;login:

THE USENIX MAGAZINE

August 2004 • volume 29 • number 4

inside:

USENIX

The Advanced Computing Systems Association

USENIX Upcoming Events

The 4th International System Administration and Network Engineering Conference (SANE 2004)

SEPT. 27–OCT. 1, 2004, Amsterdam, The Netherlands
Web site: http://www.sane.nl

Internet Measurement Conference 2004 (IMC 2004)

Sponsored by ACM SIGCOMM in cooperation with USENIX
OCTOBER 25–27, 2004, Taormina, Sicily, Italy
Web site: http://www.icir.org/vern/imc/

18th Large Installation System Administration Conference (LISA '04)

NOVEMBER 14–19, 2004, Atlanta, GA, USA
Web site: http://www.usenix.org/lisa04

6th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA '04)

Co-sponsored by TCOS/TCI and USENIX
DECEMBER 2–3, 2004, English Lake District, UK
Web site: http://wmcsa2004.lancs.ac.uk/

First Workshop on Real, Large Distributed Systems (WORLDS '04)

DECEMBER 5, 2004, San Francisco, CA, USA
Web site: http://www.usenix.org/worlds04
Paper submissions due: August 1, 2004

Sixth Symposium on Operating Systems Design and Implementation (OSDI '04)

DECEMBER 6–8, 2004, San Francisco, CA, USA
Web site: http://www.usenix.org/osdi04

2005 USENIX Annual Technical Conference

APRIL 10–15, 2005, Anaheim, CA, USA
Web site: http://www.usenix.org/usenix05
Paper submissions due: October 18, 2004

2nd Symposium on Networked Systems Design and Implementation (NSDI '05)

MAY 2–4, 2005, Boston, MA, USA
Web site: http://www.usenix.org/nsdi05
Paper submissions due: October 8, 2004

10th Workshop on Hot Topics in Operating Systems (HotOS X)

JUNE 12–15, 2005, Santa Fe, NM, USA
Paper titles and abstracts due: February 1, 2005
Full paper submissions due: May 2, 2005

14th USENIX Security Symposium

AUGUST 1–5, Baltimore, MD, USA

19th Large Installation System Administration Conference (LISA '05)

DECEMBER 4–9, San Diego, CA, USA

4th USENIX Conference on File & Storage Technologies (FAST '05)

DECEMBER 12–14, San Francisco, CA, USA

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

contents

;login: Vol. 29, #4, August 2004

;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.

$80 of each member's annual dues is for an annual subscription to *;login:*. Subscriptions for nonmembers are $110 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.

motd

by Rob Kolstad

Dr. Rob Kolstad has long served as editor of ;login:. He is SAGE's Executive Director, and also head coach of the USENIX-sponsored USA Computing Olympiad.
<kolstad@usenix.org>

Failing

One can hardly attend an educational institution in our country without being warned against failure. "Failure is bad," "Don't fail!" and "Don't be a failure!" are implicit, if not stated, messages.

I am not sure that failing is so awfully bad. In fact, if one looks around, one can see examples in many different venues.

- Water skiing without falling (failing)? Not trying hard enough.
- Performing sysadmin activities without failing at least in test environments? Not pushing hard enough.
- Make a mistake on a calculation (failure), catch it, and fix it? No problem.

I don't think "failure" is the problem. The real problem is "failing slowly" or "failing in some undetected manner" (ignoring, for the moment, "failing fatally," e.g., driving a car over a cliff).

Consider writing code in a scripting language. In these days of pretty darn fast computers, one can write some erroneous code, try to run it (but fail), fix the bug, and succeed—all in a manner of seconds. That sort of failure doesn't hurt at all! The amount of time it takes to make sure you typed semicolon instead of colon dwarfs the time an interpreter can check it for you. In fact, one can even develop incrementally, testing five lines at a time. The series of tiny failures pales next to trying to debug a 1,000-line script riddled with mistakes.

This idea generalizes, I think:

- Home repair: turn the little screw on the sprinkler head. Too much water (failure)? Turn it the other way!
- Riding a bicycle: turn the wheel one way and fall (failure). Turn it the other way!
- Cooking: burned the {toast, etc.}? Cook it less!

Quick failures (short of catastrophes and deaths) are dandy learning opportunities and can help us all grow as humans.

Failing slowly, on the other hand, has serious drawbacks. Think of the poor college student who majored in Albanian Poetry when she really wanted to be a computer programmer. What a miserable career she has until she realizes she took the wrong fork in the road and backtracks. Unfortunately, this can take years.

Certain projects or even relationships can fail still more slowly. The US divorce rate is a symptom of this. Couples work hard for some period of time and then realize that all those years invested in the marriage are not going to enable it to succeed. Of course, this is sometimes hard to foresee by those involved.

The initial foundations of "Extreme Programming" worked against failing slowly. Frequent (sometimes daily) checkpoints enabled implementors to know when they had taken even the slightest wrong turn. There is wisdom here, even if XP is not for everyone.

Failing fatally is still no good, of course. One must supervise infants and children. One must use power tools of all sorts very carefully. Using safety devices such as seat belts, safety goggles, and personal flotation devices is only common sense. I don't advocate failing when personal injury or death might result.

I routinely perform experiments so that I fail on a small scale instead of a large one:

- For my lawn extension, I planted 15 3x50 foot test plots of various grasses to see which would grow in the combination of sun, soil, and watering conditions of the new yard. An unexpected dividend was the revelation that grasses are very different—textures, widths, colors, and general impression vary widely. I chose one of the top two grasses,

and the new lawn has gone well. Of course, I failed for a year on getting a new lawn—and I failed with 13 kinds of grass. The alternative of failing slowly after the dozens of person-hours required to put in a new lawn was quite unappealing, though.

- I practice experimental cooking. I frequently glean a few dozen recipes from the Net in an effort to learn "essence of Chicken Cordon Bleu." I then create an amalgam that appears to hit the high points. I reckon I have a failure rate of only about 2%—and the successes are delicious. But this entire endeavor wouldn't be possible if failure weren't an option.
- The new wine cellar has a white LED that can shine on any given wine bottle's label. They're all individually addressable. Suggested as a stupid extreme idea by a lunch partner, the potential for "coolness" was undeniable, so we did experiments. First, we acquired a handful of different bright LEDs in a few colors. After a few experiments, a dozen of the best were acquired. Then we built a small PIC circuit to cycle the half-dozen winning LEDs across a single row of bottles. That brought the project to the third go/no-go point. I'm happy to report that 1/3 of the cabinets are ready to install and another 1/3 of the light circuits have been constructed. I hope that it turns out cool. If it's not, then this will become an example of failing slowly.

Failure *is* an option. Don't be afraid to make nonfatal mistakes if they are mistakes that can easily be recovered from. You might find your quality increasing side-by-side with your throughput.

Of course, success is always an option, too. Never underrate it!

;login:

EDITORIAL STAFF

EDITOR
Rob Kolstad *kolstad@usenix.org*

CONTRIBUTING EDITOR
Tina Darmohray *tmd@usenix.org*

MANAGING EDITOR
Jane-Ellen Long *jel@usenix.org*

COPY EDITOR
Steve Gilmartin *proofshop@usenix.org*

PROOFREADER
ProofShop *proofshop@usenix.org*

MEMBERSHIP, PUBLICATIONS, AND CONFERENCES
USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710
Phone: 510 528 8649
FAX: 510 548 5738
Email: *office@usenix.org*
login@usenix.org
conference@usenix.org
WWW: *http://www.usenix.org*
http://www.sage.org

apropos

by Tina Darmohray

Tina Darmohray, contributing editor of ;login:, is a computer security and networking consultant. She was a founding member of SAGE and has been a Director of USENIX.

tmd@usenix.org

Sysadmin Jobs: The CIO

What's It Take?

I've been thinking about system administration job descriptions again, this time in the context of who manages system administrators and what they know and do.

Just as the proliferation of computers required professionals to take care of them, keep them up and running and talking to each other, the proliferation of system administrators has necessitated a management chain to manage the overall effort.

The literature reflects that in the early 1980s we began to see CIO (Chief Information Officer) positions. This C-level executive arrived on the scene from a variety of places: some organizations simply promoted existing directors of MIS, others attached CIOs to COOs or CFOs. Depending on the organization, there might also be a CTO (Chief Technical Officer), or the CIO could be wearing both those hats. One thing these positions usually have in common, though, is that somewhere down the line they are managing folks who are managing systems.

I asked a number of colleagues, working in a variety of industries as well as education and government, "To the extent that a CIO is the 'ultimate' system administrator manager, what qualifications should the CIO have?"

First off, a couple of people who responded cautioned that the CIO in fact isn't just "the ultimate system administrator manager." They emphasized that the CIO has a larger scope than just system administration, e.g., networking, DBAs, help desk, desktop services, and more. In summary, the CIO is responsible for all aspects of the data at a particular organization; managing the systems is part of the equation, but it is not the whole of the job.

Of course I have my own bias on what it takes be a good CIO, and how I'd prioritize those qualities, but I'll leave my vote out of it and present the findings of my nonscientific survey.

The top vote-getter is a focus on funds/budgets: folks want their CIO to be able to assess, predict, and manage the bottom line. Specific experience in managing budgets and in both funding and managing IT projects was mentioned. But probably the most insightful comments centered on a CIO's understanding of running a service for their organization, and that the measure of their success was how well their efforts supported the organization's overall success. This attribute increases in importance the more information-intensive the organization is.

People management comes next. Some folks quantify it, saying that a CIO should have at least five years of management experience. Where the experience was gained may not be significant, as it's recognized that the most important management knowledge transcends the details of the environment. But no matter how you slice it, management experience is essential. One person laid it on the line with this comment, "If you haven't earned your chops, no one will respect you."

Tied with measurable management experience comes the less quantifiable "vision thing." The CIO must be able to absorb knowledge of emergent products and services and set high-level technical direction for the organization. It's not enough to be able to see the big picture; the CIO must be able to communicate that view to their employees and persuade them to head in the stated direction. So the "vision thing" in turn requires the "communication and leadership thing."

Rounding out a three-way tie is what I'll call technical prowess, because the feedback on what is needed is split between hands-on technical experience and having enough technical background to make technology judgment calls. Regardless of how you get it, the CIO needs to possess a technical background in order to understand what technical people who disagree are saying and to pick the right solution.

Perhaps more of a warning than a qualification: tenacity is felt to be key. Some war stories seemed to poke through with this recommendation: "If they aren't tenacious, they become the executive-level whipping boy," and "A CIO is more a management than an IT expert, because s/he will have to interface with other high-level management and without the correct political skills, s/he won't survive or won't be able to help her/his staff."

Clumped together at the bottom of the list are the need to be aware of security issues and a proven track record in technology integration and implementation. These two are near and dear to my own heart, and since they are the only fairly specific areas of knowledge mentioned, I took that as a positive sign.

Financial and management experience, vision, leadership, technical prowess, tenacity, security awareness, and a proven track record in technology integration and implementation: These are the qualities that industry professionals need their CIOs to have. Does yours have what it takes?

Thanks to Wendy Nather, Debby Hungerford, Shawn Instenes, Philip de Louraille, and a dozen anonymous respondents.

USA Computing Olympiad correspondence

[Editor's note: USENIX is a major sponsor of USACO. These are a sampling of the thank-you notes received at USENIX subsequent to the USACO finals.—RK]

Thank you so much for your generous support of the USACO program. My son, Adam, has participated in USACO for two years. He was fortunate to be invited to the training camp both years, and he was chosen for the B-team at last year's IOI. These experiences have been among the most meaningful of his high school career. The program is very well run. The coaching staff has been nothing short of outstanding. USACO has been not only educational but also a lot of fun for Adam. He has always been keenly interested in computer science, but now, as a result of his USACO involvement, he is more excited than ever about studying computer science next year at MIT.

So, once again, thank your for your support of this very important program.

Sincerely,

Nancy, Don, and Adam Rosenfield

I am a sophomore at Barrington High School in Rhode Island, and a finalist at last month's USACO training camp. I would like to thank USENIX for supporting the USACO program, and urge you to continue your sponsorship of this unique and invaluable experience in programming.

The USACO program is challenging, extremely difficult, and intense. It is also very educational and instructive, and is excellent training for future study and careers. I can easily exhaust my course work in CS, but I can never exhaust USACO problems and contests. USACO also gives me a shot at competing internationally, which I could never do with training at my school. I also get a chance to meet talented programmers from all over the country, and form friendships with some of the best people in the field.

I would like to thank you again for sponsoring this program, and I hope that you continue your commitment to America's young computer scientists by supporting USACO.

Sincerely,

Joe Zimmerman

I would like to thank you for supporting the USA Computing Olympiad. The USACO provides challenging algorithmic problems which have improved my programming skills enormously. My school's computer club has looked for other contests to use, but none are nearly so high in quality and difficulty as the USACO. In fact, my brother (who attended the ACM finals the past two years) says that ACM is easier and less well done than USACO and IOI. Also, the USACO provides online training pages which help newcomers learn algorithms. The training camp was a very valuable experience where I could meet other students as interested in computers as myself. I was fortunate enough to attend the International Olympiad in Informatics last year and look forward to going this year, and the USACO program provides the opportunity to meet international students interested in computers as well.

Sincerely,

Eric Price

My son has just qualified as a USACO team member to IOI in Greece this September. I am so grateful to you for your sponsorship of this wonderful competition and series of contests leading to the team selection. These high-level contests have been pivotal in my son's life. He tends to be shy, and has not been well understood by most of the rest of the world, who cannot begin to relate to him and his passion for computer programming and problem solving. Through the USACO programs you sponsor, he has met others like him, both peers and adults, and has found a world where he fits in and excels. I know you may think all you are sponsoring is excellence in computing, but for my son you are sponsoring a wonderful life to reach an amazing potential as well. I cannot thank you enough.

Many blessings,

Vicky Kaseorg

letter to editor Rik Farrow

Hey Rik,

I was wondering if you could help me with some work advice. I am a software engineer for a major financial firm based in NYC. As you know, there was a huge jump in off-shoring development to India and other countries. I hear some really bad forecasts as far as software development in the US is concerned. I love the whole process of developing software. I love creating something from nothing. It's just a grown-up's version of building with Lego blocks which I *loved* to do as a kid.

What do you think will happen with the US software industry? What can I do to circumvent negative results? What would be a good graduate degree to pursue?

Thanks,

Isaac
compuder@aol.com

Rik responds:

Hi Isaac:

What I have seen in the US is that a lot of the most senior people have lost their jobs, and many have managed to get hired at about half of what they were making before.

In other words, having an advanced degree is not something that will help you in today's market—if anything, it will hurt you.

That is not to say that things will not change in the future. There is always a place for people who become experts in niche areas. If you have a passion for a particular technology niche, such as advanced networking, databases, data "farming," to name a few, then getting an advanced degree that focused on that niche would be a good step for the future of your career.

As for the US programmer market, the US, and the world, is currently in a slump for two reasons. First, the dot-bomb has shifted people's perceptions of technologists from gods to dogs, and we are suffering because of that. Over time, the pendulum will swing back—our society cannot function without its computers, and the software they use is more complex and demanding than ever.

The second is the outsourcing of computer jobs (not just programming, but help desk and system/network management). Modern communications means that the people doing this work can be located anywhere. The downside of this is that real communication between software developers and program managers will get even worse (it has never been very good). Soon enough, people will realize that they get what they pay for—cut-rate prices for cut-rate programming. And, on top of that, they spend an enormous amount of money that goes into the pockets of middlemen who do nothing to increase the quality of the end product.

That's what I currently think.

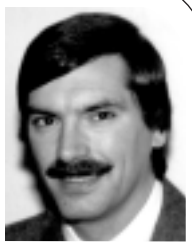
Regards,

Rik

[Editor's note: Rik's assessment is defensible. Do you have a different one? Please send it to *login@usenix.org.*—RK]

the jail facility in FreeBSD 5.2

by Kirk McKusick



Dr. Marshall Kirk McKusick writes books and articles, consults, and teaches classes on UNIX- and BSD-related subjects. He has twice served on the Board and as president of USENIX.

mckusick@mckusick.com

The FreeBSD access control mechanism is designed for an environment with two types of users: those with and those without administrative privilege. It is often desirable to delegate some but not all administrative functions to untrusted or less trusted parties and simultaneously impose system-wide mandatory policies on process interaction and sharing. Historically, attempting to create such an environment has been both difficult and costly. The primary mechanism for partial delegation of administrative authority is to write a set-user-identifier program that carefully controls which of the administrative privileges may be used. These set-user-identifier programs are complex to write, difficult to maintain, limited in their flexibility, and prone to bugs that allow undesired administrative privilege to be gained.

[**Editor's note:** This article is a partial excerpt from Chapter 4, "Process Management," from *The Design and Implementation of the FreeBSD Operating System*, by Marshall Kirk McKusick and George Neville-Neil. Reprinted with permission from Pearson Education, Inc. (0-201-70245-2). Copyright 2005. To learn more: *http://www.awprofessional.com/title/0201702452*.]

Many operating systems attempt to address these limitations by providing fine-grained access controls for system resources [P1003.1e, 1998]. These efforts vary in degrees of success, but almost all suffer from at least three serious limitations:

1. Increasing the granularity of security controls increases the complexity of the administration process, in turn increasing both the opportunity for incorrect configuration, as well as the demand on administrator time and resources. Often the increased complexity results in significant frustration for the administrator, which may result in two disastrous types of policy: running with security features disabled and running with the default configuration on the assumption that it will be secure.
2. Usefully segregating capabilities and assigning them to running code and users is difficult. Many privileged operations in FreeBSD seem independent but are interrelated. The handing out of one privilege may be transitive to many others. For example, the ability to mount file systems allows new set-user-identifier programs to be made available that in turn may yield other unintended security capabilities.
3. Introducing new security features often involves introducing new security management interfaces. When fine-grained capabilities are introduced to replace the set-user-identifier mechanism in FreeBSD, applications that previously did an appropriateness check to see if they were running with superuser privilege before executing must now be changed to know that they need not run with superuser privilege. For applications running with privilege and executing other programs, there is now a new set of privileges that must be voluntarily given up before executing another program. These changes can introduce significant incompatibility for existing applications and make life more difficult for application developers who may not be aware of differing security semantics on different systems.

This abstract risk becomes more clear when applied to a practical real-world example: Many Web service providers use FreeBSD to host customer Web sites. These providers must protect the integrity and confidentiality of their own files and services from their customers. They must also protect the files and services of one customer from (accidental or intentional) access by any other customer. A provider would like to supply

substantial autonomy to customers, allowing them to install and maintain their own software and to manage their own services, such as Web servers and other content-related daemon programs.

This problem space points strongly in the direction of a partitioning solution. By putting each customer in a separate partition, customers are isolated from accidental or intentional modification of data or disclosure of process information from customers in other partitions. Delegation of management functions within the system must be possible without violating the integrity and privacy protection between partitions.

FreeBSD-style access control makes it notoriously difficult to compartmentalize functionality. While mechanisms such as chroot provide a modest level of compartmentalization, this mechanism has serious shortcomings, both in the scope of its functionality and the effectiveness of what it provides. The chroot system call was first added to provide an alternate build environment for the system. It was later adapted to isolate anonymous FTP access to the system.

The original intent of chroot was not to ensure security. Even when used to provide security for anonymous FTP, the set of operations allowed by FTP was carefully controlled to prevent those that allowed escape from the chrooted environment.

Three classes of escape from the confines of a chroot-created file system were identified over the years:

1. Recursive chroot escapes
2. Escapes using ..
3. Escapes using fchdir

All these escapes exploited the lack of enforcement of the new root directory.

Two changes to chroot were made to detect and thwart these escapes. To prevent the first two escapes, the directory of the first level of chroot experienced by a process is recorded. Any attempts to traverse backward across this directory are refused. The third escape, using fchdir, is prevented by having the chroot system call fail if the process has any file descriptors open referencing directories.

Even with stronger semantics, the chroot system call is insufficient to provide complete partitioning. Its compartmentalization does not extend to the process or networking spaces. Therefore, both observation of and interference with processes outside their compartment is possible. To provide a secure virtual machine environment, FreeBSD added a new "jail" facility built on top of chroot. Processes in a jail are provided full access to the files that they may manipulate, processes they may influence, and network services they may use. They are denied access to and visibility of files, processes, and network services outside their jail [Kamp & Watson, 2000].

Unlike other fine-grained security solutions, a jail doesn't substantially increase the policy management requirements for the system administrator. Each jail is a virtual FreeBSD environment that permits local policy to be independently managed. The environment within a jail has the same properties as the host system. Thus, a jail environment is familiar to the administrator and compatible with applications [Hope,

The administrator of a FreeBSD machine [can] partition the host into separate jails and provide access to the superuser account in each of these jails without losing control of the host environment.

2002].

Jail Semantics

Two important goals of the jail implementation are to:

1. Retain the semantics of the existing discretionary access-control mechanisms.
2. Allow each jail to have its own superuser administrator whose activities are limited to the processes, files, and network associated with its jail.

The first goal retains compatibility with most applications. The second goal permits the administrator of a FreeBSD machine to partition the host into separate jails and provide access to the superuser account in each of these jails without losing control of the host environment.

A process in a partition is referred to as being "in jail." When FreeBSD first boots, no processes will be jailed. Jails are created when a privileged process calls the jail system call with arguments of the file system into which it should chroot and the IP address and hostname to be associated with the jail. The process that creates the jail will be the first and only process placed in the jail. Any future descendants of the jailed process will be in its jail. A process may never leave a jail that it created or in which it was created. Any given process may be in only one jail. The only way for a new process to enter the jail is by inheriting access to the jail from another process already in that jail.

Each jail is bound to a single IP address. Processes within the jail may not make use of any other IP address for outgoing or incoming connections. A jail has the ability to restrict the set of network services that it chooses to offer at its address. An application request to bind all IP addresses is redirected to the individual address associated with the jail in which the requesting process is running.

A jail takes advantage of the existing chroot behavior to limit access to the file system namespace for jailed processes. When a jail is created, it is bound to a particular file-system root. Processes are unable to manipulate files that they cannot address. Thus, the integrity and confidentiality of files outside the jail file-system root are protected.

Processes within the jail will find that they are unable to interact or even verify the existence of processes outside the jail. Processes within the jail are prevented from delivering signals to processes outside the jail, connecting to processes outside the jail with debuggers, or even seeing processes outside the jail with the usual system-monitoring mechanisms. Jails do not prevent, nor are they intended to prevent, the use of covert channels or communications mechanisms via accepted interfaces. For example, two processes in different jails may communicate via sockets over the network. Jails do not attempt to provide scheduling services based on the partition.

