information diversity meets its Tertiary boundary, and our systems will once again have to deal with loads that tax their resources to extinction. Limits will be reached, and we shall be operating once again on the edge where the balance of trade proves crucial to the survival of the species.

Please Sir, ISO Want to Buy Your BS-Enabled ITILity!

So what of the tail end of this tale? Services. The service paradigm has arrived to stay, in business, in commerce, and certainly in computing. This paradigm has all but wiped out the conventional notion of system administration in the eyes of network research and development and the telecom service providers. To them UNIX is an application service; Windows is something to be updated over a network.

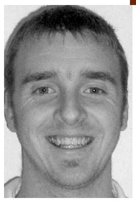
Businesses are gearing up for service management. Standards of good practice for service delivery management were developed by the British government in the late 1990s and have grown into a ubiquitous standard of practice in business today called the Information Technology Infrastructure Library (ITIL). The manuals and documents for this standard are still sold at great expense, and a summary was published as British Standard BS15000 and now also as ISO20000. These documents mention configuration management, although they say nothing about how (or indeed if) it can be implemented. They merely recommend that a software engineering type of versioning be used for the management for all kinds of configuration data. This is not the same story I have been telling in these articles, but it is a higher-level aspect of it.

It is probably possible to describe any enterprise as a number of interacting services, trading with one another for mutual advantage. Once the advantage disappears, the enterprise falls apart. ITIL and its bureaucratic rival the NGOSS/eTOM (enhanced Telecom Operations Map) help managers to see some aspects of good governance, so that service providers will be able to document their organizations and associated feel-good behavior, but they speak only of quality of the process surrounding the service (a sort of managerial version of the meditation discussed here). They say nothing about practical implementation or of the technical challenges.

In this series I have tried to show that the matters of system configuration and policy-guided behavior have a technical basis that is sometimes ignored. The future of low-level configuration management (in the sense of computer governance) lies with automation, whereas the high-level behavior of interaction lies in commerce. There is much research to be done in this area. We must understand the science and the economics of this, not merely the tradition and the doctrine—and that science is of communication. Communication, in turn, requires language: There must be a language to express the changes, desires, and policies of our self-regulating systems. And only when all of these pieces of the puzzle are in place can we say that we have fully understood configuration management.

LEIGH GRIFFIN AND JOHN RONAN

Xen installation and configuration



Leigh is a student assistant researcher at the TSSG, Waterford Institute of Technology. He has now returned to WIT to complete his studies in the B.Sc. in Applied Computing Degree.

lgriffin@tssg.org

John Ronan is a senior researcher at the TSSG, Waterford Institute of Technology, and also a radio ham (EI7IG). When not experimenting with new network technologies, he can be found bouncing AX25 packets through LEO satellites.

jronan@tssg.org

THIS GUIDE IS DESIGNED TO GET A Xen server up and running in no time. It is a simple copy-and-paste guide which should get you through a bare-bones install with minimal trouble and time. Xen is an exciting and still relatively new technology; more information can be obtained by visiting the Xen homepage at http://www.xensource.com and also by reading "The Inevitability of Xen," by Crowcroft et al. [1]. The only limitations you will find with Xen will derive from your hardware or your ingenuity.

As this guide is aimed at beginners, it is going to cover the installation of Xen from the binary packages. The binary packages are recommended for people who are new to Xen and are uncomfortable with the range of configuration options that the source install offers.

The latest stable release of Xen can be found at http://www.xensource.com/products/downloads/. The sources are also available from BitTorrent sites, among others; however, we recommend going with the official releases to ensure validity, security, and stability [2]. We obtained the 3.0.1 binary install file and thus will use this as a reference for the rest of the guide; simply replace 3.0.1 with your 3.0.x in the relevant positions in order to install the software successfully.

The Linux Filesystem Hierarchy Standard recommends placing the source file in the /usr/src folder, and that's where we will put it and unpack it [3].

Prerequisites

We ran the following commands to install the dependency packages and to remove some outdated and unnecessary packages (using Debian):

```
xen:# apt-get remove exim4 exim4-base lpr nfs-
common portmap pidentd pcmcia-cs pppoe
pppoeconf ppp pppconfig

xen:# apt-get install screen ssh debootstrap
python python2.3-twisted iproute bridge-utils
libcurl3-dev
```

Now that we have the necessary files installed, let's extract the software from its .tar file and run the install script:

```
xen:# /usr/src$ cd xen-3.0.1-install
xen:# /usr/src/xen-3.0.1-install$ ./install.sh
xen:# /usr/src/xen-3.0.1-install$ mv /lib/tls /lib/tls.disabled
```

The last command is necessary to avoid the emulation slowdown problems with the glibc libraries that are installed by default [2].

You should now have the Xen software installed on your computer. To start the Xen services at boot time, the following commands need to be run:

```
xen:# update-rc.d xend defaults 20 21
xen:# update-rc.d xendomains defaults 21 20
```

The final additions to be made are to add the Xen kernel to the bootloader program (Grub). Scroll through the file until you find the line that reads:

```
### BEGIN AUTOMAGIC KERNELS LIST
```

Just above that is the place where we must make our addition to the file. Enter the following text:

```
title Xen 3.0 / XenLinux 2.6.12
kernel /boot/xen.gz dom0_mem=64000
module /boot/vmlinuz-2.6.12-xen0 root=/dev/hda1 ro console=tty0
```

Note that it is important to make sure that your root is indeed /dev/hda1. If it is not, simply change the value after root= to match it. If you are unsure what your root name is, scroll further down the menu.lst file and you will see the default kernel and its root value.

Reboot the machine; at the boot prompt, Grub will now list Xen 3.0/ XenLinux 2.6.12 as the first kernel and boot it automatically. Everything should load normally and you will be given your standard login. If the machine does not boot, the following may fix the problem.

If the Xen machine executes a hard reboot as it is starting up, the problem rests with the amount of RAM in your machine. You will get no error message with this problem, and the last thing you will see is a line that says:

```
"Scrubbing free RAM. . . . . . . ."
```

Then the screen will go black and do a hard reboot. The solution is to remove the excess RAM (while still keeping the DIMMs balanced) and reboot the machine. Currently, the binary install can only cope with a maximum of 3583 MB of RAM. To use more RAM, a source install needs to be performed and PAE support must be built into the kernel; however, this topic lies outside the scope of this guide.

Creation of Domains

Now that we have the Xen software installed, it is time to get to the creation of the virtual machines. First, we are going to create a storage area for our virtual machines:

```
xen:# mkdir /virtual && cd /virtual
```

Here we are going to create two directories in which to store and configure the images:

```
xen:/virtual#  mkdir vm_base
xen:/virtual#  mkdir images
```

We will create a default image and swap image from which our virtual machines will be derived. Execute the following commands:

```
xen:/virtual#  dd if=/dev/zero of=/virtual/images/vm_base.img bs=1024k
count=xxxx
xen:/virtual#  dd if=/dev/zero of=/virtual/images/vm_base-swap.img
bs=1024k count=xxx
```

Note that the value that count= specifies is the size the image will be in megabytes. Simply change it to a value that will suit your needs; only your machine capacity is the limit. (See p. 65 for sample file sizes.)

Now we need to format the base image to be ext3 so that it can serve as our journaling filesystem. We have chosen ext3 as it is faster than ext2 and has stronger guarantees for data integrity [4].

```
xen:/virtual# mkfs.ext3 /virtual/images/vm_base.img
```

Answer yes to the question prompted regarding the warning about the block special device. Now we need to configure the swap file to be a swap area:

```
xen:/virtual# mkswap /virtual/images/vm_base-swap.img
```

Next it's time to install the Debian base system to our newly created image. First, though, we need to mount our image:

```
xen:/virtual# mount -o loop /virtual/images/vm_base.img /virtual/vm_base
```

Debootstrapping the Base Image

We run the debootstrap command to download all the prerequisite packages, using the following command:

```
xen:/virtual# debootstrap —arch i386 sarge /virtual/vm_base/
http://ftp2.de.debian.org/debian
```

Now change root and configure the images apt program to specify how we want to pull down our software and updates:

```
xen:/virtual# chroot /virtual/vm_base
xen:# apt-setup
```

During the standard apt setup, you will be asked some basic questions regarding your location and which mirror you wish to use to speed up the process. When this is done, edit the sources.list that comes with apt and change the word testing to stable wherever it appears in the file. Now update your software repository:

```
xen:# apt-get update
```

The next step in the installation process involves setting up the locales for your region:

```
xen:# apt-get install localeconf
```

Choose the locales to install depending on your country (e.g., en_IE ISOxxxx for Ireland or en_US ISOxxxx for the United States).

Next, configure the base system using base-config. A menu with various installation options will be presented to you. The important things to configure are:

1. Users and passwords. This is where you set the default user name, password, and root password. This is an important part as each image subsequently created from the base image will have these default passwords, which will need to be changed.
2. The time zone.

3. Which software to install. When the program prompts for additional software to be installed we choose “none,” as this is the base image, from which the other virtual machines will later be derived. Each derived machine can be customized when it is ready.

When you are satisfied with the system, simply hit return and you are finished configuring the base system.

There are some small configurations still to be completed. First, remove the hostname from the system. We remove the hostname because debootstrap copies this from the host machine to the newly created image so both will have the same hostname:

```
xen:# rm -f /etc/hostname
```

Now we need to create our networking interfaces by editing /etc/network/interfaces:

```
auto lo
iface lo inet loopback
        address 127.0.0.1
        netmask 255.0.0.0
```

Next we edit the fstab file; it must end up looking exactly like the following in order to represent the internal structure of the virtual image, its mountpoints, and its filesystem types:

```
/dev/hda1  /          ext3     defaults   1 2
/dev/hda2  none       swap     sw         0 0
/dev/pts   devpts     gid=5,mode=620      0 0
none       /dev/shm   tmpfs    defaults   0 0
```

These values will map to the configuration file values for the root and swap later on in the configuration of the virtual domains themselves.

Our last configuration option sees the creation of the hosts file:

```
127.0.0.1      localhost.localdomain   localhost
# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
```

Now we leave the chroot environment with exit. All that is left is for us to copy the kernel modules to our virtual machine and unmount the image.

```
xen:/virtual/vm_base# cp -dpR /lib/modules/2.6.12.6-xenU\
   /virtual/vm_base/lib/modules/
xen:/virtual/vm_base# mv /virtual/vm_base/lib/tls /virtual\
   /vm_base/lib/tls.disabled
xen:/virtual/vm_base# umount /virtual/vm_base
```

The base image is now complete.

Creation of Virtual Domains

Now that we have a base image to work off of, it is time to go and make some virtual machines. We will do this by copying the base image like so:

```
xen:/virtual/vm_base# cp -pf /virtual/images/vm_base.img\
   /virtual/images/vm01.img
```

```
xen:/virtual/vm_base# cp -pf /virtual/images/vm_base-swap.img\
    /virtual/images/vm01-swap.img
```

Now we create a configuration file for this new domain. Xen is located in /etc/xen, so that is the place where we will leave the configuration files, because the Xen software automatically scans this directory for the matching file. We use /etc/xen/myfirstdomain.sxp as the name for our first domain.

Here is a copy of the domain that we created:

```
name="myfirstdomain"
kernel="/boot/vmlinuz-2.6.12.6-xenU"
root="/dev/hda1"
memory=64
disk=['file:/virtual/images/vm01.img,hda1,w','file:/virtual/images\
    /vm01-swap.img,hda2,w']

# network
vif=[ '' ]
dhcp="off"
ip="10.0.0.50"
netmask="255.0.0.0"
gateway="10.0.0.254"
hostname="myfirstdomain.yourdomain.org"

extra="3"
```

The ip addresses should match ranges within your organization's network. It is simply a case of sorting the networking out more then anything else. We set DHCP to be off in our instance, but if your network requires DHCP to be on as a means of dealing with addresses it's simply a matter of changing "off" to "on."

The root value is the value that we set earlier in the /etc/fstab file. The mappings between the swap and root values that we set earlier in the /etc/fstab file are also evident in the specification of the disk value.

Note that the memory value is the amount of RAM in megabytes that you are going to give to this domain. In this case, our domain is going to receive 64 MB of RAM to work with.

Now to start the machine, you need to be logged in as root.

Once in as root, you have access to the Xen software. If you type in xm help, you will get a listing of the available commands and how to use them. To create a domain we will execute the following command:

```
xen:# xm create -c myfirstdomain.sxp
```

There is no need to specify the exact path to the myfirstdomain file, as the Xen software automatically looks in /etc/xen for a file matching the configuration file you are using. If you placed the configuration file elsewhere, simply insert the complete path to the file.

The -c flag is used to ask for a console for the domain you have just launched. If all goes ok, you should see the machine booting up and eventually you will get to the login prompt. If you get an error saying the domain failed to balloon, it is an error associated with allocating too little memory to the virtual machine. You have not allocated enough RAM to allow domU to boot successfully. You will have to use the xm destroy myfirstdomain to stop the domain, then edit the configuration file to allocate more memory to your domain, and use the create command to launch the domain.

Log in with the default username and password that you specified in the configuration of the base system. It is a good idea now to change the default password. This is a major security issue, as each domain is created from the same base system and thus has the same username and passwords!

With your domain you should be able to ping the master Xen server, other xen domains floated, and other hosts on the same network as yours. It should also be possible to ssh into the domain.

When you are finished with your domain and wish to exit it, you can do a shutdown as normal, which will send you back to the original Xen domain from where you came, or if you wish to leave it running and wish to return to Xen, simply hold down CTRL +]. This will take you back to Xen. If you run `xm list` you should see your domains that are successfully floated, including information such as how much memory they are allocated and their domain name. To get a console to one of them simply run `xm console myfirstdomain` where myfirstdomain is the name of the domain we specified within the configuration file and is the name that appears in the list of domains we see when `xm list` is run.

If you wish to create more domains it is simply a matter of copying the base image:

```
xen:# cp -pf /virtual/images/vm_base.img /virtual/images/vm0X.img
xen:# cp -pf /virtual/images/vm_base-swap.img /virtual/images\
    /vm0X-swap.img
```

The `vm0X` just needs to be changed to a new unique number or name.

A corresponding config file needs to be created in /etc/xen, which references the newly created image file in its `disk=` parameter.

If you wish to have your domains started automatically at startup, a link must be created in the auto folder that Xen scans as the system boots. This can be achieved by doing this:

```
xen:# ln -s /etc/xen/myfirstdomain.sxp /etc/xen/auto
```

Restart the machine and see if the domains come up successfully.

Extra Configuration

The final configuration that must be done in order to create more than three domains may need to be performed now.

Each virtual image and its swap area run on a loop each. The default number of loops is 7. If you attempt to float a fourth or fifth domain you will get this error:

```
"Error: Device 769 (vbd) could not be connected. Backend device not found."
```

This means that we can only create at most three domains with this setup (as each requires two loops to run). So we need to do some editing to vital files. Again, ensure a backup has been made in case things go wrong.

We need to edit the modules configuration file /etc/modules.conf and add these options anywhere in the file:

```
options loop max_loop=64
rmmod loop
modprobe loop
```

Once done, if you are running devfs, the new loops should have been automatically created.

If you still only see seven values for loop, you need to edit /dev/MAKEDEV and recompile it to make the changes take place. This is a very big file; you need to scroll down until you see the following:

```
loop)
        for part in 0 1 2 3 4 5 6 7
        do
            makedev loop$part b 7 $part $disk
        done
        ;;
```

This needs to be changed to:

```
    loop)
        for part in `seq 0 63`
        do
            makedev loop$part b 7 $part $disk
        done
        ;;
```

Then recompile by running `makedev loop`.

Verify in /dev that there are now 64 loops created, which is enough for 32 machines to be created. If you need more, change the 63 to a number you desire.

When this is done, restart the machine and everything should be working fine.

RESOURCES AND LINKS

[1] J. Crowcroft et al., "The Inevitability of Xen," *;login:*, 30, no. 4 (2005): 10–13. Available at http://www.usenix.org/publications/login/2005-08/pdfs/crowcroft.pdf.

[2] *Xen User's Manual*. Available at http://www.cl.cam.ac.uk/Research/SRG/netos/xen/readmes/user/.

[3] Filesystem Hierarchy Standard Group, R. Russell et al., eds., *File System Hierarchy Standard*, 2004. Available at http://www.pathname.com/fhs/pub/fhs-2.3.pdf.

[4] M.K. Johnson, "Red Hat's New Journaling File System: ext3," 2001. Available at http://www.redhat.com/support/wpapers/redhat/ext3/#advantages.

[5] F. Timme, "The Perfect Xen 3.0 Setup for Debian," 2006. Available at http://www.howtoforge.com/perfect_setup_xen3_debian.

Acknowledgment: This work is partially supported by the IST ENABLE project under the European Commission's 6th framework program.

ROBERT MARMORSTEIN AND
PHIL KEARNS

debugging a firewall policy with policy mapping

Robert Marmorstein will graduate from the College of William and Mary this summer with a Ph.D. in Computer Science. When he is not actively researching ways to manage and analyze firewalls, he spends his time avoiding grues in the Great Underground Empire.

rmmarm@cs.wm.edu

Phil Kearns is an Associate Professor of Computer Science at the College of William and Mary. His research interests lie in the general area of computer systems.

kearns@cs.wm.edu

IF YOU MANAGE A LARGE NETWORK, chances are good that lurking somewhere in your firewall policy is an error that could compromise the security of your network. Firewalls are subject to many different kinds of configuration errors. Although many of these glitches are relatively harmless, some can seriously compromise your policy's correctness and reliability. Inserting a rule into the policy twice may cause a negligible performance hit but usually does not affect the correctness or security of the policy. However, a typo in a critical rule may give an untrusted host access to your important servers. Until recently, debugging a firewall policy with more than a few rules was a challenging and tedious task. In the past few years, however, new tools for analyzing the policy have made finding firewall gremlins much easier. With these tools, you can find and repair many common firewall problems quickly and easily.

ITVal is a framework we designed for testing iptables-based Linux firewalls. It was originally intended to provide an open-source alternative to existing commercial testing tools such as the Fang firewall analysis engine [7, 9] and the Internet Security Scanner [3]. Like those tools, it relied either on the user's ability to create logical queries describing the desired behavior of the firewall or on a pregenerated set of generic tests. We quickly discovered, however, that devising meaningful and effective queries required nearly the same effort as inspecting the firewall by hand. To address this problem, we came up with a new method of debugging the firewall rule set.

Our new technique is easily usable by any system administrator. It doesn't need extensive preparation of queries, cases, or tests. In fact, the only input requirement is a textual representation of the policy (easily generated by the `iptables -L -v -n` command). As output, it produces a map of the network that can be used for detecting anomalies in the policy.

The policy map divides hosts into groups based on their interaction with the firewall. If the firewall treats two hosts the same, they will belong to the same group. If they are treated differently, they belong to different groups. A correct policy will usually divide the hosts of a network into logical, easily identifiable groups based on their function within the network. One group might be the “all workstations” group. If the network distinguishes between different kinds of workstations, there may, instead, be separate groups for “Solaris workstations” and “Linux workstations.” Other groups might be the “Web servers” group and the “wireless hosts” group.

It is significant that the map generated by the component of ITVal described in this article does not rely upon user input. The map is generated by processing only the iptables rules set that defines the firewall policy. In a very real sense, it is merely a different representation of the set of iptables rules, but we contend that it makes the difficult task of finding some errors in firewall configuration much easier.

Since you probably have a good intuitive idea of the various types of hosts on your network, it is relatively easy to check that the firewall policy map represents a correct policy. The policy map probably won’t reveal every error in your policy, but it will make many of the most significant errors instantly visible. A quick glance at the policy map will reveal significant bugs in the firewall policy that query-based tools could not easily uncover.

Debugging a Firewall

To create a policy map, you need to create a few important input files. The first step is to dump a copy of your firewall policy to disk. If you have root access, this can be done simply by typing `iptables -L -n -v > myRules` at the command line. If your firewall uses packet mangling, you also need to dump the NAT table to disk with `iptables -t NAT -L -n -v > myNatRules`.

You also need to provide a query file, myQuery, containing the single statement `QUERY CLASSES;`. This tells ITVal to generate the policy map for the input firewall.

You can now use the command `ITVal -F myRules -N myNatRules -q myQuery` to generate the policy map. ITVal will create a list of the various host groups in the firewall policy and display them on stdout.

The firewall policy in Table 1 protects subnet 128.30.40.0/24 from the outside world. The network has a few key servers: a mail server (128.30.40.10) and two identical Web servers (128.30.40.11–12). Hosts on the network also talk to two special machines outside the network: a name server (128.30.1.128) and a network time server (64.15.175.5). The policy contains several errors, which can easily be detected by inspecting the policy map.

	Target	Source	Destination	Port
1	ACCEPT	Anywhere	128.30.40.0/24	DNS
2	ACCEPT	128.30.40.0/24	Anywhere	DNS
3	ACCEPT	128.30.40.0/24	64.15.175.5	NTP
4	ACCEPT	128.30.40.0/24	128.30.40.0/24	
5	DROP	!128.30.40.0/24	128.30.40.13	
6	ACCEPT	128.30.40.0/24	218.30.40.10	SMTP
7	ACCEPT	128.30.40.0/24	128.30.40.10	IMAP
8	ACCEPT	128.30.40.0/24	128.30.40.12	SSH
9	ACCEPT	Anywhere	128.30.40.11	HTTP
10	ACCEPT	Anywhere	128.30.40.12	HTTP

TABLE 1: A BUGGY FIREWALL POLICY

This policy is intended to enforce a few important rules. We want to allow mail traffic only between trusted clients and the mail server. (This is unrealistic, since the mail server will also need to send and receive messages from the outside world, but it makes the example much simpler.) We also want both Web servers to allow HTTP connections from any host and SSH connections from clients on the trusted subnet. Furthermore, any of our systems should be able to access domain name service and the network time service, but only from appropriate servers. Using ITVal, we can generate a map of this firewall policy. The policy map for the firewall in Table 1 looks like this:

```
QUERY CLASSES; There are 6 host classes:
Class1: #Untrusted Hosts and DNS Server
          <Everything not explicitly listed in the other classes>
Class2: #NTP Server
          64.15.175.5
Class3: #Trusted Network, including the Mail Server
          128.30.40.[0–10]
          128.30.40.[14–255]
Class4: #Web Servers
          128.30.40.[11–12]
Class5: #Anomalous Host 128.30.40.13
          128.30.40.13
Class6: #Anomalous Host 218.30.40.10
          218.30.40.10
```

The policy map consists of a list of various classes of hosts, which correspond to the different types of systems on the network. Each class consists of a set of hosts with some common properties. Two hosts belong to the same class if and only if any packet sent or received by one of them is treated the same as if it had come from (or to) any of the others. If two hosts are in different classes, there is some essential difference between them in the firewall policy that distinguishes them from each other.

In the listing given here, each element of a class is given as an address or a range of addresses. For instance, in class 4, the element 128.30.40.[11–12] represents a set containing the addresses 128.30.40.11 and 128.30.40.12. A brief inspection of the various classes enables you to easily identify their members. For instance, class 2 corresponds to the external NTP server. Class 3 represents the trusted hosts of the network. Class 4 represents the Web servers of the network. Classes 5 and 6 are anomalous. Class 5 consists of a decommissioned print server that no longer belongs on the network. Class 6 is an artificial class caused by an error in the firewall. All other hosts belong to class 1. For easier reference, we have added comments to the output that identify each class.

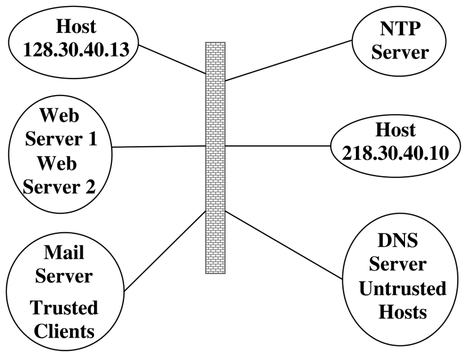

Host 128.30.40.13
NTP Server
Web Server 1 Web Server 2
Host 218.30.40.10
Mail Server Trusted Clients
DNS Server Untrusted Hosts

FIGURE 1: AN ITVAL NETWORK MAP

Figure 1 is a graphical depiction of the policy map. By inspecting the map for anomalies, you can detect several important policy errors.

Catching Typos

In a policy of more than a few dozen rules, there is a good chance that one of the rules contains a typo. In the sample policy, a typo on line 6 prevents SMTP traffic from reaching the mail server. The policy map immediately highlights this error. Since 218.30.40.10 doesn't correspond to any of the important servers, the existence of class 5 is a clear indication of an error in the policy. A search through the rule set for the address 218.30.40.10 uncovers the typo and allows us to patch the problem by transposing the first two digits of the address. Repairing the problem gives us a new policy map. The new map looks like this:

```
QUERY CLASSES; There are 5 host classes:
Class1: #Untrusted Hosts and DNS Server
          <Everything not explicitly listed in the other classes>
Class2: #NTP Server
          64.15.175.5
Class3: #Trusted Network, including the Mail Server
          128.30.40.[0–10]
          128.30.40.[14–255]
Class4: #Web Servers
          128.30.40.[11–12]
Class5: #Anomalous Host 128.30.40.13
          128.30.40.13
```

Detecting Outdated Rules

The new policy contains yet another anomalous class. Host 128.30.40.13 in class 5 of the new policy map does not correspond to any of our important servers. Why has it been distinguished as a special class?

It is very easy to forget to change the firewall policy after altering the network infrastructure. The sample policy contains rules for an experimental print server that has been taken offline and no longer needs special protection from external hosts. In the meantime, the server's IP address has been reused as the address of a new workstation. As a result, the firewall makes a distinction between that address and the other systems. Rule 5 of the original policy, originally designed to protect the print server from untrusted hosts, is now blocking network traffic to the new workstation. Removing the outdated rule from the policy resolves this issue and gives us the following policy map:

```
QUERY CLASSES; There are 4 host classes:
Class1: #Untrusted Hosts and the DNS Server
          <Everything not explicitly listed in other classes>
Class2: #NTP Server
          64.15.175.5
Class3: #Trusted Network, including the Mail Server
          128.30.40.[0–10]
          128.30.40.[13–255]
Class4: #Web Servers
          128.30.40.[11–12]
```

Detecting Overly Broad Rules

Sometimes the easiest way to temporarily allow hosts to access an external service is to open that service up to any external host. This is a bad practice which often leaves a network open to intrusions from untrusted hosts, but it is very convenient when bringing a new system online for the first time. In the long run, however, it is usually much better to use a more specific rule that allows only trusted hosts to provide that service.

The DNS server does not appear in any of the classes explicitly listed in the policy map. This is because the members of class 1, the "everything else" class, have been hidden to save space. Counter to our expectations, the DNS server has been lumped into this class with all of the untrusted hosts. To permit DNS traffic only from the appropriate server, the firewall policy ought to distinguish that server from other hosts outside the network. The fact that the DNS server is grouped with untrusted hosts indicates either that DNS traffic is allowed from any external host or that all DNS traffic is blocked by the firewall.

Examining the firewall policy reveals an error in rules 1 and 2. Those rules should grant DNS access only from the DNS server (128.30.1.128), and not from other external hosts.

This error can be easily repaired by inserting the correct IP address into each rule. DNS traffic will then be permitted only to the appropriate server. Running ITVal on the new policy gives us a new policy map:

```
QUERY CLASSES; There are 5 host classes:
Class1: #Untrusted Hosts
        <Everything not explicitly listed in the other classes>
Class2: #NTP Server
        64.15.175.5
Class3: #DNS Server
        128.30.1.128
Class4: #Trusted Network, including the Mail Server
        128.30.40.[0–10]
        128.30.40.[13–255]
Class5: #Web Servers
        128.30.40.[11–12]
```

Detecting Shadowed Rules

Another common error is to create a rule in the policy that shadows other rules. Since iptables considers rules in sequential order, a rule that accepts or drops packets from an entire subnet will take precedence over a narrower rule that occurs later in the chain. These shadowed rules can be easily identified in the policy map.

In the policy map, the mail server does not have its own class. Instead it is grouped with the trusted workstations on the network. This is a good sign that one or more of the rules protecting the mail server has been shadowed. By looking through the rule set for rules corresponding to the mail server and/or the trusted hosts, you can easily see that rule 4 shadows rules 6 and 7. It turns out that the rule is unnecessary and can be deleted. Removing the rule creates a new rule set with the following policy map:

```
QUERY CLASSES; There are 7 host classes:
Class1: #Untrusted Hosts
        <Everything not explicitly listed in the other classes>
Class2: #NTP Server
        64.15.175.5
Class3: #DNS Server
        128.30.1.128
Class4: #Trusted Clients
        128.30.40.[0–9]
```

```
            128.30.40.[13–255]
Class5: #Mail Server
            128.30.40.10
Class6: #Primary Web Server
            128.30.40.11
Class7: #Secondary Web Server
            128.30.40.12
```

The removal of rule 4 introduced two new classes into the policy map. As expected, there is a new class containing the mail server. The class containing the Web servers has also changed. The erroneous rule hid some discrepancies in how the two Web servers are treated by the firewall that need to be addressed. Now that the rule has been removed, these errors have become visible.

Detecting Missing Rules

In the new policy, the primary and secondary Web servers belong to separate classes. Why is this? Rules 8, 9, and 10 allow HTTP access to both servers, but they allow SSH access only to the secondary Web server. The policy is missing a rule that would permit SSH access to the primary server.

Inserting a new rule to fix this problem gives us the policy shown in Table 2, which has the following policy map:

```
QUERY CLASSES; There are 6 host classes:
Class1: #Untrusted Hosts
            <Everything not explicitly listed in the other classes>
Class2: #NTP Server
            64.15.175.5
Class3: #DNS Server
            128.30.1.128
Class4: #Trusted Clients
            128.30.40.[0–9]
            128.30.40.[13–255]
Class5: #Mail Server
            128.30.40.10
Class6: #Web Servers
            128.30.40.[11–12]
```

	Target	Source	Destination	Port
1	ACCEPT	128.30.1.128	128.30.40.0/24	DNS
2	ACCEPT	128.30.40.0/24	128.30.1.128	DNS
3	ACCEPT	128.30.40.0/24	64.15.175.5	NTP
4	ACCEPT	128.30.40.0/24	128.30.40.10	SMTP
5	ACCEPT	128.30.40.0/24	128.30.40.10	IMAP
6	ACCEPT	128.30.40.0/24	128.30.40.11	SSH
7	ACCEPT	128.30.40.0/24	128.30.40.12	SSH
8	ACCEPT	Anywhere	128.30.40.11	HTTP
9	ACCEPT	Anywhere	128.30.40.12	HTTP

TABLE 2: A CORRECT FIREWALL POLICY

A graphical depiction of a map for the new policy is shown in Figure 2. Both Web servers now belong to class 6. The map now conforms to our expectations. There are separate classes for the mail server, the DNS server, and the NTP server. The Web servers are grouped into a single class. The set of trusted clients forms a group and all other systems are grouped as untrusted hosts.

