;login:

THE USENIX MAGAZINE

June 2004 • volume 29 • number 3

inside:

USENIX '04

Annual Technical Conference

USENIX

The Advanced Computing Systems Association

USENIX Upcoming Events

2004 Linux Kernel Developers Summit

JULY 19-20, 2004, Ottawa, Ontario, Canada
http://www.usenix.org/events/kernel04/

13th USENIX Security Symposium

AUGUST 9–13, 2004, SAN DIEGO, CA, USA
http://www.usenix.org/events/sec04

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

Organized by NLUUG, co-sponsored by USENIX and Stichting NLnet

SEPT. 27–OCT. 1, 2004, Amsterdam, The Netherlands
http://www.nluug.nl/events/sane2004/

Internet Measurement Conference 2004 (IMC 2004)

Sponsored by ACM SIGCOMM in cooperation with USENIX

OCT. 25-27, 2004, Taormina, Sicily, Italy
http://www.icit.org/vern/imc

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

Sponsored by USENIX and SAGE

NOVEMBER 14–19, 2004, Atlanta, GA, USA
http://www.usenix.org/events/lisa04/
Final Papers due: September 7, 2004

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
http://wmcsa2004.lancs.ac.uk/

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

DECEMBER 5, 2004, San Francisco, CA, USA
http://www.usenix.org/worlds04

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

DECEMBER 6–8, 2004, San Francisco, CA, USA
http://www.usenix.org/events/osdi04
Notification of acceptance: August 2, 2004
Papers due for shepherding: September 3, 2004
Camera-ready final papers due: October 4, 2004

2005 USENIX Annual Technical Conference

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

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

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

Tenth Workshop on Hot Topics in Operating Systems (Hot OS X)

JUNE 12–15, 2005, Santa Fe, NM, USA
Paper titles and abstracts due: October 8, 2004
Finall Paper Submission due: October 15, 2004

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

contents

;login: Vol. 29, #3, June 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 Editor, and also head coach of the USENIX-sponsored USA Computing Olympiad.

<kolstad@usenix.org>

Outrage

Do you keep up with various controversies? I'm not talking about the Iraq War, which seems to be the number one controversy right now. I'm not even talking about voting machine issue in California. I'm talking about a little known controversy surrounding a chemical that is found through the world and is extremely dangerous.

What does this have to do with *;login:* and computers? The main data clearinghouse is located on the Web at *http://www.dhmo.org* . For whatever reason, little information has made it into the print media or other news services. In fact, this site is almost the only information source that presents the DHMO data in this way. I think it's important that this site be understood – and outrage about it put into context.

Basically, the Web site provides pages and pages of discussions, facts, and recommendations about the chemical known as DHMO, dihydrogen monoxide. The site includes:

- Environmental impact statements – "DHMO contributes to global warming and the "Greenhouse Effect"; it is one of the so-called 'Greenhouse gases'. It is a causative agend in most instances of soil erosion. . . . Measurable levels of DHMO have been verified in ice samples taken from both the Artic and Antarctic ice caps. Recent massive DHMO exposures have led to loss of life and property in California, the US-Midwest, and the Philippines, just to name a few.
- Relationships to cancer – "The causative link . . . has not been established, although a significant amount of evidence seems to suggest that DHMO at least plays a role in the formation of cancer, including: Hodgkin's lymphoma, . . . chondrosarcoma, fibrosarcoma, colorectal cancer, leukemia . . ." [another half dozen are listed.] Links to cancer agencies are included.
- DHMO and the dairy industry – DHMO is used throughout the US and international dairy industries. DHMO is found in every nation's milk supply. No regulations exist for using DHMO in dairy cattle, though it is widely used to increase milk production.
- The link between DHMO and child safety – inhaling it even in small quantities causes far too many deaths (actual statistic for just one age group: 3.1 deaths per 100,000 children ages 1-4 in the year 2000 died this way in the USA).

Other Dangers

Solid DHMO causes tissue damage in prolonged exposure; DHMO is a major component of acid rain. Gaseous DHMO causes severe burns; liquid DHMO "leads to corrosion and oxidation of many metals." "DHMO is often associated with killer cyclones in the U.S. Midwest and elsewhere."

DHMO, is used "as an industrial solvent and coolant, in nuclear power plans . . . by elite athletes to improve performance, in the production of styrofoam . . . in abortion clinics, in cult rituals . . . by many terrorist organizations, in animal research laboratories, and in pesticide production and distribution."