Jailed processes are subject to the normal restrictions present for any processes including resource limits and limits placed by the network code, including firewall rules. By specifying firewall rules for the IP address bound to a jail, it is possible to place connectivity and bandwidth limitations on that jail, restricting the services that it may consume or offer.

The jail environment is a subset of the host environment. The jail file system appears as part of the host file system and may be directly modified by processes in the host environment. Processes within the jail appear in the process listing of the host and may be signaled or debugged.

Processes running without superuser privileges will notice few differences between a jailed environment and an unjailed environment. Standard system services such as remote login and mail servers behave normally, as do most third-party applications, including the popular Apache Web server. Processes running with superuser privileges will find that many restrictions apply to the privileged calls they may make. Most of the limitations are designed to restrict activities that would affect resources outside the jail. These restrictions include prohibitions against the following:

- Modifying the running kernel by direct access or loading kernel modules.
- Mounting and unmounting file systems.
- Creating device nodes.
- Modifying kernel runtime parameters such as most sysctl settings.
- Changing security-level flags.
- Modifying any of the network configuration, interfaces, addresses, and routing-table entries.
- Accessing raw, divert, or routing sockets. These restrictions prevent access to facilities that allow spoofing of IP numbers or the generation of disruptive traffic.
- Accessing network resources not associated with the jail. Specifically, an attempt to bind a reserved port number on all available addresses will result in binding only the address associated with the jail.
- Administrative actions that would affect the host system, such as rebooting.

Other privileged activities are permitted as long as they are limited to the scope of the jail:

- Signaling any process within the jail is permitted.
- Deleting or changing the ownership and mode of any file within the jail is permitted, as long as the file flags permit the requested change.
- The superuser may read a file owned by any UID, as long as it is accessible through the jail file system namespace.
- Binding reserved TCP and UDP port numbers on the jail's IP address is permitted.

These restrictions on superuser access limit the scope of processes running with superuser privileges, enabling most applications to run unhindered but preventing calls that might allow an application to reach beyond the jail and influence other processes or system-wide configuration.

Jail Implementation

The implementation of the jail system call is straightforward. A prison data structure is allocated and populated with the arguments provided. The prison structure is linked to the process structure of the calling process. The prison structure's reference count is set to one, and the chroot system call is called to set the jail's root. The prison structure may not be modified once it is created.

Hooks in the code implementing process creation and destruction maintain the reference count on the prison structure and free it when the last reference is released. Any new processes created by a process in a jail will inherit a reference to the prison structure, which puts the new process in the same jail.

Some changes were needed to restrict process visibility and interaction. The kernel interfaces that report running processes were modified to report only the processes in the same jail as the process requesting the process information. Determining whether

Making the jail environment appear to be a fully functional FreeBSD system allows maximum application support and the ability to offer a wide range of services within a jail environment.

one process may send a signal to another is based on UID and GID values of the sending and receiving processes. With jails, the kernel adds the requirement that if the sending process is jailed, then the receiving process must be in the same jail.

Several changes were added to the networking implementation:

- Restricting TCP and UDP access to just one IP number was done almost entirely in the code that manages protocol control blocks. When a jailed process binds to a socket, the IP number provided by the process will not be used; instead, the preconfigured IP number of the jail is used.
- The loop-back interface, which has the magic IP number 127.0.0.1, is used by processes to contact servers on the local machine. When a process running in a jail connects to the 127.0.0.1 address, the kernel must intercept and redirect the connection request to the IP address associated with the jail.
- The interfaces through which the network configuration and connection state may be queried were modified to report only information relevant to the configured IP number of a jailed process.

Device drivers for shared devices such as the pseudo-terminal driver needed to be changed to enforce the restriction that a particular virtual terminal cannot be accessed from more than one jail at the same time.

The simplest but most tedious change was to audit the entire kernel for places that allowed the superuser extra privilege. Only about 35 of the 300 checks in FreeBSD 5.0 were opened to jailed processes running with superuser privileges. Since the default is that jailed superusers do not receive privilege, new code or drivers are automatically jail-aware: They will refuse jailed superusers privilege.

Jail Limitations

As it stands, the jail code provides a strict subset of system resources to the jail environment, based on access to processes, files, network resources, and privileged services. Making the jail environment appear to be a fully functional FreeBSD system allows maximum application support and the ability to offer a wide range of services within a jail environment. However, there are limitations in the current implementation. Removing these limitations will enhance the ability to offer services in a jail environment. Three areas that deserve greater attention are the set of available network resources, management of scheduling resources, and support for orderly jail shutdown.

Currently, only a single IP version 4 address may be allocated to each jail, and all communication from the jail is limited to that IP address. It would be desirable to support multiple addresses or possibly different address families for each jail. Access to raw sockets is currently prohibited, as the current implementation of raw sockets allows access to raw IP packets associated with all interfaces. Limiting the scope of the raw socket would allow its safe use within a jail, thus allowing the use of ping and other network debugging and evaluation tools.

Another area of great interest to the current users of the jail code is the ability to limit the effect of one jail on the CPU resources available for other jails. Specifically, they require that the system have ways to allocate scheduling resources among the groups of processes in each of the jails. Work in the area of lottery scheduling might be leveraged to allow some degree of partitioning between jail environments [Petrou & Milford, 1997].

Management of jail environments is currently somewhat ad hoc. Creating and starting jails is a well-documented procedure, but jail shutdown requires the identification and killing of all the processes running within the jail. One approach to cleaning up this interface would be to assign a unique jail-identifier at jail creation time. A new jailkill system call would permit the direction of signals to specific jail-identifiers, allowing for the effective termination of all processes in the jail. FreeBSD makes use of an init process to bring the system up during the boot process and to assist in shutdown. A similarly operating process, jailinit, running in each jail would present a central location for delivering management requests to its jail from the host environment or from within the jail. The jailinit process would coordinate the clean shutdown of the jail before resorting to terminating processes, in the same style as the host environment shutting down before killing all processes and halting the kernel.

References

Hope, P. 2002. "Using Jails in FreeBSD for Fun and Profit," *;login:*, vol. 27, no. 3, pp. 48–55, *http://www.usenix.org/publications/login/2002-06/pdfs/hope.pdf*, USENIX Association, Berkeley, CA (June 2002).

Kamp, P. & R. Watson. 2000. "Jails: Confining the Omnipotent Root," *Proceedings of the Second International System Administration and Networking Conference (SANE), http://docs.freebsd.org/44doc/papers/jail/* (May 2000).

P1003.1e. 1998. Unpublished Draft Standard for Information Technology—Portable Operating System Interface (POSIX)—Part 1: System Application Program Interface—Amendment: Protection, Audit and Control Interfaces [C Language] IEEE Standard 1003.1e Draft 17, ed. Casey Schaufler, Institute of Electrical and Electronic Engineers, Piscataway, NJ (1998).

Petrou, D. & J. Milford. 1997. Proportional-Share Scheduling: Implementation and Evaluation in a Widely Deployed Operating System, *http://www.cs.cmu.edu/~dpetrou/papers/freebsd_lottery_writeup98.ps* and *http://www.cs.cmu.edu/~dpetrou/code/freebsd_lottery_code.tar.gz* (1997).

CFS travails

by Travis Howard

Travis Howard is a independent security researcher currently in search of employ-
ment. His home page is *http://travcom.tripod.com.*

auto92089@hushmail.com

This is a story about a mysteriously corrupted encrypted file system and my attempts to recover the data. Forgive me if my narration of it is a bit rough, as I am not exactly sure what happened; all I can do is make observations. This is my first attempt to turn a chronological log of observations into a good narrative. In doing so I am gaining a new appreciation for the work of investigative journalists, who must go through a similar process to turn their notes into a compelling story. This is a work in progress. If you have solutions to any problems I mention herein, please email them to me.

The encrypting file system under observation is CFS, Matt Blaze's "Crypting File System." Parts of it date back to 1987, making it a rather venerable piece of code. Matt Blaze is well known in the computer security community, and the code has been available for scrutiny for some time (*http://www.crypto.com/software/*). The code itself is fairly portable and might be the most popular encrypting file system in UNIX.

First, a little background on how CFS works. CFS stores each encrypted file as a separate entity in the file system (i.e., it is not an encrypted disk device). Storage for each file also includes an initialization vector (IV). Early on, CFS used the inode number to store the IV. Subsequently, it used the GID field. Finally, Matt decided to use a separate symlink (with the prefix ".pvect_") to hold it (by pointing to it). In CFS, the IV is used to make files with identical contents encrypt to different data streams to prevent snoops from noticing correlations between files.

I encountered a bit of file system corruption using CFS.

It's hard to know exactly when the corruption occurred or why. My logs say I had some disk problems around 2 May 2001. I believe some of the encrypted files were moved to lost+found. My notes say that I restored a copy of the encrypted directory hierarchy to make comparisons and moved some files. I was not aware of the significance of the symlink/IVs at that time and probably made some mistakes de-synchronizing them.

I did make some edits to sensitive files on 15 Dec 2001. The files I was working on were not apparently corrupted. Subsequently, a period of hassles ensued, getting CFS to co-exist with my OS, which had a new v3 RPC mechanism. I recompiled various versions of CFS around this time, for better or for worse using an OS release with a buggy compiler (egcs-1.1.1). This compiler is known to generate incorrect code, and could be the source of some of the data corruption.

On 6 Dec 2003, decrypting certain files yielded corruption. That is, ASCII text files suddenly appeared to be binary files. How can I quantify this corruption? I ran a tool called "ent" on the files, and it yielded some interesting results:

```
$ ent corrupted_text_file
Entropy = 5.558949 bits per byte.
Optimum compression would reduce the size of this 6588 byte file by 30
percent.
Chi square distribution for 6588 samples is 40067.26, and randomly
would exceed this value 0.01 percent of the times.
Arithmetic mean value of data bytes is 63.7025 (127.5 = random).
Monte Carlo value for pi is 4.000000000 (error 27.32 percent).
Serial correlation coefficient is -0.087718 (totally uncorrelated = 0.0).
```

The entropy is one measure of the amount of unpredictable information in the file. For totally random files, it should be 8 bits per byte, and for a file full of nulls, it

should be 0 bits per byte. The optimum compression measurement merely describes the percentage of predictable information in the file. Chi square is a statistical text that involves sorting data in "bins" (probably 256 bins in this case, one for each possible byte), squaring the difference between the actual bin contents and the predicted values, then dividing by the probability of that bin being selected (in this case, 1/256 since all bytes are equally probable). Thus, the more a file deviates from random, the higher the chi square score, and the lower the percentage of cases that would exceed it. The arithmetic mean is obvious; the Monte Carlo approximation involves constructing a virtual 1 x 1 dart board with a circle of diameter one inside it, and using the samples as coordinates of darts, then dividing the number that land inside the circle by the total number of darts, and using that to compute a value of pi (it converges rather slowly). Finally, the serial correlation merely describes how much each sample depends on the prior sample.

Normally when encryption goes awry (from using the incorrect key, for example), you get something indistinguishable from random bits. As you can see, the files are definitely not random. Random looks like this:

```
$ ent random_file
Entropy = 7.970002 bits per byte.
Optimum compression would reduce the size of this 6144 file by 0 per-
    cent.
Chi square distribution for 6144 samples is 250.58, and randomly would
    exceed this value 50.00 percent of the times.
Arithmetic mean value of data bytes is 126.5417 (127.5 = random).
Monte Carlo value for pi is 3.152343750 (error 0.34 percent).
Serial correlation coefficient is -0.009299 (totally uncorrelated = 0.0).
```

On the other hand, they are more random than ordinary text files:

```
$ ent uncorrupted_text_file
Entropy = 4.436244 bits per byte.
Optimum compression would reduce the size of this 5134 byte file by 44
    percent.
Chi square distribution for 5134 samples is 76869.97, and randomly would
    exceed this value 0.01 percent of the times.
Arithmetic mean value of data bytes is 92.6139 (127.5 = random).
Monte Carlo value for pi is 4.000000000 (error 27.32 percent).
Serial correlation coefficient is -0.008061 (totally uncorrelated = 0.0).
```

I then wrote two programs, freqcount.pl and freqgraph.pl, to explore the files further. The former counts the frequency of each byte, and the latter represents it graphically (in text mode) as bars of asterisks running horizontally, so you can compare two files in side-by-side windows. What I saw appeared to be disruption of the large frequency spike of the space character; however, it was definitely not uniform. It looked as though it had gone through some polyalphabetic substitution with a periodicity of four or so.

My first reaction was to go to backup tapes. However, my backup tapes for 15 Oct 2003 turned out to be faulty; I had run out of tape but hadn't noticed. Although I have many backups, this corruption went unnoticed for so long that I didn't have a backup set old enough.

My next reaction was to see if cfsd, the NFS-like daemon that serves up the files, was at fault. I decided to try ccat (a stand-alone utility) on some of the files to eliminate cfsd

as a source of errors. Unfortunately, it warns that it only works on old-format CFS files. CFS went through several stages of evolution; stand-alone tools like ccat have not been kept up to date with the various evolutionary changes.

I should mention here some of the other tools I used. I used bc quite a bit, even though it has the annoying property of not accepting lowercase letters as hex digits and does not have a bitwise XOR operator. So I often ended up using Perl instead. If you try stuff like this you'll need the chr and ord functions. I also referred to an ASCII chart quite a bit.

I read Matt Blaze's notes.ms, and read through some of his code, particularly cmkdir.c. I found a few errors that are security-relevant; I must question how much trust I put in this software without doing even a cursory review of the code. My findings:

1. The type of cipher is included, apparently erroneously, in some key manipulations:

```
struct cfs_admkey {
    ciphers cipher;
    union {
            cfs_adm_deskey deskey;
            cfs_adm_3deskey des3key;
            ...
    } cfs_admkey_u;
};
cfs_admkey k;
...
/* now we xor in some truerand bytes for good measure */
bcopy(&k,ekey,32);  /* assumes key material < 32 bytes */
for (i=0; i<32; i++) {
    ekey[i] ^= randbyte();
}
encrypt_key(&k,ekey);
bcopy(ekey,ek1,32);
decrypt_key(&k,ek1);
/* new &k is our real key */
```

2. A file called "..." is created. Apparently this was to provide a somewhat-known plaintext so that the program can tell whether you have given the correct key or not. CFS attempts to make this 8-byte value half random, but due to shifting in the wrong direction, the attacker knows 7 of the 8 bytes, almost giving him or her a full known-plaintext. By contrast, PGP uses two bytes of known plaintext to determine if you have the correct key (meaning that it accepts the wrong key 1 in 2^{16} times).

```
char str[8];
...
strcpy(str, "qua!");
/* now randomize the end of str.. */
r = trand32();
for (i=0; i<4; i++)
    str[i+4]=(r<<(i*8))&0377;
```

At this point I thought that the IV was stored in the inode numbers. CFS used to do this, and I almost certainly changed the inode numbers on several files during my restoration efforts. There are 2^{32} possible inodes, and I would have to search for the correct one for each file. This could have taken a while. I would have had to be clever and crafty to finish this project in my lifetime. But perhaps it was not as impossible as

it seemed; the stream generated from the inode number is XORed with the data at one point; perhaps I wouldn't have had to try all of them in a brute-force method. Also, the file system may not have 2^{32} inodes to try out—perhaps much fewer. Yes, this might be possible.

I thought my problem was that there's a different inode used to seed a pRNG of some kind to create a stream that is being XORed with my program's text. To extract the original text, I would need to know the proper inode number. The size of the inode number tells me its maximum theoretical size, and if the file system it is on has only grown over time, its maximum inode value would tell me an even lower upper bound, while the inodes of nearby files might give me a good place to start searching. I also needed to recognize when I'd deduced the correct inode number.

Knowing whether the inode number is right or not would probably require running statistical tests on the contents of the file. On top of all this, these files' contents are sensitive (that's why they're encrypted), so I wouldn't be able to distribute any brute-forcing. I would either have to use Matt Blaze's CFS code or develop my own and test it to make sure it was doing exactly the same thing as his.

At this point I decided to print out all the papers on CFS and dive deep, really deep, into the belly of the beast. I really needed to understand the CFS encryption technology. Perhaps I could be clever and find an algorithmic shortcut, or a known-plaintext situation.

On 6 Jan 2004, I found there were "only" about 280,000 inodes on that file system. That meant 280,000 trials, far more reasonable than the earlier estimate, which ran as high as 4,294,967,296.

Checking each decryption for the bytewise frequency count of the space character could do a 1:256 winnowing. I'd need about another 1:100 reduction of those candidates to perform manual inspection (I figured 10 manual inspections per file is my upper limit).

On 7 Jan 2004, I thought I might have found my answer, due to the equations in Matt's papers:

```
D_p = DES^-1(K_2, E_p xor DES^1(K_1, g(p mod m)))
    xor DES^1(K_1, f(p mod m)) xor i
```

All the data are 8 bytes wide, except perhaps the keys. I think in this case, everything is known except *i*, which "is a bit representation of a unique file identifier derived from the UNIX inode number and creation time of the encrypted file."

Still, in the equations above, *i* is the only variable that is unknown to me. Since it is not used to seed any RNGs or to key any ciphers, I was in luck. Looking at the decryption equation, if *i* were wrong it would give me the correct data but XORed with some constant. That would be consistent with the entropy measurements I had made, which indicated that the file was definitely non-random but did not have the strong frequency spike of the space character in English prose.

Stated differently: If you plugged the wrong value of *i* (let's call it *i'*) into the above equation, you'd get this:

```
D_p' = D_p xor (i' xor i)
```

That's what was happening to me. Each 8-byte chunk of data that I saw was the original plaintext, XORed with some 8-byte constant value, which was the error (XOR) of *i'*

and *i*. Statistically, if I know the most common D_p′ and the most common D_p, and I know *i′*, I can solve for *i*:

```
i = D_p' xor D_p xor i'
```

Now, in that equation, each variable is an 8-byte array. At first, I thought *i* was 4 bytes long, and it simply concatenated it to itself to form an 8-byte value (later I learned that *i* was indeed an 8-byte value). Each byte of *i* can be solved independently:

```
i[0] = D_p'[0] xor D_p[0] xor i'[0]
i[1] = D_p'[1] xor D_p[1] xor i'[1]
…
i[3] = D_p'[3] xor D_p[3] xor i'[3]
i[0] = D_p'[4] xor D_p[4] xor i'[0]
…
i[3] = D_p'[7] xor D_p[7] xor i'[3]
```

In other words, I essentially had a polyalphabetic substitution cipher, with four substitution tables all built on bitwise XOR with a constant. For text files, I might even be able to solve each byte of the value *i* independently, suggesting only 4 x 256 = 1024 guesses! This was workable! In fact I could vary each byte of *i* at the same time, meaning only 256 decryptions!

On closer reading of the release notes (notes.ms), I found that CFS now stores an IV for filename in a symbolic link .pvect_filename. Amazingly, I had traced corruption to those files missing symbolic links! Hooray! Now I knew exactly what was missing, which files were corrupted, and how to fix it! I verified that these .pvect_ files were missing on my oldest backup too.

This new development meant I needed to guess eight hex digits, 32 bits of randomness. Quite an improvement, although I had to read the source to see exactly how it was used to see if I could search each byte independently. I suspected that I would be able to.

I could test my theories by creating a known-plaintext file, noting then removing the .pvect_whatever file, and then conducting a search for the proper (known) value. This might be faster than understanding CFS's code.

On 9 Jan 2004, I wrote a little program to find files without .pvect_ files. There were 19, and had been exactly 19 for a while. Not too bad. I had about 500 files in CFS.

I also modified ccat to the extent necessary to work with new-style dirs. It wasn't pretty, but it worked. Then I modified ccat to accept both an IV and a passphrase.

I wrote a program that tries IVs with each byte ranging from 0 to 255. For each of the 256 possibilities, it runs ccat with an IV of that value, repeated (all bytes equal). Then it does a simple frequency count for offsets of 0, 1, 2, and 3 into the file. After doing this it dumps all the information using Perl's Data::Dumper. My plan was to analyze this output later using statistical tests. Estimates of the time to do this were around 38 minutes, which turned out to be surprisingly accurate.

Thinking back, I do think I recall having seen symlinks pointing to garbage in the lost+found; I must have decided they were deletable and didn't restore them. All this work due to that miscalculation! Oh well, I had learned quite a bit about CFS.

I then noticed that the IV was 8 bytes in ASCII-encoded hexadecimal, which is a convenient way to store a 32-bit value. However, it was not converted into binary before

use; the ASCII values were used instead! So basically you had a 64-bit IV, but it could only hold 2^{32} different values. Thus, my program should have really calculated frequency counts for offsets of 0–7 into the file (not 0–3):

```
i[0] = D_p'[0] xor D_p[0] xor i'[0]
i[1] = D_p'[1] xor D_p[1] xor i'[1]
…
i[7] = D_p'[7] xor D_p[7] xor i'[7]
```

It's worth noting here, as a sidebar, that the more I knew about the file in question, the better I could narrow down the possibilities for the IV. If I had known nothing about the file, if it had been truly random data, I wouldn't have been able to tell whether one IV was better than another. The more I knew about the file's lack of randomness, though, the better I could weed out all the irrelevant IVs.

This knowledge comes from knowing the names of the files, remembering their contents, that kind of stuff. One thing about the IVs is that they don't affect the high bits of the file. That is, since the IV is all in the ASCII range of 0–127, being XORed with the file will not change the high bits. That is unfortunate, as the files typically don't have high bit set anywhere. However, this makes sense from a cryptographic perspective of trying to hide similarities between files; you are modifying bits that people care about instead of ones that are typically not used.

Taking all of this into consideration, I wrote a program (smart_iv) that very quickly deduced the missing IV. The basic idea behind this was run ccat once and break its output down by which byte of the IV it is being XORed against (forming 8 bins). Then count the frequencies of the characters in that bin. The whole data structure is essentially an 8 x 256 array. It then tests for the IV that gives me the greatest number of space characters. One could map the frequency counts based on the IV being tested, but I found it easier to go the other way: that is, to XOR the space character with the IV in question and look at the occurrences (frequency) of that entry in the frequency chart. This program was capable of recovering the longer files of English text, probably about five of the 19. I then renamed this program (find_blank) because I had some better ideas.