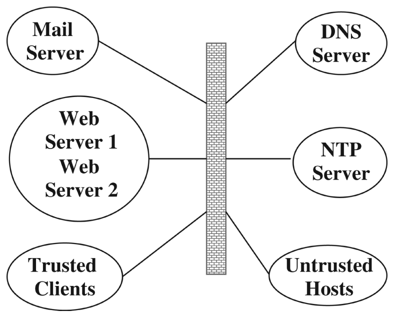


FIGURE 2: MAP OF THE NEW POLICY

Penetration Testing

In addition to making certain kinds of policy errors immediately obvious, the policy map can be used with other penetration testing techniques for more comprehensive and accurate verification. Since the firewall treats all hosts in a class the same, testing one address from each class gives complete coverage of the entire firewall policy. For instance, when using nmap [2] or hping2 [1] to search for unfiltered ports, it can be useful to use source address spoofing to test that the firewall rejects packets from a variety of sources. Instead of using a randomly selected source address, you can take one address from each class to be sure that all interesting behaviors of the firewall have been tested.

What If I Don't Use iptables?

ITVal currently only parses iptables firewalls, but the general technique of generating a policy map can be used with any type of firewall. A more technical description of policy mapping can be found in the proceedings of LISA '06 [6], which outlines the algorithm used to compute the various classes of the policy map. A quick-and-dirty approximation to the policy map can be created by listing all the addresses (and address ranges) explicitly mentioned in the firewall policy. While such a listing is far less precise and useful than the policy map, it can reveal some of the behaviors uncovered by the policy map and can provide a good sample of addresses to use during penetration testing. You might also try converting your rule set into an iptables policy using some of the scripts Bill Stearns has made available on his Web site [8] (mileage will vary!).

Conclusion

Maintaining a well-tested and tightly configured firewall is an important part of overall network security. Thanks to tools such as ITVal, it no longer needs to be an arduous task. By periodically testing whether your firewall policy conforms to your general expectations about the organization of your network, you can quickly and easily identify and repair significant firewall errors.

In addition to generating a policy map, ITVal provides many other useful tools for detecting problems in your firewall, including various ways to test for spoofing protection and to check whether viruses and Trojans can access backdoors through your firewall. More information about ITVal is available in the proceedings of Freenix '05 [5] and LISA '05 [4]. The tool itself can be downloaded from http://itval.sourceforge.net.

REFERENCES

[1] P. Bogaerts, HPING Tutorial, August 2003: http://www.radarhack.com/dir/papers/hping2_v1.5.pdf.

[2] Fyodor, "The Art of Port Scanning," *Phrack* 7, no. 51 (September 1997).

[3] Internet Security Systems, *Internet Scanner User Guide Version 7.0 SP 2* (2005): http://documents.iss.net/literature/InternetScanner/IS_UG_7.0_SP2.pdf.

[4] R. Marmorstein and P. Kearns, "An Open Source Solution for Testing NAT'd and Nested iptables Firewalls," in *19th Large Installation System Administration Conference (LISA '05)* (December 2005), pages 103–12.

[5] R. Marmorstein and P. Kearns, "A Tool for Automated iptables Firewall Analysis," in *FREENIX Track: 2005 USENIX Annual Technical Conference* (April 2005), pages 71–82.

[6] R. Marmorstein and P. Kearns, "Firewall Analysis with Policy-based Host Classification," in *20th Large Installation System Administration Conference (LISA '06)* (December 2006).

[7] A. Mayer, A. Wool, and E. Ziskin, "Fang: A Firewall Analysis Engine," in *Proceedings of the IEEE Symposium on Security and Privacy* (May 2000).

[8] B. Stearns, http://www.stearns.org/.

[9] A. Wool, "Architecting the Lumeta Firewall Analyzer," in *Proceedings of the 10th USENIX Security Symposium* (August 2001).

DAVID BLANK-EDELMAN

practical Perl tools: spawning

David N. Blank-Edelman is the Director of Technology at the Northeastern University College of Computer and Information Science and the author of the book *Perl for System Administration* (O'Reilly, 2000). He has spent the past 20 years as a system/network administrator in large multi-platform environments, including Brandeis University, Cambridge Technology Group, and the MIT Media Laboratory. He was the program chair of the LISA '05 conference and was one of the LISA '06 Invited Talks co-chairs.

dnb@ccs.neu.edu

AS THE PARENT OF A NEW BABY, I'VE really come to appreciate the idea of multitasking. Remind me to tell you about the times I've successfully fed my child from a bottle while simultaneously going to the bathroom and petting a cat that asserted it could not live another moment without some attention. (Ok, maybe I won't tell you about them, though wouldn't that make a swell column?) Having the number of uninterrupted time slots shrink considerably since our child was born has made me a fan of things in the Perl world that help get tasks done more quickly or efficiently. Multitasking is one of those techniques, and that's just what we're going to talk about in today's column.

Fork()ing

Let's start out with one of the basic building blocks of most multitasking Perl code: fork()ing. The fork() function comes from the UNIX system call of the same name, though it works on most other operating systems as well. This now includes Windows operating systems, thanks to work in 1999 by ActiveState that Microsoft itself sponsored.

Here's a quick review of fork() for those of you who didn't grow up teething on (or perhaps even loading) the V7 magtapes. When you call fork() from a Perl program, a copy of the running process is made. This copy receives all of the context of the program that called fork(), including the run-time environment, any variables in play at the time, and open file handles. The new process is called the "child process" with the original one dubbed the "parent process." Since the program that called fork() continues to run in both the child and the parent processes it is up to your code to ensure that the child process knows to do childlike things (processing something, etc.) and that the parent acts parental (e.g., creating new children or waiting for existing children to finish).

How does your code know whether it is running as the child or the parent process? Almost everything about the two running processes is the same. The key difference is that the parent receives back the pid of the child spawned by fork() as a return code from that fork() call. The child receives a 0 from the same call. This leads to the following idiom you see in most code that uses fork():

```
my $childpid = fork();
die "Fork failed:$!\n" if !defined $childpid; # fork returns undef on failure

# if $childpid == 0, we are in the child process and need to do stuff
if ($childpid == 0) { ... do stuff }

# we're the parent and so we need to reap the fork()d process when done
else {
  # waitpid waits for a particular pid vs. wait() which will
  # reap any child process that has completed
  waitpid($childpid,0);
}
```

A parent process must retrieve the exit status of all of its children once they have completed, a process known as "reaping," or the child processes continue to live on as "zombies" until the parent process dies. Reaping is performed by wait() or waitpid() as seen in the code above. The waitpid()/wait() functions will (you guessed it!) wait until the child in question has finished before returning.

Warning: Because we have a student fork-bomb one of our machines at least once or twice a year (mostly unintentionally) I feel compelled to mention that code that either looks like, acts like, or boils down to this:

```
while (1) { fork(); } # bad bad bad bad
```

is, as the comment says, bad bad bad bad. A program that fork()s out of control like this is a fork-bomb. If process limits allow (e.g., Solaris defaults; see the maxuprc kernel parameter) this program will consume all available spots in your process table, causing the system to melt down and be a real pain to clean up to boot. If you are going to write code that repeatedly fork()s, always build in some kind of big red button to shut the process down, for example:

```
while (1) { fork() unless (-f /tmp/stop); } # create /tmp/stop to end
```

or impose limits to keep the problem self-quashing:

```
while (1) { die "Over the fork limit" if $limit++>100; fork(); }
```

With this little snippet we've covered all of the basics you have to know to begin programming using fork(). Toward the end of the column we'll see an easier way to use this functionality.

Basic Threading

The second simple method for multitasking involves threading. Before we go much further I want to insert a number of caveats about threading under Perl:

1. Thread support is comparatively new to Perl. Actually, it is more accurate to say that *this* kind of thread support is new to Perl. There was an attempt in the past (around Perl 5.005 or so) to add threads but that model never proved to be stable and was eventually moved to "deprecated" status and will leave the code as fast as the Perl 5 developers can get it out (with version 5.10, as I understand it). Support for the current model looks pretty good and is under active maintenance, so you probably don't need to worry. Still, I want you to know that the

ground around this issue is a little softer than usual, so you'll need to step carefully. Things such as debugger support for threads are still a little shaky (better in 5.8.5+ but still incomplete), but threads are definitely usable today.

2. Perl threads are likely to be different from any of the other threading models you have seen before. They are not your granpappy's lightweight processes or precisely any other thread implementation you've encountered before. We'll talk about what they are, but I thought it best to prime you for what they are *not* first.
3. To use the current threading model, your Perl interpreter has to be built with a special option at compile time. This is not enabled by default in a source build, and different OS vendors are more or less adventurous. For example, until Solaris 10 Sun did not have it turned on in the Perl interpreter that ships with Solaris; Apple does turn it on for OS X. To tell if you have it enabled, look for the line that contains useithreads= in the output of perl -V. If it says define you are all set. If it says undef, then you will have to rebuild the interpreter. As a related aside, there is a module called forks that provides the same API as the threads module we'll be using but does it using fork() calls. If you'd like to play with the threads stuff on an OS that doesn't provide a threaded Perl but does support fork() natively, this module may do the trick for you.

If you are still with me after passing the "Danger, This Means You!" paragraphs above, it means you are interested in how threads work in Perl. Let's look at that now. Modern versions of Perl, when enabled at compile time, provide something called "interpreter threads" or "ithreads" for short. I'll use "thread" and "ithread" to mean the same thing in this column.

A standard Perl program gets run by a single Perl interpreter thread that handles the interpretation and execution of a program from start to finish. Perl threads allow you to start up additional Perl interpreters that independently execute parts of your code. Each interpreter thread gets its own copy of the state of the Perl program at that point. If this sounds to you a little bit like a fork() situation we've already covered, that shows you are on the ball.

One difference from fork() is the ability to actually share data between threads. With fork(), the children get a copy of all of the variables in play but changes made by a child aren't seen in the parent's copy unless they work out some sort of external synchronization mechanism. With ithreads, the situation is a little different. By default, ithreads don't actually share any data; the copies they have of the program's state are completely independent. However, if you'd like two ithreads to share access to a variable, you do have the ability to mark that variable as shared using a separate pragma. A shared variable can be changed by one thread and all other threads will see this change as well. This provides considerable power to the programmer but also potential peril, since reading and writing to a shared resource need to be carefully coordinated. Let's take a look at some sample code and then we'll see how such issues are addressed in Perl.

```
use threads;

sub threaded_sub {
                print "running in thread id: " . threads->self->tid() . "\n";
        }

my $thread = threads->create(\&threaded_sub);
```

```
# this shows a scalar return result, but we could also pass a list
my $result = $thread->join;
```

This code shows nearly the simplest use of ithreads I can demonstrate. To use ithreads, we define a subroutine whose code will be executed in a thread. As a thread is created, it is assigned an id (with the main or initial thread when the program first runs being called thread id 0). The thread being created here isn't particularly exciting, since it only prints its id and then exits, but you get the idea.

The create() line takes that code and spins off a new thread to run it. At that point the subroutine threaded_sub is happily executing in a separate thread. The result of the create() command is a thread object we can use for thread control operations. The next line executes one of these operations called join(), a method similar in intent to wait()/waitpid() from our fork examples. join() waits for the desired thread to complete and retrieves the return value from that thread. If we did not include the join() statement, Perl would complain when the program exits leaving behind an unjoined thread:

```
Perl exited with active threads:
0 running and unjoined
1 finished and unjoined
0 running and detached
```

If we didn't care at all about the results of that thread, we could replace the method join() with one called detach().

Now let's get a bit more sophisticated. I mentioned before that different threads do not share the same data unless explicitly instructed to do so. That sharing is performed by adding the threads::shared pragma and marking certain variables with this status:

```
use threads;
use threads::shared;

my $data : shared = 1; # share($data) can also be used
# now all threads will read and write to the same variable
```

Congratulations: With this step we've now stepped onto the rickety bridge over the deadly gorge of Parallel Programming Peril (cue the dramatic alliteration music)! With race conditions and other nasty beasts waiting for us at the bottom of the gorge we have to step very carefully. As soon as you begin to deal with a shared resource, you need to make sure that the right piece of code updates that resource at the right time. The other pieces of code running at the same time must also follow the right protocol to avoid having that update get inadvertently overwritten. To deal with these circumstances Perl offers a set of functions such as lock().

As you can probably guess, lock() attempts to place a lock on a variable and blocks until it succeeds. It's useful when several threads want to modify a shared value:

```
{ lock($data); $data++; }
```

Why use curly braces in that example? Perl's threading model has no function called "unlock()"; locks are released when they pass out of scope. By using curly braces around these statements we've set up a temporary scope that just includes the update to the variable $data after the lock is in place.

There are other idioms for thread programming that avoid doing this sort of dance. I don't want to go too deeply into parallel programming techniques, but this one bears a quick look because it comes up so frequently.

Here's a modified version of the example used in the official Perl threads tutorial (perldoc perlthrtut):

```
use threads;
use Thread::Queue;

my $DataQueue = Thread::Queue->new;
$thr = threads->create(
    sub {
        while ( defined( $DataElement = $DataQueue->dequeue ) ) {
            print "Thread "
                . threads->self->tid()
                . " popped $DataElement off the queue\n";
        }
        print "Thread " . threads->self->tid() . " ready to exit\n";
    }
);

print "Thread " . threads->self->tid() . " queues 12\n";
$DataQueue->enqueue(12);
print "Thread " . threads->self->tid() . " queues A, B and C\n";
$DataQueue->enqueue( "A", "B", "C" );
print "Thread " . threads->self->tid() . " queues undef\n";
$DataQueue->enqueue(undef);
$thr->join;
```

Let's go over the code in some detail, because it might not be immediately clear what is going on. First, we create a queue for the two threads we are going to use to share. One thread will place new values at the end of the queue (enqueue()); the other will grab values from the top of the queue (dequeue()) to work on. By using this scheme the threads don't have to worry about bumping into each other.

After creating a queue, the second thread (i.e., the one that is not the main thread) gets defined and launched via the create command. The subroutine defined in this command just attempts to pop a value off the queue and print it. It will do this for as long as it can retrieve defined elements from the queue. It may not be readily apparent from the code here, but when faced with an empty queue, dequeue() will sit patiently (block/hang), waiting for new items to be added. Think of the second thread as always waiting for new elements to appear in the queue so it can retrieve and print them.

The rest of the program takes place in the main thread while the second thread is running. It pushes several values onto the queue: the first a scalar, the second a list, and the third an undefined value. It ends with an attempt to join the second thread. The net result is that the code prints something like this:

```
Thread 0 queues 12
Thread 0 queues A, B and C
Thread 0 queues undef
Thread 1 popped 12 off the queue
Thread 1 popped A off the queue
Thread 1 popped B off the queue
Thread 1 popped C off the queue
Thread 1 ready to exit
```

This looks like all of the action first takes place in the main thread (0) followed by the second thread's work, but that's just the order the output is received. If you step through the main thread with a debugger, you'll find that the main thread will queue a value, the second thread prints it, the main thread queues another value, the second thread prints it, and so on. Thread::Queue has its limitations (some of which are solved by other modules), but in general it provides a fine way to pass things around among threads that all have to coordinate tasks.

Now that you know about queues, we've finished a good surface look at threads in Perl. Be sure to see the documentation for the threads and threads::shared pragmas and the Perl thread tutorial (perlthrtut) for more information on other available functionality.

Convenience Modules

We could spend a lot more time talking about threads, but I want to make sure I mention one more topic before we come to an end. As you probably guessed, Perl has its share of modules that make working with fork() and threading a little easier. Let me show you one that I'm particularly fond of using: Parallel::ForkManager.

I like Parallel::ForkManager because it makes adding parallel processing to a script easy. For example, I have a script I use when I want to rsync the individual directories of a filesystem to another destination. I use this in cases where it is necessary to copy over each subdirectory separately for some reason. Here are some choice pieces from the code:

```
opendir( DIR, $startdir ) or die "unable to open $startdir:$!\n";
while ( $_ = readdir(DIR) ) {
    next if $_ eq ".";
    next if $_ eq "..";
    push( @dirs, $_ );
}

closedir(DIR);

foreach my $dir ( sort @dirs ) {
    ( do the rsync );
}
```

One day I realized that directory copies like this don't have to take place serially. Several directories can be copied simultaneously with no ill effects. Adding this parallel-processing functionality was just a matter of changing the code that said:

```
foreach my $dir ( sort @dirs ) {
    ( do the rsync );
}
```

to:

```
# run up to 5 copy jobs in parallel
my $pm = new Parallel::ForkManager(5);

foreach my $dir (sort @dirs){
    # ->start returns 0 for child, so only parent process can start new
    # children, once we get past this line, we know we are a child process
    $pm->start and next;

    ( do the rsync );

    $pm->finish; # terminate child process
}

$pm->wait_all_children; # hang out until all processes have completed
```

The added code creates a new Parallel::ForkManager object that can be used to fork a limited number of child processes (->start), have them exit at the right time (->finish), and then clean up after all children with one command (->wait_all_children). The module does all of the scut work behind the scenes necessary to keep only a limited number of fork()ed processes going. I find the ease of adding parallel processing to my scripts (just four lines of code) has made me much more likely to create scripts that handle several tasks simultaneously. There are other convenience modules that are worth looking at (e.g., Parallel::Forker does all that Parallel::ForkManager can do, but it allows you to specify that certain child processes must wait to run after others have completed). Be sure to do searches for "fork," "parallel," and "thread" at search.cpan.org to see what is available. If you find yourself needing a really sophisticated multitasking framework, you'd be well served to check out the POE framework at poe.perl.org.

Oops, I have to go back to getting many, many things done at the same time. Take care, and I'll see you next time.

ROBERT HASKINS

ISPadmin: DHCP services

Robert Haskins has been a UNIX system administrator since graduating from the University of Maine with a B.A. in computer science. Robert is employed by Shentel, a fast-growing network services provider based in Edinburg, Virginia. He is lead author of *Slamming Spam: A Guide for System Administrators* (Addison-Wesley, 2005).

rhaskins@usenix.org

IN THIS EDITION OF ISPADMIN, I TAKE a look at the area of DHCP [1] services. DHCP stands for "Dynamic Host Configuration Protocol" and is used by many Ethernet-based networks for handing out IP addresses to client devices (PCs) in an easy, scalable manner. It is based upon the older BOOTP protocol though it does have its own IETF standards (RFC2131 [2] and RFC2132 are primary; see [3] for a more complete list). DHCP is closely related to other network protocols such as TFTP and DNS, which is why many commercial software vendors package their DHCP offerings with these (and other) related protocol servers.

Background

Of course, traditional enterprise networks are a big user of DHCP services, allowing easy dynamic and persistently available IP addresses for business users. In provider networks, DHCP is used only in certain access methods, such as cable modems and traditional end subscriber Ethernet access (e.g., switches and some wireless access points) for use in deployments such as apartment buildings. Wireless access points (such as those from Colubris) often support multiple IP address assignment methods, including RADIUS and DHCP [4]. Other access technologies (such as DSL and dialup) are typically assigned IP addresses via RADIUS or similar authentication protocol, and therefore they are not usually associated with DHCP services.

DHCP Feature Requirements

At a basic level, the service provider requires very similar features to an enterprise needing address assignment services. Some of the more important features required in just about every DHCP deployment include:

- Assignment of "persistent" IP addresses to specific MAC addresses
- A graphical user interface (GUI) to help support personnel troubleshoot problems
- Flexibility in managing address pools

You might be wondering why I included a GUI as a must-have. Support personnel are a different breed of folks and need a GUI in order to efficiently handle customer trouble requests. Also, a small organization will want a Microsoft-type GUI instead of a command line for managing DHCP on a small network.

For a service provider (or large enterprise), additional DHCP features required often include:

- The ability to easily extend the server's functionality
- Command-line access
- Large-scale deployments (millions of clients)
- An interface to provider provisioning systems
- An easy-to-use Application Programming Interface

Many of these features are found in the DHCP servers targeted at carrier-class service providers covered in this article.

Deploying a DHCP Server

Planning is key to bringing up any new network infrastructure and DHCP is no exception. DHCP startups can be phased in, by pointing only a limited number of LAN segments (say, a single class C of 254 possible clients) at the new DHCP server in question. Going back to the original DHCP server is easy, as it involves just changing the DHCP helper address on the LAN segment you moved in the first place.

Designing advanced options such as RFC 3046 (Option 82), called the Relay Agent Information Option [5], can be tricky, because they are often specific to the hardware vendor in question. Option 82 is a DHCP feature where the client gives the server additional information about itself so that the server can select the proper IP address for the client and assist the client in self-configuration. It pays to test these features completely in the lab prior to rollout, so that vendor promises can be turned into reality without nasty surprises late in the rollout!

Solutions

There are many DHCP solutions available on the market, and most have decent support for service providers. Many of the service-provider-directed solutions are part of "suites" that handle other functionality, such as overall IP address management, DNS, TFTP, and similar functions.

ISC DHCPD

The reference DHCP implementation is ISC's full-featured DHCPD [6]. Here is a listing of DHCPD features from the ISC Web site:

- DHCP Failover Protocol support
- OMAPI, an API for accessing and modifying the DHCP server and client state
- Conditional behavior
- The ability to store arbitrary information on leases
- Address pools with access control
- Client classing
- Address allocation restriction by class
- Relay agent information option support
- Dynamic DNS updates
- Many bug fixes, performance enhancements, and minor new DHCP protocol features

Many smaller service providers utilize ISC's DHCPD on their networks. However, to use it in a large production network, additional development work would probably be required. For example, a GUI would likely be needed for support personnel, and a provisioning interface/system would have to be developed.

MICROSOFT DHCP

Microsoft's DHCP server [7] is probably the leader in terms of the sheer number of DHCP servers in use, but there are likely few large deployments. (I don't know of many good sources of DHCP server market data, though Birds-eye.net [8] offers one somewhat dated source.) Anecdotal evidence suggests that Microsoft is only used in small-service-provider deployments. The Microsoft DHCP server has come a long way since NT 3.5 and now includes features such as clustering/failover [9] and better management of address scopes and ranges.

LUCENT VITALQIP

The VitalQIP [10] server is a product from Lucent that contains a DHCP server along with other functions in one application, including IP address management and a DNS server. VitalQIP's real strength is its scalability, as the software is designed from the ground up for very large networks. To this end, it is a "manager of manager" of sorts, able to manage different underlying DHCP platform types (MS Windows, IBM AIX, and Lucent's own) all from one interface. Of course, if you are a service provider who doesn't have several million clients, then this solution is probably too large (and likely too expensive) for you.

NOMINUM

Nominum [11] is a big player in the carrier-class service provider software market for DNS and DHCP servers. In fact, according to its Web site, its "products were developed by Nominum engineers based on lessons they learned from writing BIND (version 9) and ISC-DHCP (version 3)." Its DHCP product offering, called the Dynamic Configuration Server (DCS), is part of its "Triple Play" product line, including authoritative as well as caching DNS servers. Similar to the other carrier-grade solutions, the Nominum solution is designed around scalability, reliability, speed, and ease of use and integration into service provider provisioning systems. Like Lucent's offering, unless you support millions of subscribers, this solution probably isn't for you.

INCOGNITO

Incognito Software [12] is a relative unknown in the DHCP market, though the company has been around since 1992. It has a nice product suite, with solutions to many of the issues service providers face:

- DHCP
- DNS
- ENUM [13]
- SIP
- IP address management
- TFTP

Incognito's DHCP product is called IP Commander and is a native application running on a number of platforms, including Microsoft Windows NT/2000/XP/2003, Sun Solaris 8, and Red Hat 9/AS3/ES3. It includes support for a number of protocols, including DHCP, DNS, and TFTP. This product has all of the features one would expect in a carrier-grade product, including scalability, reliability, performance, and easy integration with back-end provisioning systems. The only feature missing is a Web interface for support personnel, though it wouldn't be very hard to build one on top of the provided command-line interface. All in all, this is a good product for any provider to look into further.

CISCO CNR

Cisco's carrier-class DHCP offering is called Cisco Network Registrar (CNR) [14]. This is a very scalable, flexible, and robust DHCP server, offering TFTP, DNS, and IP address management capabilities. One of the biggest benefits of CNR is how it scales from one server to one cluster of servers to multiple clusters of servers over very large networks. Another big benefit to CNR is how extensible it is: One can write embedded scripts to handle all the customizations one needs when running a DHCP network of any size. CNR has both a CLI and a Web-based GUI, though the GUI could use some extensive human factors improvements. (In other words, I found it hard to use when evaluating the GUI.) CNR is another good option for a small but growing provider who needs DHCP and associated services.

Conclusion

For the service provider, DHCP is a core part of the cable modem and traditional Ethernet-based networks (apartment complexes and such). Service providers have many of the same DHCP-related requirements a traditional enterprise network operator has, plus a few others. These additional requirements include ease of extending the capabilities server, provisioning, CLI access, and very large-scale deployments.

The reference DHCP implementation is the ISC DHCPD server, which is good for many smaller installations. Microsoft incorporates a DHCP server as part of many versions of Windows, which is a good option for small providers. A small to mid-size provider would do well to look at Incognito's IP Commander and Cisco Registrar, both excellent choices for these markets. A large carrier might consider any of the DHCPs listed previously, in addition to Lucent's VitalQIP or Nominum.

REFERENCES

[1] Wikipedia DHCP page: http://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol.

[2] RFC2131 text: http://www.ietf.org/rfc/rfc2131.txt.

[3] Network Sorcery DHCP page: http://www.networksorcery.com/enp/protocol/dhcp.htm.

[4] Colubrus MGW-3500: http://www.colubris.com /global-wireless-network-management/wlan-accessories.asp.

[5] Option 82 Relay Information Option: http://www.networksorcery.com/enp/protocol/bootp/option082.htm.

[6] ISC DHCPD: http://www.isc.org/sw/dhcp/.

[7] Microsoft TechNet home for DHCP: http://www.microsoft.com/technet/itsolutions/network/dhcp/default.mspx.

[8] Birds-eye.net Enterprise DHCP Market Research: http://www.birds-eye.net/analysis_archive/enterprise_dhcp_market_research.shtml.

[9] Cluster support for Microsoft DHCP servers: http://technet2.microsoft.com/WindowsServer/en/library/5df3d4e9-e846-413a-bd9a-99645ac580991033.mspx?mfr=true.

[10] Lucent VitalQIP home: http://www.alcatel-lucent.com/wps/portal/products/detail?LMSG_CABINET=Solution_Product_Catalog&LMSG_CONTENT_FILE=Products/Product_Detail_000143.xml.

[11] Nominum Dynamic Configuration Server home: http://www.nominum.com/products.php?id=4.

[12] Incognito IP Commander: http://www.incognito.com/products/IPCommander/overview.jsp.

[13] ENUM (mapping telephone numbers via a DNS-like protocol): http://en.wikipedia.org/wiki/Telephone_Number_Mapping.

[14] Cisco Network Registrar: http://www.cisco.com/en/US/products/sw/netmgtsw/ps1982/index.html.

HEISON CHAK

virtualizing Asterisk

Heison Chak is a system and network administrator at SOMA Networks. He focuses on network management and performance analysis of data and voice networks. Heison has been an active member of the Asterisk community since 2003.

heison@chak.ca

AS CPU AND MEMORY MODULES BEcome faster and more affordable, setting up what we used to call a "big box" to host multiple virtual machines (VMs) for server consolidation and still achieve close-to-native performance is becoming a reality without costing a fortune.

Although commercial virtualization products such as VMWare, Parallels, and Solaris Zones focus on delivering ease of deployment and administration of VMs, they all have their limitations, ranging from minimum CPU requirements to supported OSes. Xen, however, is an open source virtualization software that is designed with goals similar to those of some commercial products. It also has certain limitations, the biggest one being its inability to run a nonmodified OS (with the kernel being the issue), which has made running the Windows OS impossible in the past. This shortcoming has been addressed with the added support of Intel VT-enabled CPUs or the AMD Pacifica equivalent. (See the Hand article about hardware virtualization, this issue, p. 21.)

Virtualization

Xen was chosen mainly because of the available support for running Windows as a guest OS, with hardware virtualization. The object was to find a virtualization platform that enabled Windows, Linux, and Solaris operating systems. My initial reasons for virtualization were:

- To reduce energy consumption
- The ability to bring up a test environment within minutes
- To stage and test Asterisk 1.4

Until recently, SCSI drives have always been chosen for their performance and reliability. With Serial ATA (SATA) drives becoming more prevalent and larger in capacity, the gap is closing. Replacing an array of RAID 5 SCSI drives with big mirrored SATA drives can increase capacity and reduce electricity consumption. The reduced spindles of SATA drives will generate less heat, reducing cooling requirements.

Building the Virtualization Host

To get started, I installed Xen from source onto a Linux box running Debian Sarge. I then patched a Linux 2.6 kernel with Xen modifications and compiled as a dom0 kernel (for the host OS). After booting the Linux box with the newly built dom0 kernel, you can build a domU kernel (for the guest OS) and a virtual machine template based on Debian. This template can be used to quickly deploy new Linux VMs.

Whereas the kernel for domU remains under /boot of dom0, the root (/) of the VMs resides in image files (.img). Each image file has an ext3 file system and contains a Debian install:

```
vm:/# ls -l /boot/*xen[0U]
-rw-r—r— 1 root root 2347714 2006-11-27 22:57 /boot/vmlinuz-2.6.16.29-
   xen0
-rw-r—r— 1 root root 1263925 2006-11-27 14:40 /boot/vmlinuz-2.6.16.29-
   xenU

vm:/# ls -l /vserver/images/
-rw-r—r— 1 root root 4194304000 2006-10-30 21:54 debian01.img
-rw-r—r— 1 root root 2097152000 2006-10-30 21:56 debian01-swap.img
-rw-r—r— 1 root root 4194304000 2006-10-30 23:35 debian02.img
-rw-r—r— 1 root root 2097152000 2006-10-30 23:38 debian02-swap.img
-rw-r—r— 1 root root 4194304000 2006-10-28 08:43 debian_base.img
-rw-r—r— 1 root root 2097152000 2006-10-28 09:50 debian_base-
   swap.img
```

Deploying a new VM involves duplicating debian_base.img and debian_base-swap.img image files.

See http://www.howtoforge.com/debian_sarge_xen_3.0.3 for step-by-step instructions on how to install Xen on a Debian Sarge system.

Asterisk and Xen

Two VMs have been set up to test Asterisk 1.2 and 1.4, with 128 MB of memory and one virtual CPU assigned to each VM:

```
vm:/# xm list
Name            ID    Mem(MiB)   VCPUs     State     Time(s)
Domain-0        0     374        1 r——     200.4
asterisk-12     1     128        1 -b——    20.6
asterisk-14     2     128        1 -b——    13.1
```

Both virtual machines have access to the Internet to check out the latest source of Asterisk via subversion. Compiling libpri and asterisk was relatively easy; as one might guess, the tricky part is with the Zaptel drivers. These are loadable kernel modules that may need access to hardware, specialized PCI interfaces that communicate with the PSTN. Aside from CPU and memory, virtualizing hardware is more involved and sometimes difficult or impossible.