The Web site goes into great detail about DHMO's chemical properties, why it should be banned, and the fact that DHMO is found in almost every river, lake, ocean, and stream – worldwide.

Other Assessments

One of my old graduate school friends told his daughter's chemistry teacher about the DHMO Web site. She assigned the class a quick research project to learn about the dangers of DHMO, assess its

current impact on society, and write a short report/action plan. 100% of them came back with alarmist summaries and action plans that tended toward banning the substance altogether. I think this speaks worlds about the site and about how chemistry students view today's academic challenges.

A Challenge For You

Outrage is easy; action is more challenging. Please go to the Web site yourself and check it out. Get the facts you need. If you get out of it what I did, please seek out others to spread the word about its data – and about outrage. I fear that our society is way too manipulated by various sources of information and data. It's sites like this that expose our manipulation by the media and potentially enable us to get a better viewpoint on how to deal with information we are presented. We must conserve our outrage and direct it towards topics that are truly deserving!

;login:

EDITORIAL STAFF

EDITOR
Rob Kolstad *kolstad@usenix.org*

CONTRIBUTING EDITOR
Tina Darmohray *tmd@usenix.org*

MANAGING EDITOR
Alain Hénon *ah@usenix.org*

COPY EDITOR
Steve Gilmartin

PROOFREADER
jel *jel@usenix.org*

TYPESETTER
Festina Lente

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

certs for the masses

A Community-Oriented Certificate Authority

by Adam Butler

Adam Butler is a board member and marketing/PR director for CAcert, Inc. This past April, as part of his "PKI Roadshow," he traveled throughout the European Union to meet with CAcert users and other PKI enthusiasts. Look for Adam and the rest of the CAcert crew at this year's USENIX conference in Boston or email him direct at

adam@donkeyrequiem.com

"Secure authentication and encryption methodologies want to be free." Okay, I admit it. Compared with all the other OSS anthropomorphisms floating around, that one's a bit of a mouthful. Nevertheless, the need for strong and reliable data security is as old as data itself.

While the Internet community has championed the "information wants to be free" cause for as long as I can remember, this concept has always been tempered with a profound respect for personal privacy. Consistently, the heroes of the open source movement trumpet the emancipation of innumerable ones and zeroes across the globe while contemporaneously applauding the individual's right to keep his or her ones and zeroes private and secure.

Savvy computer users recognized this need from the very beginning, not because they had anything in particular to hide; rather, they merely realized that private data wasn't safe from prying eyes unless specific steps were taken to ensure that safety.

Long before buggy WEP-encrypted WLAN access points dotted the landscape – hell, even before the 1990s Internet retailing explosion – countless individuals sent countless petabytes of God-knows-what to God-knows-who without realizing that every bit of their communications could be (and often were) intercepted by others.

Over time, folks wised up. For the sysadmins among us, ask yourself: When was the last time you accessed one of your boxes in an open, untrusted environment, using Telnet rather than SSH?

And even Joe User caught on, eventually learning to check his browser for that nifty lock/key icon before submitting his online purchase. Sure, he probably still has little or no idea what is meant by terms like "Secure Sockets Layer" or "128-bit encryption," but at least he knows to check first before spiriting his credit card information off into the ether as cleartext.

I doubt anyone would seriously dismiss the role of PKI, SSL, et al. in strengthening consumer confidence in secure Web transactions and thereby laying the groundwork that allowed companies like Amazon and eBay to succeed, but the Public Key Infrastructure allows for so much more than mere virtual mercantilism.

For the most part, the Internet community exploits only a tiny fraction of what this valuable technology has to offer, and with gross privacy violations occurring at disturbingly increasing frequencies,[1] it would seem that now, more than ever, the importance of publicly available cryptography tools and techniques cannot be overstated.

It's time to take the next steps in securing our personal data and that of our users. For that, we're going to need a certificate authority.