I changed ccat to accept an IV, but what IV do I specify? Ideally, I would use a null vector. That is, if I let i'=0, then it drops out of the equations mentioned above because XOR with 0 is an identity operation. In fact, cfsd probably assumes i'=0 when there are no .pvect_ files. However, ccat uses C string functions to parse the IV from the command line, and eight nulls would be read as a zero-length string. So, instead, I specified "00000000" and dealt with the XOR deltas between that and the actual IV instead of with absolute values (that is, they are relative to 0x3030303030303030 since ASCII for "0" is 0x30).

Around this time I noticed that there were other files which had .pvect_ files but that they were also corrupted. Is there no end to this pain? I would have to examine every file on the encrypted file system, by hand, to determine whether it looked reasonable. Argh! I put this off until I had completed recovering the 19 files without .pvect_ symlinks.

11 Jan 2004

I noticed that there was a class of files that were not English prose, and therefore did not have the prevalence of the space character in their frequency profiles. So I wrote another program, called maximize_printables, which found the IVs that maximized

the number of printable characters. This program did not work as well as I expected, since some IVs that maximized printables also contained junk like vertical tabs (!), which virtually never appear in ASCII files.

The next logical step was another program, called minimize_nonprintables, which did what you'd expect. However, unlike the maximizing spaces heuristic, which tended to pick only a handful of IVs, the minimize_nonprintables generally printed out several (on the order of 16 or so). I found that since it usually got half of the bytes in the IV correct, I could try one possibility, with ccat, see halves of words that I recognized, then exploit these inter-byte dependencies to infer the correct values of the currently incorrect IV bytes. I was able to recover around five more of the 19 files without .pvect_ files.[1]

Finding those halves of words in the plaintext gave me an idea. Perhaps I should write another program, for files containing a known string. For example, an RCS file has the words "head", "access", "symbols"—surely this knowledge could be helpful in determining an IV!

Again, the more I know about the plaintext, the easier it is to recover an IV. For simplification, consider the case of an RCS file. In this case, I know it begins with "head\t". In most cases I never increment the major number, so the next characters are usually "1.", which is followed by one or more digits. If I know that a fixed string of length l occurs at a particular offset into the file, I can easily compute l bytes of the IV starting at that offset modulo 8 and wrap around in the obvious fashion.

12 Jan 2004

It's worth noting that this applies to other types of files too; for scripts, I start with "#! /". The extra space is there for portability because Dynix uses a four-byte "magic number" to identify scripts. I never expect to see Dynix, but it's only one byte and portability is not a bad habit to have. In many cases the following letters are "usr/" too.

I like to start from simple and specific cases, and enhance scripts to handle general cases. So the next program I wrote, known_header, cracked an IV based on known plaintext at the beginning of the file. In this case, I simply XORed "0" and the known byte and the byte from the decrypted file, and that gave me the IV. Simplicity itself! This recovered about five of the 19 files. The rest of the 19 were automatically generated files which I simply removed, so I consider these techniques moderately successful.

But let's say I'm looking for a known string of some length l in the plaintext at an unknown location. If the IV were fixed, I would look for one of eight patterns, with the pattern being dependent on my offset into the file (modulo 8 of course). At each step I might have to look ahead as far as *l*–1 bytes, to see if I was at the beginning of a match. That might be acceptable, but the IV isn't fixed. What I'm really doing is trying to find the IV that induces the most occurrences of that string, and I'm trying to do it without iterating over all possible IV values (2^32 iterations over the length of the file sounds like too much computation). Surely there has to be a better way.

14 Jan 2004

To count strings of length *l*, I thought of an (*l*+1) state DFA with multiple "tokens" that move around to keep track of the state. Specifically, there will be 2^32 logical tokens, but in implementation I'll consolidate them into *l*+1 tokens, one for each state.

1. Theoretically, minimize_nonprintables and maximize_printables should give the same results since they are complementary. However, my implementation of one or the other must have been in error since I think I noticed different results.

Every time a logical token hits the final state, it increments a counter associated with its IV. In actuality, there may be multiple logical tokens for a given IV. For example, if the IVs were only two bytes long, and the string we want to find is ABABC and we read in ABAB, then there will be two tokens, one on the first B and one on the second. Note that the string must be longer than the IV to have two logical tokens in the works.

To simplify the matter, I modified ccat to accept a null IV. I merely tested to see if it was a null string and, as a special case if it was, accepted it as the null vector (which would be impossible to pass on a command line anyway). That helped a bit.

17 Jan 2004

It's interesting to note that in some cases I might know two different types of things about a file: for example, that space is the most frequent character and that it contains a given string. Each of these individually induces a distribution on the IV. However, it is more difficult to represent and combine this distribution between these two programs than achieving the same level of confidence using one technique alone. That is, it is easier to print out the most probable IV than it is to communicate the top 10 IVs. And even that is easier than communicating all IVs and their associated probabilities.

19 Jan 2004

I've written known_strings. I started by examining all the relevant finite-state automata (FSA) classes on CPAN; none really fit the bill. I then wrote the main body of the program, proceeding as though I had already written any relevant classes. I knew that I'd need some kind of custom FSA and that I'd need some structure in which to store results.

Representing the results of the searching posed an interesting challenge. Did I want to allocate a Perl structure with 2^32 entries? Even at a byte per entry, that's the entire address space of the machine! I could do it with a large file, but that lacked elegance. If the known string were short, there would be many matching IVs and the program would be quite slow.

One alternative would seem to be storing only non-zero entries, but this is only useful if the resulting array is sparse. The sparseness of this array depends on the length of the string I'm seeking and the content of the file. Another alternative would be to represent the partial matches of IVs, and after parsing the file, to loop through all IVs to see how many of these partial matches a specific IV actually matches. Put another way, my results might be "any IV that starts with A" and "any IV that starts with AB". I would then enumerate all IVs, starting with AAAAAAAA, noting that it matches once and so outputting "AAAAAAAA 1". When I got to ABAAAAAA, which matches twice, I'd output "ABAAAAAA 2". Thus, instead of storing the results array in memory, I could actually store it temporally by walking through IVs and dynamically generating the values it would have. Of course, if I spent 1 ms on every IV, iterating through them all would take 49 days.

In this light, perhaps it would be better to iterate over the file than to deal with the whole IV space. Most of the files were small text files. In fact, most UNIX files are small; for this reason the file system is optimized for dealing with relatively small files.

And that's the current situation. I took a little detour investigating alternatives to CFS and will let you know in the future how it all ends up.

ISPadmin

by Robert Haskins

Robert D. Haskins is currently employed by Renesys Corporation in Hanover, NH.

rhaskins@usenix.org

Recent Spam-Fighting Developments

Introduction

In this edition of ISPadmin, I will look at a wide range of recent developments in the fight against spam. The following topics are covered:

- DSPAM
- Sender Policy Framework (SPF) and related ideas
- Selective port 25 blocking
- "Filters That Fight Back"
- Being paid to spam
- The CAN-SPAM law (and Do Not Email registry)
- Recent spam-related prosecutions and lawsuits
- L.L. Bean and overstock.com anti-spyware cases

DSPAM

"DSPAM (as in De-Spam) is an extremely scalable, open-source statistical hybrid anti-spam filter," according to its Web site. They recently announced that the newest version (3.0) is nearing production release. This software is essentially a sophisticated statistical filter designed for high-volume mail systems. It is written in a compiled language (C), which makes it very scalable with low overhead. DSPAM uses a database back end for saving and tracking scores. Some of the benefits of the DSPAM approach over its competitors are:

- Speed/performance
- Scalability
- Low administrative overhead
- All major MTAs supported

However, it would be nice if DSPAM included support for some of the features in SpamAssassin, namely:

- Support for distributed blacklists (i.e., MAPS, SPAMHAUS)
- Support for distributed checksum networks (DCC and Vipul's Razor)

Nice as these would be, DSPAM as it currently exists is certainly very useful for anyone who needs a scalable, easy-to-use statistical analyzer for their mail infrastructure.

Sender Policy Framework (SPF)

Sender Policy Framework (formerly Sender Permitted From) has garnered a lot of press lately. SPF uses DNS records that indicate the hosts from which a mailserver should accept email for a given domain. For example, if an email envelope had the from address "bob@aol.com" but the actual SMTP server sending the message wasn't listed in AOL's SPF record, the receiving server would reject the message, presuming the header to be "faked." The larger the email-box hosting provider, the more helpful something like SPF is going to be.

AOL adopted the SPF protocol in December 2003. Microsoft's proposal (in its unfortunately named Caller ID for Email proposal/standard) has been merged with the SPF. Yahoo's DomainKeys standard (one of the proposals backed by Sendmail) is a similar approach, though more difficult to implement in the short term due to the need for signed keys. However, this is arguably more secure, as each message would be signed, thus improving the trustworthiness of email being sent using the DomainKeys proto-

col. Both Caller ID for Email and DomainKeys have been submitted to the IETF for adoption as a standard.

While anything that can be done to reduce the amount of spam is a good thing, the SPF protocol (and related solutions) suffers from a few shortcomings:

- Email forwarding is problematic.
- It doesn't do anything about the "spam zombie" problem.
- It binds email address owners closely to their providers.
- SPF is much more likely to be adopted by the large email-box hosting providers than smaller ones.

The real way to fix email is to replace RFC822 with a more secure protocol. Unfortunately, until there is a critical mass, this is unlikely to happen. As a result of not having a standard, we will have to rely on incremental approaches such as SPF.

Selective Port 25 Blocking

In June 2004, the large US-based cable-modem ISP Comcast began blocking SMTP (port 25) on customer cable modem connections that generate a large amount of traffic. This is an effort to block the many customer machines that have been turned into "spam zombies." Comcast should be commended for taking a bold step against spam.

Many ISPs have globally blocked port 25 on their networks for years. However, this type of global blocking can cause no end of headache for legitimate customers who host their own email server that connects to the Internet via the ISP's network connection. If they take the proper precautions (such as not being an open relay), there should be nothing wrong with hosting a mail server so long as the ISP's terms of service are not violated.

Selectively blocking port 25 on those customer connections who are most likely compromised and being actively used by spammers is a big step in the right direction. If every provider followed Comcast's lead, there would be a significant reduction in the amount of spam (at least until the spammers found the next method to use).

"Filters That Fight Back"

In his August 2003 essay, Paul Graham argues for using email-client–based filters that follow the links listed in spam messages and then pound the spammers' sites with HTTP requests. While this is a noble idea and would probably achieve the goal of putting (some of) the spammers out of business, it suffers from a number of shortcomings.

First of all, it would be easy to "joe job" someone if the plan was implemented widely enough. (A "joe job" occurs when an unsuspecting third party is listed as the From: address in a spam message and is inundated with complaints from email users who don't know better.) There would be nothing to stop a prankster (or anyone else, for that matter) from sending out a wide message that gets defined as spam, causing the innocent party's Web site or other electronic presence to be interrupted.

Second, any spammer who didn't have a URL listed in their message would be immune. If the spammer used a telephone number to collect sales, there would be no easy way for filters to fight back.

Third, most providers would not take kindly to this type of network traffic on their networks. If the plan were implemented and successful, service providers would be the

A company called VirtualMDA is offering to pay $1 per CPU hour of time to use your computer and network connection to send marketing messages on behalf of its clients.

ones who would bear the brunt of the cost, by virtue of having to buy additional bandwidth on their networks to handle the increase in network load.

Being Paid to Spam

A company called VirtualMDA is offering to pay $1 per CPU hour of time to use your computer and network connection to send marketing messages on behalf of its clients. This rate of pay is a big win for VirtualMDA, as it would take a huge number of messages before the user would get paid for his/her effort. In any case, before they ever saw a penny from VirtualMDA, the user's ISP would probably shut the the account off for violating the ISP's terms of service.

The CAN-SPAM Law

Disclaimer: I am not a lawyer!

The US CAN-SPAM Act went into effect on January 1, 2004, with much fanfare. Marketers are required to do the following under this law, according to the spamlaws.com site:

- Label their email
- Include opt-out instructions and the sender's physical address
- Refrain from using deceptive subject lines and false headers

Unfortunately, no standard label for bulk email was specified in the law, which makes this provision almost meaningless. The act also authorizes the FTC to establish a "Do Not Email" list. I believe the establishment of a "Do Not Email" list could result in much *more* spam reaching users' email boxes. The temptation for a confirmed email list such as this to be abused by spammers is simply too great, no matter what controls are placed on it.

Has this law had any effect on spam to date? Not much. Enforcement of the law is now only beginning. Time will tell, unfortunately. At best, laws are only part of the solution. At worst, they are part of the *problem*!

Recent Spam-Related Prosecutions and Lawsuits

On April 29, 2004, the FTC announced the first four prosecutions under the US CAN-SPAM law. At an average of one prosecution per month, there will be no effect on the average user. However, I believe the FTC and FBI are just beginning their work, and we will start to see much more of an impact once large numbers of spammers are brought to justice. Of course, a small percentage of the spammers are responsible for a large percentage of the spam.

In a May 24, 2004, Boston Globe article, Hiawatha Bray makes the point that phishing (scammers who pretend to be credit card companies in order to get a victim's credit card numbers and other personal information) will be good for the fight against spam. When the scammers, spammers, and spyware marketers target the companies with deep pockets and lots of market share (and money) to lose, only good things can come of it.

In fact, L.L. Bean and overstock.com both recently announced lawsuits against advertisers who used spyware in their marketing efforts. The marketing companies utilized spyware to generate pop-unders for the named advertisers when users (who had the spyware software installed on their computers) visited the L.L. Bean site. It is only a

matter of time until the spammers market the wrong product and get sued by the product's trademark holders.

References

DSPAM: *http://www.nuclearelephant.com/projects/dspam/*

MAPS RBL: *http://www.mail-abuse.com/services/mds_rbl.html*

SPAMHAUS: *http://www.spamhaus.org/*

DCC: *http://www.rhyolite.com/anti-spam/dcc/*

Vipul's Razor: *http://razor.sourceforge.net/*

SPF: *http://spf.pobox.com/*

Yahoo DomainKeys: *http://antispam.yahoo.com/domainkeys*

AOL SPF: *http://postmaster.aol.com/info/spf.html*

MS Caller ID for Email: *http://www.microsoft.com/mscorp/twc/privacy/spam_callerid.mspx*

Comcast selective port-25 blocking: *http://zdnet.com.com/2100-1104_2-5230615.html*

"Filters that Fight Back": *http://www.paulgraham.com/ffb.html*

VirtualMDA: *http://www.virtualmda.com*

spamlaws.com CAN-SPAM: *http://www.spamlaws.com/federal/108s877.html*

First CAN-SPAM prosecutions: *http://www.cnn.com/2004/LAW/04/28/internet.spam.ap/index.html*

Boston Globe phishing article: *http://www.boston.com/business/globe/articles/2004/05/24/best_news_in_the_war_on_spam_phishing/*

L.L. Bean spyware lawsuit: *http://biz.yahoo.com/prnews/040517/nem044_1.html*

musings

by Rik Farrow

Rik Farrow provides UNIX and Internet security consulting and training. He is the author of UNIX System Security and System Administrator's Guide to System V.

rik@spirit.com

Many years ago, I heard Michelle Crabb, a sysadmin at NASA Ames in the San Francisco Bay Area at that time, tell a story about discovering an intrusion there. Ames was the site of the satellite feed that connected the nascent Internet in Australia to the continental US. A NASA manager wanted Internet access added to his own desktop Sun workstation, and eventually Crabb agreed to do so. Within a week, the manager came to her and mentioned that the login prompt on his Sun had changed. Instead of "login" with a small "l", it now appeared as "Login".

This subtle change was the overt signature of a trojaned login program. Had the Australian teenage attacker been a bit more careful, his successful intrusion might have gone unnoticed much longer.

I have always wondered about how many really skilled attackers there are. People who can break into a system, cleverly trojan/rootkit it, and only visit that system occasionally. With little traffic and no overt signs of an intrusion, such as the usual port scanners or IRC relays like psyBNC generating traffic, a skillfully completed intrusion might not be noticed for months, perhaps years.

Little, if any, data about the really skillful attacks seems to exist. But a very big hint did emerge in late May, when new overflow exploits, as root, for CVS server software were divulged. Some people claimed that they had been using the CVS exploits since 2001 and had owned many well-known open source CVS servers. All that was needed was anonymous read-only access in order to exploit the server.

I found these assertions somewhat hard to believe. Partially because I didn't want to believe them. And largely because so few attackers (apparently) have the discipline to take over a system and behave in such a low-key manner as not to be noticed. Also, one would hope that any changes to open source software, such as the addition of back doors, would have been noticed by now (after the claimed three years).

There were the trojaned configure scripts that were discovered in 2003. Each configure script included an addition designed to look as though it belonged in a configure script. The trojan was not rocket science, as it used a hardwired address to connect to and failed to loop properly (at least the version I played with). The trojaned configure script would hardly be the work of a skillful attacker.

The attacks on supercomputers at Stanford University and in the TeraGrid at SDSC, NCSA, and other locations were closer to a skillful assault, but they were not actually successful. The attackers, instead of treading lightly, just kept abusing more accounts and taking over more systems, making it only a matter of time before their intrusions were discovered. The attackers were certainly persistent, returning even after being discovered. I am hoping that one or more of the defenders will discuss their experiences with these intrusions in the Security edition of *;login:*.

Open Source in Danger?

If open source (OS) CVS servers were owned for a long time, what effect will this have on the security of open source software? There have been attempts to install back doors in OS software before, such as the slight modification to the wait() system call late last year. But that attempt was noticed because the attackers bypassed the code

management system, drawing attention to the change. Two other attempts, on CVS servers, were also noticed.

One reason for these changes being noticed is that any time someone submits a change, many eyes will scrutinize the code. It would be one thing to make a change that would subvert the entire CVS system, and quite another to slip something strange past people who are competing (to a degree) to produce the best code. Still, the reputation of OS code is at stake, and weaknesses in CVS code, responsible for maintaining the integrity of OS, certainly do not look good.

Other Dangers

Problems with CVS have a much greater impact, I would hope, than the latest bit of tomfoolery from the Alexis de Tocqueville Institution (ADTI). Ken Brown, president of ADTI, paid a visit to Andy Tanenbaum at Vrije University in Amsterdam, where he teaches computer science. Tanenbaum is the author of MINIX, a UNIX-like operating system created mainly as a teaching tool. Brown had traveled from Washington, DC, to the Netherlands, apparently convinced that there was some ill will between Linus Torvalds, creator of Linux, and Professor Tanenbaum, creator of the not nearly as popular MINIX. You can read Tanenbaum's words for yourself at *http://www.cs.vu.nl/~ast/brown/*.

Brown has written a book which may have appeared by the time this column gets published. His book will apparently make the claim that Linus did *not* write Linux, because it is impossible for one person to write an operating system. Imagine making that claim after interviewing a college professor who had himself done exactly that.

There are more incredibly bogus claims floating about, such as the one by Dan O'Dowd, CEO of Green Hills Software, who said in a speech, "The very nature of the open source process should rule Linux out of defense applications." O'Dowd used as his example Ken Thompson's famous trojan code in early versions of AT&T UNIX. How Thompson's wonderful hack would affect an entirely different code tree (it wasn't actually a *magical* hack) is beyond most thinking individuals. Perhaps O'Dowd was trying to recapture Linux business his company had lost to OS versions of Linux?

The closed source world is just as vulnerable to back doors, if not more so, than OS. Some security software vendors have had difficulty staying on NSA's approved software list because when NSA attempted to build their software from the source, it didn't match the distributed version. Did this signify back doors or merely different compiler flags and configuration options? And Thompson's back door was in closed source, not OS.

Cash Registers and Voting

The most egregious example of closed source software appears in US voting systems. Electronic voting has been successfully used in Australia and India. The source code is available for public inspection, not kept as a trade secret as is the code in the most widely used electronic voting systems in the US. (See *http://www.elections.act.gov.au/Elecvote.html* for information about an Australian system, *http://www.theregister.co.uk/2004/06/23/open_source_voting_software/* for information about a Dutch system.)

What gets me about companies like Diebold is this: Diebold also makes touch-screen cash registers, like the ones you have probably seen in bars and restaurants. Similar touch-screen cash registers can be purchased from Internet sites for about $1200, with

printers included. Diebold's touch-screen voting systems cost over $3000, without printers. And Diebold has a well-known reputation for installing last-minute patches to their voting systems, a practice that resulted in Diebold machines being barred from use in several counties in California.

Now what is so hard about a touch-screen voting machine that it would require last-minute patches? Diebold doesn't have this problem with their cash registers. Cash registers have a paper trail to prevent fraud. But not voting systems? Something is fishy here—actually, it stinks to high heaven.

I think the problem with some US voting machine companies is so serious that people prefer not to think about it.

Open source, or at least publicly audited code, is more secure than proprietary code. If you have to use a touch-screen system to vote, vote by mail so that there will be at least as much of a paper record as there would be when you buy lunch.

California online privacy act has widespread effects

by John Nicholson

John Nicholson is an attorney in the Technology Group of the firm of Shaw Pittman in Washington, D.C. He focuses on technology outsourcing, application development and system implementation, and other technology issues.

John.Nicholson@ShawPittman.com

and Rachel Wheeler

Rachel Wheeler is a summer associate at the firm of Shaw Pittman in Washington, DC. She is a student at the William & Mary School of Law.

Rachel.Wheeler@ShawPittman.com

Editor's Note: This issue features two different approaches to describing the new California Online Privacy Protection Act. John Nicholson's article includes motivations, reasonings, and discussions of all sorts of nuances of OPPA. Dan Appelman's article addresses businesses and contains only the nitty-gritty, though sometimes with a more business-oriented slant. While there is overlap between the articles, seeing the different points of view is fascinating.

I used the information in these articles to create a potentially compliant privacy notice for the USA Computing Olympiad. It was interesting to perform the mental exercise that answers the question, "Just how is this information actually used, and who else can see it?" I imagine this will be very challenging for larger organizations. Some, as Nicholson's article points out, have simply opted out: Children under 13 are not allowed at their sites.—RK

1. California Business and Professions Code, §22575 et seq. (2003).

On July 1, 2004, the California Online Privacy Protection Act of 2003[1] (OPPA) took effect. OPPA requires owners of commercial Web sites that collect personal information from California residents to post *conspicuously* a privacy policy explaining the types of information collected and the parties with whom the information is shared. Complying with OPPA's requirements is relatively straightforward. Not only does OPPA apply to all businesses that collect information from California consumers, regardless of the location of those businesses, but OPPA also exposes businesses to civil lawsuits, including class actions, even for negligent violations.