With Xen 3.0.3, PCI devices can be assigned solely to a VM (domU) but it is not used (or hidden) in the host OS (dom0). ("Tiger" refers to the Zaptel card.)

```
vm:/# lspci | grep -i tiger
00:09.0 Network controller: Tiger Jet Network Inc. Tiger3XX Modem/ISDN
   interface
00:0a.0 Communication controller: Tiger Jet Network Inc. Tiger3XX
   Modem/ISDN interface
00:0c.0 Communication controller: Tiger Jet Network Inc. Tiger3XX
   Modem/ISDN interface

vm:/# cat /etc/xen/debian01-config.sxp
name="asterisk-12"
```

```
kernel="/boot/vmlinuz-2.6-xenU"
root="/dev/hda1"
memory=128
disk=['file:/vserver/images/debian01.img,hda1,w',
      'file:/vserver/images/debian01-swap.img,hda2,w']

# Assign FXO interface to asterisk-12 domU
pci = [ '00:0a.0' ]

# network
vif=[ '' ]
dhcp="off"
ip="10.155.200.1"
netmask="255.255.0.0"
gateway="10.155.1.1"
hostname="asterisk-12.ykz.zealnetworks.com"

extra="3"
```

By assigning a PCI to a domU, one can build Zaptel drivers and use the FXO interface to make calls to the PSTN or to use it as a timing device for Meetme conference.

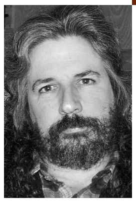
Limitation with ztdummy

Ideally, a virtual machine should not remain hardware-independent. In Asterisk, conferencing requires a reliable clock to mix audio. Such a timing source is usually featured in the Digium hardware PCI cards, or one can choose to use the ztdummy driver. In Linux 2.6, ztdummy defaults to using the kernel's real-time clock, which is not available to a Xen domU. Falling back to using the USB clock (i.e., OHCI_UCD) didn't work either, as USB support in domU seemed lacking at the time this article was written.

Until ztdummy works properly in domU, one may need to assign the PCI interface to the virtual machine if conferencing is required. Hardware dependency for a virtual machine may not be elegant, but it works.

ROBERT G. FERRELL

/dev/random



Robert is a semiretired hacker with literary and musical pretensions who lives on a small ranch in the Texas Hill Country with his wife, five high-maintenance cats, and a studio full of drums and guitars.

rgferrell@gmail.com

HAVING BEEN IMMERSED IN COMPUTER-related jargon for most of my life, I tend to scoff at the notion that said vocabulary is anything other than perfectly logical and apropos. On those rare occasions when the planetary gears align and free radicals roam the wild prairie unfettered, however, objectivity sneaks in for a brief visit and I find myself reflecting on the mystical nature of the terminology adopted by our austere and august profession.

Since operating systems are on the plate for this issue, let's start there. The name UNIX itself is a famously wry cut at what must have seemed at the time the semimythical Multics, whose operational status the Creators grew weary of anticipating in futility. This saucy philosophical underpinning ensured that lexicographic mayhem was more or less inevitable. Take as evidence of this postulate the cascade of variants that followed as a result of AT&T having trademarked "UNIX": ULTRIX, XENIX, AIX, Dynix, AUX, POSIX, HP-UX, DG/UX, Linux, and, yes, USENIX. The fact that computer geeks tend to have active, if somewhat bizarre, senses of humor just adds an extra layer of rich, velvety goodness to the mix.

In this issue my peculiar phraseology focus is on the "log" bog. Somewhere along the tortuous path of multiuser computing evolution, it became de rigueur to refer to the action of connecting to a mainframe or server as "logging." That in itself is curious: Why "log"? I have to date three roughly equally unlikely hypotheses as to the origin of this term:

1. Watching the tape drives spin reminded them of the lumberjacks' sport of birling.
2. Users waiting for an available terminal looked like harvested lumber getting jammed together on a river.
3. The perniciously high error rate of their programs spawned a deep-seated urge to have at the CPU with an uprooted tree.

[Oh, and if you just happen to know "log"'s true origin, please keep it to yourself. However flawed, I prefer my etymological fantasies unsullied by prosaic factoids.]

We're not quite finished with log, however, for we have yet to consider the Great and Uniquely Topical to This Publication Preposition Proposition: to log "in" or "on"? I'm reminded thereby of George Carlin's reply to being instructed to get on an airplane (paraphrased for your protection): "No, thanks. I'm getting *in* the plane."

So, which is it? Does one "log in" or "log on"? The answer seems to be platform-specific, at least according to Bob DuCharme's useful and unexpectedly entertaining *The Operating Systems Handbook*. One logs *on* to IBM mainframes, but logs *in* to UNIX or VMS. One can generally circumnavigate this prickly semantic conundrum altogether by *hacking in*, of course.

Disclaimer: I'm only kidding (as far as you know).

Moving along, I've always been fascinated in a "look-at-that-funky-insect-crawling-around-in-the-sink-I-hope-it-doesn't-sting" sort of way with kernel programming. I don't really understand it at any deep level, but I've never been proficient at resisting the urge to muck about under the hood. Most of the time my more creative attempts are stillborn, but once in a while the compiler is so traumatized by my complete lack of comprehension of C syntax that it gets confused and kicks out a viable kernel. Actually, "viable" is not nearly so accurate a term as "insane." Well, we all have our hobbies.

Another of my hobbies is writing new front ends in Perl for common system functions. I've rewired `rm`, for example, so that it merely moves the target file into an obscure directory. I've also added undocumented options that accomplish pointless tasks, such as list processes in numerical order by the hex representation of the letters in their names. There is no conceivable use for this listing scheme, admittedly, but paucity of utility is pretty much my hallmark where coding is concerned. My crowning achievement in this arena was `lsuf`, which listed all files in a given directory *not* in use by a process.

I once messed with `kill` so that it generated a little ASCII art animation of whatever was entered as arguments being run through with a spear by a tiny Viking warrior. In the same vein I've animated `mount`, `umount`, and `fsck`, the nature of which embellishments I leave to the reader's fertile imagination.

Perhaps my most ambitious foray into this twisted realm was when I decided to create an entire "artificial horizon" for one of my Internet-facing Sun boxes. I mangled every command line utility that could conceivably be of use to anyone gaining illicit access to the system for the sole purpose of driving said intruder crazy. I substituted characters when renaming, created spurious columns for `ps`, generated nonsense for `netstat`, and redirected `ls` to random commands such as `file /dev/*`. I made a text file of all the magical personages I could think of from literature and the cinema to be displayed when someone typed `which`; `more` triggered the response, "Don't be greedy!" but not much else. The command `perl` (regardless of the arguments) brought up a man page expounding the natural history of the oyster. Try to `grep` something and up popped a Freud-in-the-Box:

```
# grep nameserver /etc/resolv.conf

You seem to have unresolved nameserver issues. Tell me about your
childhood.
```

In short, I rendered the system thoroughly useless to anyone not privy to the fact that I'm a nut case. Come to think of it, awareness of my mental status probably wouldn't be of any real benefit. Some things just can't be dealt with in a rational manner.

On a distantly related closing note, did I ever mention that my friends of the "dabbling in the supernatural on a boring Saturday night" persuasion have declared me a psychic damper? Apparently my very presence in the general area is equivalent to the Blue Screen of Death for Ouija board sessions and seances. It's not that I'm some huge skeptic of ghosts, UFOs, and their ilk, surprisingly. In truth I take no stance, since in the absence of evidence an absolute belief in the existence or nonexistence of something is equally bogus. If a bona fide ghost floated up and shook my hand, I'd just shrug and cross ectoplasm off the list of things I've never seen.

Overall, I find natural phenomena more inconvenient than frightening. I got struck in a half-hearted way by lightning a few years ago, for example, but it just made me spill my beer. I was hoping super-powers might develop as a result, but all I really have to show for the experience, other than a slightly more robust liver, is a tiny hole-shaped scar on the heel of my right hand, where several palmar folds intersect, and a tendency to attract foam packing peanuts from a considerable distance. All hail Static Boy!

book reviews

ELIZABETH ZWICKY
WITH
PAUL ARMSTRONG,
SAM STOVER, AND
RIK FARROW

MASTERING REGULAR EXPRESSIONS: UNDERSTAND YOUR DATA AND BE MORE PRODUCTIVE

Jeffrey E. F. Friedl

O'Reilly, 2006. 484 pages.
ISBN 0-596-52812-4

By a strange twist of fate, a significant percentage of all the Java code I have ever written went directly into production on mission-critical systems. For all I know, it's still deployed: all five lines of it. What has this got to do with regular expressions? I ended up writing this delicate, important code that we were going to have to release without proper testing in a language I didn't know because I wasn't scared of regular expressions. Not, you will note, because I was a wizard with them—no, I was merely competent, confident enough about regular expressions, and nervous enough about breaking things that people felt it would be in good hands. And, indeed, it worked better after I fixed it than it did before.

Had I read this book first, I probably would have done a better, faster, more efficient job and felt calmer doing it. I started reading this book with the warm confident glow you get when you're reviewing on good, firm ground—here's a topic I know something about. No, I didn't. I mean, obviously I did have some basic competence, but I still spent a lot of time going "Wow. Now that's cool." And even more time going "Huh? Now if I read that again, I'm going to really understand something complicated and interesting." The author at one point says, "It's not necessarily easy to wrap one's brain around this, but once it 'clicks,' it's a valuable tool." This is certainly true of the *recursive* regular expression he's discussing at that point, but it's more generally true of the book as a whole.

At some point, you are going to need to do regular expressions. You could do it by trial and error and reading manuals, but believe me, I've seen a lot of people do that, and it's not pretty. Instead, you should buy this book. If you don't already know something about regular expressions, it's going to be slow going. That's pretty much the nature of the territory; take heart from the idea that it's even slower when you do it by trial and error. If you've managed to solve a number of interesting problems with regular expressions, but it was harder than you'd hoped, there are some odd failures, and you have a feeling that there was luck involved, you should find this not an easy read, but an enlightening one.

Does it matter what language you're struggling with? Probably not. The book covers Perl, Java, .NET, and PHP in chapters of their own, and it has tables for a number of other languages, plus information on how to figure out what's going on inside your regular expression engine if it's not covered. Besides, the basic concepts are language-independent.

WHY SOFTWARE SUCKS . . . AND WHAT YOU CAN DO ABOUT IT

David S. Platt

Addison-Wesley, 2006. 242 pages.
ISBN 0-321-46675-6

This book is not really meant for us. Instead, it is meant for nontechnical people who are computer users and want to know why technical people are torturing them. It will be enlightening, sometimes in a "self-knowledge hurts" kind of a way and sometimes more in a "Gee, those other techies are just like us, I guess" kind of a way.

I enjoyed it thoroughly—I like a good rant, when I agree with it—and I did learn a few things. I feel warmer and fuzzier about software phoning home when it crashes, for instance. Mostly, I thought it was good fun, and actually surprisingly positive, given the title.

CATASTROPHE DISENTANGLEMENT: GETTING SOFTWARE PROJECTS BACK ON TRACK

E. M. Bennatan

Addison-Wesley, 2006. 270 pages.
ISBN 0-321-33662-3

This is a practical guide to figuring out if you are dealing with a certifiable catastrophe and then getting your project back on track, if at all possible. You can tell it's a practical guide, because sometimes it tells you that, in fact, it is not possible to recover, and you should give up. It is a systematic, process-oriented book, better suited to large companies with official "Projects" than to startups, but even so, it has a refreshing willingness to admit the existence of politics and personal feelings.

This book is best suited for somebody with some management experience. It would be helpful if you think the project is going down the tubes, but everybody else is either more experienced or on too many happy drugs, you're not sure which. It would also be helpful in a situation where everybody agreed there was a disaster at hand, but nobody knew how to get out of it again.

HACK THE STACK: USING SNORT AND ETHEREAL TO MASTER THE 8 LAYERS OF AN INSECURE NETWORK

Michael Gregg, Stephen Watkins (Technical Editor), George Mays, Chris Ries, Ron Bandes, and Brandon Franklin

Syngress, 2006. 442 pages.
ISBN 1-59749-109-8

I began worrying about this book when I got to the subtitle. No, I don't mean I objected to the bit about the 8 layers—I figured (correctly) that they'd intentionally added a layer to the OSI model that involves people. I mean the bit about Snort and Ethereal, which are lovely tools, but minimally helpful at hacking people and downright useless at the hardware level. However, subtitles, like the rest of the cover, tend to be editorial decisions over which the authors have very little control.

The idea of using the OSI networking stack as a way to understand networking as a whole, in a practical context, is a good one. It's made more difficult by the poor match between the OSI model and the way TCP/IP works in practice, but a bit of jiggering makes it come out functional.

Nevertheless, this book tries to cover everything about network security. And I do mean everything—from the breeds of dogs that might be appropriate as guard dogs through the bits in an Ethernet frame through encryption, secure software development, and security policies. The results range from OK to so bad they're funny: "Depending on placement, the windows should be either opaque or translucent." Yes, this is referring to physical windows. It is unclear to me whether the authors meant "Windows in high-security areas should be either opaque or translucent" (although if you're going to make the thing opaque, surely getting rid of it would be a better option) or whether they really meant "translucent or transparent." But in any case, in my world, network security is not incompatible with having at least the occasional transparent window.

The coverage of physical security and encryption is worth avoiding at all costs. About Basic XOR encryption it says, "This type of encryption does not contain any mathematical theory or functions that would introduce diffusion or confusion, which helps prevent cryptanalysis attacks that are based on statistical analysis." Now, my understanding of cryptographic theory is basic, but it's sufficient to tell me that XOR is a mathematical function, that diffusion and confusion are complicated, interesting concepts that you can't mention once without further explanation, and that XOR with a key that's significantly shorter than the encrypted text is in fact vulnerable to statistical analysis.

The interesting part of the book is the demonstrations of practical uses of snort, ethereal (or wireshark, as it is now known) and other tools, some of them intended for good, some of them neutral, and some of them distinctly ill-intentioned. This book would be even more useful if the screen shots were more legible, but there's lots of installation information. On the whole, I'd recommend choosing a different book on hacking tools and system administration.

IPV6 ESSENTIALS, 2ND ED.: INTEGRATING IPV6 INTO YOUR IPV4 NETWORK

Silvia Hagen

O'Reilly, 2006. 418 pages.
ISBN 0-596-10058-2

REVIEWED BY PAUL ARMSTRONG

If you're after a solid technical reference on IPv6, this is certainly a good resource. The writing style makes for easy reading and the book is well laid out, starting with some history and then working its way up the stack as you progress through.

There's also a vast amount of information about not only IPv6 but all the networking technologies that you might employ when using it. Making the book even more enjoyable to read is that everything is referenced by RFC and there are notes when an RFC has been superseded. If you're thinking you've forgotten too much about the ins and outs of networking, an introduction to the required information to understand a chapter is, for the most part, also included.

Although the technical side of the book is excellent, I found a few of the sections on deployment to be a little thinner than I would have liked. Particularly frustrating was the case study of the University of Porto, where the sentence "They see the project as a very interesting experience with some peculiar and proactive measures to overcome various problems that had to be solved by the network administrators" left me with the question, "What problems and how did you solve them?"

Although it's aimed at a technical audience, if you're a manager and your team is considering deploying IPv6, you should consider reading Chapter 1, which covers history, why you might deploy it, and misconceptions, and pages 285–310 of Chapter 10, covering integration, case studies, what's missing in IPv6, security, cost, and vendor support.

REAL DIGITAL FORENSICS: COMPUTER SECURITY AND INCIDENT RESPONSE

Keith J. Jones, Richard Bejtlich, and Curtis W. Rose

Addison-Wesley Professional, 2005. 688 pages. ISBN: 0-321-24069-3

REVIEWED BY SAM STOVER

If you are interested in network or host-based forensics, this book belongs on your shelf. Scratch that: It belongs on your desk. Not in your hands—on your desk. You'll need your hands on the keyboard. I don't know about you, but I learn better when I can do something, instead of just reading about it—and this book is all about doing things. Not only are there a ton of exercises throughout the chapters, there are four "Forensic Analysis" chapters that walk you through different Linux, Windows, and USB device forensic processes.

The book starts out with a chapter each on Windows and *NIX Live Response, then moves into three chapters of network-based forensics. Some of the tools and tips given in the Windows and *NIX chapters might be familiar to veteran sysadmins. The network chapters were obviously written by Mr. Bejtlich, as anyone familiar with his other works will immediately recognize his idiosyncratic hostnames and writing style. There is a small bit of overlap with one of his *The Tao of Network Security Monitoring*, which he actually references (unlike some other authors I could mention).

The first five chapters are pretty basic; however, the host-specific chapters will be a good primer for the network folks, and vice versa for the network chapters and the host folks. At this point, everyone is at a common starting point, and digging into the forensic process begins. Chapter 6 walks you through forensic acquisition, which is the process by which you obtain the data to analyze. There are plenty of tips that relate directly to the legal process—probably the most intimidating aspect of forensic work. A good example is documentation: There is a whole chapter named "Before You Jump In" with a three-page section labeled "Document, Document, Document!"

After the acquisition comes the analysis, and this is where the book really shines. There are seven chapters dedicated to forensic analysis techniques ranging from Windows Registry Reconstruction (Chapter 12) to analyzing Windows and Linux files of unknown origin (Chapters 13 and 15). All seven chapters have numerous pseudo "real-life" case studies—along with actual data on the included DVD, which gives you the feeling that you're truly doing the analysis.

Now that you're hooked on the forensic process, Chapters 16 and 17 help you build your own tool kit. I found this part of the book to be a little lean, but I know any author is at a disadvantage when trying to keep up with all of the latest and greatest tools out there. Since this edition was published over a year ago, there are a number of tools that aren't discussed. Personally, I'd really like to see the next edition have a bit more coverage of open source tools. Not that the authors shy away from open source resources; quite the opposite is true. A number of great tools, such as dcfldd and Pasco, are used extensively throughout. I'd just like to see more. Several of the more common commercial tools are also discussed (e.g., FTK, Paraben's PDA Seizure, and the ubiquitous EnCase).

The remaining chapters deal with two new and rather exciting areas. There are three chapters on mobile device forensics that deal with cell phone and PDA devices. The final section focuses on online forensics, with a chapter each on "Tracing E-mail" and "Domain Name Ownership."

In all, I think this book is a must-have for any budding forensics analyst. The case studies are valid and true to real life. The formatting of the book lends itself to a classroom setting as well. Since the data is provided, it would make for an interesting exercise to see what nuggets students (and teacher!) could unearth. There are a few minor spelling and grammatical errors, but nothing worth complaining about. This is truly a fine work—I can't wait for the second edition.

PRACTICAL CRYPTOGRAPHY

Niels Ferguson and Bruce Schneier

Wiley, 2003. 432 pages. ISBN: 0471223573

REVIEWED BY ERIC SORENSON

To be clear from the outset: I cannot in this review provide an objective assessment of the accuracy of Bruce Schneier and Neil Ferguson's *Practical Cryptography*. I'm not fit to lick Bruce Schneier's cryptographical boots, let alone poke holes in his math, but I can say with certainty that the book provided me with a powerful lens through which to view the crypto systems I know and use the most (IKE, SSL, and WEP). I can heartily recommend it to anyone who wants to learn what differentiates good cryptosystems from bad ones and how to make (more) sure the ones you build are the former, not the latter.

The authors first provide a gentle introduction to first principles ("complexity is the worst enemy of security"), then step deeper into cryptographic primitives such as block ciphers and hash functions and use these building blocks to solve real-world problems, including creating a secure channel over an untrusted medium and negotiating session keys using public-key crypto. The last third of the book emerges from the heavy math to discuss technical and political issues surrounding PKI, the standards process, and technology patenting.

The book is aimed at engineers who are designing or implementing a computer system that uses cryptography, so the section on ciphers and cipher modes, hashes, and MACs has a very pragmatic format: the authors discuss the purpose of each kind of primitive, survey the landscape of options, and offer a recommendation. I found this discussion both fascinating and useful. Previously, when setting up IPSEC VPNs, I had known there were MD5 and SHA options for packet authentiation, but other than selecting the one both endpoints supported I didn't know which to choose. Now I do: The authors recommend SHA because of the higher potential for "birthday" attacks against MD5.

The book's mathematical discussions of randomness, prime numbers, and modulo arithmetic get heavy pretty quickly. The authors realize this and provide both escape hatches ("we won't use this formula [for entropy], so you don't need to remember it") and levity ("most of the math dates back thousands of years, so it can't be too difficult, right?"). They start with basic high school math concepts like the Sieve of Eratosthenes and build from there. I confess I got lost around the section discussing Garner's Formula, but I believe with time and patience one could make it through the whole discussion without needing external references or specialized training.

As I said, the book provides a lens to examine existing cryptosystems. Nowhere is this more evident than in the final chapters, on the dreams versus the reality of PKI and the evils of security protocol design by committee. Bruce's razor-sharp skewering of politial compromises to security problems will be familiar to readers of his blog and Crypto-Gram newsletters; like his nontechnical book *Secrets and Lies*, *Practical Cryptography* gives us the treat of reading his ideas in a more extended format.

Schneier and Ferguson's *Practical Cryptography* is a mine of information for people who want to know more about crypto systems, whether to create new ones or simply to understand better the protocols we use every day to read our email, conduct banking transactions, or browse the Web over wireless while sipping a mocha. The authors don't shy away from technical details, so protocol designers could use this book to avoid pitfalls they might not have known existed, but neither is this book so dense as to be incomprehensible to nonmathematicians. Highly recommended.

LINUX ADMINISTRATION HANDBOOK, SECOND EDITION

Evi Nemeth, Garth Snyder, and Trent R. Hein, with Adam Boggs, Matt Crosby, and Ned McClain

Prentice Hall PTR, 2006. 1,000 pages.
ISBN: 0-13-148004-9

REVIEWED BY
RIK FARROW

This latest in the series of administration handbooks shares its friendly yet authoritative tone with its predecessors. I felt I was constantly being provided with excellent advice and guidance as I read sections of this book, advice that I generally agreed with (a nice feeling). I started out in Chapter 7, "Adding a Disk," a thorough tutorial in the various issues involved in adding a new disk to a Linux system. I wanted to see just how much what I already knew from experience matched the chapter and to learn about features I had never used. Along the way, I picked up concepts I hadn't understood, such as the use of hdparm, software RAID, and Linux Logical Volume Management. I liked how the authors have provided accurate cross references to other sections (by page number, not section number, making these easy to find). In the chapter on TCP/IP networking, I learned about mii-tool, something I knew must be there (a way to configure autonegotiation of network interfaces) but was never able to find. This book covers all of the sysadmin topics.

Even a 1,000-page book cannot cover all there is to know about the myriad system administration topics mentioned. The authors' strategy in those cases is to suggest the best references available, be they other books, man pages, or Linux doc files. They go beyond just explaining software to covering network hardware and providing advice about which backup hardware will best fit your needs. It is this soup-to-nuts approach that has helped make this the most popular system administration series of all time.

Occasionally I did identify something that I considered an oversight. For example, although Linux does support the route command, it has largely been replaced by the much more powerful ip command, which gets no mention at all (that I could find). ip appears to be a direct connection to the IP stack in the kernel that gives you more control than route and ifconfig combined, but a section of the book on this would have been great (and is on my wish list for the next edition). Other than this minor oversight, I consider this a great book, one both new Linux sysadmins and other experienced sysadmins will appreciate having.

C IN A NUTSHELL

Peter Prinz and Tony Crawford

O'Reilly, 2005. 600 pages.
ISBN: 0-596-00697-7

REVIEWED BY
RIK FARROW

I can't say I read this book cover to cover. It is a reference book, as its title suggests, and it functions very well at that. I learned C the way I was taught to learn languages in college: with a grammar, some examples, and a compiler to practice with (BDS C). That left lots of holes in my knowledge of C, holes that *C in a Nutshell* performs well to fill in.

The authors describe the use of the auto storage class specifier as "archaic," just like my knowledge of C. They just as clearly explain what the restrict type qualifier does (provides a hint to the compiler that an object will only be referenced via the given pointer). The single largest section in the book covers libc, and it differs from the online man pages in several ways, the most important of which is to provide program examples showing how the function is used. The book starts with the grammar (223 pages), then explains the standard headers and functions (next 270 pages), then finishes up by describing gcc, make, and gdb.

Altogether, this book is a great desktop reference for anyone who needs to understand C. Somehow I have managed to read and write C programs without this book, but I will certainly appreciate having it handy in the future—it is now within easy reach of my favorite chair.

HANS BOEHM, BILL PUGH, AND DOUG LEA

standards: multithreading in C and C++

Hans Boehm is a researcher at HP Labs who is best known for his work on garbage collection. Recently he has focused on improving concurrent programming foundations, especially for C++.

hans.boehm@hp.com

William Pugh is a professor at the University of Maryland, College Park. His current research focus is on developing tools to improve software productivity, reliability, and education.

pugh@cs.umd.edu

Doug Lea, a professor of computer science at SUNY Oswego, is the author of several widely used software packages and components, as well as articles, reports, and standardization efforts dealing with object-oriented software development.

dl@cs.oswego.edu

MAINSTREAM DESKTOP and server machines increasingly require explicitly concurrent programs to achieve full performance, owing to the increasing prevalence of both single-chip multiprocessors and hardware support for multiple threads.

Currently a common way to write such programs is to program in C or C++, with the aid of a threads library, such as an implementation of the POSIX pthreads interfaces, to provide concurrency. This is also an established technique for handling multiple concurrent event streams, even on single-threaded single-processor machines.

Unfortunately, this approach has turned out not to be completely sound, primarily because reliable multithreaded execution requires certain guarantees about the language and compiler that cannot easily be provided by a library or library specification [1]. Some of the associated issues have been understood for many years. The second half of this paper briefly outlines a symptom of this issue that appears to not have been well recognized.

As a result, several of us have started an effort to address these problems by directly defining the meaning of multithreaded programs in the underlying programming language. Initially this is being done in the context of C++, building on some earlier work in the context of Java [2, 3].

There appears to be consensus that these issues should be addressed in the current ongoing revision of the C++ standard. In that context, we are addressing three somewhat separable issues:

1. Defining the meaning of existing programs in the presence of threads. Our current approach largely follows pthreads and leaves the semantics undefined if there is a data race, i.e., if a program modifies a location while another thread is accessing it. This approach appears to be the only plausible one for C and C++. However, it can only succeed if the definition of a data race is made precise enough for programmers, compiler writers, and hardware to know when data races occur and how to avoid them. Currently, it is not defined at all. Among other consequences, unexpected compiler transformations regularly break multithreaded programs (as illustrated in the example that follows).
2. Defining an atomic operations library to allow the construction of correct multithreaded programs without locks. This does not directly affect most existing application-level programs, though a significant number of them should be modified to use this library in order to ensure correctness. Such a library is necessary for development of portable core libraries and infrastructure code that increasingly use lock-free techniques to implement high-performance synchronization support. Defining an atomics library relies critically on the semantics of memory operations and data races.
3. Designing a threads API that meshes better with the rest of the C++ language.

We expect that the first two issues and their solutions also apply, with minor modifications, to C. And compatibility would be greatly desirable. We expect the last issue is mostly C++ specific, though there are likely to be exceptions, such as support for thread-local storage.

As a result, we would like to encourage members of the C committee to follow our discussions and to provide input, particularly if they see aspects of our approach that would make it less palatable to the C committee, and hence lead to unnecessary divergence between C and C++.

A Simplified Example

We illustrate some of the problems addressed by this work with a simple case in which the current language specifications for C and C++ are clearly inadequate for multithreaded programs. This is only one among many possible examples. It helps demonstrate that the problems are in fact profound and must be addressed by the language specification and compilers. It also points out that the expected impact on compilers is likely to be nontrivial.

Consider the following declarations and function definition:

```
int global_positives = 0;
typedef struct list {
  struct list *next;
  double val;
} * list;

void count_positives(list l)
{
  list p;
  for (p = l; p; p = p -> next)
    if (p -> val > 0.0)
      ++global_positives;
}
```

Now consider the case in which thread A performs

```
count_positives(<list containing
only negative values>);
```

while thread B performs

```
++global_positives;
```

This should be perfectly correct, since count_positives in this specific case does not update global_positives, and hence the two threads operate on distinct global data and require no locking.

But some existing optimizing compilizers (including gcc, which tends to be relatively conservative) will "optimize" count_positives to something similar to

```
void count_positives(list l)
{
  list p;
  register int r;

r = global_positives;
  for (p = l; p; p = p -> next)
    if (p -> val > 0.0) ++r;
  global_positives = r;
}
```

This transformation is clearly consistent with the C language specification, which addresses only single-threaded execution. In a single-threaded environment, it is indistinguishable from the original.

The pthread specification also contains no clear prohibition against this kind of transformation. And since it is a library and not a language specification, it is not clear that it could.

However, in a multithreaded environment, the transformed version is quite different, in that it assigns to global_positives, even if the list contains only negative elements. Our original program is now broken, because the update of global_positives by thread B may be lost, as a result of thread A writing back an earlier value of global_positives. By pthread rules, a thread-unaware compiler has turned a perfectly legitimate program into one with undefined semantics.

This is a contrived example, but similar issues have been encountered in practice, and these are discussed in more detail in Boehm [1]. We hope this has served as a brief introduction to the kind of problems we are trying to address and will encourage others to follow the discussion.

REFERENCES AND FURTHER READING

[1] H.-J. Boehm, "Threads Cannot Be Implemented as a Library," in *Proceedings of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation*, pp. 26–37, 2005. Also available at http://www.hpl.hp.com/techreports/2004/HPL-2004-209.html.

[2] JSR-133 Expert Group, JSR-133: Java Memory Model and Thread Specification: http://www.cs.umd.edu/~pugh/java/memoryModel/jsr133.pdf, August 2004.

[3] J. Manson, W. Pugh, and S. V. Adve, "The Java Memory Model," in *Conference Record of the Thirty-Second Annual ACM Symposium on Principles of Programming Languages*, January 2005. Also available at http://www.cs.umd.edu/users/jmanson/java/popl05.pdf.

USENIX notes

USENIX MEMBER BENEFITS

Members of the USENIX Association receive the following benefits:

FREE SUBSCRIPTION to *;login:*, the Association's magazine, published six times a year, featuring technical articles, system administration articles, tips and techniques, practical columns on such topics as security, Perl, VoIP, and operating systems, book reviews, and summaries of sessions at USENIX conferences.

ACCESS TO ;LOGIN: online from October 1997 to this month: www.usenix.org/publications/login/.

ACCESS TO PAPERS from USENIX conferences online: www.usenix.org/publications/ library/proceedings/

THE RIGHT TO VOTE on matters affecting the Association, its bylaws, and election of its directors and officers.

DISCOUNTS on registration fees for all USENIX conferences.

DISCOUNTS on the purchase of proceedings and CD-ROMs from USENIX conferences.

SPECIAL DISCOUNTS on a variety of products, books, software, and periodicals. For details, see www.usenix.org/membership /specialdisc.html.

TO JOIN SAGE, see www.usenix.org/ membership/classes.html#sage.

FOR MORE INFORMATION regarding membership or benefits, please see www.usenix.org/membership/ or contact office@usenix.org. Phone: 510-528-8649

USENIX BOARD OF DIRECTORS

Communicate directly with the USENIX Board of Directors by writing to *board@usenix.org*.

SUMMARY OF USENIX BOARD OF DIRECTORS MEETINGS AND ACTIONS

ELLIE YOUNG
ellie@usenix.org

The following is a summary of the actions taken by the USENIX Board of Directors from July through December 2006.

CONFERENCES

The following people were invited to serve as program chairs for upcoming conferences:

- Jeff Chase and Srinivasan Seshan for the 2007 USENIX Annual Technical Conference
- Paul van Oorschot for the 2008 USENIX Security Symposium
- David Wagner for the 2nd USENIX/ACCURATE Electronic Voting Technology Workshop

It was agreed to hold the 2007 Linux Kernel Developers Summit in Cambridge, UK.