Enter CAcert

Until recently, the thought of approaching a certificate authority (CA) for not one but numerous X.509 certificates might have tied your stomach in knots, caused you to break out in hives, and even prompted you to murder your entire family. Because unless Daddy's trust fund left you so much dough that you're routinely torching $100

1. "The Regulation of Investigational Powers Act (RIPA)," *http://www.homeoffice.gov.uk/crimpol /crimreduc/regulation/index.html* (July 28, 2000). See also "U.K. e-mail snooping bill passed," *http://www.cnn.com/2000/TECH/computing/07/28/uk.surveillance.idg/* (July 28, 2000).

bills just to light your Havanas, you're probably turned off a bit by the realization that the best price any CA offers is still going to require that you take out a second mortgage on the house.

Dylan quotes so often lend themselves to the OSS movement, and now is no exception: Times are indeed a-changin'.

Late last year, CAcert, a nonprofit, OSS-based certificate authority quietly stepped forward with a proposal that was as simple as it was groundbreaking: the Australian-born organization would offer signed, 128-bit X.509 certificates for personal and server-side use . . . for free.

Like so many open source mavericks before them, a small group of committed individuals simply took a long, hard look at a particular industry – in this case, the buying and selling of X.509 certificates – and realized they could do a better job.

In almost no time at all, CAcert was providing gratis what industry leaders Thawte and VeriSign were routinely hawking for hundreds or even thousands of dollars apiece.[2]

Today, CAcert offers signed, 128-bit X.509(v3) certificates for SSL, Wireless Auth, S/MIME, VPN, and other authentication/encryption schemes. And whether you're in the market for a personal or a server-side solution, you can leave your cache of Spanish doubloons at home – CAcert's expenses are still covered by donations and advertising, not exorbitant (and unnecessary) annual fees.

And that's not all. The venerable CA already offers a highly thought out "Web of Trust" assurance scheme,[3] gently lifted from the highly thought out WOT scheme offered by Thawte,[4] which was in turn borrowed from the highly thought out WOT scheme developed by Phil Zimmerman and the folks at PGP.[5] The WOT program allows CAcert's more than 5,000 members to notarize/sign/assure (depending on whose terminology you prefer) one another in pursuit of "Trust Points."

As a user increases his or her number of trust points with CAcert, advanced features are unlocked and become available for use. One such feature allows users to submit their PGP/GPG key to be signed by the CAcert master key, a novel integration of multiple PKI technologies.

Another feature, expected to be in place by the time you read this, will be the availability of so-called "code signing" certificates, similar in concept to those used in Microsoft's Authenticode initiative,[6] but minus the evil. CAcert sees this as a chance to give back to its fellow open source compatriots, empowering developers on various OSS projects with the means to digitally sign their work without having to rely on certs from expensive, corporate CAs who could not care less about the OSS community.

Supporting the OSS Infrastructure

Undoubtedly, the most important role of a community-oriented CA is to provide an affordable alternative to commercial certificate authorities. This enables thousands of smaller Web presences to abandon their current hackneyed PKI implementations and fall under the umbrella of a true CA, rather than relying on self-generated certificates in which users are (rightfully) leery of placing their trust.

As the situation currently stands, webmasters who wish to employ some type of Public Key Infrastructure – SSL, for example – usually feel that they must choose between (1) paying hundreds of dollars each year for a "trusted" certificate signed by some big

2. As of March 15, 2004, Thawte offered two 128-bit SSL server solutions, priced at $199 and $449 per year. On that same date, VeriSign offered a host of 128-bit SSL certificate packages ranging from $895 to $1495 per year. (all figures in US$ unless otherwise noted).

3. CAcert, "Assurance Programme," *http://www.cacert.org/index.php?id=8* (March 18, 2004).

4. Thawte, "Freemail Web of Trust System," *https://www.thawte.com/cgi/personal/wot/contents.exe* (April 15, 2004). See also Thawte, "Thawte: Web of Trust," *https://www.thawte.com/wot/index.html* (April 18, 2004).

5. William Stallings, "The PGP Web of Trust," *Byte* (February 1995).

6. Roger Grimes, "Authenticode," Microsoft TechNet, *http://www.microsoft.com/technet/security/topics/secapps/authcode.mspx* (March 18, 2004).

A CAcert solution requires less work on the part of the webmaster, and it's safer for the users.

name CA or (2) grabbing a current copy of the SSL libraries and generating their own self-signed, "untrusted" cert for $0. Unsurprisingly, many of these webmasters opt for the second choice, necessitating that each of their (apparently quite trusting) users download and install their sites' home-brewed root certificates, always assuming/trusting that webmaster X really is webmaster X, even if no one has ever confirmed this in any form or fashion.