Summary of Requirements

OPPA specifies both the information "operators" (see definition in Section II, below) must include in a privacy policy and the methods by which operators must post the policy. Under OPPA, a privacy policy must include: (1) a list of categories of personal information collected and parties with whom the collected information is shared; (2) if applicable, a description of the process by which the users can update collected personal information; (3) a description of the process by which the operator notifies users of "material" changes to the privacy policy; and (4) the date the privacy policy becomes effective. To post a privacy policy conspicuously, an operator must do at least one of the following: (1) include the policy on the home page of the Web site or the "first significant page" after the home page; (2) hyperlink to the policy from the home page or the "first significant page" with an icon containing the word "privacy" in colors that contrast with the background of the page; or (3) hyperlink to the policy from the home page or the "first significant page" with text including the word "privacy" in a font style and size distinguishable from surrounding text.

Who Must Comply with OPPA

OPPA applies to all "operators" of "commercial website[s] or online service[s]" if they collect "personally identifiable information" from consumers residing in California. Internet service providers and other groups that transmit and store information on behalf of third-party operators, such as Web site development companies, are expressly exempt from OPPA's requirements. Under OPPA, "consumers" are California residents who "seek or acquire" goods, services, money, or credit online. "Personally identifiable information" includes names, addresses, telephone numbers, social security numbers, or any other information that allows operators to contact consumers. As will be discussed below, companies that might collect personal information from a California consumer via a Web site should implement a privacy policy that satisfies the requirements of OPPA (including those regarding the location of the

privacy policy, its appearance, and its content) and a policy for evaluating, investigating, and responding to complaints under OPPA.

But My Business Isn't In California! How Can It Be Subject to a California Law?

The sweeping language of OPPA means that operators anywhere in the United States and possibly abroad are potentially subject to suit. Further, under California case law regarding Internet jurisdiction, most operators collecting information for commercial purposes will be subject to the jurisdiction of California courts. All operators in the following categories are subject to California jurisdiction: (1) those incorporated in California, (2) those with a principal place of business in California, and (3) those with "systematic and continuous" ties with the state due to the presence of offices, personnel, bank accounts, and other tangible assets in the state. In addition, those operators who have limited or no physical ties to California will still be subject to the jurisdiction of California courts if the operators are found to have certain "minimum contacts" with the state.

California courts endorse a sliding-scale approach to assessing the concept of "minimum contacts" in Internet jurisdiction cases. Those operators who are outside of California and who maintain completely passive Web sites that simply advertise services are the least likely to be subjected to California jurisdiction. For example, in *Advanced Software, Inc. v. Datapharm, Inc.*,[2] a California federal district court found that an Ohio operator had not subjected itself to California jurisdiction simply by posting a Web site describing its services and employees, listing contact information, and providing links to other pharmaceutical sites.

At the other end of the scale, those operators who post interactive Web sites allowing them to contact California residents repeatedly or to form contracts with California residents are the most likely to be subject to jurisdiction in California. In *Snowney v. Harrah's Entertainment, Inc.*,[3] a Nevada operator was subjected to California law based on the operator's Web site solicitation of California residents to make hotel reservations on its Web site, evidence that Californians actually made reservations using the Web site, and the fact that the Web site targeted Californians by providing directions from California to hotel locations in Nevada. In *Panavision Int'l, L.P. v. Toeppen*,[4] the Ninth Circuit Court of Appeals found the required "minimum contacts" in a transaction in which an operator registered as a domain name the trademark of a California business and then attempted to profit from reselling the domain name to the trademark holder. Finally, in *Colt Studio, Inc. v. Badpuppy Entertainment*,[5] a federal district court found the required "minimum contacts" where the operator had entered into contracts with 2,100 Californians to provide them with monthly subscriptions to an adult Web site.

OPPA is specifically aimed at operators who are closer to the *Colt Studio* end of the sliding scale. Because OPPA is focused on actively collecting personally identifiable information rather than passively advertising goods and services, and because that personally identifiable information will potentially give the operator notice that the subjects of the information are California residents (e.g., the information may include their address), California Internet jurisdiction case law is likely to give California's courts subject matter jurisdiction over operators who collect data from California citizens for commercial purposes, making them subject to OPPA enforcement actions.

2. 1998 U.S. Dist. LEXIS 22091 (D. Cal. 1998).

3. 11 Cal. Rptr. 3d 35 (Cal. Ct. App. 2004).

4. 141 F.3d 1316 (9th Cir. 1998).

5. 75 F. Supp. 2d 1104 (D. Cal. 1999).

Noncompliance

An operator can be held liable for failure to comply with OPPA if either (1) the operator is negligent and the failure is "material"[6] or (2) the operator knowingly or willfully violates OPPA, regardless of the "materiality" of the violation. OPPA provides a 30-day grace period to allow operators to come into compliance once they are notified of a violation. It should be noted, however, that intentionally violating the requirement would probably qualify as "notice." OPPA, however, does not specify a particular party to fulfill the role of notifier. Accordingly, it is possible that a California consumer's complaint to an operator regarding noncompliance may serve as a trigger for the 30-day grace period.

OPPA does not include an explicit enforcement provision. Commentators have suggested that California's Unfair Competition Law[7] (UCL) will provide the means of enforcement. The UCL governs "unfair, unlawful, and fraudulent business acts" and specifies that the attorney general, district attorney, or city attorney may bring civil actions. More important, any "person, corporation, or association, or . . . any entity acting for the interests of itself, its members, or the general public" can initiate actions under the UCL. Damage awards can reach $2500 per violation. Commentators have also suggested that the UCL's allowance for personal actions may result in class action lawsuits under OPPA.

Recommended Business Response

Because OPPA contains no jurisdictional limitations and because OPPA violations could result in costly fines, companies that might collect personal information from a California consumer via a Web site should implement a privacy policy that satisfies the requirements of OPPA (including those regarding the location of the privacy policy, its appearance, and its content) and a policy for evaluating, investigating, and responding to complaints under OPPA. In creating a privacy policy, companies should determine the types of personal information they collect, the ways in which they use the information, the parties with whom they share the information, and the means by which they notify customers regarding changes to their policies.[8]

6. Note: "material" is one of those words used by lawyers to express a concept that can only be determined in context in a specific situation. Basically, something is "material" if a reasonable person would care. So, in this situation, a violation of OPPA is material if a reasonable person would care about (i.e., suffers harm from) the violation.

7. California Business and Professions Code, §§17200–17209 (2003).

8. This article provides general information and represents the authors' views. It does not constitute legal advice and should not be used or taken as legal advice relating to any specific situation.

OPPA and Web site operators

by Dan Appelman

Dan Appelman is a partner in the international law firm of Heller Ehrman, White & McAuliffe, LLP. He practices intellectual property and commercial law, primarily with technology clients, and is the current chair of the California Bar Association's Standing Committee on Cyberspace Law.

dan@hewm.com

In mid-October of 2003, Governor Gray Davis signed the Online Privacy Protection Act (OPPA). The new law took effect on July 1, 2004, and became part of the Business and Professions Code at §§22575 through 22579. OPPA requires every person and business entity in the United States (and, presumably, anywhere in the world) who owns a Web site and collects personal information about California residents to post a conspicuous privacy policy on that Web site stating what information they collect and with whom they share it, and to comply with that policy. Violations of the new law can result in civil penalties and can be the basis for suits brought by the California attorney general as well as by private individuals.

Federal law generally does not require Web sites to include privacy policies, although there are exceptions related to the banking and health services industries and to Web sites directed at children. The new California law is one of several recent examples where the California legislature has gone further than Congress in imposing significant restrictions on the way companies, regardless of where they are located, conduct their business online or with the aid of computers, to the extent that such business practices affect California residents.[1] Compliance with OPPA is not intuitive, and companies must become familiar with its particular requirements.

Who Must Comply

The new law applies to all "operators" of commercial Web sites and online services that collect personally identifiable information about California consumers. This includes out-of-state operators as well as those based in California.

The term "operator" means the owner of a Web site or an online service. It does not include third parties who may operate, host, or manage the site or service or who process information on behalf of the owner. The term "personally identifiable information" means individually identifiable information collected online about individual consumers, such as a first and last name, a street address, an email address, a telephone number, a social security number, or any other identifier that would permit the operator, or others who obtain access to that information, to contact a specific individual. The term "consumer" means any individual who seeks or acquires goods, services, money, or credit for personal, family, or household purposes. OPPA does not apply to owners of Web sites that only collect information about other businesses.

What the New Law Requires

The new law requires an operator to conspicuously post a privacy policy on their Web site, or, in the case of an online service, to use reasonable means to make that policy available to consumers. In order to meet the "conspicuously posted" requirement, the privacy policy must:

- appear prominently on the home page of the Web site;
- be directly linked to the home page by means of an icon that contains the word "privacy" and uses a color that contrasts with the background of the Web page; or
- be linked to the home page by means of a hypertext link that includes the word "privacy," is written in capital letters equal to or greater in size than the surround-

[Reprinted with permission from Heller Ehrman Venture Law Group]

1. Other examples include a new law requiring companies that maintain databases that include personal information about California residents to disclose any breach in security of those databases, which became effective on July 1, 2003, and a broad prohibition against sending unsolicited commercial email messages ("spam") to California email addresses, absent a clear opt-in by the recipient. This law was preempted by the new federal CAN-SPAM Act.

ing text, or is otherwise readily distinguishable from the surrounding text on the home page.

The privacy policy itself must do all of the following:

- It must identify the categories of personally identifiable information the operator collects.
- It must identify the categories of third parties with whom the operator may share the personally identifiable information that it collects..
- It must describe the process (if any) by which consumers can review and request changes to any of the collected information.
- It must describe the process by which the operator will notify consumers of material changes in its privacy policy.
- It must identify the effective date of the privacy policy.

OPPA contains a built-in "cure period"—it expressly provides that an operator who has been notified that they are not complying with the requirement to post a privacy policy will not be considered in violation of the law unless they fail to post the privacy policy within 30 days of such notification. Other provisions of the new law indicate that an operator intentionally failing to comply with the law will be considered in violation of OPPA even if the noncompliance is immaterial, while an operator whose noncompliance is not intentional, but negligent, will be considered in violation of OPPA only if the noncompliance is material.

The Consequences of Not Complying

The new law does not itself contain enforcement provisions. It is expected that OPPA will be enforced through California's Unfair Competition Law (the UCL), which is located at Business and Professions Code §§17200–17209.

Under the UCL, the attorney general, district attorneys, and certain city and county attorneys may bring civil actions based on acts of "unfair competition" as defined in Business and Professions Code §17200. Acts of "unfair competition" include acts, in business, that violate any law, which means that once OPPA takes effect in July 2004, a violation of OPPA will also be a violation of the UCL. The identified law enforcement officials may seek civil penalties and injunctive or other equitable relief.

Of greater concern, under the UCL "any person" may bring an action "in the interests of itself, its members, or the general public." This "private attorney general" provision has been broadly interpreted to allow a person with no personal interest and who has suffered no harm to bring a private action for restitution (to be paid to those who have suffered harm) or injunctive relief. In a private attorney general action brought under this special standing provision, the plaintiff may also recover attorneys' fees. Therefore, private plaintiffs will be able to use alleged violations of OPPA as a basis for asserting private UCL claims.

Recommendations

It is imperative that companies start complying with the new law immediately or risk fines and penalties and, perhaps, adverse publicity. Even if OPPA is preempted or overturned, posting a privacy policy and complying with it is becoming standard practice for those doing business on the Internet. To comply with the new law, privacy policies should disclose:

- What information Web site operators collect from those visiting their sites.
- How that information is used.

- With whom that information is shared.
- How consumers can review and correct the collected information.
- Whether consumers can "opt out" of having that information retained by the operator or shared with others.
- How consumers can communicate with the operator.

The Federal Trade Commission (*http://www.ftc.gov*) also has Web pages devoted to recommended best practices for privacy policies. Readers who have questions about how to comply with the new California law or federal requirements should contact the author of this article.

using C# reflection

by Glen McCluskey

Glen McCluskey is a consultant with 20 years of experience and has focused on programming lan-guages since 1988. He specializes in Java and C++ performance, testing, and technical documentation areas.

glenm@glenmccl.com

In this column we want to consider a set of C# features that go by the name "reflection." This term is a little hard to describe but in general refers to the ability of a running C# program to examine data about itself and modify its behavior accordingly. We'll look at several examples to help clarify this idea.

For reflection features to work, a C# system needs to keep data about the program around, so that it can be accessed at runtime. We can call this information "metadata," to distinguish it from the application data the program operates on.

Finding Types in a Running Program

Let's start with a fairly simple use of reflection, as illustrated in this program:

```
using System;
using System.Reflection;

public class DumpTypes {
    public static void Main(string[] args) {
        string targstr = (args.Length == 0 ?
            null : args[0].ToLower());

        Assembly asm = Assembly.Load("mscorlib.dll");

        Type[] typelist = asm.GetTypes();

        foreach (Type type in typelist) {
            string typestr = type.ToString().ToLower();
            if (targstr != null && targstr != typestr)
                continue;

            Console.WriteLine(type);

            MemberInfo[] memlist = type.GetMembers();
            foreach (MemberInfo mem in memlist)
                Console.WriteLine("    " + mem);
        }
    }
}
```

If you run this demo without any arguments, it displays about 25,000 lines of output, a list of all the classes and interfaces that the C# runtime system knows about, together with the members of each class. This process is sometimes called "type discovery."

The first few lines of output are as follows:

```
System.Object
    Int32 GetHashCode()
    Boolean Equals(System.Object)
    System.String ToString()
    Boolean Equals(System.Object, System.Object)
    Boolean ReferenceEquals(System.Object, System.Object)
```

```
System.Type GetType()
Void .ctor()
```

System.Object is a class known to C#, and it has class members such as GetHashCode and Equals. These members have particular parameter types such as System.Object and return types like Int32.

You can also run the program and specify the name of a class, like System.String, and only the details of that class will be displayed. The program works by opening an assembly, which is something like an archive file or dynamic link library. It then reads the types and members in a standardized way and displays them.

Obtaining a list of classes and members may not seem like much, but it's impossible to do in languages like C and C++. About the most you can do in C is to open an archive file like libc.a and read through it. There is no standard way of doing this operation, and information about parameter and return types for the various C functions is not preserved.

Dynamic Invocation

Let's go on and look at another use of reflection, one that's a little more sophisticated. Suppose that you have an application such as a C# interpreter, or a debugger, or browser, or something, and you'd like to invoke methods by name at runtime.

Or to say it another way, imagine that you are doing C programming and you have a program like this:

```
void f1() {}
void f2() {}
int main(int argc, char* argv[]) { … }
```

The user specifies "f1" or "f2" as a string on the command line, and you call the right function based on what is specified. The only way you can program such a feature is by means of a big if-then statement, or by keeping a table of function names/pointers around so that you can do dynamic dispatch. This approach is cumbersome.

Here's some C# code that solves this problem:

```
using System;
using System.Reflection;

public class InvokeDemo {
    static object run(string classname, string methodname) {
        Assembly asm = Assembly.Load("mscorlib.dll");

        Type type = asm.GetType(classname);

        object obj = asm.CreateInstance(classname);

        object[] args = new object[0];

        object ret = type.InvokeMember(
            methodname,
            BindingFlags.Default | BindingFlags.InvokeMethod,
            null,
```

```
            obj,
            args
        );

        return ret;
    }

    public static void Main(string[] args) {
        while (true) {
            try {
                Console.WriteLine();

                Console.Write("Enter class name: ");
                string classname = Console.ReadLine();

                Console.Write("Enter method name: ");
                string methodname = Console.ReadLine();

                object ret = run(classname, methodname);

                Console.WriteLine("ret value = " + ret);
            }
            catch (System.Exception exc) {
                Console.WriteLine("exception: " + exc);
            }
        }
    }
}
```

A typical interaction looks like this:

```
Enter class name: System.Object
Enter method name: GetHashCode
ret value = 21
Enter class name:
```

This interaction specifies that an object of the System.Object class is created, and then the GetHashCode method is called on the object. The return value (the hash code) is 21.

This program first queries the standard assembly for type information about the specified class (System.Object), creates an instance of that class, and then calls the method (GetHashCode) on the instance. This simple demo works only for parameterless methods, but can easily be extended to accept a list of arguments to be passed to the method.

The demo also illustrates exception handling, used to catch errors such as specifying an unknown class or method name.

Custom Attributes

A third way that you can use C# reflection is to specify and use custom attributes. An attribute is metadata about a C# program. It is not the program instructions or application data, but rather something like a "smart comment."

Let's illustrate the idea of attributes by looking at a sample application. Suppose that you'd like to incorporate code-change logs into your source code. That is, whenever

the source is edited, some information is added that describes the date, author, and nature of the change. One way to do this is through comments in the code. But comments have no structure, and are not queryable except in an ad hoc way.

Another way to solve this problem is by using C# custom attributes. Here's some code that shows how this can be done:

```
using System;
using System.Reflection;

[AttributeUsage(
    AttributeTargets.Class |
    AttributeTargets.Property, AllowMultiple = true
)]

public class CodeChangeAttribute : Attribute {
    private string id;
    private string who;
    private string date;
    private string descr;

    public CodeChangeAttribute(string id, string who,
    string date, string descr) {
        this.id = id;
        this.who = who;
        this.date = date;
        this.descr = descr;
    }

    public string GetId() {
        return id;
    }

    public string GetWho() {
        return who;
    }

    public string GetDate() {
        return date;
    }

    public string GetDescr() {
        return descr;
    }
}

[CodeChangeAttribute("123", "Jane Smith",
    "5/27/04", "initial checkin")]
[CodeChangeAttribute("456", "Bill Jones",
    "5/29/04", "fix three bugs")]

public class TestAttr {}

public class AttrDemo {
```

```
    public static void Main(String[] args) {
        MemberInfo meminfo = typeof(TestAttr);

        object[] attrlist = meminfo.GetCustomAttributes(
            typeof(CodeChangeAttribute), false);

        foreach (object attr in attrlist) {
            CodeChangeAttribute cca = (CodeChangeAttribute)attr;
            Console.WriteLine("id = " + cca.GetId());
            Console.WriteLine("who = " + cca.GetWho());
            Console.WriteLine("date = " + cca.GetDate());
            Console.WriteLine("descr = " + cca.GetDescr());
            Console.WriteLine("————————");
        }
    }
}
```

The first part of the demo defines a class called CodeChangeAttribute. This is a custom attribute, represented as a class, and objects of the class contain information on particular code changes. The information with each object is the change ID, the name of the person making the change, the date, and a description of the change.

This attribute can then be used with other classes you define, such as TestAttr. The attribute values are delimited by [...] and appear before the definition of the class.

The last part of the demo, the code in Main, shows how to query the attributes. When you run this program, the output is

```
id = 123
who = Jane Smith
date = 5/27/04
descr = initial checkin
————————
id = 456
who = Bill Jones
date = 5/29/04
descr = fix three bugs
————————
```

Attributes are annotations that you place on source code elements such as classes and method names. They can be used to provide information about an element, such as the code-change log example above, or affect the runtime behavior of a C# program. Attributes are not program data in the conventional sense, and they are not comments. They can be queried using C# reflection facilities.

Reflection is a powerful tool that you can use in your C# programs. It supports a rich and dynamic programming style, and opens the door for many innovative applications.

practical perl

Database Modeling with Class::DBI

by Adam Turoff

Adam is a consultant who specializes in using Perl to manage big data. He is a long-time Perl Monger, a technical editor for *The Perl Review*, and a frequent presenter at Perl conferences.

ziggy@panix.com

Database programming is often boring and tedious. Even with modules like DBI, using a database routinely involves writing nearly the same code over and over again. But it doesn't need to be that way. Class::DBI removes most, if not all, of the tedium and makes it easier to use a database than to avoid it.

I started a new job recently, and now I spend about an hour every day commuting on the train. This affords me a nice stretch of time to catch up on my reading, and I now regularly read one or two books per week. However, as I plow through my library, I'm starting to forget some of the books I've read recently.

Of course, there are many ways to solve this admittedly minor annoyance. I could whip up a quick little Web-based application in Perl, and store the list of books I've read recently in some database. But that strikes me as an approach that is simple, obvious, and wrong. The last thing I want to do, actually, is spend my time at home writing the same kind of database-centric Web application that I build at work.

So I looked for a simple, lazy solution. I just wanted something to jog my memory, and for a while I kept a list of book titles on a 3x5 card. But a list of book titles is a little limiting. A simple list of titles doesn't give me a whole lot of room to keep summaries of what I've read, or write notes on things I want to remember.

As this simple 3x5 card system started to get unwieldy, I started to think again about writing a Perl program to help me manage this information. After all, I could use SQLite as a database, which is about as painless as relational databases can possibly get, especially for quick hacks like this.

Even with SQLite, I wasn't looking forward to writing the Perl code to add, insert, and update records in a database. Again. I've done it for years, and while it's not that hard, it is repetitive and somewhat tedious. So I looked into Tony Bowden's Class::DBI module, which makes almost all of that pain just go away.

Step 1: Creating a Database

If you have been writing Perl code to use a database for even a short period of time, you know the story by heart:

- Create an empty database to hold your data.
- Connect to the database with the DBI module.
- Prepare SQL SELECT statements, and retrieve rows from the database as necessary.
- Process those rows, one by one.
- Occasionally prepare SQL statements to insert, update, or delete rows in the database.

The first step in writing a database-centric application is to set up the database. Class::DBI cannot help here, so I still need to do this the old-fashioned way with SQL CREATE TABLE statements.

To start, I only care about maintaining a list of books. Books are written by authors, so I will need two tables to start—one for books and one for authors. This will allow me to find all books written by a single author, but it will only let me track the primary

author for any one book. Not a perfect design, but good enough to start—I'm not trying to replace the Library of Congress here.

The SQL statements to create these tables in SQLite look like this:

```
CREATE TABLE author (
    authorid INTEGER PRIMARY KEY,
    first,
    middle,
    last);

CREATE TABLE book (
    bookid INTEGER PRIMARY KEY,
    author,
    title,
    subtitle,
    pubyear,
    edition,
    isbn,
    format);
```

The first thing to note is that SQLite is a typeless database, so all columns will be stored as strings of arbitrary size. This means that it is not an error to store a value like "199x" in the pubyear column, something which would normally store only integer values. So SQLite will never raise an error when interpreting the value "199x" as an integer, because all values are always strings, and there are never any type errors to worry about. Nor will SQLite raise an error when it tries to store a 1024-character string in a 32-character field—it will always store the full 1024 characters.