It was agreed that USENIX would sponsor a workshop on Hot Topics in Understanding Botnets (HotBots '07), to be held alongside NSDI, per a proposal by Niels Provos.

It was agreed that USENIX will sponsor the Distributed Event-Based Systems Workshop and fund $3,000 in student stipends for it.

It was agreed that USENIX be an in-cooperation sponsor of the ACM Symposium on Computer Human Interaction for Management of Information Technology (CHIMIT) and the ACM Workshop on Experimental Computer Science.

USENIX will be a co-sponsor of the SETIT 2007 conference.

It was decided that when booking future hotels for the LISA conference, we will alternate venues between the West Coast and the Boston and D.C. areas.

TRAINING

It was agreed to proceed with discussions with the SANS Institute on offering tutorials at each other's events.

OUTREACH/GOOD WORKS

It was agreed, per a proposal from Garth Gibson, that USENIX would host a repository of computer anomaly data.

It was agreed to contribute $15,000 to the USA Computing Olympiad in 2007, per a proposal from Rob Kolstad.

Niels Provos will liaison with Kirk McKusick on BSD issues/conferences.

It was agreed to continue funding POSIX and ISO SC22 standards activities in 2007, with an additional amount to support attendance at the C++ meeting in Oxford.

FINANCES

The registration fees for the USENIX Annual Technical Conference were raised by $10 per day, and for FAST and NSDI by $20 for the three-day technical session registration. The registration fees for HotOS were reduced to match those of the other USENIX three-day workshops.

Member dues, which had not been raised since 2005, were increased by $5.00 for USENIX and SAGE, in order to cover increased costs.

A first-draft budget for 2007 was approved in December 2006, and the auditor was selected to perform the audit of the 2006 financial statements.

SAGE

Upon notification that Strata Rose Chalup was stepping down as SAGE Programs Director, the USENIX Board of Directors passed a motion thanking Strata for her services to USENIX, SAGE, and the profession of system administration. At this time, SAGE activities will be managed by Jane-Ellen Long, with assistance from Tony Del Porto and oversight and guidance from Alva Couch, who is chair of the SAGE subcommittee of the USENIX Board of Directors.

SAGE UPDATE

JANE-ELLEN LONG
jel@usenix.org

ALVA COUCH
alva@usenix.org

A lot has been happening lately. The sad news first: Strata has moved on to nurture her growing consulting practice at Virtual.Net Inc. We'll miss her vision, her dedication, and her humor. We hope she'll drop in from time to time to cheer us and nudge us into the future, and of course SAGE members can always catch her on the sage-members discussion list. (Haven't joined yet? See www.sage.org/lists/.)

ALL ABOUT LISA

Those of you who weren't lucky enough to be among the over 1,200 attendees of this year's LISA Conference can now listen to Cory Doctorow's fascinating keynote address and to Alva Couch's own provocative "lunch & learn" session on "The Future of System Administration: How to Stop Worrying and Learn to Love Self-Managing Systems." Even if you did attend the four-track LISA, you probably had to make some hard choices among sessions. Now's your chance to catch up on what you missed: All the refereed papers are open to USENIX and SAGE members online, and many talk slides are up as well, at www.usenix.org/events/lisa06/tech/.

We are delighted that Paul Anderson has agreed to chair LISA '07. Paul has been a contributor to SAGE and USENIX for many years. He is the author of a SAGE booklet on system configuration, as well as a SAGE white paper on the topic; he's organized the Configuration Management Workshop at LISA every year since 2002; and he's the principal author of LCFG. Watch the SAGE Web home page for the Call for Papers, coming in early 2007: www.sage.org/index.html#confs.

We are pleased to announce that, once again, two SAGE awards were presented at LISA '6. The Outstanding Achievement Award went to Tobias Oetiker and Dave Rand, and the Chuck Yerkes Award went to Doug Hughes. Read more at www.sage.org/about/outstanding.html and www.sage.org/about/yerkes.html.

SAGE SHORT TOPICS GET LONGER

Internet Postmaster: Duties and Responsibilities, by Nick Christenson and Brad Knowles, is now online at www.sage.org/pubs/15_postmaster/. Available very soon: *Host Configuration and Maintenance with Cfengine,* from The Source, a.k.a. Mark Burgess and his henchwoman Æleen Frisch, who have taught many of you in LISA training classes. This booklet was inspired by a discussion on sage-members.

YouChoose: SAGE members have told us you love the SAGE Short Topics series and want more. Some of you have said you only want PDFs. Some have said you'd rather have a different booklet instead of whatever one came out most recently. We took a good hard look at the program and came up with an answer that, we hope, addresses these issues. Instead of waiting for one booklet to have its day in the sun before publishing another, we plan to publish at will. All the

PDFs will be available anytime to all SAGE members. We'll put up an online ordering form as well. Each year of your membership you may choose *any* one booklet to be sent to you for free, and you may order any additional print booklets for $10 per booklet, as of yore. BTW, have you noticed that the booklets are getting longer? Greater depth of information as well as greater breadth is the goal.

Want to write a booklet? Have an idea for a subject and/or author? Please let us know: send a note to sagebooklets@sage.org.

NEW WHITE PAPERS ONLINE

White papers from the LISA '06 Hit the Ground Running Track are now online. If you don't feel ready to write a booklet, but your job has offered you a learning experience and you'd like to keep others from having to go through that particular torture, write a white paper to share with the world via www.sage.org/pubs/whitepapers/whitepapers.html.

WHAT DO YOU WANT FROM SAGE?

We have our own ideas, but to truly serve the membership, we need to know what's most important to you. How can we best serve you? Please send your ideas, fully formed or carry-out to bake at home, to us at suggestions@sage.org.

Thanks for supporting SAGE, and we'll catch up with you again in April.

JOHN LIONS FUND WRAP-UP

The USENIX Board voted to match up to $250,000 in 2006 donations to establish the John Lions Chair in Operating Systems at the University of New South Wales.

USENIX received $39,224 from outside donors in 2006. Linux Australia made the largest donation, $18,022. USENIX matched dollar for dollar all donations, so the grand total donated to UNSW for this campaign will be $80,000.

Theodore Ts'o, USENIX Treasurer, presented a check for this amount to Gernot Heiser, who represented UNSW, at linux.conf.au 2007.

2007 USENIX ANNUAL TECHNICAL CONFERENCE

JUNE 17–22, 2007, SANTA CLARA, CALIFORNIA, USA

Join us in Santa Clara, CA, June 17–22, for the 2007 USENIX Annual Technical Conference. USENIX Annual Tech has always been the place to present groundbreaking research and cutting-edge practices in a wide variety of technologies and environments. USENIX '07 will be no exception.

✦

USENIX '07 WILL FEATURE:

- An extensive Training Program, covering crucial topics and led by highly respected instructors
- Technical Sessions, featuring the Refereed Papers Track, Invited Talks, and a Poster Session
- Plus BoFs and more!

JOIN THE COMMUNITY OF PROGRAMMERS, DEVELOPERS, AND SYSTEMS PROFESSIONALS IN SHARING SOLUTIONS AND FRESH IDEAS.

http://www.usenix.org/usenix07

conference reports

IN THIS ISSUE

WORLDS '06: 3rd USENIX Workshop on Real, Large Distributed Systems

Seattle, Washington
November 5, 2006

Summarized by Iulia Ion

MEASUREMENT AND MONITORING

Geolocalization on the Internet through Constraint Satisfaction

Bernard Wong, Ivan Stoyanov, and Emin Gün Sirer, Cornell University

Bernard Wong presented a new method of determining the location of Internet hosts, based on constraint satisfaction. With existing techniques such as static databases, users have to get precise city location data from the database and constantly update this information. The authors propose a dynamic approach that uses latency information to determine the location of nodes. The main challenge is caused by network congestion, which might introduce extra delay and therefore makes it difficult to locate node positions.

Bernard presented the Octant system, which extracts constraints based on network measurements, anchors them to the globe, and makes a latency-to-distance relationship. Given the basic assumption that there is a strong correlation between latency and distance, Octant shows the likeliness that a node is a particular distance away. The data is afterward formalized as positive and negative constraints. Given different landmarks, the speaker proposes taking the intersection of the constraints. To deal with overaggressive constraints, different weights can be assigned. Therefore, results have different degrees of confidence, which increases exponentially as the latency reduces linearly. An additional technique used in Octant to address indirect routing and its effects on constraint construction involves piecewise localization, wherein each node is localized depending on another. Some other results-improving techniques are congestion estimation (reducing or increasing the actual latency as measured) and removing regions where people are unlikely to live, such as oceans, rural areas, and deserts.

To evaluate Octant, the authors collected traceroute data among 51 PlanetLab nodes and compared the results to previous geolocalization techniques. GeoLim proved to be the next best system. Octant is able to extract more aggressive constraints with lower error rates. The use of geographic and demographic constraints can further reduce the size of the estimated target region. A demo can be viewed at https://www.cs.cornell.edu/~bwong/octant/query.html.

Audience questions addressed the following: (1) How would Octant perform on DSL nodes of PlanetLab? Bernard answered that although currently Octant uses traceroutes to get the latency information, different measurement techniques could also be used to fit with the PlanetLab DSL nodes model. (2) How well does Octant work outside the United States? Bernard answered that the system has not yet been evaluated. The results would depend on the concentration of nodes in the region (e.g., in Europe it would work better than in Australia). Octant pins down the intermediate nodes in series, but pinning down the first router depends on the previous ones. (3) How accurately are you able to pin down intermediate routers toward the end? Bernard replied that it is difficult to measure accuracy. (4) How often do the final regions end up being disconnected parts? Answer: It happens a lot.

■ *A Platform for Unobtrusive Measurements on PlanetLab*

Rob Sherwood and Neil Spring, University of Maryland

The talk was given by Rob Sherwood. Rob started by explaining the need for measurements and the benefit that these would bring to many applications such as performance optimization, overlay construction, and network diagnosis. The grand challenge would be to record a day in the life of the Internet. The problems encountered in making such measurements are mainly due to firewalls, abuse reports, and limited bandwidth. Abuse reports occur because exceptional measurement traffic is often considered suspicious and prevents one from measuring everything.

The measurement platform used was Sidecar, which works by injecting probes into normal traffic. Rob presented the tool, how it works, and a couple of quick examples. Basically, Sidecar tracks connections by recording data as it passes by. Neither side knows that there is any sort of probing going on. Sidecar can traverse NATs and firewalls. Probes are retransmissions and require no end-point support. The authors can modify probes for specific measurements such as reducing TTLs, sending probes in trains, and adding IP options.

Rob explained that most often the abuse reports were caused by the application logs. The lesson learned is that when doing traffic generation, one must pay attention to what abuse records are generated. Other problems were caused by clock irregularities (clocks would change rate and jump backward), causal packets reordering, and lag in I/O Systems Calls. The Artrat tool can be used to try to decide from the receiver side where the bottleneck is. The technique uses the IP timestamp option with ICMP echo to measure queuing delay.

Rob was asked whether the measurement platform requires symmetric routes. He answered no. Sidecar doesn't require symmetric routes, but it does assume that the Sidecar listening machine is on both the forward and the reverse path. In their experiments, they simply ran Sidecar on one of the end-hosts to accomplish this.

■ *ConfiDNS: Leveraging Scale and History to Improve DNS Security*

Lindsey Poole and Vivek S. Pai, Princeton University

The talk was given by Lindsey Poole.

Although cooperative DNS resolver systems, such as CoDNS, have demonstrated improved reliability and performance over standard approaches, their security has been weaker, since any corruption or misbehavior of a single resolver can easily propagate. The authors addressed this weakness in a new system called ConfiDNS, which augments the cooperative lookup process with configurable policies that utilize multisite agreement and per-site lookup histories.

The threat model used focuses on client-side attacks because they are easier to carry out and harder to catch. The advantage of the technique is that there is no need to change the server infrastructure. An incrementally deployable client-side solution can be carried out. The authors evaluated the system and proved that ConfiDNS can provide better security than CoDNS and local resolvers, while retaining the other benefits of CoDNS, such as incremental deployability, improved performance, and higher reliability.

Lindsey was asked whether it would be more useful to source-route DNS lookups from each client and avoid local nameservers entirely. The reasons for not using this approach would be a large increase of lookup traffic to potentially constrained nameservers, and failure to defend against adversaries operating on the local network. ConfiDNS can use encrypted peer traffic to avoid local adversaries, and the peer traffic can be absorbed at peer resolvers, mitigating the impact on remote nameservers. The final observation was that only 10% of failures are client-side, and 30% are server-side.

MANAGING SCARCE RESOURCES

■ *Resource Management for Global Public Computing: Many Policies Are Better Than (N)one*

Evangelos Kotsovinos, Deutsche Telekom Laboratories; Iulia Ion, International University in Germany; Tim Harris, Microsoft Research Cambridge

The talk was given by Evangelos Kotsovinos.

Management responsibility in global public computing systems is distributed among different individuals and organizations. Such stakeholders can be network administrators, server owners, or infrastructural authorities. Unfortunately, high-level resource management facilities are often absent from public computing systems. Whereas federation is crucial for scalability and cost-efficiency, it introduces important resource management challenges related to expressing policies and managing policy overlaps. Usually, there are different users asking for resources on a server. The challenge is reaching a decision given the different policies and interests of different stakeholders.

Evangelos explained that these overlaps occur because different stakeholders may have different views on how server resources are to be apportioned. The authors proposed a practical system that allows the different stakeholders to independently express federated policies. The Role Based Resource Management system (RBRM) provides mechanisms for resolving potential constraint overlaps automatically and reaches decentralized decisions.

Role-based resource management policies are defined using a Web interface and consist of the following elements: role declarations (a group of users to which common policies apply), role entry conditions, constraint definitions (reservation or usage limitation on a resource that applies to all members of a role), and constraint relationships. When more than one constraint is applicable to a user request for a certain resource there is an overlap; the system needs to determine how much of the resource can be made available to the user. The system supports advanced pattern-matches for existing constraints, together with variable bindings, and generates a replacement constraint for the ones that overlap.

When a user requests resources from a server, resource allocation is determined based on the policies deployed on the server, resource availability, and user properties and credentials. Depending on these inputs, the system allocates the requested resources, denies access, or starts negotiation.

The authors demonstrated experimentally on the XenoServers platform that the system scales gracefully and introduces only a very low performance overhead. RBRM is suitable for operating in realistically large and complex settings. Furthermore, it has been able to express a large set of real-world policies that users of existing platforms have requested.

The addressed questions were: (1) How would the system work for servers or services as opposed to users who do it for themselves? (2) Give us some examples of where it would be useful to negotiate. Would the user be happy with less? Evangelos answered that users will ask for a much higher allocation than they actually need. In such cases they might reconsider it. (3) Do you have a model for online negotiation? Can you do RBRM recursively? The answer was yes, it's supported by the framework. (4) Suppose I gave all the bandwidth away and somebody else comes. Is there a way I can renegotiate resource allocation for running sessions? Evangelos answered that, with the current model, once resources are allocated, they cannot be revoked for the lifetime of the session.

Evangelos invited everyone to attend the demo session scheduled later that day where the authors would present a demo of the running system. Further information can be obtained at http://www.xenoservers.net/.

■ *Optimizing Grid Site Manager Performance with Virtual Machines*

Ludmila Cherkasova and Diwaker Gupta, Hewlett-Packard Labs; Eygene Ryabinkin, Roman Kurakin, and Vladimir Dobretsov, Russian Research Center "Kurchatov Institute"; Amin Vahdat, University of California, San Diego

The talk was given by Lucy Cherkasova and dealt with analysis of Grid workload for the past year from a Tier-2 Resource Center at the RRC Kurchatov Institute (Moscow, Russia). The analysis revealed that a large fraction of Grid jobs have low CPU utilization.

Virtualization can add many desirable properties to Grid computing, such as customized environments, QoS provisioning, and policy-based resource allocation. Additionally, the authors sought to justify an economic and performance incentive to move to a VM-based architecture. Using a simulation model, they showed that a half-size infrastructure augmented with four VMs per node can process 99% of the load executed by the original system in RRC Kurchatov Institute.

The authors described a prototype design built on top of the Xen VMM. The goal of the developed prototype is to integrate VMs in the Grid workflow. The prototype offers a clear separation between mechanisms and policies, was deployed for testing, and was integrated with the Grid workflow in the Kurchatov Institute.

Future work involves getting data from the running system, investigating how often migration is needed, and addressing scalability issues. The authors plan to determine the best migration policies and develop a management suite for enterprise applications.

These follow-up questions were asked: (1) To what extent are the presented resource usage figures representative of global public computing platforms in general? While the data we have are quite representative (spanning more than a year), this is the first Grid workload study of a single data center. We do not have enough publicly available data from other data centers to draw general conclusions.

(2) What fraction of resources are used by short versus long jobs? Study has shown that 20% of overall CPU usage is consumed by the jobs that run less than one day (which represent 92% of all the jobs). Jobs that run around 3 days (representing 4% of all the Grid jobs) are responsible for 42% of overall CPU usage.

For further information check out http://www.hpl.hp.com/personal/Lucy_Cherkasova/projects/grid-vm.html.

PLANETPANEL

■ *Learning from PlanetLab*

Thomas Anderson, University of Washington; Timothy Roscoe, Intel Research Berkeley

The session was started by Thomas Anderson.

Although PlanetLab has been enormously successful in fostering distributed system research, it is not as successful as it could be. Thomas looked at nine important reasons why PlanetLab is not yet the platform for huge distributed systems. PlanetLab is not viral, as is BitTorrent, where people are contributing resources to the system. There are no people contributing resources in order to get access to the system. PlanetLab has enduring limitations. In terms of scalability, although the number of participants is increasing, the number of online nodes remains constant.

Thomas's hypothesis is that not enough has been learned from past experience. He describes nine decisions that have been crucial to PlanetLab's success but which, he argues, should be rethought now that PlanetLab is successful. His points are:

1. Centralizing trust: PLC is a trusted intermediary between node owners and node users. Thomas argues that a single point of trust is unsustainable and that trust should be explicit and flexible. Each site should be able to select its own PLC.
2. Centralizing resource control: PlanetLab Central controls resource allocation. Site administrators have very limited control over what runs on their site, or which jobs get which resources. Thomas argues for better incentives for management.
3. Decentralizing management: By design, PlanetLab provides minimal services to users, which has not worked in practice. The authors argue that encouraging community contributions is inconsistent with centralized trust/control. Instead, we need a set of initial versions of services to demonstrate that the API is complete.
4. Treating bandwidth as free: PlanetLab does not charge users for bandwidth. The authors believe that the lack of accounting offers perverse incentives. Instead, accurate fine-grained cost accounting, visible to applications, is needed.
5. Providing only best effort: PlanetLab provides no resource reservations or resource predictability. There are no limits on the number of jobs that run on each node. The authors' view is that power users crowd out everyone else and that there is room for much better short-term/long-term schedulers.
6. Using Linux as the execution environment: Thomas argues that Linux is the wrong API for distributed systems. The audience argued for the advantages of Linux and gave as examples the existence of top, personal folders, and ssh agents.
7. Distributing OS services: PlanetLab is a distributed OS with few distributed services.
8. Evolving the API: PlanetLab was designed to evolve.
9. Focusing on the machine room: PlanetLab focused on large machines. Thomas brings up the issue of running PlanetLab on PDAs and other small devices.

Thomas concludes that for PlanetLab or GENI to thrive requires large-scale community involvement in defining and improving the platform.

■ *The Lessons of PlanetLab*

Thomas Anderson, University of Washington; Marc Fiuczynski, Princeton University; Michael Freedman, New York University; Rob Ricci, University of Utah

Michael Freedman talked about the problem of misaligned incentives and pointed out that the success of PlanetLab is largely judged by the number of nodes and of slices, not by impact and result.

Another problem with PlanetLab is that slices cannot specify policies. The proposed solution is to provide an ability to easily determine the current status of sessions. However, the concern is that virtualization and isolation of resources is not a panacea.

Michael also argued that decentralized trust/control exists and that sites rarely enforce their local AUPs, and only then in a haphazard manner; therefore he proposes an explicit method for expressing rules and policies.

Marc E. Fiuczynski brought again into discussion the lack of incentive for people to contribute with resources and argued that not enough resources are available, because people do not contribute. Again the discussion was brought to ssh forwarding, which got broken as PlanetLab evolved. The reason for this is that a degree of isolation between researchers was needed, but instead the result was resource isolation. Are there things that PLC could be doing differently to provide incentives or recognize input from users? The future envisages Private PlanetLab (MyPLC) where you can bring up your own private PlanetLab in 30 minutes. MyPLC lets you have complete control over software and API, allowing you to implement your own resource control and have several PlanetLab deployments. In such a context, Marc referenced the work on Role Based Resource Management and expressed his hope of using such a framework to do policy management in a scalable fashion.

Real challenges remain in specifying peering agreements between private PlanetLabs (e.g., what to do when one party is in violation of an agreement) and resource management and control (expressing federated resource management policies and managing conflicts). Finally, the session ended with a reference to the Prisoner's Dilemma: If I buy 10 boxes, what's the benefit to me?

TRENCHES

Summarized by Rik Farrow

■ *Towards Fingerpointing in the Emulab Dynamic Distributed System*

Michael P. Kasick and Priya Narasimhan, Carnegie Mellon University; Kevin Atkinson and Jay Lepreau, University of Utah

Mike Kasick began by explaining that the number of errors produced while Emulab is used overloads operators. Emulab has 1300 users, 430 local nodes, and 740 remote nodes (including PlanetLab) and uses software created over five years comprising 490,000 lines of code, within many scripts. Emulab allows users to create virtual networks as well as load operating systems and applications on the PCs within Emulab.

Since the existing error-reporting system was deficient, they first built tblog, a set of scripts that logged all errors to a database, and included new functions that can be called from within scripts. tblog performs call-chain analysis to determine which of a cascade of error messages contains information about the event that caused the failure to occur. In the previous system, operators would have to examine several email messages and collect context from other log files to determine the triggering event.

Tblog improved the situation, but it did not solve the root problem. Existing scripts produced opaque messages, designed to be human readable but not machine parsible. Their second approach, tbreport, focuses on producing structured error messages that are easily parsible by scripts because they include consistent error types and sufficient context, and always propagate the primary errors, avoiding "me too" error messages. Mike provided some examples of how tbreport helped to improve error analysis through live use on Emulab. David Anderson of Carnegie Mellon asked, "What five-page document should I read now to avoid the problems you found in Emulab?" Mike answered that there isn't such a document, but he advises people to modularize code. When people write code, they focus on the success case and don't includes labels in code so you can grep through it searching for failure points. Also, use RPC mechanisms to do global finger-pointing.

■ *Data Management for Internet-Scale Single-Sign-On*

Sharon E. Perl, Google Inc.; Margo Seltzer, Harvard University and Oracle Corporation

Sharon Perl described her experiences while building a unified login system for Google Accounts that supports both Gmail and AdSense. Google began life without any notion of customer accounts, but work began in April 2002 to create a very simple backend database. Version 2 used the same API, but it replaced the database with a replicated Berkeley DB–based system. Sharon pointed out that "at Google's scale, even rare events happen often." They needed consistency, automated failover, and low latency.

Sharon knew about Paxos, a consensus algorithm designed so that all nodes in a distributed system will agree on a value. After a Google tech talk by Margo Seltzer, Sharon became aware that Berkeley DB could do replicated backups and already provided the simple key-to-value support needed, so she decided to make a Berkeley DB system that worked like Paxos. The production single sign-on system relies on a single master system which holds a lease that allows the master to commit changes. If there is a timeout by the master leaseholder (on the order of seconds), replicants vote to select a new master. The actual data gets replicated across several datacenters. Locking depends on the Chubby Lock mechanism (see the relevant paper in OSDI '06).

A lively Q&A session followed. One person wondered how they would know if they had multiple masters at some point, and Sharon answered that they could look at timestamps in logs and see that sometimes there were two masters. Another person asked how clock skew would affect lease timeouts. Sharon answered that someone within Google who understood hardware came up with a safe timeout value (which is on the order of several seconds, as I found out later). Someone asked about server replacements. Sharon answered that reboots are okay, as no state gets lost, but losing a disk would be a problem.

■ ***A Distributed File System for a Wide-Area High Performance Computing Infrastructure***

Edward Walker, University of Texas at Austin

Ed Walker works in the Texas Advanced Computing Lab and uses NSF TeraGrid, a national high-performance computing infrastructure for performing large-scale engineering and scientific problems. TeraGrid currently uses GPFS crossmounts for supporting remote file sharing. But because of operating systems issues, not all sites can use IBM's GPFS, and in a survey of users in 2005, scp was cited as the most important data management tool.

Ed pointed out that many desktops are becoming computation science–capable and that the majority of links within TeraGrid participants have less than 2% utilization, so that bandwidth can be used. He then described XUFS, a userspace overlay that hooks file system calls by interposing a shared object before libc to get transparent file system redirection. XUFS has goals of location transparency (since laptops and even desktops move easily), performance, and private name space, but not file sharing, as his research has shown that scientific computing files are rarely shared (umask of 077). XUFS aggressively uses local caches, and it also performs write-on-close to sync up locally made changes with the remote copy.

In performance testing, XUFS does as well as or better than GPFS in most cases (the exception being smaller files). XUFS has a command-line tool for flushing the local write cache in case of a client crash, and automatic recovery in case of host crashes or network outages. Gunnar Sirer asked about the lack of support for file sharing. Ed answered that out of nearly 2000 GPFS users, only one had changed the default permissions to all group read permissions within directories.

OSDI '06: 7th USENIX Symposium on Operating Systems Design and Implementation

Sponsored by USENIX in cooperation with ACM SIGOPS

Seattle, Washington November 6–8, 2006

■ *Opening Remarks*

Summarized by Rik Farrow

OSDI 2006 began with record rains in Seattle, but the rain and local flooding did nothing to dampen the mood in the conference. Jeff Mogul started out with the usual summary of the number of papers submitted versus those accepted (149/27), telling us that each paper was reviewed multiple times and that shepherds helped with each accepted paper. Papers that did not meet the format required by the CFP were rejected without review. As OSDI, together with SOSP, is the top venue for publishing refereed operating-system-related papers, hopeful authors are strongly motivated to do much more than adhere to formatting.

Jeff thanked the program committee members and the people and organizations who sponsored OSDI. He pointed out that registration fees do not begin to cover the cost of the conference, but through the work of Robbert van Renesse and USENIX, enough money was raised to pay for registration fees and travel expenses for 72 students, as well as two receptions. Jeff also told us that he had set up osdi2006.blogspot.com so that summaries and comments on papers could be posted in real time during the conference.

Brian Bershad announced the winner of the SIGOPS Hall of Fame award, "Safe Kernel Extensions without Runtime Checking," by George C. Necula and Peter Lee. The two Best Paper awards went to "Rethink the Sync" and "Bigtable: A Distributed Storage System for Structured Data."

I found most of the papers exciting and was busy emailing links to abstracts to friends and acquaintances who I thought would likely be interested (most were). OSDI certainly has become one of my favorite conferences, being full of great information and new ideas.

LOCAL STORAGE

Summarized by Anthony Nicholson

■ ***Rethink the Sync***

Edmund B. Nightingale, Kaushik Veeraraghavan, Peter M. Chen, and Jason Flinn, University of Michigan

Awarded Best Paper

The authors note that asynchronous I/O provides good user-perceptible performance, but it does not provide reliable and timely safety of data on disk. Synchronous I/O provides data safety guarantees but incurs significant overhead. Ed Nightingale presented a new model of "externally synchronous" I/O that resolves this tension by approximating the performance of asynchronous I/O while providing the data safety of synchronous I/O. The main reason synchronous I/O is slow is that applications must block until data has been written safely to disk. The authors argue that only the user, not applications, should be considered "external" to the system. Therefore, under their model, applications need not block on I/O operations but can perform other work while writes are queued. The system only blocks when some output that depends on a pending write is about to be externalized to the screen, disk, or network—in other words, when any event occurs that would make the user think that the I/O has completed.

External synchrony preserves the same causal ordering of writes as synchronous I/O. The fact that unrelated I/O operations can be batched and overlapped without violating causal ordering is what enables the performance wins in their system. Their implementation leverages their prior work (Speculator, SOSP '05) to track causal dependencies across multiple applications and throughout the kernel. "Commit dependencies" inside the kernel track all the processes and objects that are causally dependent on a given pending write. Commit dependencies are forwarded to applications that become tainted by uncommitted data, to ensure preservation of causal ordering. They modified the Linux ext3 file system to support external synchrony, and they compared the performance of their system to native ext3 in both asynchronous and synchronous I/O mode and to ext3 synchronous with write barriers. Their evaluation results show that ext3, even using synchronous I/O, does not guarantee data durability across crashes or power failures, but ext3 with write barriers provides the same data safety of external synchrony, although at a severe performance cost. Their performance on various file system benchmarks shows performance close to that of asynchronous ext3, while providing superior security guarantees to that of synchronously mounted ext3. The performance of ext3 using synchronous I/O is also an order of magnitude slower than that of external synchrony. Their performance on the specweb99 benchmark also shows that external synchrony adds minimal latency overhead as compared to asynchronous file systems.

David Anderson from Carnegie Mellon asked whether there was a corner case where an application would do an asynchronous write, see it failed, and change behavior based on that. Ed responded that in their system, by the time an application discovers a write has failed it would have already moved on, complicating recovery. He argued, however, that "failure notification" usually means kernel panic and crashing owing to hardware failure, so the user has bigger problems in that case. George Candea from EPFL asked about network latency in the mySql benchmark from their paper. Ed said that the client and server were on the same box in their evaluation, because the SpecWeb benchmark was more concerned with network latency. George explained that he was more interested in group commit latency. In that case, Ed said, their system gets the benefit of group commit but wouldn't be able to commit multiple transactions from the same client because of the rules of external synchrony. Micah Brodsky from MIT asked whether performance would suffer under external synchrony for high bandwidth, sequential I/O operations. Ed said you would still see improvement, because the OS wouldn't be blocking between operations if they weren't dependent on each other. Micah wondered how their performance would compare to that of asynchronous I/O for such large sequential operations. Ed noted that asynchronous I/O is limited by the speed to write to memory, and external synchrony would be similarly limited.

■ ***Type-Safe Disks***

Gopalan Sivathanu, Swaminathan Sundararaman, and Erez Zadok, Stony Brook University

Gopalan Sivathanu began by noting that data on disk consists of two things: data and pointers. The structure of pointers on disk implies how disk blocks are organized via higher-level abstractions into files and directories. Unfortunately, today's disks are pointer-oblivious. The file system knows all about the semantics of data and the disk knows about the hardware details. But because the interface between OS and disk is so constrained, little information is exchanged between the two. Everything is just reading and writing blocks. The disk doesn't know the high-level reason for a read/write. For example, it would be nice for a RAID system to prioritize the handling of metadata blocks, because those are more important. Type-safe disks try to bridge this semantic gap. The authors propose an extended interface between the kernel and disk, to enable both type-awareness (disks tracking pointers) and type-safety (disks using pointer information to enforce constraints).