With CAcert, a new option unfolds. Rather than fool around with generating a homebrew SSL cert, a webmaster unwilling to pony up for one of VeriSign's products can instead obtain one signed by CAcert. And unlike the self-signed certificate, CAcert "vouches for" its certificate and reveals to site visitors (via trust points) how well-known/trusted the webmaster is by the CA, giving visitors to the site straightforward, independent verification that Bob's Porn Palace is indeed operated by Bob.

Additionally, as more webmasters abandon self-signed certificates for flexible, widely available CAcert products, they free themselves from having to publish site-specific root certificates, revocation lists, and the like. Users simply install CAcert's root certificate – which isn't that much to ask considering that CAcert (as an independent CA) employs the same methods of member verification as its for-profit competitors – and, voilà, they'll be able to work with not just that one site, but all other sites that fall under CAcert's umbrella.

Thus a CAcert solution requires less work on the part of the webmaster, and it's safer for the users. The latter point has the added advantage of potentially driving more traffic to certain sites, as users who didn't trust the homebrew PKI solution might be more inclined to accept the CAcert trust model instead.

So CAcert is rocking and rolling along, expanding on traditional PKI and offering gobs of cool new options for encryption, authentication, digital signing, and the like, and all without robbing its users blind. What's the catch?

Well, there's no catch – just head over to *http://www.cacert.org* and check it out for yourself. But there are a few small flies in the ointment.

Fortunately, hackers are well known for jumping into the thick of things and coming to the aid of worthwhile projects . . . the perfect audience for a subtle call to action.

Root Cert Inclusion in Browsers

Obviously, a major goal for CAcert is to have its root certificate included with all of the popular Web browsers, so visitors to secure sites are neither required to download and install the cert themselves nor subjected to whatever awkward error messages their browser of choice decides to toss at them.

With something like 300 billion people using Windows in southern Georgia alone, it's no shock that Internet Explorer is by far the leader when it comes to browser market share. Anecdotal evidence (and common sense) seems to suggest that back during the Browser Wars, commercial certificate authorities probably greased the wheels with a healthy chunk of change to ensure that their root certificates would be included in both Navigator/Communicator and IE – ah, the hidden costs of "strategic partnerships"!

These days, the various browsers have dramatically different requirements in terms of root certificate inclusion.

In true Microsoft style, Redmond adopted a new metric for determining whether a CA's root certificate is to be included with its browser/OS/kitchen-sink product: In order for a CA's root certificate to be accepted – I swear I'm not making this up – Redmond said CA must pay a WebTrust-licensed member of the American Institute of Certified Public Accountants up to $250,000 for an initial evaluation/inspection, plus additional tens of thousands of dollars in fees on a periodic "follow-up" basis .[7]

The makers of the Opera Web browser did not respond to email queries regarding their inclusion policies/requirements; however, a Bermuda-based CA representative stated in the netscape.public.mozilla.crypto newsgroup that "as of [his] last contact in 2003, Opera wanted cash to add a CA [root certificate]. *They did not appear to have a standards policy*."[8] Nice to see somebody's got their priorities straight, eh?[9]

Rather than getting into all the other browsers under the sun, e.g., Safari, Konqueror, Lynx, and whatever crappy little program came with my Palm Pilot, let's jump ahead a bit and discuss open source's favorite son: Mozilla/Firefox.

Getting in Good with the Lizard

The open source advocates among us look forward to a time when software is finally wrenched free from the clutches of its faceless captors – massively proprietary organizations whose interests in innovation seldom reach beyond their own shortsighted marketing strategies, leaving less profitable technologies to stagnate.

And while collaborative software initiatives flourish across the globe, services designed to support and expand the underlying OSS infrastructure continue to face significant challenges. These barriers sometimes arise from corporations leveraging their de facto monopolies against newcomers, but often there's no evil empire to blame. Frequently, bumps in the road are merely the result of various open source advocates and developers disagreeing about one thing or another.

Earlier, I mentioned the Mozilla Foundation and its (apparently) nonexistent root certificate inclusion policy.