The one minor exception to this rule is the case when the first field in a table is declared as the INTEGER PRIMARY KEY for that table. In this case, SQLite will automatically assign a unique integer ID value for every row added to this table. This will be the record ID for each record.

With these two CREATE TABLE statements, creating an SQLite database is a breeze. SQLite stores its databases as regular files on the file system, so if I have access to this file, I have access to this database—no database user IDs and passwords to worry about! If these two SQL statements are stored in a file called create.sql, I can create the database like this:

```
sqlite library.db < create.sql
```

(Or I could write a small Perl program and use DBI calls to create the database. But this way is quicker and easier—something lazy and impatient programmers everywhere will appreciate.)

Step 2: Using Class::DBI

Now that there is an empty database, I need Perl code to add, edit, update, and delete rows. This is where things start to get very boring, very quickly.

Normally, I could start to write a Perl program by loading the DBI module, creating a database handle, and issuing SQL statements against that database handle, mixing Perl and SQL code all along the way. Alternatively, I could write a series of modules that hide these low-level operations to access my database, keeping all of the SQL code neatly isolated in one place.

However, the only things that differ between this application and the database applications I wrote last week, last month, or last year are:

- How to connect to the database
- The tables in this database
- The connections between these tables
- The columns in these tables

Class::DBI enables me to access a database just by specifying these four key pieces of information. In exchange, it provides an object-oriented interface for accessing the database, and hides all of the low-level details, like constructing SQL statements. For simple applications, I can forget that there is a SQL database lurking underneath (after I issue the CREATE TABLE statements, that is).

Using Class::DBI is simple. Very simple. It starts with a module that defines the first piece of information in my little application—how to connect to the database:

```
package MyLibrary::DBI;
use base "Class::DBI";
MyLibrary::DBI->connection("dbi:SQLite:library.db", "", "");
```

Next, I need to define one module for each of the two tables in my database. Each of these modules should declare three things: the name of the table in the database, the columns in that table, and the relationship between this table and other tables in this database. Here are those two module definitions:

```
package MyLibrary::Author;
use base "MyLibrary::DBI";

MyLibrary::Author->table("author");
MyLibrary::Author->columns(All => qw(authorid first middle last));
MyLibrary::Author->has_many(books => "MyLibrary::Book");

package MyLibrary::Book;
use base "MyLibrary::DBI";

MyLibrary::Book->table("book");
MyLibrary::Book->columns(
  All => qw(bookid author title subtitle pubyear edition isbn format)
);
MyLibrary::Book->has_a(author => "MyLibrary::Author");
```

And that is it. By writing these three trivial modules, Class::DBI will provide the guts of a database application to manage my list of books. All of the tedious, repetitive database management code is automatically provided by Class::DBI.

Step 3: Programming with Class::DBI

But what does Class::DBI actually do?

Class::DBI works by defining the common behaviors that generic database application share. It does all of the hard work, but it doesn't know how to use this particular database. To make it work in a real application, I need to plug in the specifics—where to find the database, and the structure inside that database. That's what the three modules do: customize a general-purpose framework for database applications into the framework I need for my database application today. Each of those customizations is made through inheritance (those important use-base clauses peppered about).

That's fine, but how do I use the modules I just created? It's quite simple, really. To create a new row in the author table, create a new MyLibrary::Author object. If I use the MyLibrary::Author module to search for an author record, I will get an object that represents a row in the author table. I can make changes to one of these objects, and commit those changes back to the database at a later time.

Creating a new record is easy. Just create a new object, as if it were any other Perl module:

```
# Create an empty author object,
# and an empty record in the database
my $author1 = new MyLibrary::Author;

# Create another object/row, and specify
# values for two fields
my $author2 = new MyLibrary::Author({
        first => "Isaac",
        last  => "Asimmov"
});
```

When I created the MyLibrary::Author module, I defined three important fields in the author table: first, middle, and last. Class::DBI makes these fields in MyLibrary::Author objects, and provides methods to get and set the values of these fields. Changes to these objects can be sent back to the database using the update method, and records can be deleted using the delete method on these objects:

```
## This was a blank record in the database.
## Add some values to it, and write it back.
$author1->first("Edgar");
$author1->middle("Rice");
$author1->last("Burroughs");
$author1->update();

## Wait a second — I don't have any books by this author!
$author1->delete();

## Oops! "Asimov" was misspelled
$author2->last("Asimov");
$author2->update();
```

Of course, book records are just as easy to create, update, and delete.

Class::DBI and Database Design

There is an additional wrinkle, though. Most relational databases are "normalized" to some degree. Class::DBI knows about normalization and understands that most databases are at least somewhat normalized. In my simple design for the MyLibrary database, each book record is uniquely identified by the value in its bookid field, and each author record is uniquely identified by its authorid field. Because books have authors, the author field in the book table stores one of these authorid values, which can be used to find the author of a book. (A fuller discussion of database normalization is beyond the scope of this article, but this is generally how it works in practice.)

Unique record IDs are such a common feature in relational database designs that Class::DBI automatically assumes that the first field in a table contains the record ID for that table—a safe assumption, since this practice is quite common.

Class::DBI has two main methods to describe the common relationships between tables: has_a and has_many. In this simple database design, an author has many books, and a book has a single author. (Remember, this is an admittedly simplistic model of bibliographic data.) With the MyLibrary::Book->has_a(author=> "MyLibrary::Author") declaration, I've told Class::DBI a few things:

- The author field in the book contains a unique ID from the author table.
- When I select a record from the book table, the value isn't a plain number, it is a reference to the author table. Class::DBI should use that value to instantiate the MyLibrary::Author object that corresponds to that ID value.
- If I assign MyLibrary::Author object to an author field in a MyLibrary::Book object, Class::DBI should use the ID for that author, not some other value for that author.

In this database, a book can have one author, but an author can have many books. This means that many records in the book table can share a single author value. This relationship is stated in the MyLibrary::Author->has_many(books=>"MyLibrary::Book") declaration, and it adds a method to the MyLibrary::Author module named add_to_books. This method allows me to create or edit a book record, and set that record's author field to the unique ID of this MyLibrary::Author record:

```
## First, create an author.
my $author = new MyLibrary::Author({
      first => "Isaac",
      last  => "Asimov"
});

## Create a new, standalone book record.
my $book1 = new MyLibrary::Book({
      title => "Foundation"
});
my $book2 = new MyLibrary::Book({
      title => "Foundation and Empire"
});

## Three ways to set the author of a book.
$book1->author($author);
$author->add_to_books($book2);

## Create a new book, and set its author.
$author->add_to_books({
   title => "Second Foundation"
});
```

There are many, many more methods that Class::DBI provides for manipulating data in relational databases. Some of the more important ones are:

- retrieve(): fetch an object given its unique ID value
- search(): fetch many objects given a list of fields to match
- find_or_create(): find an existing record that matches the fields specified, or create a new record with those values

Conclusion

This article just barely scratches the surface of using Class::DBI. This module has many more features, and it supports the standard databases such as MySQL, PostgreSQL, Oracle, and DB2. While this sample program demonstrates that the combina-

tion of SQLite and Class::DBI makes a quick database hack simple and easy, Class::DBI is a heavily used, robust package for simplifying all kinds of database programs.

For good measure, here is the full source of the MyLibrary module that uses Class::DBI to create an interface to my library database, and the addbook script to add a book to this database. Note that most of the work is not in managing the database, but in managing user input (editing a text file with $EDITOR and processing the result).

MyLibrary.pm.

```
#!/usr/bin/perl -w
use strict;

package MyLibrary;

package MyLibrary::DBI;
use base "Class::DBI";
MyLibrary::DBI->connection("dbi:SQLite:library.db", "", "");

package MyLibrary::Author;
use base "MyLibrary::DBI";

MyLibrary::Author->table("author");
MyLibrary::Author->columns(All => qw(authorid first middle last));
MyLibrary::Author->has_many(books => "MyLibrary::Book");

package MyLibrary::Book;
use base "MyLibrary::DBI";

MyLibrary::Book->table("book");
MyLibrary::Book->columns(
  All => qw(bookid author title subtitle pubyear edition isbn format)
);
MyLibrary::Book->has_a(author => "MyLibrary::Author");

1;
```

addbook.

```
#!/usr/bin/perl -w

use strict;
use MyLibrary;

sub get_input {
    my @template = @_;
    my $tmpfile = "/tmp/library.data.$$";

    ## Write out the template.
    open (my $fh, ">$tmpfile");
    print $fh @template;
    close($fh);

    ## Edit it...
    $ENV{EDITOR} ||= "vi";
    system("$ENV{EDITOR} $tmpfile");

    ## Read it back.
    open($fh, $tmpfile);
```

```
    my @data = &lt;$fh>;
    close($fh);

    ## Remove the temp file, and return the user data.
    unlink $tmpfile;
    return (@data);
}

sub process_input {
    my @lines = @_;
    chomp(@lines);

    my ($group, %author, %book);

    foreach my $line (@lines) {
        ## Group name/value pairs into Author and Book blocks.
        if ($line =~ m/^(\w+)$/) {
            $group = $1
        } elsif ($line =~ m/^\s+(\w+):\s*(.*)$/) {
            ## Ignore fields that weren't set.
            next unless $2;

            if ($group eq "Author") {
                $author{lc($1)} = $2;
            } elsif ($group eq "Book") {
                $book{lc($1)} = $2;
            }
        }
    }

    return \%author, \%book;
}

my ($author, $book) = process_input(get_input(&lt;DATA>));

my $author_rec = MyLibrary::Author->find_or_create($author);
my $book_rec   = MyLibrary::Book->find_or_create($book);

$book_rec->author($author_rec);
$book_rec->update();

__DATA__
Author
        First:
       Middle:
         Last:
Book
        Title:
     Subtitle:
      Edition:
         ISBN:
      PubYear:
       Format:
```

the tclsh spot

by Clif Flynt

Clif Flynt is president of Noumena Corp., which offers training and consulting services for Tcl/Tk and Internet applications. He is the author of *Tcl/Tk: A Developer's Guide* and the *TclTutor* instruction package. He has been programming computers since 1970 and a Tcl advocate since 1994.

clif@cflynt.com

The previous Tclsh Spot article showed how to extend a FORTRAN program by embedding the Tcl interpreter into the application.

This architecture—compiled mainline code with an embedded interpreter—offers a number of advantages. It provides a robust runtime configuration facility and enables the developer to concentrate on one portion of a task at a time, rather than mixing calculation, communication, and GUI elements. Writing the GUI in a higher-level language reduces development time, since this subsystem usually requires several redesigns and iterations before users are happy with it. And, from a marketing end, it means you can easily distribute the core product with different front ends.

The previous article demonstrated how to rewrite the old FORTRAN Lunar Lander using FORTRAN for the mainline code and calculation subroutine and Tcl for the user interface.

This article will expand the user interface without touching the core FORTRAN code (much).

The one change to the original FORTRAN code is to move the hard-coded FORTRAN constants into Tcl variables so that they can be defined at runtime.

The original code was:

```
! Set constants
impulse = 2000
fheight = 10000.0
speed = 100.0
fuel = 1000.0
gross = 900.0
CALL ftcl_start('config.tcl')
```

And the new code is:

```
CALL ftcl_start('config.tcl')
! Fetch constants defined in Tcl script
CALL ftcl_get_int('impulse', impulse)
CALL ftcl_get_real('ht', fheight)
CALL ftcl_get_real('speed', speed)
CALL ftcl_get_real('fuel', fuel)
CALL ftcl_get_real('gross', gross)
```

This is the obvious use for a Tcl configuration file. Config.tcl can contain code like:

```
set impulse 2000
set fheight 10000.00
set speed 100.0
...
```

Or the configuration file could contain a script to let the user set configuration options. This could be a glass tty set of questions and answers, a form to fill out, or even a button bar like this to set gravitational acceleration for different destinations:

The 8.4 release of Tcl/Tk (2003) introduced a new widget (the labelframe) and a new option to the button command (-compound) to make it easier to create button bars in Tk.

The labelframe widget (described in the previous Tclsh Spot) behaves like a normal frame; it holds other widgets, and it also supports options to control the outline and label.

Syntax: labelframe widgetName ?-option value?

The code to create the labelframe that holds the button bar is simply:

```
set w [labelframe .planet -text "Choose your Destination"]
```

The new button option -compound makes it easy to create buttons with both an image and text. Prior to 8.4, a button could contain either an image or text, but not both.

The Tk image command is quite powerful, and the newer versions of Tk are adding more facilities. Pure Tk supports images in GIF, PBM or X-Bitmap format. The img extension adds support for JPG, TIF, BMP and other formats.

You can create a Tk image either from a file or by embedding the data as base-64 data. The syntax for the image create command is:

Syntax: image create type ?name? ?options?

image create	Create an image object of the desired type, and return a handle for referencing this object.
type	The type of image that will be created. May be
	bitmap a two-color graphic.
	photo a multicolor graphic.
?name?	The name for this image.
?options?	Options that are specific to the type of image being created.

Embedding the image as a base-64 data is an easy way to create task button bars.

For example, this code will create a simple two-button taskbar that will invoke procedures named do_open and do_save when the appropriate button is clicked.

```
image create photo open -data {
        R0lGODlhEgASAPIAAAAAAICAAMDAwPj8APj8
        +AAAAAAAAAAAACH5BAEAAAIALAAAAAASABIA
        AAM7KLrc/jAKQCUDC2N7t6JeA2YDMYDoA5hs
        WYZk28LfjN4b4AJB7/ue1elHDOl4RKAImQzQ
        cDeOdEp1JAAAO/// }

image create photo save -data {
        R0lGODlhEgASAPEAAAAAAICAAMDAwAAAACH5
        BAEAAAIALAAAAAASABIAAAI3lI+pywYPY0Qg
        AHbvqVpBamHhNnqlwIkdeoJrZUlPcML0jde0
        DOnxxHrtJA6frLhC8YiNpnNRAAA7//// }

set column 0
foreach img [lsort [image names]] {
```

```
    set w [button .b_$img -image $img -command do_$img]
    grid $w -row 1 -column [incr column] }
```

The button bar that opens the Lander program is a bit more complex. It uses the -compound option to show both text and images, and it waits for the user to select a destination before continuing.

The -compound option specifies that a button should show both text and image, and defines where to place the image. The -compound key can be bottom, center, left, right, top, or none, to define where to place the image relative to the text. The value none defines the button as having either image or text, but not both. This is the default.

The buttons, created using inline base-64 data as done in the previous example, are named for and contain small images of the destination they represent: Venus, Earth, Moon, Mars.

The buttons are created with this code:

```
set w [labelframe .planet -text "Choose your Destination"]

foreach planet {Venus Earth Moon Mars} {
    set b [button $w.b_$planet -compound bottom -text $planet \
        -image $planet -command "setConditions $w $planet"]
    pack $b -side left
}
```

The button bar can be turned into a modal interaction using the vwait command (discussed in the previous "Tclsh Spot"). The vwait command will cause the Tcl interpreter to wait until a variable has changed value.

After the window is displayed, a vwait can hold the Tcl interpreter in the event loop (waiting for the user to select a destination) until the variable changes state.

```
    # Create buttons
    #...   pack $w
    vwait gravity
```

When the user clicks a button, it will invoke the setConditions procedure with the name of the parent window (the labelframe) and the destination. This procedure assigns a value to the gravitational acceleration and destination variables and destroys the parent window.

```
array set Gravities {Venus 8.8 Earth 9.8 Moon 1.7 Mars 3.9}

proc setConditions {parent dest} {
    global Gravities gravity ready destination
    set destination $dest
    set gravity $Gravities($destination)
    destroy $parent
    set ready 1
}
```

When setConditions assigns a value to gravity and returns control to the event loop, the vwait command is satisfied, and the evaluation of the script continues.

The previous version of the Lander program would wait for the user to type in a new fuel burn and click the **Go** button before calculating the new lander conditions. A more modern version of the lander will use the Tcl scale command to accept the input and run the simulation in realtime.

The scale widget allows a user to drag a slider to select a numeric value.

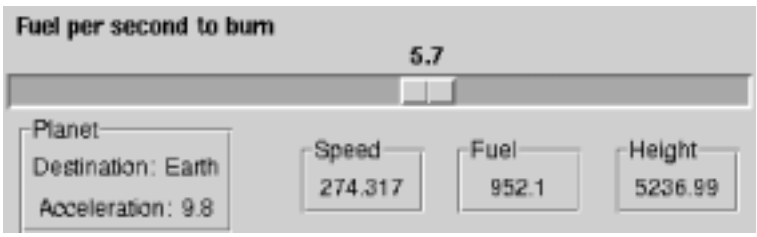
Syntax: scale scaleName ?options?

scaleName	The name for this scale widget.	
?options?	There are many options for this widget. The minimal set is:	
	-orient orientation	Whether the scale should be drawn horizontally or vertically: orientation may be horizontal or vertical. The default orientation is vertical.
	-length numPixels	The size of this scale. The height for vertical widgets, and the width for horizontal widgets. The height may be any valid Tk distance format, including inches, mm, points, or pixels.
	-from number	One end of the range to display. This value will be displayed on the left side (for horizontal scale widgets) or top (for vertical scale widgets).
	-to number	The other end for the range.
	-label text	The label to display with this scale.
	-command script	The command to evaluate when the state changes. The new value of the slider will be appended to this string, and the resulting string will be evaluated.
	-variable varName	A variable which will contain the current value of the slider.
	-resolution number	The resolution to use for the scale and slider. Defaults to 1.
	-tickinterval number	The resolution to use for the scale. This does not affect the values returned when the slider is moved.

Several Tk widgets support linking a variable to the widget, as is done using the -variable option in this scale widget. When the user moves the scale, the value of that variable automatically changes, and if the script modifies the value of the variable, the scale widget will move to reflect the change.

A horizontal scale bar to control the fuel burn can be built with this code.

```
scale .s -digits 3 -from 0 -to 10 \
   -resolution .1 -showvalue true -length 400 \
   -label "Fuel per second to burn"  \
   -variable burn -orient horizontal
grid .s -row 2 -column 1  -sticky ew -columnspan 5
```



Using this scale widget and some labelframes and labels, we can make a control GUI that looks like this to set the fuel to burn and display the current status of the lander:

You may notice that this GUI lacks the **Go** button the previous lander had for accepting the amount of fuel to burn.

We can even change the behavior of the application from stoptime to realtime in the Tk GUI, without modifying the compiled code.

The FORTRAN code calls a Tcl procedure wait4click to wait for a user to hit the **Go** button. In the stoptime version of this GUI, that procedure used vwait to pause execution until the user clicked the **Go** button.

```
proc wait4click {} {
      global ready
      vwait ready
}
...
          button .b -text "Go" -command "set ready 1"
```

To make the application run in realtime, we can use the Tcl after command to set the ready variable. The after call acts like an automated user clicking the **Go** button once a second.

```
proc wait4click {} {
      global ready
      vwait ready
      after 1000 {set ready 1}
}
```

The ready variable is modified the first time the user selects the **Gravity** (in the setConditions procedure) and is then modified by the after script once a second.

A realtime Lander game with a slider, though an improvement over the old FORTRAN type interface requiring numbers to be typed in, would be nicer yet with a graphical display. The next "Tclsh Spot" will describe better ways to represent the data.

the bookworm

by Peter H. Salus

Peter H. Salus is a member of the ACM, the Early English Text Society, and the Trollope Society, and is a life member of the American Oriental Society. He owns neither a dog nor a cat.

<peter@matrix.net>

There are many books I want to mention this month, so I'll jump right in.

Administration

Mark Burgess has written a number of worthwhile articles, including a series in *;login:*, and has delivered several excellent papers at USENIX (LISA) and other events. I even published an article by him in *Computing Systems* nearly a decade ago.

So I wasn't surprised to receive his *Analytical Network and System Administration.* I was surprised by the approach that Burgess takes to system administration; he's not really so worried about keeping the network(s) or the machines up and running as he is concerned with the functioning of the elaborate and complex "human-computer systems" that make up the information technology conglomerate.

This may well be the first book to approach system administration by means of complexity theory. It's certainly the first one I've read that doesn't concentrate on the how-tos and recipes. It's a very fine piece of work.

Interestingly, while security appears to preoccupy the journalists and the publishers, Burgess devotes only a few paragraphs to it. But I received two works on network security to compensate.

McNab's work focuses on the methodology of poking at your setup to ascertain its weaknesses. It is an insightful guide to rendering your network less permeable as well as a guide to testing.

Lockhart's book—reviewed in this issue by Rik Farrow, so I'll just give it a brief mention here—really does serve up 100 "tips and tools" to help you harden your network. I found it interesting and learned a good deal from it.

OpenBSD is Theo de Raadt's 1995 branch off the NetBSD tributary of UNIX. I've always thought of it as the variant designed for the paranoid hacker. Palmer and Nazario have written an excellent exposition concerning the sorts of secure architectures you can construct with OpenBSD. My sole problem is that I'm unclear who their audience is. If it's the elementary one, then some chapters (notably the ones on packet filtering, IPSec, and IPv6) may be too tough; if the target is the experienced sysadmin, then I think the chapters on installation, booting, packages, etc., are just padding.

A Penguin Quartet

Hardly anyone plugs in anymore. I'm not even certain that you can buy a notebook without a wireless card. So *Linux Unwired* is a handy and useful guide to Wi-Fi, Bluetooth, infrared, cellular networking, GPS, and other good stuff. If you're not clear on 802.11, 802.11a, 802.11b, . . . , 802.11i, this is the book for you.

If you want to demonstrate to the least geekish person you know that they can, indeed, run Linux (instead of another, nameless, product), get them Grant's *Linux for Non-Geeks.* There are a few things about it that irked me—e.g., Grant's use of "boot" and "boot disk" without explanation; and while Fedora core is provided with the book, at a time when I can buy a machine pre-loaded with Linux at Wal-Mart, there ought to be a chapter that bypasses the install phase entirely—but it's certainly the best

BOOKS REVIEWED IN THIS COLUMN

ANALYTICAL NETWORK AND SYSTEM ADMINISTRATION
MARK BURGESS
Hoboken, NJ: John Wylie, 2004. Pp. 366.
ISBN 0-470-86100-2.

NETWORK SECURITY ASSESSMENT
CHRIS MCNAB
Sebastopol, CA: O'Reilly, 2004. Pp. 371.
ISBN 0-596-00611-X.