Because type-safe disks are conscious of the relationship between data and pointers, they can do such things as automatically garbage-collect data blocks that are no longer referenced by any file or directory. Offloading these tasks to the disk lets the file system component of the operating system shrink in size and complexity. The authors added additional API calls between file system and disk, such as "allocate block," "create pointer," and "delete pointer." They have implemented a prototype in Linux as a pseudo-device driver and have ported the ext3 and vfat file systems to support TSD. The porting effort was minimal (approximately two person-weeks). Their case study is a security application (ACCESS: A capability conscious extended storage system), which is disk-enforced access control. To access data, an application must provide the disk with a valid capability (an encryption key). Thus the maximum amount of data that can be exposed is that which is currently in use or cached at a higher level. Traditionally, if the OS were compromised, then all data on disk would be compromised. ACCESS establishes a security perimeter on the disk itself instead.

Michael Scott from the University of Rochester asked how TSD would work with a file system that doesn't use the hierarchical pointer design, such as a file system that stores file data in a chain of blocks interconnected by pointers. Gopalan answered that they can support such file systems because the pointers tracked by type-safe disks need not be the same as those maintained by the file system in its own metadata. Margo Seltzer from Harvard asked how this was different from the semantically smart disk work. Gopalan responded that semantically smart disks need to do a lot of operations at the disk level to infer the sort of information that type-safe disks explicitly are given through the API. Margo concluded that the two solutions are basically the same but the implementation cost is different. Gopalan disagreed.

Another questioner asked where private keys for disk blocks can be stored, if we don't trust the operating system that can read application memory. That discussion was taken offline. Emin Gün Sirer from Cornell asked how they settled on the API between the kernel and disk. Why not just move the whole file system into the disk? Gopalan answered that because we often want to run multiple file systems (or none at all, for certain databases) at once, not all functionality can be pushed into the disk. He was not confident that their API was the most minimal interface possible.

Finally, Chad Verbowski from Microsoft Research asked how they would properly keep the parity blocks in a RAID system. Gopalan answered that if one wants to use type-safe disks, then all layers of the file system software stack must be modified, including the RAID software.

■ ***Stasis: Flexible Transactional Storage***

Russell Sears and Eric Brewer, University of California, Berkeley

Rusty Sears stated that systems researchers in particular often abandon off-the-shelf storage solutions because of their poor performance and reinvent the wheel on every project that juggles a large amount of data. The goal of their project was to provide a transactional storage framework that provides good performance off the shelf but is easily extensible to meet the needs of users without requiring that applications be rewritten because they are too tied into the underlying plumbing. This saves users from having to concern themselves with details of logging, recovery, etc., unless they really want to. Stasis has the following three design principles: (1) Provide simple, thin APIs to low-level components. (2) Ensure high-level semantics via local invariants. (3) Make all module interactions explicit—this lets them place policy decisions in replaceable modules.

Russell gave an example of implementing a concurrent hash table on top of Stasis. They wrap operations with Stasis calls so that the underlying layers know how to undo the operation that is being wrapped, in case a transaction needs to be rolled back. Russell also discussed their case study: persistent objects. In other words, these are systems that support transactional updates over a series of objects. To optimize these updates, Stasis can conserve log bandwidth by only logging diffs. Also, they halve memory usage by deferring page cache updates. Low-level modules implement log updates and defer writes to the page cache in order to save memory. They implemented group commit in the log manager, and evaluation shows good performance gains. Since the log manager sees all updates that are produced, it could be extended to transparently implement automatic replication, etc. Stasis can also ensure temporal ordering of certain writes (such as external synchrony). Similar to type-safe disks, the type-system of the disk could be coupled to a type-system of a higher-level application, since these modules are all extensible. The authors are interested in implementing zero-copy I/O for the page file or replicating the page file on multiple servers.

Compared to atop Berkeley DB and SQL, performance is reasonable, with the optimizations described here doubling throughput and halving required memory. There were no questions. The project Web page is http://www.cs.berkeley.edu/~sears/stasis/.

RUNTIME RELIABILITY MECHANISMS

Summarized by Geoffrey Lefebvre

■ *SafeDrive: Safe and Recoverable Extensions Using Language-Based Techniques*

Feng Zhou, Jeremy Condit, Zachary Anderson, and Ilya Bagrak, University of California, Berkeley; Rob Ennals, Intel Research Berkeley; Matthew Harren, George Necula, and Eric Brewer, University of California, Berkeley

Feng Zhou began his talk by noting that many operating systems and applications run loadable extensions. These extensions are often buggier than their hosts and execute in the same protection domain. To address this issue, the authors present SafeDrive, a language-based approach to extension safety. SafeDrive can be decomposed into two principal components: the Deputy source-to-source compiler and the run-time recovery system. Although the principles presented could be applied to other systems, the talk and the paper focused on adding type-based checking and restart capability to existing Linux device drivers. The core idea is to transform code written in C into a safe variant, addressing issues such as out-of-bound array accesses, null terminated strings, and unions.

Previous approaches to retrofitting type safety to C, such as CCured, required the use of fat pointers, which contain both the pointer and bound information. This approach unfortunately requires changing the memory layout of data structures, making the modified extensions incompatible with their host's binaries. Instead of using fat pointers, Deputy relies on the programmer adding lightweight annotations to header files and extension source code. Since SafeDrive relies on run-time checks, it must provide mechanisms to deal with violations. SafeDrive enforces the invariant that no driver code will execute after a failure. The SafeDrive run-time system tracks all kernel resources used by a device driver, using wrappers around kernel API functions. Each tracked resource is paired with a compensation operation that performs an undo when a fault occurs in a driver. As an example, the compensation operation for a spinlock is to release the lock.

Compared to approaches using hardware memory protection such as Nooks, SafeDrive provides finer-grained memory protection. This allows it to catch more errors at compile time and exhibit less run-time overhead.

Andrew Baumann from the University of New South Wales stated that many bugs are concurrency-related and asked whether SafeDrive can detect and recover from such errors. Feng Zhou answered that SafeDrive does not currently address this issue. Brad Karp from University College London asked whether SafeDrive handled integer overflow, specifically the case where a signed integer overflow could result in a different memory allocation than the size requested. Feng Zhou answered that SafeDrive doesn't deal with integer overflow but that, in most cases, memory allocation routines should handle this issue. Rik Farrow asked what happens when programmers make errors writing the annotations. The answer is that the Deputy type system will catch some of the errors. Michael Swift from the University of Wisconsin asked how easy would it be to integrate this with Nooks to allow catching errors that only Nooks is able to catch. Feng Zhou said that it should be feasible to do so. Finally, someone from the University of California, Santa Cruz, asked whether SafeDrive was able to cope with complex data structures. Feng replied that by being able to deal directly with pointers, Deputy is able to do so.

■ *BrowserShield: Vulnerability-Driven Filtering of Dynamic HTML*

Charles Reis, University of Washington; John Dunagan, Helen J. Wang, and Opher Dubrovsky, Microsoft; Saher Esmeir, Technion

Charles Reis began by pointing out that vulnerabilities in browsers are dealt with by patching, but there will often be a long delay between the time a patch is released and its application. This delay opens a dangerous time window when attackers can even use the patch as a blueprint to create exploits. A previous approach, Shield, aims to provide protection equivalent to patching but with easier deployment and roll-back. Shield allows filtering of malicious static content using vulnerability signatures. BrowserShield's goal is to provide similar protection for dynamic Web content by rewriting embedded scripts into safe equivalents on their way to the browser.

BrowserShield consists of a JavaScript library and a logic injector, which could be located at the server, at the firewall, or even as a proxy on the client. Both components are configured using flexible policies that can be tailored to address specific vulnerabilities. The injector performs a first translation which modifies the HTML to remove exploits and wraps all embedded scripts to force them to be invoked via the BrowserShield library. The library performs a second translation during page rendering by dynamically rewriting scripts to access the HTML document tree via an interposition layer. The evaluation shows that, combined with anti-virus and HTML filtering, BrowserShield provides patch-equivalent protection for all 19 vulnerabilities in IE for which patches were released in 2005.

George Candea from EPFL asked about the guarantees that can be provided that pages will be rendered correctly. The answer is that it's much easier to roll back a policy than an applied browser patch when something goes wrong. BrowserShield policy only affects pages, not the browser itself. George then asked how to evaluate the policies. Charles answered that the two metrics are how easy it is to write a policy and how to thoroughly test policies. Jason Flinn from University of Michigan asked whether the scripting was part of the TCB. Charles Reis said it is. Brad Chen from Google asked about specific things that the authors would like to see added or removed from JavaScript. Charles Reiss answered that a smaller API would make it easier to achieve complete interposition. Benjamin Reed from Yahoo! inquired how to debug a policy once deployed. Charles Reis answered that it's important to first distinguish whether a problem is due to BrowserShield or due to the page or browser. A solution would be to render the page in a unprotected browser running in a virtual machine. Benjamin further asked what the core dump should look like. Charles answered that BrowserShield could be enhanced with a set of debugging policies to generate the appropriate information.

Finally, Diwaker Gupta from UCSD asked how to prevent infinite loops in the script translation. Charles answered that if the script had an infinite loop then its interpretation would fall into an infinite loop, but he didn't think that it was possible for a script to force the translation step into an infinite loop.

■ ***XFI: Software Guards for System Address Spaces***

Úlfar Erlingsson, Microsoft Research, Silicon Valley; Martín Abadi, Microsoft Research, Silicon Valley, and University of California, Santa Cruz; Michael Vrable, University of California, San Diego; Mihai Budiu, Microsoft Research, Silicon Valley; George C. Necula, University of California, Berkeley

Úlfar Erlingsson began by introducing XFI, a software-based protection system that provides safe system extensions. He proceeded with a demo of a JPEG image containing an exploit being rendered by both an unmodified JPEG library and an XFIed version of the same legacy library. The first case resulted in an application crash, whereas the XFIed version properly trapped and gracefully aborted the rendering.

The XFI implementation creates safe extensions from existing Windows x86 Portable Executables by performing binary rewriting. The rewriter adds inline guards or short machine-code sequences that perform checks at run-time. Guards are placed on computed control-flow transfers using unique labels as valid target identifiers. A guard verifies that the label is present at the target site before performing a computed jump or call. This mechanism ensures that all transfers remain within the control-flow graph. The rewriter also adds memory access guards to ensure that computed memory accesses lie within valid memory regions. The validation uses a fast path when the address lies within bounds established at load time. Memory accesses outside of these bounds fall through a slow path that validates the address using data structures similar to page tables. The binary is also modified to use two stacks, a scoped stack and an allocation stack. The scoped stack contains return address and variables that are accessed by name only. The allocation stack contains data that can be accessed using computed access. This mechanism allows the integrity of the scoped stack to be established through static verification.

Because rewriting binaries is a tricky process, XFI doesn't require this step to be trusted. Instead, XFI relies on a trusted verifier to parse the binary at load time, ensuring that binaries have the appropriate structure and the appropriate guards. The verifier is simple, fast, and consists of only 3000 lines of straightforward code, mostly x86 decoding tables, increasing confidence in its correctness.

Bryan Ford from MIT asked how XFI ensures that the heap is not incorrectly seen to contain matching identifiers. Úlfar answered that a guard checks whether computed control flows are to targets within the binary, in addition to matching identifiers and the verifier ensuring that identifiers are unique. Jim Lawson from MSB Associates asked whether XFI rejects self-modifying code. Úlfar acknowledged this. Rik Farrow asked about binary size increase. Úlfar answered that code size can increase by a factor of two or more but that most of the additional code is either in nonexecutable verification hints or in out-of-band trampolines, which are not invoked often. Therefore the increase in code size has a minimal impact on performance and i-cache behavior.

OS IMPLEMENTATION STRATEGIES

Summarized by Andrew Miklas

■ *Operating System Profiling via Latency Analysis*

Nikolai Joukov, Avishay Traeger, and Rakesh Iyer, Stony Brook University; Charles P. Wright, Stony Brook University and IBM T.J. Watson Research Center; Erez Zadok, Stony Brook University

Nikolai presented a new approach to operating system profiling. He showed that the logarithmic distributions of an OS's latencies can reveal most, if not all, aspects of the OS's internal operation. He presented several methods to analyze these profiles and provided interesting examples of the sorts of conclusions that can be drawn using the profiler data. Among others, the authors were able to diagnose a locking error in the Linux kernel without having to examine the source code. Also, because the latency can be measured entirely outside of the kernel, the method can be used to analyze operating systems even if the source is not available.

The basic technique involves repeatedly measuring the times required to complete OS requests. The latency of each request depends on the path taken through the code and interactions with other processes. Therefore, the distribution of these values can be used to make inferences about an operating system's inner workings. The latencies are used to generate histograms that can be analyzed either visually or by a provided automatic data-analysis toolset. Various events, such as being blocked on a lock, show up as spikes on the histogram. Correlations between different requests can reveal their contention on a shared resource. For example, similar spikes on the histograms of two different system calls that only appear when the two are run together suggest that they contend on the same lock.

Although the profiler can be used without any kernel changes whatsoever, it can take advantage of kernel instrumentation points if they are available. Loadable drivers can also be used as vantage points from which to measure latencies. The authors created a tool that can patch Linux file systems to automatically produce latency data. The presented profiler adds negligible overheads. The generated profiles are usually smaller than 1 kilobyte. The profiler adds less than 200 CPU cycles per profiled call, whereas existing profilers add overhead per profiled event. For example, if a system call is trying to acquire several semaphores and to perform I/O, existing profilers would add overhead for each such event. As a result, the presented profiler is efficient. As measured with the Postmark benchmark, the system's CPU time was affected by less than 4%.

Marcel Rosu of the IBM T.J. Watson Research Center asked whether the technique makes heavy use of CPU cycle counters in order to cheaply measure intervals of time and if they tested their system on CPUs that use variable clocks, such as those found in notebook computers. Although they didn't test on variable-clock CPUs, "it should be possible," Nikolai said, "to simply apply a scaling factor to handle the cases where the CPU isn't running at its maximum frequency. Also, remember that our system doesn't rely on using CPU cycle counters. We could use an off-CPU high-precision timer if the CPU lacked an appropriate method of measuring intervals." Brad Chen of Intel asked whether the profiler can be used to analyze anomalous cases, rather than just bad implementations of the main case. The paper describes a "sample profile," where instead of folding all the latencies for a given test together, these were separated based on fixed time intervals. Nikolai's group used this method to analyze ReiserFS's performance during the relatively infrequent periods of time when the Linux buffer flushing daemon bdflush was active.

■ *CRAMM: Virtual Memory Support for Garbage-Collected Applications*

Ting Yang and Emery D. Berger, University of Massachusetts Amherst; Scott F. Kaplan, Amherst College; J. Eliot B. Moss, University of Massachusetts Amherst

Ting Yang began by saying that the memory management component of an OS and the heap-management component of a garbage-collected (GC) run-time environment must cooperate with each other to ensure good performance under all memory loads. Today's operating systems don't provide enough VM sophistication to support GC run-times effectively when memory pressures are significant. As a result, current run-times are reactive; they only resize the heap once performance has degraded. In contrast, CRAMM is predictive; it can determine the best heap size and suggest that the run-time adjust it before the system begins to experience performance loss.

CRAMM consists of two components: an extension that sits in the kernel's VM subsystem, and a heap-size model that exists in the run-time environment. The kernel-mode component tracks each process's working-set size (WSS): the amount of memory that the process needs so that it does only a small amount of swapping. The model component computes the heap size that would cause the process's WSS to just fit within the maximum amount of memory the operating system is willing to allocate. If the current heap size and the computed optimal heap size differ, the model asks the run-time to adjust the size of the heap. The system can therefore quickly respond to changes in memory pressure by reducing the size of the heap before it is swapped out, avoiding reclamation-triggered thrashing. CRAMM adds only about 1–2.5% overhead during ordinary test runs where memory is plentiful. In exchange, CRAMM dramatically improves the performance in situations where memory pressure suddenly increases.

Chris Stewart, University of Rochester, asked whether the authors assumed that when using copying garbage collectors, the number of pages used to hold "copied survivors" does not change rapidly and wondered whether their system therefore would be able to handle flashloads when the run-time uses copying GCs. Ting explained that they keep track of the CS value from one invocation of the GC to the next and apply a smoothing function. The maximum value ever seen for the CS value is weighted more heavily to ensure that they don't underestimate this parameter.

- ***Flight Data Recorder: Monitoring Persistent-State Interactions to Improve Systems Management***

Chad Verbowski, Emre Kıcıman, Arunvijay Kumar, and Brad Daniels, Microsoft Research; Shan Lu, University of Illinois at Urbana-Champaign; Juhan Lee, Microsoft MSN; Yi-Min Wang, Microsoft Research; Roussi Roussev, Florida Institute of Technology

Chad presented the Flight Data Recorder (FDR), a new tool which will ship with Windows Vista that allows all changes to the persistent state of a system to be logged for later analysis. He explained that various system management tasks that up until now have been something of a black art essentially reduce to queries over the logs gathered by the FDR. As a motivating example, Chad told of a server at Microsoft that would exhibit extremely poor performance every few weeks. A system administrator eventually determined that this was because the system's page file was being inappropriately shrunk. Unfortunately, he was unable to determine why this was happening. The best he could do was to send out email to the other admins asking them to make sure they weren't resizing the file. However, after running the FDR for a few weeks, the logs were used to quickly pinpoint the offending script.

Currently, system management tools break down into roughly three categories. Some use a similar logging approach but, owing to space constraints, activate only on demand. These types of tools are of limited usefulness when trying to determine why a particular piece of configuration data changed. Another class of tools uses signatures to look for known-bad configurations. However, creating signatures general enough to be useful across a wide variety of machines is a time-consuming process. Finally, manifest-based approaches require that applications provide the system with a list of all configuration dependencies. However, the authors point out that the uninstall tools included in many software packages leave behind files and configuration data, suggesting that building complete manifests is impractical.

The proposed approach simply logs all changes to the system's persistent state. The main contribution of the work is its novel method of encoding the activity logs. This method requires on average just 0.5–0.9 bytes per event. Since typical systems generate on the order of 10 million events per day, the resulting logs are small enough to be practically sent over the network, archived, correlated with other systems, and quickly queried. The authors report that they are able to execute common queries against a day's worth of stored data in as little as three seconds. They also note that it should be possible to serve as many as 5000 systems running the FDR with a single archive server.

The authors see the FDR as being useful not only for system management but also for ensuring that various security and management policies are being followed. For example, the logs captured by the FDR can be used to determine how often a locked-down production server is modified without proper approval. The FDR can also be used to assist in locating system "extensibility points": configuration settings that control the loading of extra system services or plug-ins. This has important implications for detecting and removing malware. In summary, the FDR has made it possible to know about everything that is happening on a system.

Q: Can the FDR be used to predict how a system will respond to a configuration change? A: We can search for another system that already has the configuration change but is otherwise similar to the machine in question. If we can find such a machine, we can use it to approximate how this system would behave with the change. Q: What is the right way to query the logs generated by the FDR? Should we be using SQL, or perhaps a customized query engine with a programmatic API, etc.? A: Right now, we just expose the raw tables as they are in the log file. Any special-built query engine should be optimized for queries of the form "What files have been changed since time T?" since the most common requests seem to be those that look for modified configuration entries. Q: Other types of events might be worth logging. For example, did you consider logging all socket activity? A: We did think about logging IPC activities, but the system doesn't currently implement this feature.

WORK-IN-PROGRESS REPORTS (WIPS)

Summarized by Andrew Miklas

■ ***Taking the Trust out of Global-Scale Web Services***

Nikolaos Michalakis

If you were to contract out the hosting of a dynamic Web site to a number of content delivery networks, how could you be sure that every system serving your customers was running the exact software you supplied to the CDNs? This problem becomes even more severe for content dynamically generated by ordinary Internet users, where contract law might not be sufficient motivation to ensure that the distributors don't behave maliciously.

Nikolaos is researching ways to certify that dynamic content is served correctly in an environment where the delivery systems are not fully trusted by the content providers. The basic design has clients forward a fraction of the signed responses from one server to other replicas for verification. If the verifying replica computes a different result, it publishes the erroneous signed response so that other hosts can learn of the misbehaving server.

■ ***Information Flow for the Masses***

Max Krohn, Micah Brodsky, Natan Cliffer, M. Frans Kaashoek, Eddie Kohler, Robert Morris, and Alex Yip

Today's Web sites are serving an increasing amount of user-contributed content. They are no longer places where users go to passively download information, but instead they act as meeting points where people can exchange content. For example, consider Wikipedia, which is built on content contributed by its users.

However, sites today do not allow users to contribute to the applications running on them. Wikipedia does not allow anonymous users to patch up the MediaWiki software running on the live servers, as it does with its content. The main reason why this can't be allowed in today's hosting environments is security; a malicious patch could be used to leak ordinarily inaccessible information.

Traz is a novel Web-hosting environment that applies Asbestos-like reasoning to Web servers. User-contributed code may be allowed to manipulate data ordinarily inaccessible to the contributing user, but it will not be allowed to leak this information back to that user. Traz therefore allows Web sites to safely execute user-provided code against privileged information. Traz runs on ordinary commodity operating systems and allows developers to write their applications using any programming language.

■ ***AutoBash: Hammering the Futz Out of System Management***

Ya-Yunn Su and Jason Flinn

We've all experienced it: a system that isn't performing quite the way we'd like. Maybe the video hardware doesn't set the appropriate resolution when plugging an external monitor into a notebook. Perhaps the wireless card doesn't reassociate with the nearest access point correctly on wake-up. No matter what the problem, we typically use the same approach to solve it: Type the symptoms into Google, find a page that describes a fix, apply the steps described in the fix, and check to see whether the problem is corrected. If not, back out the changes, and repeat. This process, which Ya-Yunn termed "futzing," requires a substantial amount of manual intervention and can lead to serious frustration.

AutoBash is a tool that automates the futzing process. When the user notices a configuration error, he or she describes the symptoms to the tool. AutoBash will search its database for scenarios where a user started with the current configuration, applied some changes, and ended up with a new configuration that satisfies the desired description. Once a record is found, the steps required to adjust the user's configuration will be automatically replayed. If the user is dissatisfied with the result, he or she will be given the opportunity to have the changes automatically rolled back and will possibly be presented with another solution. Finally, should the tool be unable to automatically correct the error, it will watch as the user does so manually, so that other users may benefit from the futzing done by this user.

■ ***Dynamic Software Updating for the Linux Kernel***

Iulian Neamtiu and Michael Hicks

Software updates are a necessarily evil. In order to apply them, a service must typically be restarted. For many applications, the downtime that must be incurred in order to restart is undesirable. Worse, updates to system software such as the kernel can necessitate a full reboot of the machine, resulting in an even longer stretch of downtime.

Gingseng, a tool worked on by the presenter, can apply updates to running user-mode services. It does this by determining strategic locations in a service's execution where the code can be safely updated. Patching kernel code, however, is far more difficult, owing to its low-level and highly concurrent nature. Iulian described some of his work to make Ginseng able to update a live Linux kernel.

■ ***iTrustPage: Preventing Users from Filling Out Phishing Web Forms***

Troy Ronda and Stefan Saroiu

The U.S. economy loses billions of dollars each year to phishing attacks. Even worse, phishing erodes the public's trust in the Web as a platform for e-commerce. Troy claimed that many forms used to legitimately gather information originate from well-established Web sites, whereas phishing attacks are usually done from newly created sites. Fortunately, there are a number of services that can be used to estimate the popularity and thus the trustworthiness of a site. Phishing pages must also appear similar to their targets. Although it is difficult to design an algorithm to compare two Web pages, Troy mentioned that a person can usually determine with ease if one Web site is mimicking another.

These two key observations form the basis of iTrustPage, a Firefox extension that helps users avoid phishing attacks. When a user tries to fill out a form on a page that is not well established, iTrustPage stops the user from proceeding and asks the user to describe the task that he or she is trying to accomplish. Using this description, iTrustPage makes a query to Google and shows the user the Web sites associated with the first few hits. The user then indicates which Web site looks most similar to the page the user was expecting. Finally, the tool redirects the user to the organization's legitimate Web page and away from the phishing attempt. iTrustPage is currently available for download at http://www.cs.toronto.edu/~ronda/itrustpage/.

■ ***Failures in the Real World***

Bianca Schroeder and Garth A. Gibson

A major challenge in running large-scale systems is that component failure is the norm rather than the exception. Unfortunately, most work on dealing with failures is based on simplistic assumptions rather than real failure data. Bianca has been collecting and analyzing failure data from several real-world installations. The initial results indicate that many commonly used failure models are not supported by real data. For example, the probability of a RAID failure can be an order of magnitude larger, based on actual observation, than one would expect given the standard model, which uses exponentially distributed intervals between failures.

Motivated by these initial results, Bianca is continuing to collect and analyze failure data from a large variety of real-world installations. By carefully grounding new failure models in real data, researchers will be able to more accurately model a system's response to component failure.

■ ***Dynamically Instrumenting Operating Systems with JIT Recompilations***

Marek Olszewski, Keir Mierle, Adam Czajkowski, and Angela Demke Brown

Operating systems, like applications, grow more complicated each year. Unfortunately, the techniques and tools used to instrument, trace, and debug kernel-level code have not advanced as quickly as their user-mode counterparts. For example, tools such as Valgrind have made it possible to inject instrumentation into running user-mode code using JIT recompilation techniques. Because these probes are dynamically compiled directly into the surrounding code, they perform much better than traditional patch-and-redirect techniques. Unfortunately, JIT instrumentation tools are currently unable to instrument kernel code.

Marek plans to create JIT instrumentation tools that can be used with kernel code. Given the time-sensitive nature of much of a kernel, this approach may make it possible to instrument code that previously could not be probed for performance reasons. By bringing the proven benefits of JIT instrumentation to the kernel, Marek will assist systems programmers in better understanding their operating systems and ultimately will help them to produce more efficient and correct kernels.

■ ***Pattern Mining Kernel Trace Data to Detect Systemic Problems***

Christopher LaRosa, Li Xiong, and Ken Mandelberg

Profilers, debuggers, and system call tracers can all be used to diagnose performance issues within a process. However, diagnosing performance problems that result from the interplay of two or more processes can be complicated. For example, determining why an X server is exhibiting poor performance can involve gathering and correlating traces from both the X server and any active X clients. Unfortunately, there are few tools to automatically correlate traces, and programmers must usually resort either to poring over the gathered data by hand or writing ad-hoc scripts.

Christopher plans to apply data-mining techniques to system-wide activity traces. Using these techniques, anomalous conditions that might impact system performance can be automatically detected and isolated, even if they span multiple processes. He provided an example involving a stock-ticker toolbar applet that unnecessarily flooded the X server with requests. His trace analyzer was able to automatically detect the excess of X calls and pinpoint their origin. He would appreciate it if DTrace or LTT users would share their hard-to-find bugs with him, so that he can test the effectiveness of his system's automatic detection.

■ ***Spectrum: Overlay Network Bandwidth Provisioning***

Dejan Kostić

Overlay networks are currently used to efficiently disseminate content. However, because of their decentralized nature, it can be difficult to ensure that there is enough outbound capacity to support all receivers as well as prevent other overlays from "stealing" bandwidth from higher-priority services. This presents a serious problem when overlays are used to transfer streaming media, where timely delivery of content is necessary for the system to operate correctly.

Dejan is working on algorithms to measure and disseminate bandwidth availability information throughout an overlay network. By doing so, the system can make globally optimal decisions about how much bandwidth to dedicate to a media stream. This is especially useful when the same overlay is carrying a variety of different content. For example, a BitTorrent–like transfer through the overlay might be permitted as long as it doesn't cause anyone's video stream to drop below a certain bit-rate.

■ ***An Infrastructure for Characterizing the Sensitivity of Parallel Applications to OS Noise***

Kurt B. Ferreira, Ron Brightwell, and Patrick Bridges

Many commodity operating systems do not scale well to the number of processors found in today's supercomputers. When running such operating systems, as much as 50% of the system's performance can be used by the operating system itself. For this reason, many of today's largest supercomputers run stripped-down operating systems that impose as small an overhead as possible.

Kurt's research seeks to understand exactly how this overhead, termed "OS noise," affects the bottom-line performance of various scientific computing applications running on large supercomputers. He is also interested in finding ways to reduce the overheads found in ordinary operating systems in order to make them more suitable for use on large supercomputers.

■ ***Limits of Power and Latency Reductions by Intelligent Grouping***

David Essary

Disk accesses are a very expensive operation; entire classes of applications are limited by the I/O capability of their hosts, rather than its raw processing power. Improving an I/O subsystem's ability to quickly respond to requests can greatly improve the overall efficiency of such systems.

David's research seeks to improve storage access time and throughput by carefully controlling how the data is physically laid out on disk. Data can even be stored on multiple drives in an array to give the reading process more flexibility when deciding how to optimally read the data back. These techniques can result in a 70% reduction in disk-related latencies and energy consumption. David also discussed some of his work on predictive retrieval algorithms and how he had explored theoretical limits. Finally, he pointed out that his work to reduce seek operations not only improves performance but also reduces drive power consumption.

■ ***Distributed Filename Look-up Using DNS***

Cristian Tapus, David Noblet, and Jason Hickey

One of the challenges in building a distributed file system is finding a way to locate the data and metadata associated with files. Usually, this information is replicated to provide reliability and thus might be distributed across a wide-area network. A centralized directory service is undesirable because it provides a single point of failure and may behave as a bottleneck for the system.

Cristian noted that many of the problems faced when serving file metadata and data are in fact the same as those solved by DNS. For example, both DNS and FS metadata are used to resolve names in a hierarchical name space to addresses. For these reasons, Cristian suggested using DNS itself as a localization service for the data and metadata of files in a distributed FS. Looking up a file would involve making a DNS query for a name such as "passwd.etc.mojavefs.caltech.edu." The query would return the addresses of the replica file servers that could serve the named file. Replication of the metadata is handled automatically by the caching mechanisms built into DNS.

■ ***Bounded Inconsistency BFT Protocols: Trading Consistency for Throughput***

Atul Singh, Petros Maniatis, Peter Druschel, and Timothy Roscoe

Many protocols exist for ensuring the high availability of a system despite the potential for byzantine failure of its components. However, these protocols have negative scaling properties: The more nodes added, the higher the performance penalties associated with keeping all of the components synchronized.

Atul proposed a solution where replicas return results that differ slightly from the correct result. The key is that the variability of the response is bounded; a client can be sure the true value is within some range of the returned quantity. By allowing the replicas to run slightly out of sync, the overall performance of the system can be improved. This approach can be useful for applications that don't require precise results. For example, it might be acceptable for a disk quota system to allow a user to consume at most 5% more disk space than the allotted quota.

■ ***EyesOn: A Secure File System That Supports Intelligent Version Creation and Management***

Yougang Song and Brett D. Fleisch

Versioning file systems are often used to enhance the security capabilities of an operating system. Since they preserve the change history for each file, they can help system administrators both detect intrusions and roll back unauthorized changes. However, maintaining a comprehensive change history can become overwhelmingly expensive in both disk space and performance overhead.

The EyesOn system aims to preserve normal file operations and existing file structures while leaving the complexity of recovery operations to the time they are requested. EyesOn extends the same strategy used by file system journaling to record its in-memory modified data in a log without significant additional processing. EyesOn uses these logs to create file versions that can be used to accelerate the retrieval of a file's change history. Versions are created based on user-supplied predicates that can make use of statistics stored in the log. For example, predicates can use the elapsed time since a file was last modified, the total size of the change, or whether the user has explicitly requested that a snapshot of the file be taken at this time. Two types of versions are created in EyesOn. Normal versions are created for quickly retrieving recent changes and will automatically be culled once they reach a certain age. Landmark versions are created for keeping valuable information for a longer time.