After Netscape disappeared, leaving no one to make "executive decisions" about boring stuff like root certificates and the like, the Mozilla Foundation apparently embraced a policy of maintaining the status quo, keeping all the old faves (like VeriSign and S-Trust) installed without really considering what would happen when/if any new CAs came knocking.[10]

This installed base remained the same even after VeriSign erroneously issued multiple Authenticode certificates labeled "Microsoft Corporation" to a couple of crafty social engineers,[11] arguably demonstrating once and for all that money can't buy you love *or* security.

Trying to go through all the proper channels, developers submitted a "feature enhancement" request to Bugzilla, asking that the CAcert root certificate be included with Mozilla.[12] (This inventive maneuver would later pop up again in Konqueror's feature request system.)[13]

Six months after the Bugzilla feature enhancement request was submitted, an announcement was made indicating that the CAcert root certificate would be included as part of Mozilla 1.6.[14]

And then the whole world started crying.

7. Microsoft TechNet, "Microsoft Root Certificate Program Requirements," *http://www.microsoft.com/technet/security/news/rootcert.mspx* (March 18, 2004). See also American Institute of Certified Public Accountants, "WebTrust Program for Certification Authorities: Version 1.0," *http://ftp.webtrust.org/webtrust_public/tpafile7-8-03fortheweb.doc* (August 25, 2000).

8. Emphasis added.

9. Name withheld, "RE: Proposed CA Certificate Metapolicy," *news://netscape.public.mozilla.crypto* (March 3, 2004). See also "Re: Why and How VeriSign, Thawte Became a Trusted CA?" *news://comp.security.misc* (March 15, 2004).

10. For a list of all the CA root certificates shipped with Mozilla browsers by default, open your copy of Mozilla or Firefox and select Edit ➔ Preferences ➔ Privacy & Security ➔ Certificates ➔ Manage Certificates ➔ Authorities.

11. Microsoft Knowledge Base, "How to Recognize Erroneously Issued VeriSign Code-Signing Certificates," *http://support.microsoft.com/default.aspx?scid=kb;enus;293817&sd=tech* (March 18, 2004). See also Microsoft Technet, "Erroneous VeriSign-Issued Digital Certificates Pose Spoofing Hazard," *http://www.microsoft.com/technet/security/bulletin/MS01-017.mspx* (March 18, 2004).

12. You too can vote for CAcert root certificate inclusion in the next version of Mozilla. The party's right here: *http://bugzilla.mozilla.org/show_bug.cgi?id=215243*.

13. Encourage the KDE Group to include CAcert's root certificate in the next version of Konqueror. Vote at: *http://bugs.kde.org/show_bug.cgi?id=74290*.

14. Frank Hecker, "Additional Comment #20," *http://bugzilla.mozilla.org/show_bug.cgi?id=215243* (Feb. 4, 2004).

All of a sudden, everyone and their brothers, best friends, and pets were posting messages to the site and arguing back and forth, debating the wisdom of what had just happened. The discussion eventually spilled out of Bugzilla and was shepherded over to the netscape.public.mozilla.crypto newsgroup.

Despite its nonprofit status, CAcert was criticized for its failure to retain the services of prohibitively expensive third-party auditing firms. As a volunteer-led community certificate authority providing free services to thousands of users, CAcert was in no position to start handing over big wads of cash to consulting firms.

CAcert is just another two-bit, fly-by-night operation, claimed some of its detractors.

There's no oversight, they charged.

The whole operation probably just consists of a cable modem, an old Packard Bell laptop, a pirated copy of PC-DOS 3.0, and four lines of Perl code. Their certificates are all encrypted with ROT13 and their private key is stored on a purple Hello Kitty diskette that sits atop Dad's Van de Graaff generator. They spend their free time issuing certificates to serial killers, zombies, and men who bite the heads off kittens.

That's right. Kittens.[15]

The original Bugzilla feature enhancement request was subsequently blocked/superseded by a directive that the Mozilla Foundation develop a formal certificate authority acceptance policy (presumably from scratch) before accepting any new root CAs.[16] Wildly disparate proposals for the new acceptance policy flew in from everywhere – people suggested everything from AICPA/WebTrust certification (insanely expensive) to an "open door policy" that would give everybody and anybody who applied access to the root store (insanely reckless) . . . and every imaginable gradient in between.

I have tremendous respect for all of the individuals who volunteer their time for the Mozilla Project, and I can completely understand the fears voiced by those who preferred the status quo. Furthermore, I am certain everyone who participated in all the various debates had nothing but the best intentions, even though the discussion seemed a bit more like a filibuster with each passing day.