NETWORK SECURITY HACKS
ANDREW LOCKHART
Sebastopol, CA: O'Reilly, 2004. Pp. 298.
ISBN 0-596-00643-8.

SECURE ARCHITECTURES WITH OPENBSD
BRANDON PALMER AND JOSE NAZARIO
Boston: Addison-Wesley, 2004. Pp. 519.
ISBN 0-321-19366-0.

LINUX UNWIRED
ROGER WEEKS ET AL.
Sebastopol, CA: O'Reilly, 2004. Pp. 297.
ISBN 0-596-00583-0.

LINUX FOR NON-GEEKS
RICKFORD GRANT
San Francisco: No Starch, 2004. Pp. 308 + 2 CD-ROMs.
ISBN 1-59327-034-8.

HOW LINUX WORKS
BRIAN WARD
San Francisco: No Starch, 2004. Pp. 347.
ISBN 1-59327-035-6.

UNDERSTANDING THE LINUX VIRTUAL MEMORY MANAGER
MEL GORMAN
Upper Saddle River, NJ: Prentice Hall, 2004. Pp. 727 + CD-ROM.
ISBN 0-13-145348-3.

PROTOCOL
ALEXANDER R. GALLOWAY
Cambridge, MA: MIT Press, 2004. Pp. 286.
ISBN 0-262-07247-5.

HACKERS AND PAINTERS
PAUL GRAHAM
Sebastopol, CA: O'Reilly, 2004. Pp. 258.
ISBN 0-596-00662-4.

thing I've seen in years for someone who wants to get eased in.

Ward's *How Linux Works* is really very good. In under 400 pages, Ward gives the reader all the information required to understand Linux internals, as I believe you need to do in order to be a skilled user. Ward's book is valuable to both the admin and those using Linux at home or in the office.

Because of some manufacturing glitch, while there are references like "Learning the vi Editor [Lamb]" and "The UNIX Programming Environment [Kernighan and Pike]" and even "A Quarter Century of UNIX [Salus]" scattered throughout the book, there's no bibliography or list of references containing fuller bibliographic information for the reader. However, No Starch has put the PDF of the references on its Web site: *http://www.nostarch.com/download/howlinuxworks_bibli.pdf*.

It's nearly 40 years since I read Ken Knowlton's ACM paper "A Fast Storage Allocator" and over 15 since Kirk McKusick and Mike Karels' USENIX paper on the 4.3MMU. But I found Gorman's *Understanding the Linux Virtual Memory Manager* really interesting.

If you're into finding out just what makes the inner penguin tick, this is a book for you. Two fish to each author in this section.

Esoterica

Galloway's *Protocol* is a thought-provoking essay in which the author contends that, far from being a place of unrestricted, freely exchanged communication, the Internet is a regulated, constrained, structured bureaucracy governed by the technical protocols (TCP/IP, HTTP, BGP, DNS, etc.). At the end of it all, I don't agree with him, but it's worth reading and thinking about what he propounds.

Graham's *Hackers and Painters* is another thoughtful (and thought-provoking) essay. Graham points out that everything seems to be turning into a computer: the typewriter, the camera, the telephone. . . . Yet society makes fun of nerds and vilifies the hackers who make many things possible. There's a lot of insight here into the intersection of art, commerce, and technology.

History and FUD

by Peter H. Salus

USENIX Historian

peter@netpedant.com

Alexis de Tocqueville observed that it is easier for the world to accept a simple lie than a complex truth.

If you have been following the SCO Group shenanigans and/or the bizarre articles by Rob Enderle or the market analyses by Laura DiDio, you will recognize the validity of de Tocqueville's remark.

But a May press release from the head of the Alexis de Tocqueville Institution, Ken Brown, has agitated me sufficiently to devote time and space to trying to counter the FUD.

Brown released a "study" in which it is "revealed" that Linus Torvalds did not "invent" Linux, which, says Brown, has "questionable" roots.

Of course, Ken Brown doesn't go into detail—this whole thing is a teaser for a "book he is writing on open source software and operating systems." The "study" was promised for May 20, but I received email from a "customer service" person, informing me that "publication has been delayed," but that material was available online.

Eric Raymond, whom I respect, read the "excerpts" that were available. His comments were devastating. Perhaps the kindest was, "This book is a disaster." Andy Tanenbaum, author of several good books as well as MINIX, stated in a published retort that Brown "is not the sharpest knife in the drawer."

(It may be worth noting that the de Tocqueville Institution is, at least in part, funded by Microsoft.)

It's actually quite easy to question Brown's assertions. But most important, one has to realize at the very outset that I don't think Linus has ever claimed to "invent" anything. (Nor am I sure that either Dennis Ritchie or Ken Thompson ever claimed to have "invented" UNIX—their 1983 Turing Award was for "the development and implementation of the UNIX operating system.")

Anyway, the roots of Linux are far from "questionable."

All knowledge builds on previous knowledge.

Sir Isaac Newton (1642–1727) said, "If I have seen further, it is by standing on the shoulders of giants." But that was derived from Robert Burton (1577–1640), who wrote, "Pigmaei gigantum humeris impositi plusquam ipsi gigantes vident" (pygmies placed on the shoulders of giants see more than the giants), deriving this from the Roman general Didacus Stella. Operating systems build on one another. My personal feeling is that it is relatively pointless to try to go back much more than four decades. But even then, at the point where IBM had transitioned from the 701 to the 704 and was moving from the 709 to the 7090, the first transistorized computer, it is clear that the big development was time sharing.

So, the first truly important implementation was Corbato's CTSS at MIT, which led to both the Multics system and to the Dartmouth Time Sharing System.

Dennis and Ken built UNICS (its original name) on their experiences with Multics following Bell Labs' withdrawal from the Multics project in spring 1969. Many important features (like | "pipe") were suggested or instantiated by others. Pipe was suggested by Doug McIlroy and coded by Brian Kernighan.

At the 1979 USENIX Conference in Toronto, AT&T announced its new licensing fees, including $7,500 per CPU for academic institutions. This led Andrew Tanenbaum of the Free University in Amsterdam to create MINIX:

> I decided to write a new operating system from scratch that would be compatible with UNIX from the user's point of view, but completely different inside. By not using even one line of AT&T code, this system avoids the licensing restrictions, so it can be used for class or individual study. (A.S. Tanenbaum, *Operating Systems, Design and Implementation*, 1987)

Several years later, a student in Helsinki, Finland, wrote an operating system, "just for fun," which he based on MINIX. Linus Torvalds was going to call it "Freax," but his sysadmin persuaded him to use "Linux."

Linux was just a kernel. Thanks to the near-universality of the Internet, it has been augmented and improved by tens of thousands of users.

So here we are—Linux is part of an implementation of a UNIX-like operating system, inspired by MINIX, and using a large number of GNU tools and applications.

Be ashamed, Mr. Brown!

book reviews

Reviewed by Rik Farrow

NETWORK SECURITY HACKS

ANDREW LOCKHART

Sebastopol, CA: O'Reilly, 2004. Pp. 298.
ISBN 0-596-00643-8.

The latest in the series of "Hack" books, *Network Security Hacks* provides you with 100 mini-security lessons in how to use security tools and techniques. I liked the book, as I find there is always something new to learn, and the Web is not always the best place to find out how a tool works.

I found most of the hacks clearly written, with enough examples to explicate the descriptions. Occasionally, I would discover a subtlety that might confound the naive user: for example, instructions for setting up SSH port forwarding that simply assume that the reader knows that the other end of the connection must already be running an sshd.

Still, I really appreciate the work that went into the hacks. The format makes it easy to pick the book up when you have less than an hour to try to improve your skills or learn how to use an unfamiliar tool. There is a lot that will be familiar to expert sysadmins, but for a beginner-to-intermediate sysadmin, there's lots to learn here.

KNOW YOUR ENEMY: LEARNING ABOUT SECURITY THREATS, 2D ED.

THE HONEYNET PROJECT

Boston: Addison-Wesley, 2004. Pp. 740.
ISBN 0-321-16646-9.

This is the second edition of the Honeynet Project's opus, and I was interested in seeing what had been added. If you follow the Honeynet Project closely, a lot will be familiar. But the second half of the book contains forensics case studies that will alone be worth the purchase price for many readers.

The Honeynet Project began in 1999 as a loose collaboration of security researchers. They began to collect information about hacking techniques by setting up networks containing honeypots, systems intended to be attacked. Over time, their techniques have improved, making their honeynets more useful as they became easier to set up.

I was most interested in the case studies in later chapters of the book. Careful explanations of how to diagnose hacked systems are both useful and rare. I especially enjoyed reading about just how much work went into decompiling a binary attack tool. I teach the basics of understanding how to assess a hacked system, but the book goes a lot deeper than I have time for.

If you want to learn about practical computer forensics, there is a wealth of material in this book. Based on real-life experience, this book is the one you want if you are permitted to diagnose one of your own systems after it has been hacked. Ideally, you will use the examples on the included CD-ROM before that occurs.

MAC OS X: THE MISSING MANUAL

DAVID POGUE

Sebastopol, CA: O'Reilly, 2003. Pp. 762.
ISBN 0-596-00615-2.

The mere size of this book hints that there is a lot missing from the OS X online help system—and I'm not just joking about the security hole discovered in May 2004. This *Missing Manual* has already proven its worth more than once for both me and my wife.

When I bought a G5 in 2003, I set it up in my office and started configuring it. At first I thought, "This is a no-brainer!" Installing software packages from CDs went smoothly, as did configuring the G5 with static IP addresses. But then I needed root, and found it wasn't there.

Well, there is a root account in MacOS X; you just need to know how to enable it. *The Missing Manual* provides step-by-step instructions, as well as important warnings about why you don't want to use the root account and a recommendation to use sudo instead.

In another test of usability, Pogue's book passed one of the hardest tests I could imagine. I had set up an account on the G5 for my wife. Part of the reason I got a G5 was that I was tired of re-installing Mac OS 9, as it would get corrupted within a couple of weeks. Now my wife could use my G5, and I could manage it using a terminal window. Well, not quite.

First, my wonderful wife managed to rename her Library folder, which has immediate and dire consequences. She really didn't even know that was what she had done, but I needed both Pogue's help and the root account to fix the problem.

But that was not the acid test. I put *The Missing Manual* by the G5 and suggested that my wife try reading the manual instead of automatically asking me questions. Not only did she read the manual, but she learned how to use the OS X calendar tool with the help of the book. She has continued to use *The Missing Manual* to learn about the Mac and to solve other problems on her own. (I still get to help with the more mysterious issues.)

The Missing Manual is clearly written and works for both technical and non-technical users. I can highly recommend this book.

USENIX notes

USACO: The USA Computing Olympiad

by Rob Kolstad
kolstad@usenix.org

[Editor's note: USENIX is a major sponsor of USACO.—RK]

The end of the domestic programming contest season is upon us. Juniors Eric Price (from Thomas Jefferson High School of Science and Technology) and home-schooled Alex Schwendner maxed the US Open with perfect 1000 point scores. They and 14 others will attend USACO training camp in Wisconsin to compete for four spots on our international traveling team headed to the International Olympiad in Informatics (IOI) in Athens, Greece, this September.

The coaches have huddled and run the USACO finals at the University of Wisconsin–Parkside. I thought you might be interested in some of the students and their interests/accomplishments. I know that we read in the newspapers too often about all the bad things that pre-college students do, whether it's wild parties that get out of hand or serious crime (my community has seen a few deadly shootings lately). The students introduced below appear to exemplify the other side of the coin. Camp participants are chosen based on their potential to win a gold medal at the international competition this year or in some future year.

A quick survey of the seniors on the All-American team yields these results: five students bound for MIT, two for Harvard (both originally from California), and one each headed for Princeton and Carnegie Mellon in Pittsburgh.

We asked them for quick little introductions on a mailing list set up for finalists, and these are presented below, in no particular order.

USACO Finalist Competitors

Joe Zimmerman is a 15-year-old Rhode Island sophomore. "I started programming in sixth grade, having stumbled across a copy of VB somewhere in my travels. Of course, I was immediately hooked, and went on to learn C++, C, Java, BASIC, in that order. Eventually I heard of these nifty things called algorithms, and a couple of years and a lot of problems later, here I am." Joe also qualified for the USA Math Olympiad (USAMO) this year (as did many other USACO members). Joe is a Linux and MacOS fan. He also enjoys "Frisbee, learning languages, watching *Seinfeld*, reading a good book, hanging out with friends, etc."

Russian immigrant Boris Alexeev, graduating senior from Athens, Georgia, not only attended last year's camp and IOI (B-team member) but also likes "math, bouldering, driving, ultimate Frisbee, music, black currant sorbet, foosball, Perl, air hockey, philosophizing, and pretty solutions." Boris also placed second in the Intel (formerly Westinghouse) Science Talent Search, a prize worth $75,000. Boris intends to major in computer science theory.

Marcello Herreshoff attends high school across the street from Stanford University in California. He runs an extensive Web site with, among other things, free software he created, poems he's written, and political and philosophical postings. He enjoys programming, poetry, juggling, hand-blowing soap bubbles, classical piano performance, philosophy, mathematics, and playing the jaw harp. He listed languages he knows: "C/C++ (with GNU extensions, of course!), Perl, various shells, Scheme/Lisp html (if you are userly enough to actually consider it a language), LaTeX, m4, cpp and other macro languages," and others. He has

created "a Linux tutorial, a planetary simulator, a version of vi written entirely in Perl, [and] a contribution to xbindkeys allowing for scheme-based configuration files."

Fourteen-year-old John Pardon hails from Durham, North Carolina, where his father is a math professor. He is interested in computer science, math, and cello. He recounts, "I don't remember exactly when I started programming; it was sometime before 4th grade, though. I took the USAMO this year for the first time and was invited to [training camp]. I am taking Calculus BC this year. I have been playing the cello for eight years. I play with the Duke University String School in their orchestra and in a chamber group. My favorite piece is the Elgar Concerto for Cello." At camp, we learned that his fifth grade programming project was an expression parse, complete with a proper parse tree. "Dad found a graduate student to help me," he demurs.

Eric Ma, from Maryland, is another math and computer competitor. "I began programming in around seventh grade with True Basic, then learned about USACO in ninth grade and began learning C++ that year.... Like a lot of others, I play chess, the piano, cards, and of course, video games. I also like listening to music (just about anything), and watching and playing sports, especially football (go Packers!) and basketball."

Anthony Kim, junior from TJHSST, likes computer programming and math and is "a dedicated member of TJ's math team and computer team. I started programming in QBasic when I was in sixth grade. I wrote several simple programs that printed out various star pyramids, [performed] basic computations, or drew something. Then I stopped programming during the following two years in middle school for some reason. I returned to programming (this time in C++) my freshman year in an introductory computer science course and took interesting computer science courses offered at TJ later. I like to play basketball, football, Frisbee and computer games (mostly old arcade games now)."

Talented Adam Rosenfield from Lexington, Massachusetts, was another member of last year's IOI B-team and says, "I've been programming since I was about nine or so. I started with QBasic, moved on to Pascal, and then on to C++, which is what I use now. I also know Java.... I enjoy playing video games, especially RPGs, adventure, and RTS games, watching the local sports teams, biking, skiing, playing Frisbee, and playing bridge. I also play the clarinet in my school's wind ensemble." Adam has been a USAMO participant for four years.

Fifteen-year-old Richard Ho also attends high school across the street from Stanford. "My interests are in computers, math, and piano. I've taken BC calculus AP during my freshman year. Also, I qualified for the USAMO in both my freshman and sophomore years. I've also received second place category award in the California State Science Fair. I started programming with TI-BASIC on my TI-89, and moved on to C/C++ when I was 10. This year, I learned Java and Scheme in school.... I like playing the piano; I have played piano for more than 10 years. I also enjoy listening to classical music, although playing it is more enjoyable." Richard's impromptu extended piano concerts at camp were awesome. Coach Percy Liang, an excellent keyboard artist who sometimes spends 20 hours/week practicing, said, "Richard's massive repertoire is incredible."

Nate Bauernfeind, 17, attends a boarding school in northern Wisconsin. "I've been programming ever since I can remember (my parents claim since before I could talk, but well... you can never trust parents). I've learned pretty much all on my own. In sixth grade I learned VB, and the following summer I took a C++ class. I enjoy playing basketball, ultimate Frisbee, photography, reading, chess, and disturbing the peace with my guitar."

Home-schooled Alex Schwendner, 17, has attended three USACO camps and two IOIs already. Alex occasionally audits graduate math classes at the University of Texas in his home town of Austin. "My favorite data structures are tries and hash tables," he writes, "and my favorite algorithms include network-flow, FFT, Bellman-Ford, Rabin-Karp, and DP algorithms. I enjoy programming for interesting projects when I have a good idea for one. I really like LaTeX. I also do the Math and Physics Olympiads, and have been invited to both of their camps this year (and previously). I'll be going to RSI (Research Science Institute) at Caltech and to MOP (the Math Olympiad Program [training camp]) later this summer. I enjoy bridge, fencing, chess, Diplomacy, science fiction, and juggling."

Junior Tom Belulovich from New Jersey "started tinkering around with QBASIC when I was about eight or so, and started learning C++ Freshman year. I also do math competitions, like AMC, AIME, and USAMO."

Colorado junior Ben Joeris says, "I've been programming since I was 10. I worked for a startup game company during junior high, but they went belly up.... Recently I have been working on research in circular string-searching algorithms with applications in group theory with Ross McConnell, a professor at Colorado State University. I enjoy playing violin (and, occasionally, guitar), composing music, and solving fun math problems. I'm also into Science Olympiad. Of course, I also love listening to music. My personal favorite bands are Anti-Flag, Propagandhi, and Apocolyptica."

Math and computer whiz Anders Kaseorg, 18, is a home-schooled senior from Charlotte, North Carolina, who won this year's national championship for sustained performance throughout the year. He has "been programming from age five, though I only started USACO last year right before the Open, in time to make training camp and the IOI team. I also do math competitions: this will be my fifth year at the Math Olympiad Program.... I juggle approximately seven balls and five clubs, and like playing the piano, Frisbee, and chess."

TJ senior Brian Jacokes has attended two camps already. He likes "math contests, ultimate Frisbee and disc golf, piano, ska music, reading (Vonnegut and Rushdie), ice cream, ping pong, chess, running (mile, 2-mile, 5k), *The West Wing,* bridge, poker, and Minesweeper. Oh, and I like pirates (not the music/software pirates [but] the ones that sail the high seas and say 'ARRRR' and plunder booty). I'm a fan of vi, Linux, and dynamic programming." Brian left camp after contest two (about 1:30 p.m. on Friday), flew to his home near Washington, DC, ran a distance event in the regional track competitions, and returned in time to win Saturday's five-hour competition that started at 8 a.m.

Tiankai Liu, 18, from California, appears extensively in the new book *Countdown* about the USA Math Olympiad team members. He attends Exeter, a prestigious private boarding school on the East Coast, and is heading into his third training camp. He's won two gold IOI medals already and more medals in international math competitions. "I started programming in QBasic in second grade. In my childhood, I wasted a lot of time playing computer games and trying to make my own. I couldn't understand pointers, so I only switched to C in tenth grade. I am also interested in mathematics. I have competed in MATHCOUNTS, ARML, USAMO, and the IMO. My hobbies include piano (which I've played for many more years than I'm good for), foreign languages (of which only in French am I close to being competent), and Quadball."

Notice the frequent ties to early programming experience, math, piano, juggling, and Frisbee. I don't know exactly how to interpret this. They are a very interesting group of people who obviously focus a bit on the technical side of life but who universally seem to have other interests, some quite far afield from those of stereotypical nerds.

The Finals

The final competition week proceeded swimmingly, with USACO director Don Piele organizing operations and head coach Rob Kolstad coordinating the contests, grading system, and extracurricular events (miniature golf, movie night, business simulation contest, amusement park—that sort of thing).

Four coaches kept the problems and analyses running smoothly:

- CMU grad student Hal Burch. You've seen his work at the map of the Internet in *National Geographic* magazine. He also co-founded a startup company with our own Bill Cheswick. Hal earned a Best Paper award at USENIX's LISA 2000 conference.
- MIT grad student Brian Dean. Brian won MIT's highest award for graduate teaching assistants last year. He's working on a multimedia algorithm book, currently at 400 pages.
- Harvard alumnus and MIT graduate student Russ Cox. Russ recently earned his Emergency Medical Technology certificate and has directed two musical theater productions over the last couple of years.
- Recent MIT graduate Percy Liang. Percy also coached MIT's ACM contest team, which placed fifth in the international final (highest of all USA teams).

Results were often printed and distributed before the competitors rose from their seats after the intellectually strenuous three- and five-hour competitions.

The opening "fun" contest (used to make sure the competitors are familiar with the extant machines, environment, and compilers) was to recode the itoa() function for speed. The better programs were running at 20x the speed of sprintf() using very clever algorithms.

A new fun event this year was IBM's "CodeRuler," a Java-based competition in which programs role-played a medieval kingdom, directing peasants and knights to take over land and even other kingdoms. A colorful presentation of the teams battling each other in frame-by-frame animation pitted USACO programs not only against each other (in rounds with 4–6 total "rules") but also against the ACM champions from this year's world competition. My recollection is that only a program from Russia bested the best of the USACO competitors' submissions. A final round showcased the program from coaches Dean and Liang against the best of the USACO finalists. Happily, the coaches prevailed. I think the IBM representative was impressed.

We chose the four finalists to represent the USA in September's International Olympiad to be held in Greece:

- Senior Brian Jacokes
- Senior Anders Kaseorg
- Junior Eric Price
- Junior Alex Schwendner

This is potentially the strongest team we've ever had, so my hopes are high for a great medal performance.

This year's USACO program was sponsored by:

- USENIX
- SANS
- ACM
- IBM

We're seeking sponsors at the contest level for next year so that we can expand our program to individuals closer to the entry level. Long discussions at camp evolved a sort of "trickle-up" theory that suggests that higher quantity and quality at lower levels will foster higher quantity and quality at higher levels as the years go by. We're expanding the competition rule; this year's highest-level competitions had as many as 300 people—all competing a full level above the highest-level competitions of only two to three years ago.

Learn more about USACO at *http://www.usaco.org,* or even try the training at *http://train.usaco.org/usacogate.*

Alain Hénon Retires

by Rob Kolstad
kolstad@usenix.org

Al Hénon, *;login:* managing editor since 2001 and typesetter since 1997, has retired as of the June issue. Al marshaled authors (and editors) extremely successfully during his tenure. Full of life and energy, his international background contributed a real flair to everything he touched.