■ ***Robust Isolation of Browser-Based Applications***

Charles Reis

First it was online email. Next came online scheduling, photo archiving, and journaling. Today, companies have begun testing online word processors and spreadsheet applications. Eventually, it's possible that most applications will be run on racks of systems in far-away data centers and served over the Web.

If the future of applications is the Web, than in some sense the future of operating systems is the Web browser. Although they may never directly interact with hardware, they certainly will fulfill other roles traditionally handled by an operating system. For example, Web browsers should ensure that a buggy or malicious script on one site doesn't adversely affect the scripts of another. Browsers should also protect the locally stored data associated with one site from unauthorized access by scripts from another site. Charles is currently looking at ways to build these types of containment mechanisms into browsers such that changes to the server-side applications are kept to a minimum.

■ ***Stealth Attacks on Kernel Data***

Arati Baliga

Rootkits use an array of impressive techniques to hide themselves from detection. Some go so far as to rewrite portions of the in-memory kernel image to perfect the illusion. New system calls might be added to render the rootkit's processes invisible to ps. Others carefully manipulate the process lists and file system handlers to evade detection.

Arati is investigating all of the ways in which rootkits can tamper with the running kernel image by solely manipulating kernel data. She hopes to use her findings to develop monitoring systems that can't be easily fooled. In particular, she is looking at attacks that do not employ conventional hiding techniques yet are able to cause stealth damage to the system and evade detection from state-of-the-art integrity monitoring tools.

PROGRAM ANALYSIS TECHNIQUES

Summarized by Geoffrey Lefebvre

■ ***eXplode: A Lightweight, General System for Finding Serious Storage System Errors***

Junfeng Yang, Can Sar, and Dawson Engler, Stanford University

Junfeng Yang began his talk by noting that storage systems errors are the worst kind of error since they can result in corruption of persistent state, possibly leading to permanent loss of data. To address this issue, the authors presented eXplode, a system to find errors in storage systems. eXplode borrows ideas from model checking by being comprehensive, but instead of running the checked system inside a model checker, eXplode runs client-defined checkers inside the checked system. Running on a real system allows it to easily check storage stacks. A file system checker can be used to find errors in storage layers either above or below the checked system. Another advantage of this approach is that checkers are easy to write. A checker for a storage system can be written in less than 200 lines of C++ and often consists of a simple wrapping layer around existing utilities such as mkfs and fsck. The authors state that this work completely subsumes their previous work (FiSC).

Because bugs are often triggered by corner cases, eXplode's core idea is to explore all choices. eXplode provides choose(N), an N-way fork that allows checkers to fork at every decision point during testing and explore every possible operation. Before exploring a decision point, eXplode checkpoints the state of the system. A checkpoint is simply the recorded sequence of return values from choose(). To restore a checkpoint, eXplode deterministically replays this sequence from the initial state. These two mechanisms allow eXplode to perform an exhaustive state exploration. At any point during testing, a checker has the ability to force crashes. Upon a forced crash, eXplode generates crash disks based on all possible orderings of dirty buffers. A checker-supplied routine then tests all disks for specific invariants.

A challenge faced by the authors was dealing with nondeterminism. The choose() primitive could be called by nonchecking code such as interrupt handler. eXplode solves this problem by filtering on thread ID. eXplode must deterministically schedule all threads involved with the checked system. This includes the checker's thread but also all threads belonging to the checked storage system. By playing with thread's priorities, eXplode is able to enforce deterministic scheduling most of the time and can detect when it fails to do so.

Junfeng then presented an evaluation of eXplode. The authors tested various file and storage systems such as ext2, ext3, JFS, the VFS layer, NFS, Berkeley DB, and VMware with eXplode and found bugs in all of them. Someone from UCSD asked about the option not to replay instrumented kernel functions and if doing so could lead to nondeterminism. Junfeng answered that short traces were deterministic, since calls to these kernel functions succeed most of the time, but that long traces had nondeterminism. Someone asked whether it was possible to test subcomponents of file systems. Junfeng answered that normally filesystem implementations were not structured that cleanly, so it makes more sense to check a file system as a whole. Another person asked how eXplode handled nondeterminism created by disk actually performing an operation out of order internally. Junfeng answered that eXplode uses a RAM disk, which avoids this problem.

■ *Securing Software by Enforcing Data-Flow Integrity*

Miguel Castro, Microsoft Research; Manuel Costa, Microsoft Research Cambridge; Tim Harris, Microsoft Research

Manuel Costa began his presentation by noting that most of the software in use today is written in C++. This body of software has a large number of defects and there exist many ways to exploit these defects, such as corrupting control data. He presented some of the various approaches to securing software. He noted that removing or avoiding all defects is hard and that although it is possible to prevent attacks based on control-data exploits, certain attacks can succeed without compromising control flow. Approaches based on tainting can prevent noncontrol-data exploits, but they may lead to false positives and have a high overhead.

To address these issues, the authors present a new approach to secure software based on enforcing Data-Flow Integrity (DFI). Their approach uses reaching definition analysis to compute a data-flow graph at compile time. For every load, compute the set of stores that may produce the loaded data. An ID is assigned to every store operation and, for each load, the set of allowed IDs is computed. The results of the analysis is used to add run-time checks that will enforce data-flow integrity. Stores are instrumented to write their ID into the run-time definition table (RDT). The RDT keeps track of the last store to write to each memory location. Loads are instrumented to check whether the store in the RDT is in their set of allowed writes. If a store ID is not in the set during a check, an exception is raised. Because the analysis is conservative, an exception guarantees the presence of an error. There are no false positives. Manuel also noted that control-flow attacks are a form of data-flow attacks. It is possible to use the same mechanism to protect control-flow data.

Manuel described a set of optimizations to improve the run-time performance. The most important optimization is to rename store IDs so that they appear as a continuous range in the definition set. The membership test can be replaced with a subtraction and a compare. The evaluation presented demonstrates that this optimization is fundamental to the performance of DFI. On average, DFI imposes a space overhead of around 50%, and the run-time overhead ranges from 44% to 103%.

Brad Karp from University College London asked whether DFI handles function pointers and other complex program constructs. Manuel answered that DFI handles function pointers but has problems with certain pieces of software written in assembly. These problems can result in not detecting certain DFI violations. An alternative would be to use CFI in this case. It is important to understand that these limitations are a property of the program to instrument, not the attack. Bill Bolosky from Microsoft Research asked whether every store had to be instrumented. Manuel acknowledged that this was the case.

- ***From Uncertainty to Belief: Inferring the Specification Within***

Ted Kremenek and Paul Twohey, Stanford University; Godmar Back, Virginia Polytechnic Institute and State University; Andrew Ng and Dawson Engler, Stanford University

Ted Kremenek began by saying that all systems have correctness rules, such as not to leak memory, or to acquire some lock before accessing data. We can check these rules using program analysis, but the problem is that missed rules lead to missed bugs. The specification of a system is the set of these rules and invariants. The problem addressed in this talk is how to find errors that violate these rules when we don't know what the rules are or, more precisely, how we can infer the specification. The talk presented a general framework to do so and the technique is described using an example: finding resource leaks by inferring allocators and deallocators.

There are many sources of knowledge that can be used to infer system rules, such as behavioral tendencies (i.e., programs are generally correct) and function names, but there isn't a way to bind all of this information together. To address this issue, the authors present an approach based on the Annotation Factor Graph (AFG), a form of probabilistic graphical modeling. This approach reduces all forms of knowledge, either from evidence or intuitions, to probabilities. The idea is to express program properties to infer as annotation variables and infer these annotations by combining scores obtained from factors. Factors represent models of domain-specific knowledge and are used to score assignments of values to annotation variables based on the belief that an assignment is relatively more or less likely to be correct.

The talk focused on how to infer resource ownership using AFGs. First, functions return values and parameters are annotated. The domain for a return value annotation variable is to return or not return ownership, and the domain for a function parameter is to claim or not claim ownership. Ted Kremenek then described some of the factors used to infer resource ownership. Some use static analysis based on common programming axioms such as that a resource should never be claimed twice; others are based on ad-hoc knowledge such as function names.

The evaluation was based on inferring annotations on five projects: SDL, OpenSSH, GIMP, XNU, and the Linux kernel. The authors' technique obtained a 90%+ accuracy on the top 20 ranked annotations. It was also able to infer allocators unknown or misclassified by Coverity Prevent. Using their technique, the authors found memory leaks in all five projects. Ted Kremenek described a complex memory leak they were able to find in the GIMP library.

Someone from Yahoo! asked how well the tool performs when allocators that are simply wrappers around malloc are factored out. Ted answered that although many allocators call malloc, the tool doesn't use this knowledge. Brad Chen from Google asked whether the tool would be confused by realloc. Ted answered that the tool did infer that realloc was both an allocator and a deallocator. Corner cases, such as this one, are detected automatically but have to be dealt with separately. They had a similar issue with certain functions in the Linux kernel. Such outliers have to be modeled explicitly but are fairly rare. Micah Brodsky from MIT asked whether AFGs are an instance of Bayesian Net. Ted answered that AFGs and Bayes Nets belong to the same family of probability modeling. The authors actually started with Bayes Net but found the class to be too rigid. AFGs were much easier to work with. Someone asked what other things could be modeled. Ted answered that locks are very behavioral, so their approach would work well. He stated that the temporal relationship could be expressed as a grammar.

DISTRIBUTED SYSTEM INFRASTRUCTURE

Summarized by Anthony Nicholson

■ *HQ Replication: A Hybrid Quorum Protocol for Byzantine Fault Tolerance*

James Cowling, Daniel Myers, and Barbara Liskov, MIT CSAIL; Rodrigo Rodrigues, INESC-ID and Instituto Superior Técnico; Liuba Shrira, Brandeis University

The authors address the problem of building reliable client-server distributed systems. They note that the current state of the art either requires a small number of replicas (3f+1) but has high communication overhead or reduces communication complexity by requiring a much larger number of replicas (5f+1). The second case still suffers from degraded performance in cases of write contention, however. They propose a hybrid scheme that combines the best aspects of both schemes to achieve a low number of replicas (3f+1) while bounding communication overhead.

Their system uses a two-phase write protocol. First, a client obtains a timestamp grant from each replica. This grant is essentially a promise to execute the given operation at a given sequence number, assuming agreement from a quorum of replicas. In the second phase, the client forms a certificate from 2f+1 matching grants and sends this certificate to all the replicas, which then complete the write operation. A certificate proves that a quorum of replicas has agreed to a given ordering of operations. Importantly, the existence of a certificate precludes existence of conflicting certificate. Replicas are forbidden to have two outstanding grants in progress, and they return the currently outstanding grant to clients while a grant is in progress, as proof that it is busy. The authors have deployed both their Hybrid Quorum (HQ) and BFT prototypes on Emulab. Their results show that HQ performs better than BFT up until around 25% contention.

Atul Adya from Microsoft Research noted that their protocol allows clients to commit operations on behalf of other clients, and asked whether this was a security hole. James said that since certificates are free-standing and cryptographically signed, any client can send a certificate to a replica and it will commit faithfully on behalf of the originating client. Bill Blaskey, also from MSR, noted that commit could be painful because potentially thousands of operations occur per second. James noted that all data lives completely in RAM in their experiments, and a reboot is considered a failure. Petros Maniatis from Intel Research asked why the authors chose to do a two-phase protocol with 3f+1 replicas, rather than a one-phase protocol with 5f+1 replicas. James noted that they could have done so but decided that 5f+1 is just too many. The last question involved whether colluding replicas would just always tell clients that they had an outstanding grant, to force the slow path of the protocol. James responded yes, in the worst case this would happen.

■ *BAR Gossip*

Harry C. Li, Allen Clement, Edmund L. Wong, Jeff Napper, Indrajit Roy, Lorenzo Alvisi, and Michael Dahlin, University of Texas at Austin

The motivation scenario for talk is a live, peer-to-peer streaming media application. Such applications can fall apart, however, in the presence of malicious nodes, if it is not in a node's self-interest to forward a data packet on to its peers. The authors introduce the concept of BAR Gossip, a gossip protocol that makes it in the interest of selfish nodes to act for the benefit of the overall system, while detecting and punishing byzantine (malicious) nodes. BAR Gossip is in fact two protocols: balanced exchange and optimistic push.

During balanced exchange, at each round every node selects one partner, the partners exchange their transaction histories, and then they trade equal numbers of data updates. Clients must commit to their histories before discovering their partner's history; similarly, they must send the data updates first in encrypted form before subsequently sending the key. This makes bandwidth a "sunk cost," so it is illogical for selfish nodes to withhold the data key from their partner later on. The authors also introduce the notion of verifiable partner selection to prevent clients from talking to more than one client per round (to accumulate more updates).

The second part of BAR Gossip is optimistic push, which handles bootstrapping for new nodes. In this case, unequal numbers of updates may be exchanged, but the lesser peer must sacrifice the same amount of energy and bandwidth by sending junk to even out the exchange. Their simulation results show that following the protocol was the most beneficial strategy for clients in all cases.

Rob Sherwood from Maryland asked about cases where clients consider different weights for inbound and outbound traffic. Harry argued that such asymmetry can be handled by sizing key requests accordingly. Rob responded that in cases where one is downloading illegal content, it might be overwhelmingly important never to upload anything. Harry countered that in such extreme cases BAR Gossip might not work. Jim Liang from UIUC noted that the authors assume that all clients know the complete membership list. Harry said that they are currently looking at handling cases where clients have only partial membership information. Jim further asked how the BAR Gossip protocol achieves low latency for streaming media. Harry said their experiments showed an average latency of 20 seconds for a live video stream. This compares favorably with existing streaming media applications (such as the free NCAA Final Four video feed). Another questioner asked how their protocol handles collusion. Harry replied that they have no explicit mechanism for this, because handling collusion in game theory is difficult. Their results showed, however, that BAR Gossip is robust for small colluding groups (with up to 30% nodes colluding). The last question concerned the scalability of BAR Gossip. Harry cited three limitations in scaling their system: (1) handling dynamic membership churn, (2) handling nodes with only partial membership information, and (3) locality-aware partner selection. They are currently looking at all three issues.

■ ***Bigtable: A Distributed Storage System for Structured Data***

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber, Google, Inc.

Awarded Best Paper

Mike Burrows described Bigtable, a storage system designed for in-house use at Google. Bigtable developed in response to a need to store information on over 10 billion URLs, with wide variety in the size of objects associated with a given URL, and variety in usage patterns. Commercial databases are obviously unsuitable owing to the scale (petabytes of data on billions of objects, with thousands of clients and servers). Bigtable stores data in a three-dimensional, sparse sorted map. Data values are located by their row, column, and timestamp.

Since these tables can be quite large, Bigtable allows dynamic partitions by row, called tablets, which are distributed over many Bigtable servers. Clients can manage locality by choosing row keys in such a way that data that should be grouped together achieves spatial locality. A given tablet is owned by one server, but load balancing can shift tablets around. Similarly to tablets, locality groups are partitions by column instead of row. These locality groups, however, segregate data within a tablet. All data is stored as files in the Google File System (GFS), with different locality groups stored as different files in GFS. One master Bigtable server controls the many tablet servers. Clients access a tablet by requesting a handle from the Chubby lock service (described in another OSDI talk), then doing read/write directly with the tablet server that owns a given tablet. The client only talks to the master when it needs to manipulate metadata (e.g., create a table or manipulate ACLs). Currently, Bigtable is deployed on over 24,000 machines in approximately 400 clusters and is used by over 60 projects at Google.

The first question concerned the poor random read performance described in the paper and noted that the authors attributed this to shared network links. The questioner added that it seems an easy optimization to move tablet servers closer to the GFS storage that the tablets actually reside on. Mike answered that, by default, if a GFS server is collocated with the tablet server, the tablet's data will be stored on that GFS server. Atul Adya from Microsoft Research asked how they deal with hierarchical data—does this generate thousands of columns in their table paradigm? Mike noted that Bigtable is not a database, and such hierarchical data situations are not suited for Bigtable. He noted that their clients need to conform to how Bigtable works, not the other way around. George Candea from EPFL asked whether they had any technical insights to offer based on their experiences. Mike said that if you have the freedom to do so, build something with a custom API that matches the needs of both clients and users.

DISTRIBUTED SYSTEMS OF LITTLE THINGS

Summarized by Anthony Nicholson

■ ***EnsemBlue: Integrating Distributed Storage and Consumer Electronics***

Daniel Peek and Jason Flinn, University of Michigan

Daniel Peek described EnsemBlue, a framework for integrating consumer electronic devices (CEDs) into commodity distributed file systems (DFSes). This has been difficult because of the closed nature of CEDs and because such devices cannot simply run the DFS's client software to integrate its storage with all the user's other computing devices. Instead, Dan described how EnsemBlue leverages the user's general-purpose computers (such as desktops and laptops) to act as a bridge between the DFS and each CED that connects to the computer (e.g., when an iPod syncs with a user's desktop machine).

A key challenge here involves namespace conflicts between the DFS and the proprietary naming structures found on CEDs. EnsemBlue handles this by tracking mappings between the name of an object in the DFS and its name on each given CED. Since the CEDs comprise a closed system, the general-purpose computers in the system must execute all custom code in the system. These computers therefore need to know when data is updated in the system, to take certain actions such as updating custom indexes on CEDs. The authors leverage the fact that every DFS has a distributed notification protocol already—the cache consistency mechanism. Therefore, the authors introduce the concept of a "persistent query," which is an object in the DFS that indicates what the query is looking for, such as new mp3 files added to the DFS. The authors presented an example of how a persistent query for all new m4a files could be used to implement a transcoder from m4a to mp3 files. Finally, the authors described how they handle disconnected devices that cannot speak with the general file server. In such situations, several of the user's devices might be able to contact each other but not the remote file server. One of the devices becomes a "pseudo-file server," acting as a file server to the best of its ability, serving to the other devices those files that it happens to have at the time.

One questioner ask how this work would fit into the universal Plug-and-Play (uPnP) initiative. Dan responded that currently, EnsemBlue requires read and write access to the device (through USB, for example). Their future work will allow them to work with arbitrary protocols such as uPnP. Jawwad Shamsi from Wayne State University asked whether they require a separate general-purpose device for each mobile device. Dan answered that any number of CEDs can connect to any number of general-purpose computers. Christopher Stewart from the University of Rochester asked whether they had encountered any performance tradeoffs in building the protocol that interacts with the dedicated host machine. Dan responded that they hadn't measured the performance of integrating data back to the DFS through the general-purpose computer, but since their system is weakly consistent anyway, such performance would not be that important.

■ ***Persistent Personal Names for Globally Connected Mobile Devices***

Bryan Ford, Jacob Strauss, Chris Lesniewski-Laas, Sean Rhea, Frans Kaashoek, and Robert Morris, Massachusetts Institute of Technology

Currently, locating users' personal devices over the Internet is difficult. Local discovery protocols such as Bonjour don't work over long distances, and requesting DNS names for all of one's devices is impractical at best. Bryan Ford argued that people should be able to name their devices in a personal fashion and have global connectivity with each other's devices, without having to keep changing the way the devices locate each other. Their proposed solution is called UIA (Unmanaged Internet Architecture).

In UIA, users managed their own personal namespaces. When they acquire a new device (such as a phone, laptop, or camera) they assign a name to the device and "introduce" it to their existing devices. Each device has a unique endpoint identifier (EID)—just a hash of its public key. All clients running UIA belong to an overlay that lets this namespace exist atop IP. Users assign short personal names to identify their friends. Other users' devices can then be named by the combination of friend name and device name. Devices then gossip to propagate their name records. Friendship is transitive, so if Alice knows Bob directly, and Bob knows Charlie, Alice can name Charlie's phone as Phone.Charlie.Bob. The authors have implemented UIA on Linux, Mac OS X, and the Nokia 770 tablet. A UIA name daemon and router run atop the TCP/IP stack in userspace, with some GUI controls as well. Bryan also showed an entertaining demo video that highlighted the ease of use of the UIA paradigm. Their code is available for download (in a rough state!) on their project Web page: http://pdos.csail.mit.edu/uia/.

Mark Aiken from Microsoft Research asked how this system manages connectivity to devices that have moved while being communicated with. Bryan answered that in UIA's routing protocol, devices track other devices in their local social neighborhood and then hope to find a few devices that are stable enough to be rendezvous points to disseminate current IP address information. These stationary nodes help bootstrap what is essentially a distributed DNS scheme. Another questioner asked whether the authors had conducted any usability studies of their system. Bryan answered that they hadn't had any users outside their research group. Mahur Shah from HP noted that all the examples in the talk deal with devices that are fully owned by one user. He wondered how this would work for shared devices—in a family setting, for example. Bryan noted that they discuss this in the paper. Each physical device can have multiple EIDs and go by different names.

■ *A Modular Network Layer for Sensornets*

Cheng Tien Ee, Rodrigo Fonseca, Sukun Kim, Daekyeong Moon, and Arsalan Tavakoli, University of California, Berkeley; David Culler, University of California, Berkeley, and Arch Rock Corporation; Scott Shenker, University of California, Berkeley, and International Computer Science Institute (ICSI); Ion Stoica, University of California, Berkeley

Cheng Tien Ee described work at Berkeley on a modular network layer for sensor networks. The authors recognize that sensor nets software from different organizations does not interoperate easily. The specific problem they address in this work is monolithic, vertically integrated network stacks. Intuitively, one would expect that if such network layers were modularized there would be a good deal of overlay among different implementations. Since the authors argue that we are probably stuck with multiple network protocols, they focus on making it as efficient as possible to run multiple network protocols at once on one system.

Their solution decomposes the network layer into modules. First, they break the network layers into the data plane and the control plane. Each of these is further subdivided into many components, such as an output queue and forwarding engine in the data layer, and a routing engine and routing topology in the control plane. They show how diverse protocols can actually share components, such as output queues, resulting in run-time benefits and code reuse. Their evaluation of several common protocols showed that protocol-specific code made up a small fraction of the total code base for their implemented examples.

Matt Welsh from Harvard was concerned about the interplay among different network protocols with regard to such things as packet scheduling and memory usage. Cheng responded that memory management is indeed a cross-layer issue and that they are currently looking at dealing with such effects. Matt also asked whether the code was available, and Cheng said they are currently attempting to integrate their work into TinyOS. Eddie Kohler from UCLA asked about the types of protocols that would fit this model less well than the examples the authors chose. Cheng answered that they can decompose any class of protocols, but the main difficulty they have with more complex protocols is the decomposition of REs (Routing Engines) and FEs (Forwarding Engines) into smaller ones that can be better reused. An example of such a protocol is one with multiple phases during the forwarding of a packet along its path. For such protocols, it is not immediately clear how the further decomposition can be done. An indication of an improperly decomposed network protocol would be multiple similar functions, such as packet forwarding methods, being implemented within a type of component. The last question noted that there are protocols they can't support, but this is due to SP (Sensornet Protocol); for example, anything with rate limiting can't be represented to SP. To the question of whether there is anything the authors would have wanted from the lower-level abstractions that they don't currently supply, Cheng replied that he didn't need anything else from SP and, on the contrary, found it often provided more info than necessary.

OPERATING SYSTEM STRUCTURE

Summarized by Leonid Ryzhyk

■ *Making Information Flow Explicit in HiStar*

Nickolai Zeldovich and Silas Boyd-Wickizer, Stanford University; Eddie Kohler, University of California, Los Angeles; David Mazières, Stanford University

Nickolai Zeldovich stated that the HiStar operating system aims to prevent malicious and buggy software from leaking sensitive user data by making all information flow within the system explicit. The HiStar kernel implements six types of objects used as building blocks for user-level software: containers, segments, address spaces, threads, gates, and devices. Each object is assigned a security label that controls how the object can be modified or observed and can be thought of as a taint. Data in a tainted object can only be accessed by other tainted objects. Any data that flows outside the system has to be untainted first. Untainting can only be performed by a thread that has an untaint label.

Coming up with the right design of taint tracking can be difficult, if one wants to avoid covert channels. In a naive design, malicious applications could communicate by modifying and observing taint levels of different objects. HiStar closes this covert channel by making all nonthread object labels immutable. To avoid covert channels arising from resource allocation, HiStar provides a specialized IPC abstraction where the client donates initial resources to the server.

Flexibility is achieved by introducing multiple categories of taint. For example, UNIX users can be emulated by assigning a taint category to each user. A superuser can then be implemented as a thread holding untaint privilege for all user categories. As a result, root has no special privileges in the system and is not fundamentally trusted by the kernel.

Nickolai illustrated the HiStar architecture using two case studies. First, a running example of a virus scanner was used to introduce the taint-based access control model and demonstrate how HiStar allowed encapsulating application-specific security policies in a separate component. Second, an implementation of the UNIX authentication process based on an untrusted authentication service was described.

The main group of questions related to the HiStar resource management policy and dealing with different types of covert channels. Nickolai explained that static preallocation of resources is preferred in cases where you really want tight control over information flow, but in less-sensitive applications that is likely to be too restrictive. With regard to covert channels, someone asked how HiStar dealt with latency-based covert channels. Nickolai replied that although they were working on some ideas, the current implementation did not prevent such channels.

Another question was whether the authentication service could leak the password by denying and allowing login requests. The answer was that this could not happen, since every request is essentially handled by a newly forked instance of the authentication service, which is not allowed to communicate any data back to the original instance of the service. Another interesting question was whether HiStar could accommodate legacy applications requiring communication with the outside world. Nickolai said that HiStar could accommodate most legacy applications, but it may not be able to provide any added security if the application is monolithic and requires frequent network interaction.

- *Splitting Interfaces: Making Trust Between Applications and Operating Systems Configurable*

Richard Ta-Min, Lionel Litty, and David Lie, University of Toronto

In the modern OS, the trusted computing base of an application includes not only the kernel but also all the privileged user-level services that run under the root account. By compromising one of these services, the attacker gets access to all sensitive data in the system.

Proxos aims to reduce the TCB by isolating sensitive applications inside their own private instances of the OS running under control of a virtual machine. All other applications run inside the public "commodity" OS. The commodity OS is also used for communication among secure applications running inside private OSes. This is achieved by selectively routing some system calls issued by secure applications to the commodity OS. The developer specifies which calls should be routed by using the Proxos routing language. To benefit from the Proxos architecture, secure applications need to be split into components running inside the commodity OS and those running inside the private OS.

The main performance overhead comes from context switching between the private and the commodity OS, which turns out to be an order of magnitude slower than Linux kernel call. However, at least for the proof-of-concept applications that have been ported to Proxos (Web browser, SSH authentication server, and the Apache Web server with SSL certificate service), this did not prove to be a problem, as the resulting end-to-end overhead was negligible.

Q: The private OS can become as complex as Linux itself. So how does this help reduce the TCB? A: Yes, security-sensitive applications still have to trust the entire Linux kernel, but not the privileged processes running on top. Q: Is it necessary to write proxy code for each ioctl to the commodity OS? A: Yes, but in our experience things you want to isolate do not require this. Q: So, are you claiming that context-switch time is irrelevant? A: This is the case for applications that we have ported so far. Of course, for applications that do more kernel calls the performance impact would be greater. Q: In your performance evaluation you compare overhead of Proxos against Linux running on top of Xen. What would be the overhead compared to Linux running on hardware? A: We haven't done such experiments but the overhead can be estimated based on available performance data for Xen.

- *Connection Handoff Policies for TCP Offload Network Interfaces*

Hyong-youb Kim and Scott Rixner, Rice University

Hyong-youb Kim focused on efficient utilization of TCP offload capabilities available in some modern NICs. Whereas offload-capable NICs can potentially improve the performance of network-intensive applications by taking over some of the TCP processing load, it turns out that if used without care this feature can easily saturate the NIC and degrade the overall system performance.

Three techniques for optimizing connection handoff policy were proposed:

1. Prioritize packet processing on the NIC by giving packets handled by the host processor higher priority than packets processed by the NIC.
2. Dynamically adapt the number of TCP connections handled by the NIC based on the length of packet queues.
3. Compensate for the hand-off cost by offloading long-lived connections to the NIC.

The proposed techniques were evaluated using a system simulator modeling the NIC as a MIPS processor with 32 MB of RAM.

Q: If there is a sudden change in number of received packets, would there be a lot of overhead in switching? For example, what happens if someone wants to DoS by starting to send lots of long packets and then stopping? A: The NIC currently does not switch connections back to the host, so there's no overhead involved in reducing the number of connections. If the host wants to hand a connection to the NIC it has to transfer a small buffer, but it's cheap. Q: You simulate the NIC as a single general-purpose processor; however, actual network processors are more complex and have multiple specialized cores. Are your results representative of what would be observed on real hardware? A: Network processors are not good at handing NIC workloads, so they should not be used for NICs. Network processors are built for switching packets. NIC workloads are different, and network processors do not work well.

DISTRIBUTED STORAGE AND LOCKING

Summarized by Leonid Ryzhyk

■ *Ceph: A Scalable, High-Performance Distributed File System*

Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D.E. Long, and Carlos Maltzahn, University of California, Santa Cruz

Sage A. Weil described Ceph as a high-performance distributed file system designed to accommodate hundreds of petabytes of data. The main principles underlying the Ceph architecture are separation of data and metadata, use of intelligent storage devices, and adaptable dynamic metadata management.

In his talk, Sage described the two main components of Ceph, the metadata store called MDS and the distributed object store called RADOS. MDS achieves excellent scalability by dynamically partitioning the directory tree among metadata servers based on metadata access frequencies. In addition, MDS simplifies the metadata structure by replacing file allocation data with seeds to a system-wide well-known hashing function called CRUSH, which is very stable in the face of storage device failures and other storage capacity changes.

The RADOS object store provides scalability and reliability through replication, failure detection, and recovery. RADOS achieves scalability by using intelligent object store nodes that take advantage of a very compact representation of the cluster state (made possible by CRUSH), enabling them to efficiently communicate with local peers to quickly recover from failures or any other changes to the cluster. To enable the use of Ceph for efficient communication, while guaranteeing data safety, RADOS implements a two-phase write acknowledgment protocol. The first acknowledgment confirms that the update has been propagated to all replicas, while the second confirms that the update has been physically committed to disk. RADOS uses the EBOFS object file system for local data storage. EBOFS implements a non-POSIX interface that supports features such as atomic transactions and asynchronous commit notifications.

Q: Does Ceph support directory move/rename? A: Yes. The only thing that moves is the inode, not the directory. Q: What is the effect of network latency on performance numbers? A: One of the assumptions is that we are deploying in a data center environment. In principle, however, you could deploy it in a wider scenario if you have sufficient bandwidth. Q: In your model, metadata lookups are done on the server. What are the characteristics of your workload that make this more appropriate for scalability than doing the lookup on the client? A: High-performance workloads with high contention for metadata require consistency to be managed within the metadata server. Q: What is the recovery strategy for metadata? A: The short-term log doubles as a journal, so if a metadata server fails, another node can rescan its journal.

■ *Distributed Directory Service in the Farsite File System*

John R. Douceur and Jon Howell, Microsoft Research

Jon Howell described Farsite as a distributed serverless file system for networks of workstations. The focus of the current talk was on the design and implementation of a distributed metadata service for Farsite. The design goals included support for fully functional directory move operations, including moves across partitions, support for atomic moves, support for load balancing, and hotspot mitigation.

The authors observe that the conventional approach to metadata partitioning based on path names precludes load balancing. Instead they suggest implementing partitioning based on hierarchical immutable file identifiers. Since the identifier hierarchy is not tied to the path hierarchy, load balancing and directory (re)naming become completely orthogonal. File identifiers are compactly represented using a variant of the Elias y coding. Efficient lookup is achieved by storing the file map in a data structure similar to the Lampson prefix table.

To implement atomic rename operations, the recursive path leases mechanism is introduced. It enables safe locking of the chain of file identifiers from the file-system root to a file, without putting excessive pressure on the root server. Finally, the Farsite metadata service minimizes false sharing by implementing a fine-grained locking scheme, where a client acquires locks for individual file metadata fields required for the requested access mode, rather than for the entire file.

Q: Does the repeated load rebalancing have an impact on the efficiency? A: Yes, one of the design decisions we made was that when we delegate load we can never coalesce it again. In practice it seems to work well. Q: Why aren't we already all using Farsite-like things on our mostly empty disks? Is there a drawback to this approach? A: The greatest limitation is that byzantine fault tolerance depends on assumptions about how many machines may fail, but if they are all running homogeneous software and have a similar vulnerability, they can all fail/misbehave in the same way. Q: Suppose we adopt a less pure approach and put some stuff inside protected infrastructure. How does that change things? A: Without having to care about byzantine fault tolerance, things would be much simpler. However, metadata load is a huge factor, which we would like to distribute among multiple machines.

- ***The Chubby Lock Service for Loosely-Coupled Distributed Systems***

Mike Burrows, Google Inc.

Chubby is a large-scale distributed lock service used in several Google products, including GFS and Bigtable. Mike focused mainly on introducing the Chubby API and the motivation behind it and describing the ways Chubby has been used, rather than on how it was implemented.

The main purpose of Chubby is to provide distributed-systems developers with a reliable and scalable implementation of the distributed consensus protocol. However, experience has shown that even if implemented as a library, the consensus protocol is still difficult to use for developers. Therefore, Chubby encapsulates it inside the familiar lock service API.

In addition to providing lock and unlock operations, Chubby allows associating small data records with locks and adopts a UNIX-like naming scheme for them, which makes it look and feel like a file system. However, it is not well suited for storing large amounts of data and lacks a number of filesystem features such as file renaming, atomic multifile operations, and partial-file reads and writes. This lack of features helps simplify the Chubby design and prevents developers from misusing it as a distributed file system.

Lock clients can be notified of certain types of events, including file content changes, file creation/deletion, and lock acquisition. Chubby is designed to support large numbers of clients per lock. Although changes of the lock ownership are typically infrequent, clients tend to periodically poll the lock, creating a lot of read traffic. To reduce this traffic, a consistent write-through client-side cache is used.

Q: Are there any examples of interactions that you had with the user community that led to the file-system abstraction? A: No, the design decision happened before the user base. I would put the main reason down to sharing an office with Rob Pike and Sean Quinlan (Plan 9 people), so everything looked like an FS. Q: Chubby allows developers to easily get reliability guarantees by using a lock server instead of a state machine. Were there other projects that had to go off and implement a state machine? A: Well, we did. There is a state machine library that we use; at present there are no other users of it. Q: It seems like large-scale tools are becoming increasingly more integrated. Have you thought about bad interactions where one misbehaving application of a tool can cause cascading failure in other applications? A: Yes, it happens all the time. My system has managed to bring down many others. What you do is analyze exactly what happened, fix your programming steps, and fix your code so that every single problem can't happen again, and of course something else happens next time.

LARGE DISTRIBUTED SYSTEMS

Summarized by Prashanth Radhakrishnan

Experiences Building PlanetLab

Larry Peterson, Andy Bavier, Marc E. Fiuczynski, and Steve Muir, Princeton University

This talk, given by Larry Peterson, was about the authors' experience building PlanetLab (PL). PL is a global platform for deploying and evaluating planetary-scale network services. PL has machines spread around the world, with users' services running in a slice of PL's global resources.

The PL design was a synthesis of existing ideas to produce a fundamentally new system: It was experience- and conflict-driven.

Larry listed the requirements identified at the time PL was conceived and the design challenges they faced. Given its scale, PL had to rely on site autonomy and decentralized control for sustainability, while also managing the trust relationships between the users of PL and the owners of the machines. Next, it had to balance the need for resource isolation while coping with support for many users with minimal resources. Finally, PL had to be a stable, usable system, supporting long-running services and short experiments, while continuously evolving based on feedback.

PL's management architecture has the following key features to address the design challenges. PlanetLab Control (PLC), a centralized front-end, acts as the trusted intermediary between PL users and node owners. To support long-lived slices and accommodate scarce resources, PL decouples slice creation from resource allocation. Node-owner autonomy is achieved by making sure that only owners generate resources on their nodes and that they can directly allocate a fraction of their node's resources to virtual machines (VMs) of a specific slice. To support slice management through third-party services, PLC allows delegation of slice-creation by granting tickets to such services. For scalability, PL was designed so that multiple PL-like systems can coexist and federate with each other. As per the principle of least privilege, management functionality has been factored into self-contained services, isolated into their own VMs and granted minimal privileges. To address the resource allocation issues, PL provides fair sharing of CPU and network bandwidth and simple mechanisms to protect against thrashing and overuse. Finally, keeping PL's control plane orthogonal from the VMM, leveraging existing software, and rolling out upgrades incrementally helped PL evolve while also being operational.

Larry concluded with lessons learned from their experience. Key among them was the observation that decentralization follows centralization; that is, a centralized model is important for a system to achieve critical mass, and it is only by federation that the system can scale.

During the Q&A session, Sean Rhea of Intel Research Berkeley asked about Larry's comments on the proposal to set aside physical boxes for measurements. Larry said he was not convinced about reserving physical resources, but rather thought that logical isolation was sufficient. David Anderson of CMU noted that Larry's talk presented a rosy picture of PL, in contrast to the PL panel in WORLDS '06 that discussed problems with PL. David asked about the observed problems with running latency-sensitive services, disk thrashing, and scheduling. Larry said that there was room for improvement in scheduling. He also noted that since the PL code is available, the community was welcome to track down bugs that hamper their research and report patches. He said that there was a known kernel bug that could cause problems with latency-sensitive slices and that things would improve when the next kernel upgrade is rolled out.

iPlane: An Information Plane for Distributed Services

Harsha Madhyastha, Tomas Isdal, Michael Piatek, Colin Dixon, Thomas Anderson, and Arvind Krishnamurthy, University of Washington; Arun Venkataramani, University of Massachusetts Amherst

Harsha Madhyastha presented iPlane, a service that provides accurate predictions of end-to-end Internet path performance. He started off with the observation that large-scale distributed services, such as BitTorrent, depend on information about the state of the network for good performance. But most current Internet measurement efforts, such as GNP and Vivaldi, provide only latency predictions between a pair of nodes. In contrast, iPlane measures a richer set of metrics, such as latency, loss rate, and available bandwidth.

iPlane continuously performs measurements to generate and maintain an atlas of the Internet by doing traceroutes from a few distributed vantage points. For scalability, targets are clustered on the basis of BGP atoms and a representative target from each atom is used to approximate the performance of targets in the atom.

iPlane uses structural information such as the router-level topology and autonomous system (AS) topology to predict paths between arbitrary nodes in the Internet. This prediction is made by composing partial segments of known Internet paths so as to exploit the similarity of Internet routes. Next, iPlane measures the link properties in the Internet core and edge. In the Internet core, the special vantage points measure the link attributes. Link properties at the Internet edges are obtained by participating in BitTorrent swarms and measuring the links to the end-hosts while interacting with them.

Thus, to measure path properties between any two hosts, first the path between them is predicted. Then, iPlane composes the measured properties of the constituent path segments to predict the performance of the composite path.

iPlane has been demonstrated to improve the overlay performance of several representative overlay services such as content distribution networks, swarming peer-to-peer filesharing, and VoIP.

During the Q&A session, Buck Krasic of the University of British Columbia observed that participating in BitTorrent swarms to measure Internet edge link properties might result in conservative estimates; for example, BitTorrent clients may have multiple connections open and the bandwidth that iPlane observes might just be a fraction. He asked whether iPlane had some technique to compensate for that. Harsha replied that they were using BitTorrent to measure bandwidth capacity, not the available bandwidth, and that bandwidth capacity can be measured using a pair of back-to-back packets. And since measurements from BitTorrent are based on passive monitoring of a TCP connection of several packets, it is likely that at least one pair of back-to-back packets will be observed.

■ ***Fidelity and Yield in a Volcano Monitoring Sensor Network***

Geoff Werner-Allen and Konrad Lorincz, Harvard University; Jeff Johnson, University of New Hampshire; Jonathan Lees, University of North Carolina; Matt Welsh, Harvard University

Geoff Werner-Allen presented this science-centric evaluation of a 19-day sensor network deployment at Reventador, an active volcano in Ecuador. The data collected by the sensor network deployment was evaluated based on five metrics, namely, robustness, event detection accuracy, data transfer performance, timing accuracy, and data fidelity. Of these, Geoff dealt with robustness, timing accuracy, and data fidelity in his talk.

The sensor hardware they used for volcano monitoring was small and provided real-time data acquisition, unlike conventional standalone dataloggers, which are unwieldy, and it logged data to a local flash drive. Their sensor network contained 16 sensor nodes, each equipped with a seismometer, a microphone, and an antenna. These nodes continuously sample seismic and acoustic signals and log the data to local flash memory. They also run an event-detection algorithm that transmits a time-stamped report to the base station upon detection of a seismic event. The base station, located 4.6 km away from the sensor deployment, initiates data collection if it receives triggers from more than 30% of the sensor nodes within a certain time.

The overall robustness of the system was limited by power outages at the base station and a single three-day software failure. Discounting these, the mean node uptime exceeded 96%, indicating that the sensor nodes themselves were reliable. Flooding Time Synchronization Protocol (FTSP) was used for time synchronization between sensor nodes. Although predeployment results with FTSP were good, during deployment they ran into stability issues, leading to occasional incorrect time-stamp reports. They developed a time rectification approach that filters and remaps recorded time stamps to accurately recover timing despite the incorrect time stamps. They evaluated the fidelity of the collected data by performing an analysis of the seismic and acoustic signals from a seismological perspective. Their results indicate that the collected signal quality and timing match the expectations of the infrasonic and seismic activity produced by the volcano.

Geoff concluded his talk with the three lessons they learned from this deployment: Ground truth and self-validation are critical, network infrastructure is more brittle than sensor nodes, and it's important to build confidence with domain scientists.

During the Q&A session, Geoff was asked whether they look at sensor networks as a tool that scientists in other domains could use without requiring the computer scientist's presence. Geoff responded by stating that this was a great observation and that they wanted sensor networks to eventually be used like that. Someone from Stony Brook University asked whether it was possible to simply broadcast time from the base station. Geoff replied that such a broadcast would work only with single-hop networks, which isn't the case with the Reventador deployment. Mehul Shah of HP Labs asked about the fidelity of the measurements with respect to its relevance to the domain scientists. Geoff said that the scientists are still working on the results obtained and that their initial observations are encouraging.

HotDep '06: Second Workshop on Hot Topics in System Dependability

Seattle, Washington
November 8, 2006

FINDING THE NEEDLE IN THE HAYSTACK

Summarized by Yin Wang

■ ***Comprehensive Depiction of Configuration-dependent Performance Anomalies in Distributed Server Systems***

Christopher Stewart, Ming Zhong, Kai Shen, and Thomas O'Neill, University of Rochester

Presenter: Chris Stewart

Distributed server systems such as J2EE application-server systems have wide-ranging workload conditions. The assumption here is the reasonable performance expectation based on knowledge of the system design (e.g., Little's Law). The problem is performance anomalies, that is, when performance falls below expectation. Previous work shows that anomaly characterization can aid the debugging process and guide online avoidance. Chris's goal is to depict *all* anomalous conditions.

A three-step process was taken: (1) generate performance expectations by a whole-system performance model; (2) search for anomalous run-time conditions; and (3) extrapolate a comprehensive depiction. An example was shown of a submodel hierarchy (a four-level submodel for J2EE application servers), with its advantages and limitations. The next step is to determine the anomaly error threshold, which is different for online avoidance and debugging. Then Chris explained decision-tree-based depictions, why they chose to use decision trees, and how they classify anomaly conditions. For the case study of JBoss, where three performance anomalies were found and fixed, the decision tree displayed with the three anomalies described. This approach cannot detect nondeterministic anomalies, and the model accuracy requires manual investigation. Furthermore, debugging remains manual. The take-away message is that depiction of anomalies can aid debugging and avoidance.

Jay Wylie asked how to interpret anomalies that are good. Chris responded, "We don't consider that. The anomalies here are out-of-expectation anomalies." To Ken Birman's question of whether the magnitude between anomalies and normal performance is similar, Chris said that it is based on empirical observation. What about a known source of anomalies? For example, when garbage collection kicks in, does the performance degrade? Is this something you can't model? Chris agreed that this was a problem, but he said that his group hopes "to block out those known anomalies." John Wilkes asked how hard it is to build models. "The model is borrowed heavily from the NSDI '05 paper. Actually, a simple model is adequate for it, like Little's Law," was the response. In reply to a question on how controlled searches must be in order to detect anomalies, Chris commented that this work involved a benchmark-controlled environment. For a long trace of system execution, the performance variation may be huge.

Geoff Voelker asked, "In terms of the size of configurations explored, they seem to be relatively small. In a large system you may have lots of choices, for example, cache parameters. What do you do in this case?" Chris replied, "We hope to investigate a systematic method to explore system configurations. Within this work, we have eight run-time conditions and 7 million possible configurations." The final question, "Do you know when to stop exploration?" elicited a response of "It depends on end use. If you want to do avoidance, you want to stop when you manage to satisfy the performance goal. For debugging, it depends on the quality of code you want."

■ ***Static Analysis Meets Distributed Fault-Tolerance: Enabling State-Machine Replication with Nondeterminism***

Joseph G. Slember and Priya Narasimhan, Carnegie Mellon University

Presenter: Joseph Slember

State-machine replication is a standard way to add fault tolerance, but if replication is not 100% deterministic, it is difficult to do. The goal here is to target nondeterminism when it matters, and programmer intent must be respected. Joseph showed a picture of a three-tier replicated server system and explained the complexity of the problem. The approach adopted is compile-time static analysis with run-time compensation.

Next Joseph explained the taxonomy of nondeterminism (abbreviated as ND hereafter). ND–I includes pure (or first-hand) ND, for example, random(), gettime(), and contaminated (or second-hand) ND, which is the ND induced by pure ND. ND–II includes superficial ND and other ND types. For static analysis, they built an ND dictionary of C and C++. They then added data structures to store results of ND actions. Code snippets are generated and inserted as functions. For run-time compensation, there is a tradeoff between checkpoint-to-compensate (high bandwidth) and reexecute-to-compensate (high CPU).

Ken and Lorenzo broke in with "How much is the compensation overhead?" The answer is that it depends on the application-level characteristics. For example, with Apache there is no compensation at all. To the question "What is the advantage of your technique over the backup method?" Joseph explained that the backup does not work on multitier systems. In response to "If the compensation falls behind, and the replica is ahead, can the other one catch up or they will be inconsistent?" Joseph said that as soon as the replica is compensated, it is consistent. The concurrency is increased by doing it this way.

Joseph continued with the preliminary evaluation. The tier number is between 2 and 4, with clients between 2 and 4. He showed a graph on experiments with 5% forward and backward ND. The graph displayed that the technique scales well. Another graph on 60% forward and backward ND shows increased overhead. An insight from these results is that lower amounts of ND cause much less overhead. Thus application characteristics will determine the overhead.

Jay Wylie asked whether all tiers get analyzed at the same time or independently. It turns out that you can do it independently. The worst case is that tiers are fully transparent. Ken and Lorenzo asked whether there has been prior work doing replication on ND programs, writing down what the program did, then the backup waiting for the primary to know what to do. The cost seems to be not that great. Joseph added that breaking down the overhead is a subject of future work. In reply to "How does your approach compare with the method where the master decides and the slave asks for the answer?" Joseph said that the slave has to ask for everything in that case, and we don't need that, so future work will aim toward making it more efficient. Miguel Castro asked whether the project does compensation at the same time. It does, and this helps to increase the concurrency of the program. Geoff Voelker asked how to know which ND a function call is going to depend on. Joseph replied that they map calls to different ND.

- ***Correlating Multi-Session Attacks via Replay***

Fareha Shafique, Kenneth Po, and Ashvin Goel, University of Toronto

Presenter: Ashvin Goel

Typical attack characteristics include low-level or stealthy behavior, small footprint, and multiple sessions. The idea in this paper is to replay the attacks. The basic replay method is to compare outputs with replay run and the original run. But if the replay is nondeterministic, the output could differ. The solution is training using nondeterministic inputs to obtain output statistics. The outlier is classified as the attack.

Ashvin showed the experiments of unit tests on different applications and multisession attacks. The unit test result is a matrix showing the sensitivity of the method to changes in inputs. The multisession result is a diagram with attack multisession and user multisession. There is a great output difference between the attack and the user. In concluding, Ashvin proposes replaying sessions with changed inputs to correlate attacks. Future work includes nondeterministic replay and the case of long-running sessions.

Chris Stewart asked whether it is important that legitimate user outputs are not modified. If you have a false positive, you roll back, and the output is modified. Ashvin explained that false positives do not matter if it is security-critical. For analysis, we are not concerned about 100% correctness. You probably need a human to identify the attack at the end. George Candea asked about the cost of acting on false positives. What if you roll back actions that are really important? Ashvin said, "We don't get too many false positives. False negatives are more important for the work. Because we have roll-back recovery, we can reverse the rollback, but the cost is huge. It is important not to have many false positives. We still need a human." Another question concerned a paper from USENIX on trying to undo operator mistake and then traveling back in time, but it is hard to redo a change you should not have undone and the cost is high. Ashvin explained that the system has to be offline once you have an intrusion. It depends on how long it takes you to fix it. Jim Thornton's question on what to do if a legitimate change results in a different output was deferred to an offline discussion.

- ***Automatic On-line Failure Diagnosis at the End-User Site***

Joseph Tucek, Shan Lu, Chengdu Huang, Spiros Xanthos, and Yuanyuan Zhou, University of Illinois at Urbana-Champaign

Joseph Tucek showed the commonly seen Windows XP error message window that asks you to send an error report, which is mainly the core dump. It is not easy to reproduce a bug with just core dumps. The real insight is that because there is a well-defined set of steps that are generally taken during debugging, why not automate it? The proposed solution is online automated diagnosis. First capture the moment of failure; then run analysis tools only on the relevant portions of the program; finally, automate the debugging process in a humanlike protocol. For replay and reexecution, they used their checkpoint/reexecute framework in their SOSP '05 Rx paper. For analysis, their project begins with the core dump, which provides an inexpensive starting point. For the example of memory bug detection, it works by monitoring accesses. The analysis process is too expensive for production runs. It has to be modified and inserted during mid-run. It is feasible only because they limit it to recent and relevant execution.

As experimental results, there are three bugs found with failure types correctly identified. Joseph displayed in detail the example of a TAR bug. The take-away message is that a dynamic backward slice can tell a concise story of the problem and that the tool can perform analysis at the end-user side.

Solom Heddaya was interested in the number of times the system failed but no bugs were found. For deterministic bugs, Joseph claimed 100% success rate, but "occasionally, we have to go back more checkpoints, and it is possible that the earliest checkpoint is after the root cause. For the case of nondeterministic bugs, the result is not so good." George Candea said that he thought most bugs are nondeterministic, but the reproducibility varies. Joseph conceded that some nondeterminism comes from the environment, but they are eliminating this source. Ken Birman asked how far one must go in rolling back and checkpoints. Joseph said that, "according to statistics, we go back more when necessary, and give up if you cannot find it far enough." In response to whether backward slicing requires source code, Joseph said it did not and that they use binary instrumentation with the tool PIN from Intel.

Kai Shen stated that there are already many debugging tools out there. So what are the take-home points? Joseph replied that their tools are feasible because of the low overhead. "Debugging is not an art. There is a process we can use." Chris Stewart wondered how the logic here can be different for different systems. "We try to be general," said Joseph. "Java is different. We do not deal with that. If you have more specific knowledge, it would provide much better results."

PRAGMATIC CHOICES FOR THE NEW AGE

Summarized by Avishay Traeger

■ *The Case for Byzantine Fault Detection*

Andreas Haeberlen, Max Planck Institute for Software Systems and Rice University; Petr Kouznetsov and Peter Druschel, Max Planck Institute for Software Systems

Speaker: Andreas Haeberlen

Byzantine Fault Tolerance (BFT) is a well-known technique that is used in distributed systems to mask a bounded number of byzantine faults. BFT incurs large overhead in that it requires $3f+1$ replicas, where f is the number of faults to tolerate, and it does not scale well. This work describes Byzantine Fault Detection (BFD), an alternative approach that aims at detecting these faults, rather than masking them. Whereas detection is not sufficient for irreversible behavior, it is an efficient and scalable alternative for recoverable faults. It can also deter bad behavior. The detection system uses only $f+1$ replicas and requires that only one replica complete a request before returning to the client (rather than requiring that most replicas complete, as in BFT).

Each node in this system has a state machine and a detector. The detector can inspect all messages at the local node. When the detector observes a fault it informs its local application and provides evidence to other detectors. Since the detector only knows about messages on the local node, only observable faults can be detected. In the detector, each action is undeniably associated with the identity of the node that has performed the action, allowing the system to gather irrefutable evidence of faulty behavior. In addition, the detector is complete (finds evidence against faulty nodes whenever faulty behavior is observed) and accurate (does not generate valid evidence against correct nodes).

Byzantine Fault Detection is a good choice for systems with recoverable state, systems already using BFT (to ensure that faults are quickly detected), and systems that span multiple administrative domains. Other considerations include the number of nodes in the system, the delay the nodes can tolerate, and the amount of available bandwidth.

■ *Safe at Any Speed: Fast, Safe Parallelism in Servers*

John Jannotti and Kiran Pamnany, Brown University

Speaker: Kiran Pamnany

Many server applications are multithreaded, with one thread handling each request. Concurrency helps improve performance, but the programmer needs to synchronize access to shared resources, which is error-prone. The programming philosophy presented is that one should start with a serial, correct application and gradually improve parallelism, as opposed to starting with a highly concurrent but buggy program and progressively fixing bugs. In an event-driven program it is possible to improve parallelism without adding locking: If two handlers do not use the same global variables or contexts, they can run in parallel. This work uses static analysis to determine which handlers should be allowed to run together, and it enforces the constraints at run-time.

The solution uses static analysis to identify aliases, locate event handlers, determine global variables that are read or written by each handler, examine context usage by each handler, and identify system calls made by each handler. When the analysis is complete, any concurrency issues are reported. In cases where static analysis cannot determine if handlers can run concurrently, conservative behavior is used to ensure safety. The analyzer provides detailed feedback to the programmer so that constraints can be removed, either by splitting the handler or by adding an explicit lock. Profiling can help a programmer decide which constraints should be removed.

At run-time, a multithreaded event management library runs handlers concurrently, subject to the constraints generated by the static analysis. Colors and hues are assigned to event handlers: Handlers that may run in parallel are assigned different colors, and those that may run in parallel if and only if their contexts differ are assigned different hues. Two levels of queues are used to schedule event handlers; the first level has one queue for each hue and the second level has one queue for each color. This approach results in a conservative approximation of the constraints, but it enables efficient scheduling without expensive locking.

■ ***Chunkfs: Using Divide-and-Conquer to Improve File System Reliability and Repair***

Val Henson and Arjan van de Ven, Intel Open Source Technology Center; Amit Gud, Kansas State University; Zach Brown, Oracle, Inc.

Speaker: Val Henson

Today, it can take several days to run a file system check (fsck) on production file systems. As disk capacity is growing at a much faster rate than bandwidth and seek time is remaining fairly constant, the situation will only get worse. In addition, as capacity grows, the likelihood of disk errors grows as well. Existing solutions (journaling, copy-on-write, soft updates, etc.) can reduce the frequency with which one must run fsck, but not the duration of the fsck. Finally, an entire file system can fail from a small number of faults. These issues imply that file systems should be designed with repair in mind. Developers can use several techniques to achieve this: They should use on-disk formats that are conducive to repair, use simple on-disk data structures, create optimizations for reading data for repair, allow for fast incremental file system checks, and add features such as checksums, redundancy, and scrubbing.

Chunkfs is a proposed repair-driven filesystem architecture, where the file system is split into several chunks that are as self-contained as possible. Each chunk has its own block number space, allocation bitmaps, and superblock. This allows individual chunks to be fscked, greatly reducing fsck time. Other benefits include being able to change the size of the file system and defragment quickly. Since chunks do not have much common metadata, Chunkfs has built-in multithreaded scalability. It can also allow for per-chunk filesystem formats.

Chunkfs uses continuation inodes to deal with issues such as files, hard links, and renames that cross multiple chunks. To avoid having many of these continuation inodes, chunkfs uses smart allocation and sparse files so that a file has at most one continuation inode per chunk. Because these continuation inodes contain pointers to other chunks, some chunks may need to be checked together. You can find a project page for Chunkfs at http://www.nmt.edu/~val/chunkfs/.

■ ***Towards a Dependable Architecture for Internet-scale Sensing***

Rohan Narayana Murty and Matt Welsh, Harvard University

Speaker: Rohan Narayana Murty

An Internet-scale sensing (ISS) system consists of a large number of geographically distributed data sources tied into a networked framework for collecting, filtering, and processing potentially large volumes of realtime data. The infrastructure is heterogeneous, decentralized, and volatile: Failures are frequent, and the system must be reliable in the sense that it continues to process (possibly incomplete) data. ISS systems need a highly scalable solution for dependability. This work argues that ISS systems should be designed to offer feedback to end users on the fidelity and coverage of the results returned by the system and should make use of simple, lightweight replication techniques.

In many ISS applications, availability is more important than correctness. It is very difficult to guarantee correctness on such a system, and many applications are naturally tolerant to diminished quality of the data. The goal of this work is to provide mechanisms that mitigate the effects of failures (without diminishing availability) and provide feedback on the quality of answers to the end user. Query results should contain feedback about the fraction of sources represented in the answer, as well as information about the age of the data.

This work has three basic design principles to achieve these goals. First, it uses structured operator replication, which means that more resources are devoted to replications of operators that are higher in the dependency tree since they can lead to larger failures. Second, it uses free-running operators. Operators do not need to maintain consistency with each other, which obviates the need for expensive protocols. However, typical operators have a finite (and often short) causality window that defines the set of past input tuples that affect its internal state, which allows them to eventually return to a consistent state after a failure. Third, it uses best-guess reconciliation to determine the most accurate answer among possible divergent states of the free-running operators. To do so, it can use value-based reconciliation, state-based reconciliation, or a measure of replica divergence.

HIDDEN GEMS (EXTENDED ABSTRACTS)

Summarized by Geoffrey Lefebvre

■ ***Making Exception Handling Work***

Bruno Cabral and Paulo Marques, University of Coimbra, Portugal

Presented by Bruno Cabrel

Exceptions are the standard mechanism for error handling in modern programming languages. Unfortunately, dealing with exceptions is a tedious process. Programmers often avoid the issue by writing empty handlers to save time. Programmers who take the effort to write proper exception-handling code see their productivity impaired. The authors argue that exception handling should not interfere with normal programming tasks but instead become a system issue. In this scenario, the run-time environment provides a set of generic exception handlers and deals with exceptions automatically. The programmer only has to wrap the code with try blocks at the appropriate locations. Because the system may have to try multiple handlers when an exception occurs, this approach requires that try blocks be resumable and appear atomic.

■ ***Speculations: Providing Fault-tolerance and Recoverability in Distributed Environments***

Cristian Tapus and Jason Hickey, California Institute of Technology

Presented by Cristian Tapus

Cristian began his talk by noting that distributed systems are ubiquitous. It is now mandatory that we build safe and reliable systems. Failures are frequent in highly parallel machines. It is critical that systems expected to run for a long time support fault tolerance. Unfortunately, traditional checkpoint mechanisms are application-specific, their implementation is expensive in terms of man-hours, and they are also error-prone.

To address these issues, the authors present a new programming model based on speculative execution. The approach separates fault recovery code from computation. Fault recovery becomes transparent and automated and the design of distributed systems is simplified. Speculations are implemented as an extension to the Linux kernel. The implementation provides system calls to begin, abort, and commit speculative executions. Outbound messages sent while executing speculatively are marked accordingly. A process automatically switches to speculative execution when it receives a message marked as speculative.

■ ***Discrete Control for Dependable IT Automation***

Yin Wang, University of Michigan; Terence Kelly, Hewlett-Packard Laboratories; Stéphane Lafortune, University of Michigan

Presented by Yin Wang

Workflows are programs written in high-level languages and used increasingly for IT automation. These languages can express concurrency, contingency, composition, etc., making workflow programming difficult and error-prone. The authors present an approach based on discrete control theory which provides safe execution of possibly flawed workflows. Their approach uses finite-state automata to represent all execution states reachable from the initial state.

The safety specifications are represented by forbidden states or as regular expressions. The goal is to ensure that the system reaches satisfactory termination without entering forbidden states. A discrete controller is automatically generated offline. The controller dynamically disables controllable transitions based on the current execution state, avoiding transitions to forbidden states when possible. The approach presented allows workflows to be partially decoupled from the dependability requirements.

■ ***SecondSite: Disaster Protection for the Common Server***

Brendan Cully, University of British Columbia; Andrew Warfield, University of Cambridge

Presented by Brendan Cully

Brendan began his talk by noting that disaster can strike at any time. Whether caused by floods, severed power lines, or dinosaurs, site disasters can and do happen. The typical solution is to teleport your server to a new location by restoring a backup and redirecting traffic using DNS. The problem is that backups are expensive and do not always work and DNS updates can take hours, even days, to propagate.

The solution to this problem is to think under the box and use virtualization. The favored approach is to constantly replicate a primary site by performing a continuous live migration of virtual machines. Virtual machines are never suspended; their memory is simply marked copy-on-write when a snapshot is taken. Brendan stated that one of the major challenges is dealing with replication overhead. This issue can be addressed partially by using delta compression. Another challenge is how to take consistent snapshots of multiple servers. Snapshots of individual virtual machines are taken independently but coordination is required to maintain causality within the global snapshot.

■ ***Debate Panel***

All presenters took questions from the audience.

George Candea asked what applications were most amenable to the SecondSite approach. He hinted that databases have large write sets and deal with disaster naturally by shipping their log. Brendan agreed that databases were probably not the best candidate but that there are many existing server applications without built-in recovery mechanisms.

Someone asked Cristian whether programmers would be able to deal with speculations intuitively. Cristian stated that people in other communities, especially in the scientific computing community, are very excited about the idea of speculations.

Christopher Stewart asked Yin Wang, "By leaving the choice to the program, could a program go down a wrong path?" Yin explained that their approach guarantees that programs do not go down forbidden paths.

John Wilkes asked Bruno Cabrel about the metrics used to evaluate his approach. He answered that exception injection could be used as an evaluation tool.