I asked Al why he would want to retire, given the success we're having. "Rob," he said, "I've had a job continuously since 1958. That's going on 46 years without taking any real time off. I think it's time." Well, I guess so!

Please join me in thanking Al for his superb contributions and in wishing him the greatest success in his next venture. Please join me in welcoming Jane-Ellen Long back as our managing editor.

The USENIX Association Financial Report for 2003

by Ellie Young
Executive Director
ellie@usenix.org

The following information is provided as an annual report of the USENIX Association's finances. The accompanying statements have been reviewed by Michelle Suski, CPA, in accordance with Statements on Standards for Accounting and Review Services issued by the American Institute of Certified Public

Accountants. Accompanying the statements are several charts that illustrate where your USENIX and SAGE membership dues go. The Association's complete financial statements for the fiscal year ended December 31, 2003, are available on request.

FINANCIAL STATEMENTS SUMMARY

Although the trend of low conference attendance continued in 2003, USENIX is a healthy organization. In 2003, holding the same number of conferences as the previous year, USENIX had a net operating deficit of $116K (vs. $831K in 2002). During the budgeting process, USENIX attempted to break even by raising registration fees slightly, continuing to spend less in good works, and reducing staff, overhead, and expenses. Even so, a deficit resulted, due to lower conference attendance at most conferences, attrition/penalty fees imposed by the hotels for our failure to meet room block commitments, and lower membership revenue. The performance of the Reserve Fund improved, and this offset the deficit. USENIX ended the year with an increase in net assets of $457K.

USENIX MEMBERSHIP DUES AND EXPENSES

USENIX averaged 6,500 members in 2003, a 14% drop from 2002. Of these, 53% opted for SAGE membership as well.

Chart 1 shows the total USENIX membership dues revenue ($650K) for 2003, divided into membership types. Chart 2 presents how those dues were spent. Note that all costs for producing conferences, including staff, marketing, and sales and exhibits, are covered by revenue generated by the conferences.

SAGE

Chart 3 shows SAGE revenue sources for 2003 (primarily, membership dues of $130K and the revenue share from the LISA conference of $109K). Chart 4 shows all SAGE expenses (a total of $262K).

OTHER USENIX PROGRAMS

Chart 5 describes how the money allocated to Student Programs, Good Works, and Standards Activities ($108K) was spent in 2003. Chart 6 shows how the USENIX administrative expenses were allocated. (The category "other" covers such items as taxes, licenses, bank charges, and miscellaneous expenses.)

Chart 1: USENIX Membership Revenue Sources 2003

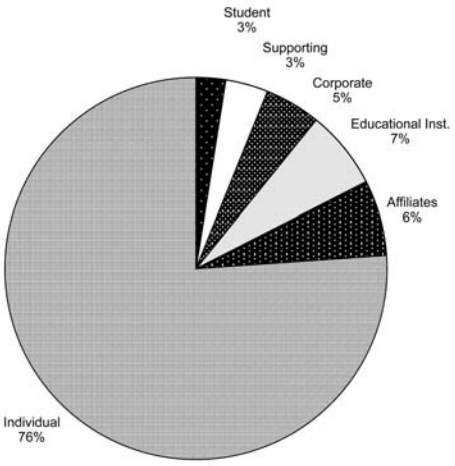


Chart 2: Where Your 2003 Membership Dues Went

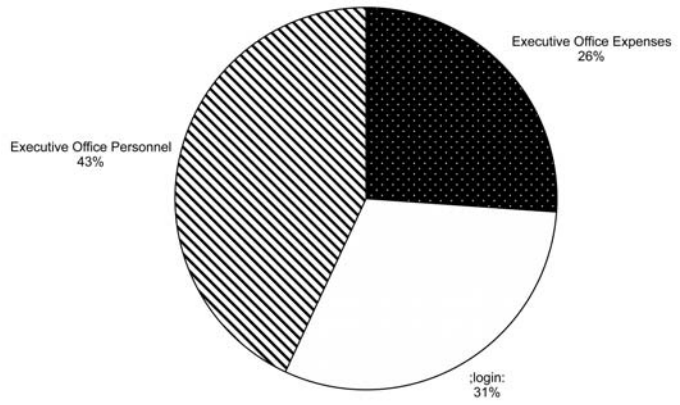


Chart 3: SAGE Revenue Sources 2003

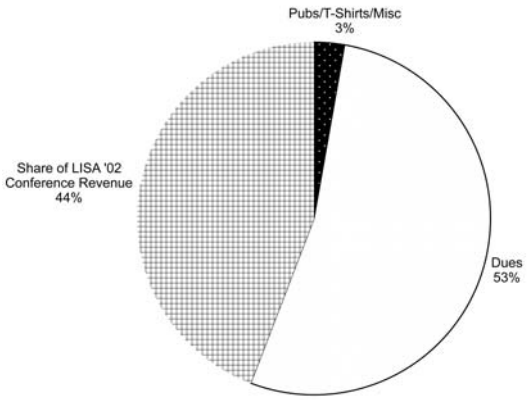


Chart 4: SAGE Expenses 2003

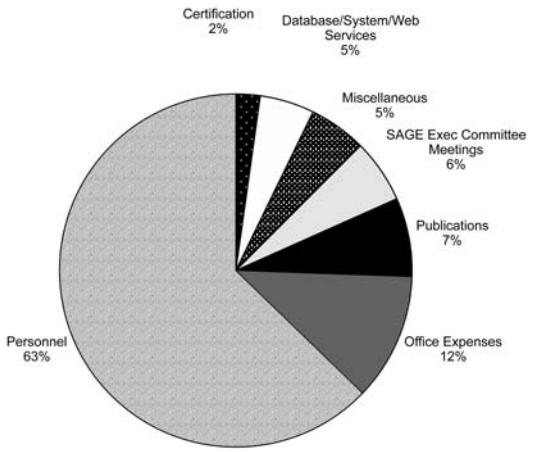


Chart 5: Student Programs & Good Works 2003

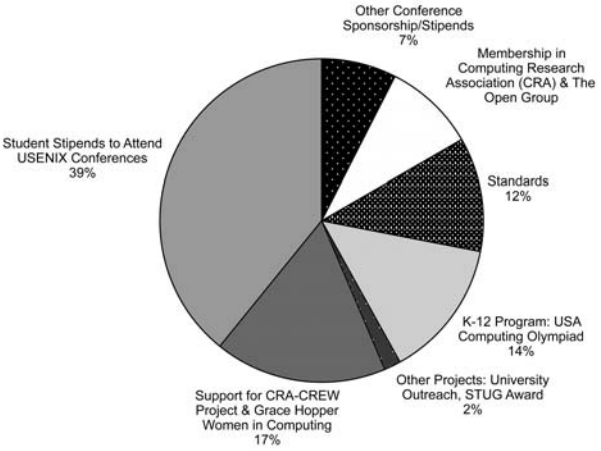
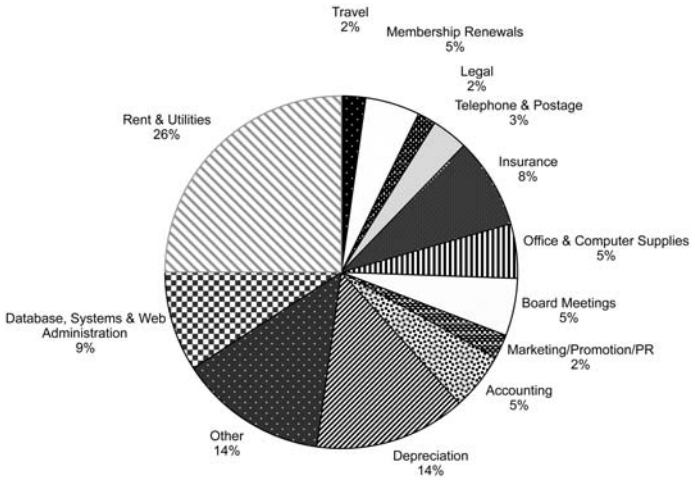


Chart 6: USENIX Administrative Expenses 2003



USENIX ASSOCIATION
STATEMENTS OF FINANCIAL POSITION
As of December 31, 2003 and 2002

ASSETS	2003	2002
Current Assets		
Cash & cash equivalents	$ 705,498	$ 1,049,294
Receivables	91,265	44,919
Prepaid expenses	24,409	27,824
Inventory	16,519	22,045
Total current assets	837,691	1,144,082
Investments at fair market value	4,916,653	4,346,762
Property and Equipment		
Office furniture and equipment	444,540	433,538
Less: accumulated depreciation	(321,667)	(256,939)
Net property and equipment	122,873	176,599
	$ 5,877,217	$ 5,667,443
LIABILITIES AND NET ASSETS		
Liabilities		
Accrued expenses	$ 108,036	$ 355,568
Deferred Revenue	13,000	12,390
Total liabilities	121,036	367,958
Net Assets		
Temporarily Restricted Assets		
Unrestricted Net Assets	5,756,181	5,299,485
Net Assets	5,756,181	5,299,485
	$ 5,877,217	$ 5,667,443

USENIX ASSOCIATION
STATEMENTS OF ACTIVITIES
For the Years Ended December 31, 2003 and 2002

	2003	2002
REVENUES		
Conference & workshop revenue	$ 3,049,096	$ 3,371,062
Membership dues	649,079	741,587
Product sales	13,524	20,129
SAGE dues & other revenue	136,119	137,182
SAGE Certification	1,200	847
Total revenues	3,849,018	4,270,807
OPERATING EXPENSES		
Conferences & Workshops	2,577,839	3,083,362
Membership; login:	470,772	553,228
Projects & GoodWorks	148,854	300,281
SAGE	242,773	442,427
SAGE Certification	6,878	316,236
Management and General	406,727	367,678
Fund Raising	110,840	39,007
Total operating expenses	3,964,682	5,102,218
Net operating surplus/(deficit)	(115,664)	(831,411)
NON-OPERATING ACTIVITY		
Donations	100	45,560
Interest & dividend income	170,155	156,327
Gains & losses on marketable securities	461,758	(865,941)
Investment fees & other	(59,653)	(69,785)
Net investment income & non-operating expense	572,360	(733,839)
Increase/(decrease) in net assets	456,696	(1,565,250)
Net assets, beginning of year	5,299,485	6,864,735
Net assets, end of year	$ 5,756,181	$ 5,299,485

USENIX ASSOCIATION
STATEMENT OF FUNCTIONAL EXPENSES
For the Years Ended December 31, 2003 and 2002

	Conferences and Workshops	Programs and Membership	Student Programs, Good Works and Projects	SAGE	Sage Certification	Total Program	Management and general	Fund Raising	Total Support		2003 Total	2002 Total
Operating Expenses												
Conference & workshop-direct	$ 1,631,705	$	$	$	$	$ 1,631,705	$	$	$ 0		$ 1,631,705	$ 1,865,746
Personnel and related benefits:												
Salaries	573,940	141,455	26,957	120,053		862,404	209,324	78,977	288,302		1,150,706	1,166,739
Payroll taxes	41,362	10,194	1,943	8,652		62,151	15,085	5,692	20,777	o	82,928	86,091
Employee benefits	107,408	26,472	5,045	22,467		161,392	39,173	14,780	53,953	o	215,345	210,737
Membership/proceedings		7,370				7,370			0		7,370	70,683
Membership/login:		187,520				187,520			0		187,520	216,016
Membership/e-learning						0			0		0	53,015
SAGE expenses				68,839		68,839			0		68,839	190,885
SAGE Certification expenses					6,878	6,878			0		6,878	309,885
Student programs, Good Works, and projects			108,019			108,019			0		108,019	285,583
General and administrative	223,424	97,761	6,891	22,762	0	350,837	143,144	11,391	154,535		505,372	646,838
	$ 2,577,839	$ 470,772	$ 148,854	$ 242,773	$ 6,878	$ 3,447,115	$ 406,727	$ 110,840	$ 517,567		$ 3,964,682	5,102,218

USENIX ASSOCIATION
STATEMENTS OF CASH FLOWS
For the Years Ended December 31, 2003 and 2002

	2003	2002
CASH FLOWS FROM OPERATING ACTIVITIES		
Change in net assets	$ 456,696	$ (1,565,250)
Adjustments to reconcile increase in net assets to net cash provided by/(used for) operating activities:		
Depreciation	64,728	73,735
Net investment income designated for long-term purposes	(108,132)	(73,987)
Realized & unrealized gains on investments	(461,758)	865,941
Decr/(Incr) in receivables	(46,346)	22,017
Decr/(Incr) in inventory	5,526	9,180
Decr/(Incr) in prepaid expense	3,415	81,153
Incr/(Decr) in accrued expenses	(247,533)	(277,935)
Incr/(Decr) in deferred revenue	610	(50,655)
Total adjustments	(789,490)	649,449
Net cash provided by operating activities	(332,794)	(915,801)
CASH FLOWS PROVIDED BY/(USED FOR) INVESTING ACTIVITIES:		
Purchase of investments	(3,730,254)	(3,765,885)
Sale of investments	3,730,254	3,765,885
Withdrawals from reserve fund		1,499,872
Purchase of property & equipment	(11,002)	(10,962)
Net cash used for investing activities	(11,002)	1,488,910
Net change in cash & equivalents	(343,796)	573,109
Cash & equivalents, beginning of year	1,049,294	476,185
Cash & equivalents, end of year	$ 705,498	$ 1,049,294

Restructuring of SAGE Governance

by Kirk McKusick
USENIX Board of Directors
kirk@usenix.org

Executive Summary

The USENIX Board of Directors, in conjunction with the SAGE Executive Committee, have decided that the current Special Technical Group (STG) model of governing SAGE has outgrown its usefulness. Beginning in July, SAGE ceased to be an STG and instead is governed by a subcommittee comprised of USENIX Board members and members of the SAGE community. The SAGE Executive Committee has been empowered to explore and undertake steps to create a separate nonprofit association to which SAGE's assets will be transferred upon successful completion of defined milestones.

Whether SAGE remains within USENIX or spins off as a separate organization, USENIX is dedicated to ensuring that current SAGE benefits and services are maintained. In particular:

- USENIX values the system administrators in its membership.
- USENIX will ensure that system administration services are supplied to its members.

If a nonprofit SAGE organization is formed, USENIX will ensure an orderly transition of benefits and services to the new organization and will cooperate with that organization to the mutual benefit of members of both organizations.

The Details

USENIX set out—over 14 years ago—to create a special technical group for system administrators. USENIX continues to want to serve sysadmins. The current system, however, does not seem to be working. Although the costs are down (SAGE almost breaks even), progress is slow.

The challenge is, then, how do we continue to serve sysadmins while changing the environment to a successful one? USENIX wishes to continue the services to system administrators that they deliver well, including the LISA conference, *;login:* (which includes much sysadmin content), the salary survey, SAGE booklets, and the sage-members mailing list. However, building a much larger, member-driven SAGE would require a significant restructuring of USENIX's business processes and probably needs to be done within a wholly different organizational structure from that of USENIX and its STG model.

Therefore, it was resolved by the USENIX Board of Directors that the STG framework for SAGE governance be dissolved effective June 30, 2004. USENIX will continue to send renewal notices to and collect dues from SAGE members and will continue to run the LISA conference, provide system administration content in *;login:*, and provide SAGE-related services including the salary survey, SAGE booklets, and the sage-members mailing list. The existing SAGE Executive Committee will serve out their terms, but no elections will be held to instate a new executive committee. During this wind-down phase of the SAGE Exec, their primary role will be to determine whether to pursue option (2) below and, if so, to initiate appropriate actions. During this time SAGE will be governed by a subcommittee of the USENIX Board composed of Geoff Halprin, Jon Hall, and Mike Jones, along with SAGE member David Parter and possibly another member of the SAGE community.

Option (1): USENIX continues to offer a SAGE membership and to provide the system administration program as an essential

part of USENIX activities. Existing programs and services are folded back into USENIX, to be governed by a subcommittee of the USENIX Board. This option starts upon the dissolution of the STG framework and will be continued for the indefinite future or until programs and services are transferred to a new organization set up under option (2).

Option (2): Separate SAGE from USENIX and allow it to go its own way under the SAGE name. A detailed memorandum of understanding was passed, containing specific milestones that are go/no-go decision points. These milestones will be further refined when and if a separate SAGE organization comes into existence.

Whichever path SAGE chooses, USENIX remains committed to its members who are system administrators and looks forward to serving their needs in the future, independently and, if SAGE becomes a separate organization, in collaboration with that organization.

2004 STUG and Flame Awards Go to M. Douglas McIlroy

Doug McIlroy, winner of both the 2004 Software Tools User Group Award and the 2004 USENIX Lifetime Achievement Award, wrote some of the most basic and timeless tools for UNIX, including sort(1), spell(1), diff(1), join(1), graph(1), speak(1), and tr(1); significantly influenced the design of macros; contributed to various computer languages; and also delved into computer security, graphics, cartography, storage allocation, and garbage collection, and even documentation techniques. For more information, see *http://www.usenix.org/about/flame.html* and *http://www.usenix.org/about/stug.html.*

conference reports

3rd Virtual Machine Research and Technology Symposium (VM '04)

San Jose, California
May 6–7, 2004

TECHNICAL SESSIONS

(The first Keynote Address, on "Virtual Machines: Past, Present, and Future," was presented by Mendel Rosenblum of Stanford University.)

2D KEYNOTE ADDRESS

The Mono VM

Miguel de Icaza, Co-Founder and CTO, Ximian

Summarized by Maria Cutumisu

Miguel de Icaza discussed the implementation of Mono, an open source execution engine for the ECMA CLI specification. The Mono VM was implemented by an enthusiastic group of people who were newcomers to the virtual machine domain but were attracted by the social, technical, and personal aspects of this project. They were interfacing with a large community of developers around the world and were observing a growing user community. The speaker started with a brief description of various systems, interesting for their capabilities with respect to the Mono project: UNIX, the Gnome Project, and Latte 2000. He continued with historical information about Ximian which was focused on making Linux succeed on the desktop.

At the time Mono was launched, the intent of the authors was to bring .NET features to Linux (C# compiler, virtual machine, core class libraries) and to provide open source features well suited to distribute the work. The team did not have any experience in compilers or virtual machines at the time. Currently, Mono has become an open source implementation of .NET that is based on the ECMA/ISO standards, includes C# and VB compilers, and works with third-party compilers, such as Delphi, Eiffel, COBOL, FORTRAN, Mercury, Python.NET, PerlSharp, and Nemerle.

The speaker described in detail several Mono features, including multi-language support, two stacks, C# compiler, virtual execution system, runtime, JIT environment, and support for optimized code compilation. Today Mono benefits from extensive inlining of intrinsic operations and an SSA-based representation. The talk concluded with an interesting discussion about research in virtual machines and compilers, as well as a brief outline of .NET limitations. Mono URL: *http://www.go-mono.com*.

A Virtual Machine Generator for Heterogeneous Smart Spaces

Doug Palmer, CSIRO ICT Centre

Summarized by Maria Cutumisu

Doug Palmer presented a virtual machine generator that provides "numerous virtual machines, each tailored to

This issue's reports focus on the 3rd Virtual Machine Research & Technology Symposium, held in San Jose, California, May 6–7, 2004.

Our thanks to the Summarizers:
Maria Cutumisu
Vivek Haldar
Yahya H. Mirza
Feng Qian
Ananth I. Sundararaj

Photo: VM '04 Program Chair Tarek Abdelrahman, with Best Paper winners Vivek Haldar and Deepak Chandra (not shown: co-author Michael Franz)

the capabilities of a class of resources." The speaker started by defining heterogeneous smart spaces as "networks of communicating, embedded resources and general-purpose computers that have a wide spread of power and capabilities." These spaces are typical of commercial, agricultural, or other outdoor environments.

The author then stated the central programming problem: Each heterogeneous smart space is unique; therefore a programming model that allows domain knowledge to be reused across smart spaces is necessary. The virtual machine generator constitutes a solution to the problem of providing a single virtual machine implementation that operates in heterogeneous smart spaces.

The speaker illustrated the virtual machine generation process and talked about the subset declaration for a virtual machine. The virtual machine is specified in an XML document, and this specification allows a stack-based virtual machine to be generated. A virtual machine specification and a subset declaration together constitute the input for the generator. The generator analyzes the virtual machine and generates source code files for Java and C that implement the subset virtual machine. These source files are compiled and linked in the presence of a standard library of support functions and classes; an assembler is generated at this point. The complete virtual machine is analyzed, and instruction codes, event codes, and stores are allocated before subsetting.

In conclusion, the advantages of the compact generator were outlined, including the fact that any optimizations that are made will propagate to future generated virtual machines. Moreover, "using a generator allows virtual machines to be quickly generated for new resources and to try new instruction sets."

MCI-Java: A Modified Java Virtual Machine Approach to Multiple Code Inheritance

Maria Cutumisu, Calvin Chan, Paul Lu, and Duane Szafron, University of Alberta

Summarized by Yahya H. Mirza

Duane Szafron presented an attempt to decouple the various roles a class plays—concept, interface, implementation, representation, factory, and extent—by separating them. The paper makes the case that most object-oriented languages do not separate these notions. The authors state that Java loses an opportunity for code reuse. This problem is illustrated by showcasing a concrete example from the Java I/O libraries. In Java a class can't inherit code from two parents, since it does not support multiple code inheritance.

A new language construct, "implementation" is presented as a solution. An implementation is essentially an interface with pure behavior code, but does not include data. With this approach, one can inherit code, but not data, from two parents, thus relaxing Java's inheritance semantics. The authors claim that they achieved significant code reuse by adding this feature to Java. The paper also adds a new "multi-super" mechanism which can be used to define an inheritance path to a particular super-implementation.

The "implementation" language feature is applied by making a minimal number of changes to the Jikes Java compiler and the Sun JVM. The compiler code generation process includes generating an invokeinterface for calls for which the static receiver type is an implementation. An invokespecial is generated for multi-super calls but with a reference to an interface instead of a class; the virtual machine can recognize these calls since all invokespecial bytecodes refer to classes. Finally, when the receiver is "this," an invokeinterface is generated instead of the usual invokespecial.

The changes to the Sun JVM include changes to the class loader (interface method table construction algorithms). These changes included detecting and reporting potential ambiguities, copying code pointers from interfaces to classes, and, finally, creating new method blocks on the JVM C-heap in two rare scenarios. The presenter stated that the resolution and dispatch of invokevirtual and invokeinterface bytecodes and the quickening of these bytecodes did not change. A key lesson learned from this project was that to make an efficient change to the VM, one must make changes when the class is loaded, but never during dispatch.