Someone asked a joint question to Bruno and Cristian, since the work of both deals with removing the need for programmers to deal with errors. Are the techniques mutually exclusive? Bruno answered that their intended targets are different. The exceptions framework aims to be a general platform solution, whereas Speculations targets distributed applications. Cristian added that although they differ, the goal is the same: to have cleaner code that is easier to reason about.

Someone stated that if a system magically handles errors, then you have a system that is slightly wrong. How do you reason about this? Cristian answered that this is a similar problem to compiler-generated errors. People do not suspect that their compiler or their operating system is wrong. John Wilkes seems to disagree on the last point. The goal is to increase the level of confidence in the application.

George Candea asked Bruno whether he learned anything surprising from some of the studies on exception handling he cited. The major surprise for Bruno was that the exception-handling code only accounts for 4 to 8 percent. John Wilkes enquired about the applications that were included in these studies. Bruno answered that the studies looked at 16 professional applications such as JBoss.

Petros Maniatis stated that many generic catch blocks will not be acceptable for certain applications. Is there a clean way to override the generic handlers? Bruno explained that the set of recovery blocks can be defined by the platform or the program.

George Candea enquired about the differences between SecondSite and some of the related work from Stanford, especially the Collective. Andrew Warfield jumped into the conversation and stated that the Collective was more about the transport format than the ability to capture instantaneous snapshots of running virtual machines.

Christopher Stewart said that there exists a subculture in the dependability community that believes that exceptions should simply be logged and not handled. He then asked Bruno and Cristian for their opinion on the matter. Bruno answered that this is similar to checked versus unchecked exceptions. He believes that all exceptions should be handled. Cristian said that not handling exceptions can result in violation of program correctness. If you transparently roll back an application without any notification, then the same problem could resurface later and you could end up in a worse state. He believes it is important to report errors to the application. The best approach is to combine the two: Provide availability and report the error. But in the end, no approach will be perfect.

Machine Learning: Theory, Applications, Experiences—A Workshop for Women in Machine Learning

San Diego, California
October 4, 2006

Organizers: Lisa Wainer, University College London; Hanna Wallach, University of Cambridge; Jennifer Wortman, University of Pennsylvania. Faculty advisor: Amy Greenwald, Brown University

Summarized by Lisa Wainer

The workshop was a one-day event offering a showcase of work by women involved in machine learning research. The main objective of the workshop was to offer female faculty, research scientists, and students in the machine learning community an opportunity to meet, exchange ideas, and learn from each other. It also gave women in other areas of computer science the opportunity to learn about cutting-edge research in a growing field. The workshop was open to anyone, male or female, to attend free of charge and was co-located with the Grace Hopper Celebration of Women in Computing. The workshop succeeded in bringing together women from different stages of their careers, from established researchers to Ph.D. candidates and even undergraduate students. It provided an opportunity for established researchers to act as mentors and for students to find much needed role models. There were ninety-six registered participants in all, and quite a few unregistered attendees (two of whom were male).

The invited faculty talks covered a diverse set of topics, were well received by the audience, and inspired much discussion during the session breaks. The student presentations were divided into short talks, spotlights, and poster presentations. Talks were generally of high quality: Roughly half fell into the category of theory and half were on applications of machine learning. The general reaction of participants was extremely positive. There was a vigorous discussion at the end of the workshop about issues for women in the machine learning community and in computer science in general. The lively discussions covered the differences in working in theoretical as opposed to application-based disciplines, the importance of role models and mentoring, how to raise profiles of women in machine learning, and whether the workshop should be run again in the future.

A few general conclusions were reached. The first was that role models are an important aspect of encouraging more women into computer science and machine learning as well as retaining women once in the field. The second was that events such as this workshop are seen as being very important to help women network, to produce collaborative work, and to meet socially. There was a unanimous response to the last point, in that the participants wanted the workshop to run again next year. Many suggested that they would like it to run alongside a machine learning conference. Many attendees commented that the invited talks and student presentations were of exceptionally high quality. Participants were eager to interact and take part by being active during the poster sessions and asking the speakers questions. Overall, the organizers and the participants felt that the workshop was very valuable, and they are planning to hold the event again next year.

Grace Hopper Celebration of Women in Computing 2006, Making Waves

San Diego, California
October 4–7, 2006

Summarized by Rae Harbird

This conference was the sixth in a series designed to bring the research and career interests of women in computing to the forefront. The presenters, from industrial, academic, and government communities, presented their current work while special sessions focused on the role of women in today's technology fields. From my perspective it was a rare and special opportunity to meet women from and hear talks on a diverse range of subject areas spanning the entire breadth of computer science. The atmosphere was truly celebratory, with a strong emphasis on women's achievements and the excitement of working in such a dynamic and fascinating field. The benefits of collaboration and networking for success underpinned the fabric of the conference, reflecting the skills at which women traditionally excel. The conference organizers made great efforts to ensure that the social events were just as rewarding as the technical sessions. I am looking forward to GHC '07.

■ ***Dasher: Information-Efficient Text Entry***

Hanna Wallach, University of Cambridge

The objective of the presentation was to introduce the audience to Dasher, a novel information-efficient text-entry system, driven by continuous pointing gestures. Keyboards, despite their ubiquity, are inefficient for two reasons: They do not exploit the predictability of normal language, and they waste the fine analog capabilities of the user's muscles. Gestural alphabets, such as those used on a Palm Pilot, use fine motor movements but are often unreliable. Devices with limited keyboards, such as mobile phones, use prediction techniques for text entry but this is also clumsy and requires two modes: word completion and disambiguation. Four important things are missing: the ability to take advantage of fine motor movements, exploitation of the redundancy of language, language independence, and single-mode operation allowing users to write and disambiguate at the same time. Dasher is based on principles of machine learning and information theory and is intended to rectify these inefficiencies. Incorporating an adaptive language model of the sort also used in speech recognition, handwriting recognition, and text compression, Dasher offers helpful predictions to the user without constraining the range of words that can be written.

Comparing Dasher with use of a standard keyboard showed that although Dasher users typed fewer characters per minute, error rates (percentage of incorrectly typed words) were lower. Experiments showed that using Dasher with an eyetracker is much faster than using an on-screen keyboard and has a significantly lower error rate. Dasher is designed to be a competitive text-entry system used wherever a full-size keyboard is not possible, such as with wearable and palmtop computers, as well as for disabled users. Anyone can use it, as no training is needed, and it is fast and fun to learn. In a recent case study conducted by Mick Donegan at the ACE Center, Paul, who suffers from cerebral palsy, used Dasher to write his thesis for a degree in Business and IT. Paul reported that Dasher required less head movement, generated fewer spelling mistakes, and was about four times faster than an on-screen keyboard.

■ ***Panel: Building and Managing a Strong Research Group***

Nancy Amato, Texas A & M University; Tracy Camp, Colorado School of Mines; Elizabeth Royer, University of California; Violet Syrotiuk, Arizona State University

As a Ph.D. student who is starting to think about my research career, I was pleased to attend this panel led by women who are experts in their chosen field, which happens to coincide with mine. Although the focus was academia in the United States, the points made were easily translated to a UK context. The talk had two principal strands: first, describing how to get funding for your research group and, second, covering tips on advising students.

As far as obtaining funding goes, some background work is necessary to get your foot in the door: The importance of networking cannot be overemphasized, as you need to establish mentors and partners for your research. Publicize your work and ideas by giving as many talks as possible, both within your institution and outside. The main recommendations for successful proposal writing were to communicate your ideas to as many people as possible and to ensure that your proposal fits the specification provided by the funding body. The panel advised researchers to read and familiarize themselves with proposals that have succeeded already. The pros and cons of working on single or multiple Principal Investigator proposals were discussed. Both are viewed as important to your career, the latter because it provides the opportunity to show the specific expertise that you can bring to a project and the opportunity to do it really well. It may also offer the opportunity of working with more senior staff. It is important to remember that proposals are often rejected, and the panel emphasized the importance of giving careful consideration to feedback received and being persistent. Acting as a panelist on funding bodies is also an important learning experience. Some of the panelists admitted to not giving enough talks as new researchers; making such presentations can be hard, but they are a great way to increase your confidence.

Finding good students for your research group was covered next; panelists advised teaching graduate classes and being proactive in recruiting students. There are distinct advantages in implementing schemes that allow undergraduates to gain short periods of experience. The techniques covered for getting the best out of your research students reflected best practice used in industry. Primarily, it is important to set clear expectations and establish clear goals and milestones; preferably, these things will be written down. Tracy Camp has a document outlining student expectations on her Web pages. Establishing a mentoring hierarchy for students not only takes some of the pressure off advisors but also gives newer researchers experience in mentoring for themselves. The visibility and reputation of your research group are, of course, important, and recommendations mirrored those given for publicizing your own research. Network whenever you have the opportunity, volunteer for activities, and be willing to host talks and give talk tours.

■ ***On Program Security***

Hongxia Jin, IBM Almaden Research Center

In an email exchange with Hongxia, she said that she aimed to give the audience a quick technical overview of program security and, in particular, to give those who are interested in finding out more a good starting point. Hongxia described some elementary design principles for achieving software security based on her years of experience working in the area. She says that even though everybody's application context may be different, the same design principles should be applicable.

In describing the motivation for her research, Hongxia explained that hackers can reverse-engineer programs to understand or even modify existing programs quite easily. Consequently, competitors may learn trade secrets or copy algorithms to reuse in competing products. They can also remove protections and redistribute the pirated program for a profit. The general problem of program protection is widely thought to be impossible, implying that program protection is an important, wide-open problem. Hongxia described two methods used to defend against attacks. Preserving code integrity by guarding against tampering is one technique. Static integrity can be enforced using cryptographic hashing of all or part of a program, although this can impose a high processing overhead. But what about run-time integrity? You can detect the presence of a debugger, for example, by measuring execution times between particular sections of the code. The actions taken if tampering is detected also need to be given some thought. On the one hand, you could abort execution; on the other, it may be attractive to delay failure until a later time, inserting plausible yet misleading operations.

Finally, Hongxia gave an example of her research in this area. Bearing in mind that eventually a determined attacker will succeed, it is advantageous to detect on-going tampering as early as possible. In this way you may be able to restrict the extent of the eventual damage. One way in which you can do this is to use an auditing log to record hacking activities. A smart hacker can tamper with the log, but to counteract this you can make the log itself tamper-resistant. Hongxia described a scheme in which log entries are encrypted and transmitted back to a central clearing point. The source of the log entries and the clearing point share a key which is continually but independently evolved in both places. Even if the hackers succeed in subverting the encryption scheme, they cannot go backward, so some evidence of the attack will still be recorded.

■ ***Shifting the Tide of Network Security: Being Safe, Being Aware, and Being Active***

Nicole A. Pauls, TriGeo Network Security

In an email exchange with Nicole, she commented that she viewed the conference as an opportunity to talk about improving things for the women of tomorrow and an opportunity for everyone to see what women are doing today. Even though the focus of the conference is on women in academia, it was equally important to see women in industry talking about their research. In Nicole's presentation, she described pragmatic strategies for improving security. Many of us think of network security tools as necessary evils of prevention, but the truth is that we can't protect ourselves from everything. Software holes can be exploited before patches can be deployed; "trusted" users can become dangerous by opening the wrong email at the wrong time, and everything happens so quickly that we might not even know until it's too late. Nicole explored a defense in-depth network security strategy, covering architecture, monitoring, and active defenses. In terms of architecture, security must be something that we think about from the outset in the design of our networks and systems. We have to think about security on many levels, protecting each element so that it can still operate securely even if, say, the firewall fails. Detection is an essential weapon in our armory, which is where logging and auditing come in. Good organization and management are really the keys to success here; centralizing logging with a syslog server coupled with automated log analysis tools is the way to go. Deciding how you are going to respond to a security incident is equally important. First and foremost, you need to have a set of clearly documented policies clarifying the risks and responses. Your biggest problem here might be wrestling with the internal politics in an organization. On a practical level, you should not expect to do everything at once: prioritize and implement what you can, when you can.

■ ***Wireless Sensor Networks and Real-World Applications***

Nirupama Bulusu, Portland State University

In her talk Nirupama encouraged us to "come learn about the opportunities and computing challenges in wireless sensor networks." Although fairly broad, this talk illustrated how sensor technology is being applied in areas such as digitized health care, energy management, condition-based maintenance, and habitat monitoring. The projects

Nirupama chose to illustrate the applications of sensor technology were very exciting; the material would make a great educational pack, encouraging children and young adults to consider a career in computer science or engineering.

Most sensor applications involve monitoring either space or objects or both. In terms of monitoring space, sensors might be used for such things as environmental and habitat monitoring and precision agriculture. For example, a vineyard in Oregon uses sensors to monitor temperature and moisture; Roger the dog collects the data using a wireless collection device in his collar. The ZebraNet project monitoring zebra movement in Kenya was a somewhat more challenging application. Nodes (or zebras) are highly mobile and only sparsely populate the environment. Transferring data to a collection point (which may also be mobile) requires nodes to self-organize and actively route data.

The final example, an application to detect the presence of cane toads in the Australian outback, had a clever twist. The problem of how you detect a cane toad is interesting. It transpires that sensors can be used to hear when a cane toad is nearby by analyzing the acoustic features of the toad call, since cane toad calls have a completely different signature from those of other amphibians. Nirupama gave an "under the bonnet" view of some of the challenges presented by sensor-based applications. For example, localization, determining the position of sensors, and distributing this information to other nodes can be a complex task, yet workable solutions now exist. And what about the future? Sensor networks for urban applications will form the "next tier of the Internet," leveraging the cell phone installed base of acoustic and image sensors to capture readings.

■ *Part of the Problem/Part of the Solution*

Claudia Morrell, University of Maryland; Revi Sterling, University of Colorado; Sophia Huyer, Women and Global Science and Technology

The panel explored the issues affecting women surrounding deployment of Information and Communication Technology (ICT) in developing countries. Nongovernmental organizations (NGOs) are increasingly employing ICT tools as part of their strategy. Whereas ICTs show great promise in alleviating entrenched economic, health, and gender disparities, they may be exacerbating gender gaps. Despite the best intentions, many aid-focused initiatives intended to assist women's unique development goals are challenged with long-term sustainability and the ability to have wide-ranging impact upon the culture. Several women-friendly projects were presented. The Grameen-Phone scheme enables women to buy cell phones and rent usage, SchoolNet Africa provides computers to schools in Africa, and TeleCenters are springing up in Latin America and Africa. In the past year or so, important synergies have emerged. The 1st Women and ICT Symposium was held in 2005. At last it is possible to combine known information on emerging markets, people, and technologies. A taskforce was formed which has been recognized by the UN Global Alliance for ICT for Development as a community of expertise.

■ *The Impact of ICT on Women in Brazil*

Dilma M. Da Silva, T.J. Watson Research Laboratories

In her presentation, Dilma Da Silva reviewed the current status of ICT deployment in Brazil with respect to women. In the 1980s, the government imposed tight controls on the market, encouraging local development of software and hardware where possible. The 1990s were characterized by dramatic improvements in phone, cell, and networking services, but by around 1995 over 35,700 jobs were cut in the IT industry (representing a 48.1% reduction). At present, technology deployment is uneven. The Brazilian business triangle (São Paulo, Rio de Janeiro, and Belo Horizonte) has high-capacity fiber, virtual private networks, and bandwidth on a par with the United States and Europe, whereas most of the countryside has no access at all. But the government is actively promoting the use of electronic voting, and the percentage of tax returns completed online is growing.

So where do women fit into this picture? In 2000, the World Bank reported that Brazil has one of the widest gender gaps in Latin America. Even with the same qualifications as male colleagues, women got only 54% of what men received as wages and comprised only 20% of the IT workforce. Socially, women are not considered suitable for ICT-related jobs and this, in turn, influences women's interests and goals. Dilma also pointed out that despite these disappointing statistics, this is not the whole picture. The situation is reflected in my own experience: In my department there are two very talented Brazilian women studying for their Ph.D.s.

Announcement and Call for Papers USENIX ACCURATE

2007 USENIX/ACCURATE Electronic Voting Technology Workshop (EVT '07)

Sponsored by USENIX: The Advanced Computing Systems Association, and ACCURATE: A Center for Correct, Usable, Reliable, Auditable, and Transparent Elections

http://www.usenix.org/evt07

August 6, 2007 **Boston, Massachusetts, USA**

EVT '07 will be co-located with the 16th USENIX Security Symposium (Security '07), August 6–10, 2007.

Important Dates

Submissions due: *Sunday, April 22, 2007, 11:59 p.m. PDT*
Notification of acceptance: *Friday, June 1, 2007*
Final files due: *Thursday, June 28, 2007*

Workshop Organizers

Program Co-Chairs

Ray Martinez, *Martinez Consulting Group*
David Wagner, *University of California, Berkeley*

Program Committee

Ben Adida, *Harvard University*
Mike Alvarez, *California Institute of Technology*
Andrew Appel, *Princeton University*
Doug Jones, *University of Iowa*
Sharon Laskowski, *National Institute of Standards and Technology*
Dave Magelby, *Brigham Young University*
Margaret McGaley, *National University of Ireland, Maynooth*
Whitney Quesenbery, *Whitney Interactive Design*
Peter Ryan, *Newcastle University*
Dan Wallach, *Rice University*

Overview

In the United States and many other countries, most votes are counted and transported electronically, but the practical and policy implications of introducing electronic machines into the voting process are emerging in this new area. Both voting technology and its regulations are very much in flux, with open concerns including reliability, robustness, security, human factors, transparency, equality, privacy, and accessibility.

The USENIX/ACCURATE Electronic Voting Technology (EVT) workshop seeks to bring together researchers from a variety of disciplines, ranging from computer science and human factors experts through political scientists, legal experts, election administrators, and voting equipment vendors. EVT seeks to publish original research on important problems, including how the software and hardware in voting might be engineered to be more robust against tampering or how it might be written to be more easily and openly verified. Papers exploring "end-to-end" approaches that strive to ensure that the integrity of the election is independent of software and hardware are also encouraged. EVT also welcomes submissions on how these systems might be engineered to be more usable by the broad voting population. EVT also seeks discussion of how election regulations and standards may evolve to support better election technologies. Additionally, EVT encourages position papers on the practicality (or impracticality) of the technological advances in electronic voting, particularly with the limited budgets available to many elections administrators. EVT will consider papers covering the gamut of technology as it is used in elections, ranging from voter registration and vote collection through tabulation and post-election auditing. We are interested in both future technologies and systems widely used today around the world.

EVT '07 will be a one-day event, Monday, August 6, 2007, co-located with the 16th USENIX Security Symposium in Boston, Massachusetts. In addition to paper presentations, the workshop may include panel discussions with substantial time devoted to questions and answers. The proceedings of the workshop will be published electronically. Attendance at the workshop will be open to the public, although talks and refereed paper presentations will be by invitation only.

In particular, we welcome papers considering:

- Design and analysis of electronic voting schemes and protocols
- Deployment and lifecycle concerns
- Mitigating threats (including insider threats)
- Usability and accessibility (both for voters and for administrators)
- Legal issues, including how voting systems must comply with the ADA and HAVA, or the effect of intellectual property rights and nondisclosure agreements on voting system testing, certification, and deployment
- The technology standards process and how it should evolve

Submission Instructions

All submissions must be in English and must include a title and the authors' names and affiliations. We will accept both short position papers (i.e., up to six [6] pages long) and longer, conference-style submissions (up to a maximum of sixteen [16] pages). Please format papers in two columns, single-spaced, using no smaller than 11 point Times Roman type in a text block of 6.5" by 9".

Each submission should have a contact author who should provide full contact information (email, phone, fax, mailing address). One author of each accepted paper will be required to present the work at the workshop.

Authors are required to submit papers by 11:59 p.m. PDT, April 22, 2007. **This is a hard deadline; no extensions will be given.** All submissions to EVT '07 must be electronic, in PDF format, via a Web form, which will be available on the EVT '07 Call for Papers Web site, http://www.usenix.org/evt07/cfp. Authors are encouraged to follow the U.S. National Science Foundation's guidelines for preparing PDF grant submissions:

- https://www.fastlane.nsf.gov/documents/pdf_create/pdfcreate_01.jsp

Simultaneous submission of the same work to multiple venues, submission of previously published work, and plagiarism constitute dishonesty or fraud. USENIX, like other scientific and technical conferences and journals, prohibits these practices and may, on the recommendation of a program chair, take action against authors who have committed them. In some cases, program committees may share information about submitted papers with other conference chairs and journal editors to ensure the integrity of papers under consideration. If a violation of these principles is found, sanctions may include, but are not limited to, barring the authors from submitting to or participating in USENIX conferences for a set period, contacting the authors' institutions, and publicizing the details of the case.

Authors uncertain whether their submission meets USENIX's guidelines should contact the program chair at evt07chairs@usenix.org or the USENIX office, submissionspolicy@usenix.org.

Accepted material may not be published in other conferences or journals for one year from the date of acceptance by USENIX. Papers accompanied by nondisclosure agreement forms will not be read or reviewed. All submissions will be held in confidence prior to publication of the technical program, both as a matter of policy and in accordance with the U.S. Copyright Act of 1976.

Authors will be notified of acceptance decisions via email by June 1. If you do not receive notification by that date, contact the Program Chairs at evt07chairs@usenix.org.

Registration Materials

Complete program and registration information will be available in June 2007 on the workshop Web site. The information will be in both HTML and PDF. If you would like to receive the latest USENIX conference information, please join our mailing list: http://www.usenix.org/about/mailing.html.

Rev. 12/22/06

Join us in Cambridge, MA, April 11–13, 2007, for the 4th USENIX Symposium on Networked Systems Design and Implementation (NSDI '07), which will focus on the design principles of large-scale networks and distributed systems. Join researchers from across the networking and systems community—including computer networking, distributed systems, and operating systems—in fostering cross-disciplinary approaches and addressing shared research challenges.

Register by Monday, March 19, and save: http://www.usenix.org/nsdi07

NSDI '07 will be co-located with the following workshops, all of which will be held on April 10, 2007:

- First Workshop on Hot Topics in Understanding Botnets (HotBots '07)
  http://www.usenix.org/hotbots07
- Second Workshop on Tackling Computer Systems Problems with Machine Learning Techniques (SysML07)
  http://www.cs.duke.edu/nicl/sysml07
- Third International Workshop on Networking Meets Databases (NetDB '07)
  http://www.usenix.org/netdb07

NSDI '07

Sponsored by

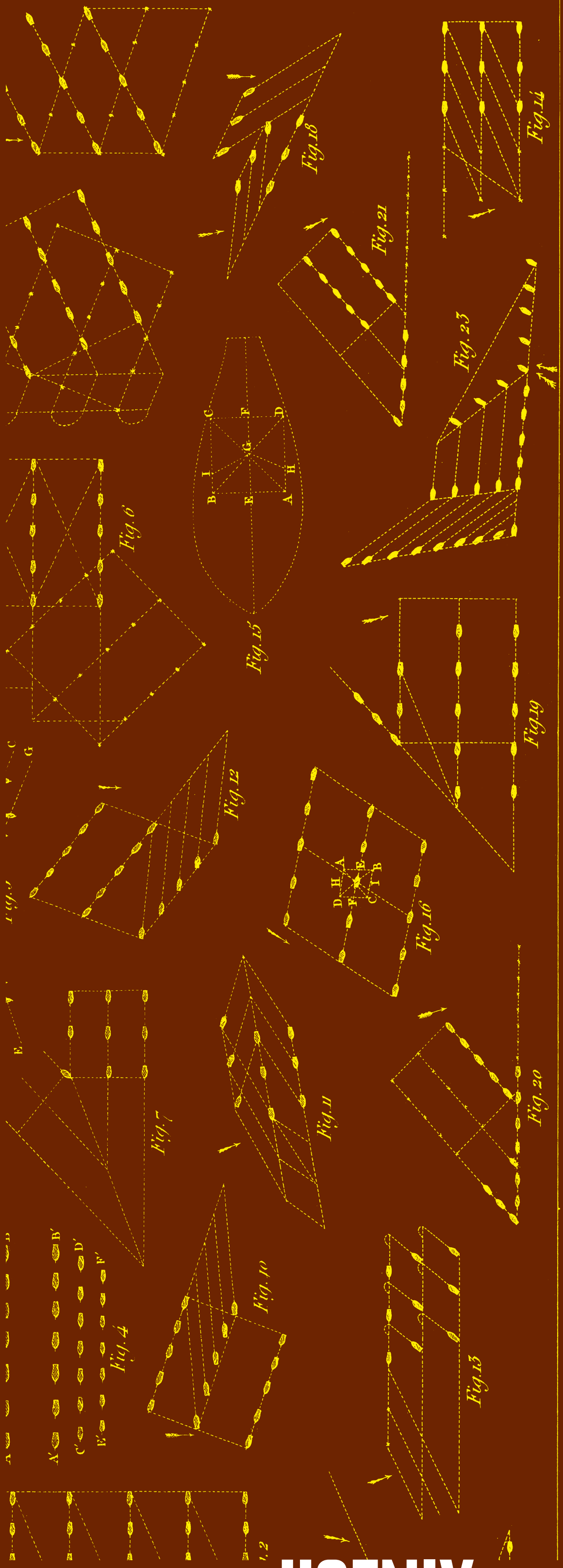
USENIX

in cooperation with ACM SIGCOMM and ACM SIGOPS

http://www.usenix.org/nsdi07

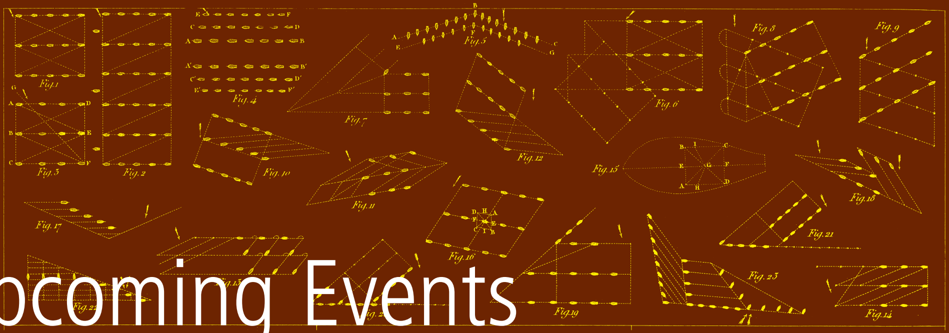
FEBRUARY 2007 VOLUME 32 NUMBER 1

;login:



USENIX
The Advanced Computing Systems Association

USENIX Upcoming Events



First Workshop on Hot Topics in Understanding Botnets (HotBots '07)

Co-located with NSDI '07

APRIL 10, 2007, Cambridge, MA, USA

http://www.usenix.org/hotbots07
Paper submissions due: February 26, 2007

Second Workshop on Tackling Computer Systems Problems with Machine Learning Techniques (SysML07)

Co-located with NSDI '07

APRIL 10, 2007, Cambridge, MA, USA

http://www.cs.duke.edu/nicl/sysml07

Third International Workshop on Networking Meets Databases (NetDB '07)

Co-located with NSDI '07
Sponsored by USENIX in cooperation with ACM SIGCOMM

APRIL 10, 2007, Cambridge, MA, USA

http://www.usenix.org/netdb07

4th USENIX Symposium on Networked Systems Design and Implementation (NSDI '07)

Sponsored by USENIX in cooperation with ACM SIGCOMM and ACM SIGOPS

APRIL 11–13, 2007, Cambridge, MA, USA

http://www.usenix.org/nsdi07

11th Workshop on Hot Topics in Operating Systems (HotOS XI)

Sponsored by USENIX in cooperation with the IEEE Technical Committee on Operating Systems (TCOS)

MAY 7–9, 2007, San Diego, CA, USA

http://www.usenix.org/hotos07

5th ACM/USENIX International Conference on Mobile Computing Systems, Applications, and Services (MobiSys 2007)

Jointly sponsored by USENIX and ACM SIGMOBILE, in cooperation with ACM SIGOPS

JUNE 11–15, 2007, Puerto Rico

http://www.sigmobile.org/mobisys/2007/

Workshop on Experimental Computer Science (ECS '07)

Sponsored by ACM SIGARCH and ACM SIGOPS in cooperation with USENIX, ACM SIGCOMM, and ACM SIGMETRICS

JUNE 13–14, 2007, San Diego, CA, USA

http://www.expcs.org/
Paper submissions due: February 9, 2007

Third International ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments (VEE '07)

Sponsored by ACM SIGPLAN and ACM SIGOPS in cooperation with USENIX

JUNE 13–15, 2007, San Diego, CA, USA

http://vee07.cs.ucsb.edu
Paper submissions due: February 5, 2007

2007 USENIX Annual Technical Conference

JUNE 17–22, 2007, Santa Clara, CA, USA

http://www.usenix.org/usenix07

Third Workshop on Hot Topics in System Dependability (HotDep '07)

Co-sponsored by USENIX

JUNE 26, 2007, Edinburgh, UK

http://hotdep.org/2007
Paper submissions due: February 15, 2007

2007 USENIX/ACCURATE Electronic Voting Technology Workshop (EVT '07)

Co-located with Security '07

AUGUST 6, 2007, Boston, MA, USA

http://www.usenix.org/evt07
Paper submissions due: April 22, 2007

16th USENIX Security Symposium

AUGUST 6–10, 2007, Boston, MA, USA

http://www.usenix.org/sec07
Paper submissions due: February 1, 2007

2007 Linux Kernel Developers Summit

SEPTEMBER 4–6, 2007, Cambridge, U.K.

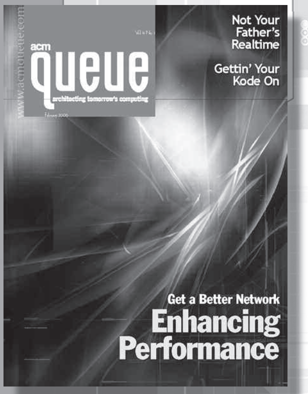
21st Large Installation System Administration Conference (LISA '07)

Sponsored by USENIX and SAGE

NOVEMBER 11–16, 2007, Dallas, TX

For a complete list of all USENIX & USENIX co-sponsored events, see http://www.usenix.org/events.

there's a whole lot of technology in the queue. are you ready?
what's next?
acm



Get ready with ACM Queue—the technology magazine focused on problems that don't have easy answers—yet.

Queue dissects the challenges of emerging technologies.
Queue targets the problems and pitfalls just ahead.
Queue helps you plan for the future.
Queue poses the hard questions you'd like to ask.

Isn't that what you've been looking for?

www.acmqueue.com

Get your FREE subscription now at www.acmqueue.com

www.acmqueue.com

2007 USENIX Annual Technical Conference

June 17–22, 2007, Santa Clara, California, USA

Join us in Santa Clara, CA, June 17–22, for the 2007 USENIX Annual Technical Conference. USENIX Annual Tech has always been the place to present groundbreaking research and cutting-edge practices in a wide variety of technologies and environments. USENIX '07 will be no exception.

USENIX '07 WILL FEATURE:

- An extensive Training Program, covering crucial topics and led by highly respected instructors
- Technical Sessions, featuring the Refereed Papers Track, Invited Talks, and a Poster Session
- Plus BoFs and more!

http://www.usenix.org/usenix07

;login:

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710

POSTMASTER
Send Address Changes to *;login:*
2560 Ninth Street, Suite 215
Berkeley, CA 94710

PERIODICALS POSTAGE
PAID
AT BERKELEY, CALIFORNIA
AND ADDITIONAL OFFICES