Semantic Remote Attestation—A Virtual-Machine-Directed Approach to Trusted Computing

Vivek Haldar, Deepak Chandra, and Michael Franz, University of California, Irvine

Summarized by Maria Cutumisu

This paper won the Best Paper award. Vivek Haldar presented a framework for semantic remote attestation, as well as two example applications built within this framework: a distributed computing client-server application (Mersenne Primes) and a Gnutella-like peer-to-peer network protocol. In contrast with current static techniques for remote applications, his team's approach uses language-based virtual machines to enable the remote attestation of dynamic program properties independently of the underlying platform. Their two examples illustrate applications that distribute trust dynamically.

One of the key questions addressed in the talk was how to transcend the notion of trust from closed systems to open systems. Trusted computing constitutes the effort of adding components and mechanisms in open systems with the goal of providing trust. As a result, the integrity of the system is checked and enforced, and the system is allowed to authenticate itself to remote systems.

The speaker talked about critical issues in trusted computing and remote attes-

tation, with a focus on integrity (ensuring a secure boot process), authenticity, and trust vs. security. Moreover, he stressed how virtual machines can make trusted computing more secure, flexible, and effective. In particular, problems with remote attestation were discussed, including issues such as the lack of program behavior attestation, the nature of the remote attestation (static, inexpressive, and inflexible), the heterogeneity of devices and platforms, and the revocation problem inherited from public-key cryptography.

The solution proposed by the authors is the implementation of a prototype framework for semantic remote attestation, i.e., the use of a trusted virtual machine (TrustedVM) for remote attestation. Virtual machines execute platform-independent code with rich meta-information. In addition, the code runs under the control of a virtual machine. A trusted VM can attest to properties of classes, as well as dynamic and system properties.

Several advantages of semantic remote attestation were outlined during the presentation, such as certified program behavior, the capability of allowing various implementations of the same program respecting certain security requirements, dynamicism and flexibility, explicit trust relationships (checked and enforced) between nodes, and a mechanism for finer-grained trust using degrees of trustworthiness. In conclusion, the speaker pointed out that currently proposed mechanisms for trusted computing are severely limited and that leveraging VM technologies can make trusted computing more flexible and effective.

Towards Scalable Multiprocessor Virtual Machines

Volkmarr Uhlig, Joshua LeVasseur, Espen Skoglund, and Uwe Dannowski, University of Karlsruhe

Summarized by Feng Qian

The paper presented a new algorithm, time ballooning, for better scheduling of virtual machines in a multiprocessor environment. The combination of techniques enables scalable multiprocessor performance with flexible virtual processor scheduling. Experimental results demonstrate that the new approach is effective.

Using Hardware Performance Monitors to Understand the Behavior of Java Applications

Peter F. Sweeney, Brendon Cahoon, Perry Chen, David Grove, and Michael Hind, IBM T.J. Watson Research Center; Mathias Hauswirth and Amer Diwan, University of Colorado at Boulder

Summarized by Feng Qian

Large Java applications have many complex components. The paper introduces the design of an extension of a Java Virtual Machine (JikesRVM) for helping programmers to understand the application behaviors.

The new extension generates traces of hardware performance monitor counters. The mechanism can generate separate traces for each thread in a multithreaded and multiprocessor environment. The events, such as instruction per circle (IPC), cache misses, etc., expose the behavior of a Java application at the architecture level. These traces are useful for JVM developers to improve JIT compilers and garbage collectors. Authors also reported the design of a tool, Performance Explorer, for visualizing trace data. The tool can extract metrics from a trace file. Using SPECjbb2000 as an example, the paper shows anomalies observed by Performance Explorer.

vBlades: Optimized Paravirtualization for the Itanium Processor Family

Daniel J. Magenheimer and Thomas W. Christian, Hewlett-Packard Laboratories

Summarized by Vivek Haldar

Daniel Magenheimer specifies that, because of their design, some processors are more "virtualizable" than others. The Intel x86 and Itanium architectures are hard to virtualize, while the PowerPC and future Intel architectures (Vandebuilt) are easier to virtualize. When an architecture cannot be fully virtualized, this has adverse impacts on both complexity and performance of a virtual machine. Parts of guest operating systems have to be dynamically rewritten, page tables need to be explicitly managed, and privilege-level leakage must be guarded against. Performance suffers due to additional ring crossings and an increased number of context switches between the virtual machine monitor and the guest OS.

The alternative to this is paravirtualization. The virtual machine monitor provides an interface similar but not identical to the physical machine. The guest OS in turn needs to be modified to accommodate this differing abstraction. The advantage of this approach is that full multi-application commercial OSes can be supported, application-level modification is not needed, and there is near-native performance. The disadvantage, of course, is that the guest OS needs to be modified. The author described vBlades, an HP Labs research prototype. It is an Itanium-based hostless virtual machine monitor that runs on bare metal. It provides the capability for full virtualization. A few sensitive instructions are statically translated. An API for paravirtualization is provided. It achieves within 2% of native performance.

Kernel Plugins: When a VM Is Too Much

Ivan Ganev, Greg Eisenhauer, Karsten Schwan, Georgia Institute of Technology

Summarized by Vivek Haldar

Ivan Ganev describes an extension mechanism for operating system kernels that provides safety, extensibility, and low performance overhead. The claim is that full virtualization is not necessary for providing strong isolation to kernel plugins—using virtual machines to solve this is overkill. Virtual machines are not lightweight and have to deal with a

whole array of low-level machine issues, such as the BIOS, I/O, and other legacy hardware.

The alternative is to use kernel plugins that employ other mechanisms for safety and isolation. This is done with a combination of hardware and software techniques. The hardware memory man-agement unit is used to enforce segmentation and memory isolation. Dynamic code generation enables arbitrary and heterogeneous adaptation on the fly. Dynamic linking maintains a clean interface between the kernel and plugins and manages namespaces.

This architecture was evaluated on a client-server benchmark. An in-kernel Web server (khttpd) was used on the server. The client was set up to be much faster than the server so that the server could be saturated. The cost of running a null plugin was negligible. The throughput of the server with and without the plugin was almost the same. Future avenues of work include fault recovery and isolation, and an IA64 port.

The Virtual Processor: Fast, Architecture-Neutral Dynamic Code Generation

Ian Piumarta, Université Pierre et Marie Curie

Summarized by Yahya H. Mirza

Ian Piumarta presented VPU, a reusable dynamic code generation infrastructure that can be used as a back end for dynamically compiled languages. Piumarta emphasized that a key element of VPU's design was to make adding dynamic code generation capabilities to an existing application essentially "plug-and-play." Today the vast majority of compiler infrastructures are either designed for static compilation, focus on low-level code generation, or are tightly coupled to their underlying source languages. These issues make it difficult to retarget current compiler infrastructures to other applications or language implementations. Additionally, Piumarta illustrated how a client interacts with the VPU's stack-based, processor-independent computational model to generate efficient native code.

The presentation also described the phases of the VPU's compilation process, including conversion to an internal abstract representation, application of several optimizations, instruction selection, register allocation, and native-code generation. Since the VPU tries to generate code as fast as possible, it only implements a small number of processor-independent optimizations. These optimizations are designed to occur in parallel with other traversals of the VPU's abstract representation, such as type or control flow analyses. Instruction selection is implemented through a table-driven approach using a small number of heuristics. The tables themselves are generated by feeding a processor-description file to a program called cheeseburg, which shares similarities with existing instruction selection generators such as iburg and lburg.

Systems using VPU are insulated from the underlying processor architecture and are supported on all VPU platforms, including the Pentium, SPARC, and PowerPC architectures. The VPU currently serves as the execution engine for the YNVM dynamic interactive incremental compiler and as the code generator for the JNJVM.

LIL: An Architecture-Neutral Language for Virtual-Machine Stubs

Neal Glew, Spyridon Triantafyllis, Michal Cierniak, Marsha Eng, Brian Lewis, and James Stichnoth, Intel

Summarized by Feng Qian

Machine code stubs are often used in implementing high-performance runtime systems for languages such as Java and CLI. To ease the task of coding, the authors presented a domain-specific language, LIL, for describing the functionality of such code stubs in a high-level, architecture-neutral manner. A special compiler transfers the description in LIL to architecture-dependent native instructions. LIL also has engineering benefits, such as improved readability and validity checks of stubs. The LIL compiler is faster and produces efficient machine code for stubs.

Detecting Data Races Using Dynamic Escape Analysis Based on Read Barrier

Hiroyasu Nishiyama, Hitachi, Ltd.

Summarized by Feng Qian

Data race can result in unexpected behaviors, and data race detection is an important method for locating potential bugs in concurrent programs. This paper proposed a new dynamic data race detection algorithm for Java. Based on the observation that only objects truly accessed by multiple threads require data-race monitoring, the new approach uses read-barrier to build the set of objects potentially subjected to data race. The number of monitored objects was reduced when compared with a write-barrier-based approach, which assumes all objects reachable from global objects are escaping. Furthermore, the author improves the dynamic escape analysis of arrays by dividing an array object into sub-blocks. The smaller number of monitored objects at runtime reduces the cost of dynamic data race detection and also improves the precision (reducing false alarms). The implementation of the proposed method and evaluation on a set of standard Java benchmarks shows the new approach is superior, both in accuracy and efficiency, to existing write-barrier approaches.

Towards Dynamic Interprocedural Analysis in JVMs

Feng Qian and Laurie Hendren, McGill University

Summarized by Vivek Haldar

The goal of this paper, presented by Feng Qian, was to perform interprocedural analysis in order to support speculative optimizations in a JIT compiler. This is a challenging problem because: (1) it is hard to construct a high-quality call graph efficiently; (2) dynamic class loading must be handled; and (3) the

analysis must accommodate unresolved symbolic references. The problem attacked in this paper was the first one: to construct a call graph dynamically.

The call graph is constructed incrementally, under conservative assumptions. Profiling stubs are inserted into methods to accomplish this. Rapid type analysis and class hierarchy analysis is used to resolve non-virtual and interface method calls. The runtime overhead for this is 2–3%. These results are optimistic, and future work hopes to undertake and make use of more advanced interprocedural analysis.

Java Just-in-Time Compiler and Virtual Machine Improvements for Server and Middleware Applications

Nikola Grcevski, Allan Kielstra, Kevin Stoodley, Mark Stoodley, and Vijay Sundaresan, IBM Canada Ltd.

Summarized by Yahya H. Mirza

IBM Canada's JVM product team presented a series of optimizations to enhance server and middleware performance. These optimizations are shipped as a part of the IBM Developer Kit for Java and the J9 Java Virtual Machine products. As a result of going over large amounts of customer-specific Java code, IBM identified three issues that significantly impacted performance: bytecode generation, finally blocks, and large usage of exceptions. To remedy these and other performance issues, IBM introduced 12 separate enhancements, including optimizations to synchronization and Java class libraries.

Server performance optimizations include both JIT and VM improvements. Many of the server enhancements target, in particular, the SPECjbb2000 benchmark. These optimizations include object allocation inlining, lock coarsening, thread-local heap batch clearing, and the utilization of the Intel SSE instructions. The performance of middleware applications have been improved through the SPECjAppServer2002 benchmark. Start-up time is improved through multiple recompilation strategies by the JIT. Interface dispatch is optimized by polymorphic inline caches. In addition, 64-bit variables, themselves used to perform unsigned 32-bit calculations, are recognized and dealt with. Finally, code reordering is utilized to minimize instruction cache misses and branch mispredictions.

The results from this project indicate that such performance improvements are not necessarily additive, and some are platform specific. Thus the performance improvements achieved are not indicative of potential improvements for future platforms. The SPECjAppServer-2002 benchmark shows the potential of these optimizations: Polymorphic inline caches and code reordering give a combined improvement of 14% for the IBM Developer Kit for Java.

Java, Peer-to-Peer, and Accountability: Building Blocks for Distributed Cycle Sharing

Ali Raza Butt, Xing Fang, Y. Charlie Hu, and Samuel Midkiff, Purdue University

Summarized by Ananth I. Sundararaj

This paper, presented by Ali Raza Butt, is based on the increased popularity of grid systems and cycle sharing across organizations. The authors attempt to build one such system that would be scalable and provide means to locate resources and further ensure that these resources are used fairly. The main goal is that all the participants should be able to utilize the system. The problem of resource discovery and management is solved using existing P2P networks. Portability is provided by leveraging the Java Virtual Machine. The ability to remotely monitor Java programs' progress provides for some security. The authors have developed a distributed credit-based system of accountability to ensure fairness.

Because cheaters can be effectively and easily removed from the system, the overhead for monitoring jobs is virtually eliminated. So the main building blocks for providing fair cycle sharing are peer-to-peer networks, Java-based progress monitoring and security, and credit-based accountability mechanisms. More information on this project can be accessed at *http://expert.ics.purdue.edu/~butta*.

Towards Virtual Networks for Virtual Machine Grid Computing

Ananth I. Sundararaj and Peter A. Dinda, Northwestern University

Summarized by Ananth I. Sundararaj

The work has been done in the context of Virtuoso. The high-level aim of the Virtuoso project is to provide arbitrary amounts of computational power to ordinary people to perform distributed and parallel computations. The traditional methodology of doing grid computing, which involves resource multiplexing using OS level mechanisms, addresses this aim. A problem with this approachis that it presents too much complexity both from the perspective of the resource provider and that of the resource user. Virtuoso proposes to do grid computing in the context of OS-level virtual machines, where the abstraction is that of a raw machine.

A very interesting networking problem shows up in this new context. A particular user's virtual ma-chines are spread over a number of foreign networks. These machines are at the mercy of the foreign network administrator for their network connectivity. The authors wish to move this network management problem back to the home network of the user. VNET is the virtual network tool that accomplishes this. The authors provided performance results for VNET and showed that its performance is quite close to that attained in the underlying network. VNET is an overlay network of VNET daemons and has a lot of potential to improve performance through, for example, network reservation. The first iteration of VNET is publicly available from the Virtuoso Web site. More information on this project can be accessed at *http://virtuoso.cs.northwestern.edu*.

WORK-IN-PROGRESS REPORTS

Efficient Code Caching for an Embedded Dynamic Adaptive Compiler

Oleg Pliss and Bernd Mathiske, Sun Microsystems, Inc.

Summarized by Maria Cutumisu

Bernd Mathiske and Oleg Pliss reported on various code caching techniques in an embedded Java Virtual Machine (JVM) for memory-constrained devices, such as mobile phones. In such environments, the presence of a dynamic adaptive compiler is salutary, as the compiled code cache management becomes performance critical.

The compiled code cache can be dynamically adjusted in size due to combined results of profiling and garbage collection feedback. Recently and frequently executed methods are profiled using a combination of sampling and instrumentation techniques with the goal of constructing a cache eviction policy based on method weight and decay.

The talk highlighted the process of code cache eviction from the perspective of the results collected from the garbage collector. Methods that are identified as nonrelevant are selected as victims, while currently executed methods are retained in the cache by setting the high bit of their weight.

The authors presented results on all EEMBC benchmarks showing large performance increases, due to the improvements in the working method set detection and cache size management.

Solaris Zones: Operating Systems Support for Server Consolidation

Andrew Tucker and David Comay, Sun Microsystems, Inc.

Summarized by Maria Cutumisu

Andrew Tucker and David Comay introduced Zones, a new operating system abstraction for partitioning systems such that multiple applications run in isolation from each other on the same hardware, within a single operating system instance. Zones has an abstraction layer that separates applications from the physical attributes of the designated machine.

Different zones can be administered in a similar manner on separate machines. A zone can have access to dedicated resources or can share resources with other zones. Each zone has its own name service identity, password file, and root user. With Zones, there are multiple virtualized views of the process table corresponding to processes running within individual zones, as reflected in the /proc file system. Moreover, each zone has a virtualized /etc/mnttab file that shows only file system mounts in that zone. Even if a zone is compromised by an intruder, the system and other zones are not affected.

The isolation provided by Zones prevents processes running in different zones from monitoring or altering each other, seeing each other's data, or manipulating the underlying hardware. The cost of running multiple workloads on the same system is reduced through a better hardware utilization, reduced infrastructure overhead, and lower administration costs. The authors presented results showing that the performance impact from using zones is negligible.

During the talk, the authors indicated several resources supporting their system, including http://www.sun.com/bigadmin/content/zones. Zones are developed as part of the N1 Grid Containers feature in Solaris 10. A version of Solaris 10 that includes an initial implementation of zones is available for download from http://wwws.sun.com/software/solaris/10.

Opcode Level Energy Consumption Model for a JVM

Sebastian Lafond and Johan Lilius, Turku Centre for Computer Science, Finland

Summarized by Vivek Haldar

Sebastian Lafond presented a simulation to measure the energy consumption of Java programs on mobile devices. Java bytecode is executed on the KVM (an implementation of the J2ME standard, which interprets Java bytecode) running on an ARM processor, and the resulting instruction trace is passed through an instruction-level energy profiler. The authors found that some KVM stages consume a constant amount of energy independently of the Java application being run. The most energy-expensive operation was the dup2_x2 instruction.

Real-Time Garbage Collector for Embedded Applications in CLI

Okehee Goh and Yann-Hang Lee, Arizona State University; Ziad Kaakani and Elliot Rachlin, Honeywell International Inc.

Summarized by Vivek Haldar

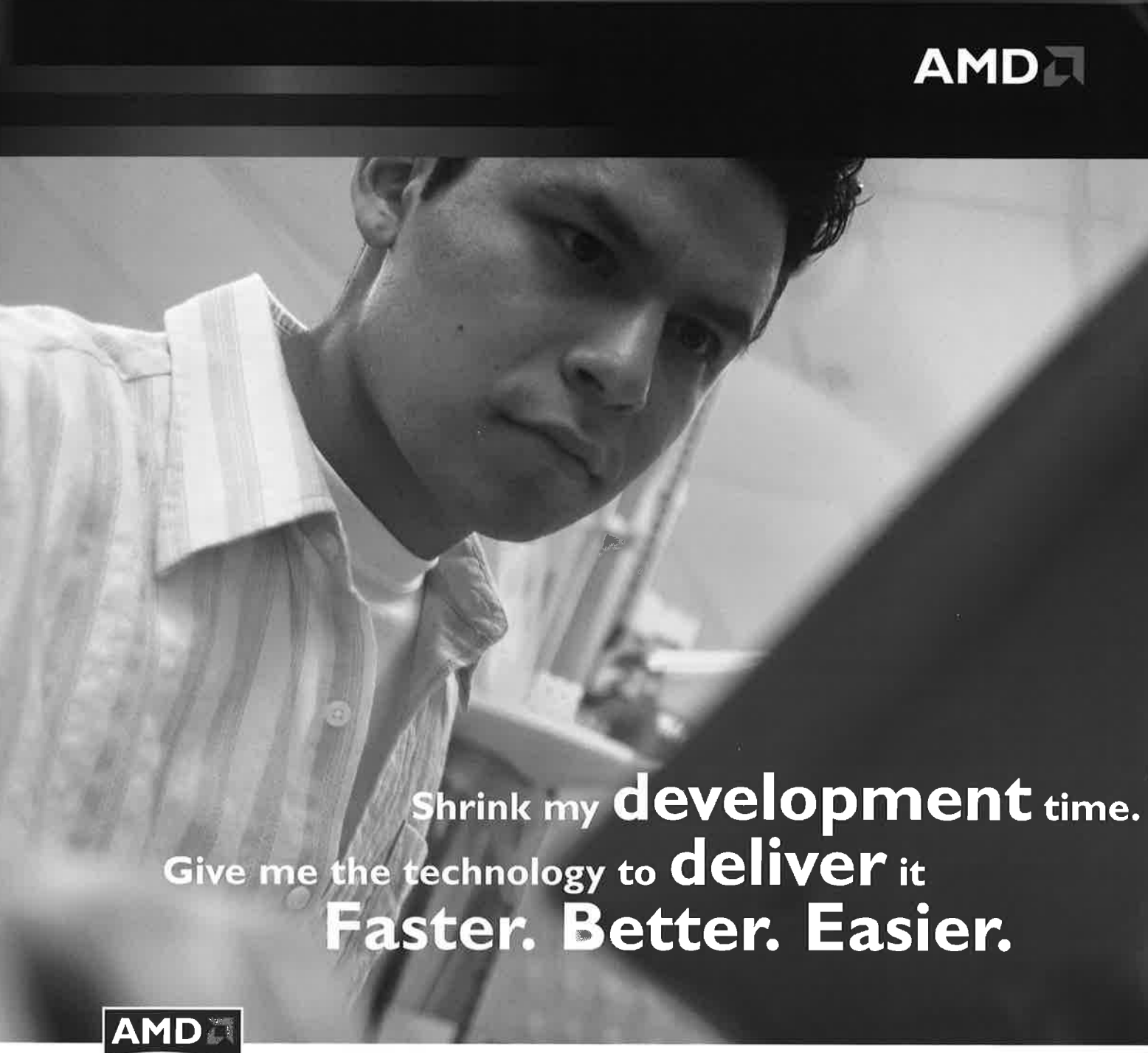
In the .NET Common Language Infrastructure (CLI), determinism is an issue for time-constrained applications. However, garbage collection is non-deterministic. The goal is to schedule garbage collection by applying real-time scheduling algorithms. This can be used to control the garbage collector's pause time and do incremental garbage collection. So far, the authors have modified the Mono runtime (which uses the mark-sweep Boehm collector) to generate write barriers. This can help to make garbage collection incremental at a fine granularity.

One-Click Distribution of Preconfigured Linux Runtime State

Richard Potter, Japan Science and Technology Corporation

Summarized by Feng Qian

Richard Potter's work-in-progress reports the idea and applications of ScrapBook for User-Mode Linux (SBUML). SBUML can take a snapshot of the transient runtime state of the Linux OS, and the state can rapidly be restored in another computer. SBUML could be used to distribute preconfigured Linux runtime state for demos or debugging. More details can be found at http://sbuml.sourceforge.net.



AMD64

Faster. Outsmart your development deadlines with AMD64 technology.

Better. Direct Connect Architecture lets you do more.

Easier. Your platform choice is simpler, since AMD64 technology excels across a wide variety of application workloads.

Register at developer.amd.com and enter a drawing for a chance to win an AMD64 system. See official rules for details and eligibility requirements.

;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