In some arguments, it was as if two or three people were simply yelling "NO" at the top of their lungs, arguing against everything, often not even taking the time to explain the basis underlying their concerns; nevertheless, these passionate appeals were frustratingly successful in their ability to steer the debate off-course, even when the overwhelming majority seemed on the verge of reaching some kind of compromise.

Though I may disagree with their views on the issue, I certainly can't fault the individuals involved for trying. After all, the minority opinion must be loud, lest it not being heard. For whatever reason, certain people apparently felt that the Mozilla Project was in imminent danger, and so they defended it to the best of their abilities. I have little doubt that I would have done the same, had the roles been reversed.

Fortunately, there is a happy end to this story. After much debate and gnashing of teeth, the CAcert root certificate once again seems on-track for inclusion in the next Mozilla release (fingers crossed).

Looking Ahead

Though the development of a community-oriented certificate authority doesn't quite reach Kuhn's definition of a true "paradigm shift," it's a revolution nonetheless. Just as

15. Actually, CAcert is a fully recognized, legally incorporated nonprofit organization with a board of directors, an organizational charter, and a strict set of bylaws that explicitly forbids strategic alliances with zombies or other members of the undead. The CA servers are stored at a secure co-location facility, complete with biometric palm scanners and other cool stuff like that. And nothing is stored or signed in ROT13 format — CAcert has always relied on the far superior Triple-ROT26 algorithm for all cryptography.

16. "Mozilla.org Needs a Public Policy on Root CA Certs," *http://bugzilla.mozilla.org/show_bug.cgi?id=233453* (March 14, 2004).

when Network Solutions lost its monopoly on domain registration, things have changed significantly for the better. And there's no going back.

None of us today would consider paying $35 a year to register a top-level domain, and very soon VeriSign's $1200+ pricing for SSL certificates will strike us as equally ridiculous. Because as you read this article, even if its root certificate still somehow remains excluded from the basic Mozilla install, CAcert will still be growing and gathering momentum. At this point, there's no sense asking if the group will accomplish one thing or another – anything's possible, and it's all just a matter of time.

Says CAcert founder Duane Groth: "[T]he established players in the certificate industry lobby hard to exclude any further competition from entering the market, which means they are able to keep charging exorbitant rates for certificates. . . . This is all set to change."

"Currently there are hundreds of thousands of Web browsers out there with our root certificate installed; companies are deploying intranets with certificates issued from CAcert and installing the root certificate on each client machine on the network . . . . [M]omentum is building at a grass roots level."

Until CAcert's root certificate is preinstalled in your browser of choice, remember that you can always install it manually by visiting *http://www.cacert.org* and clicking the appropriate link. And if you're wondering what you can do to help with the effort, join the CAcert mailing list and make suggestions and donations – contribute how you can, if you can. And see the notes in this article for the URLs where you can vote for CAcert's inclusion in Mozilla and Konqueror.

But most importantly: Visit the site, sign up, grab a certificate or two, and start securing your data. Because regardless of what politics may be going on behind the scenes and what seemingly unattainable goals the organization may set for itself, whether you can spare some time to help with the project is beside the point. CAcert's mission remains the same: to provide you with alternatives to commercial CAs like VeriSign and Thawte, to help you secure your data, and to do the same for the rest of our Internet community.

And there's no time like the present. CAcert board members and developers were at CeBIT in Sydney earlier this year, it's only been a few weeks since the "PKI Roadshow" took your humble author halfway around Europe, and now (like many of you) we have our sights set on Boston in July. Along with several of our indefatigable developers, the entire CAcert Board of Directors will be at this year's USENIX conference. (Ironically, this will be the first time most of us will have ever set eyes upon one another!)

So, what does this mean for you? Well, if you hated this article, then just hold on to that anger – and soon you'll have a chance to actually smack me in person. For other, less violent attendees, we'll be available to answer questions about CAcert, demonstrate some of the less-appreciated uses for X.509, and help you sign up for your own free certificates.

Quick, get them before it's too late! I read that those things cost thousands of dollars apiece!

california requires disclosure of database security breaches

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

A new California law took effect on July 1, 2003 which requires businesses to disclose to California residents any breach in the security of their computerized data when that breach results in the acquisition of personal information about those California residents by unauthorized users. California Civil Code Section 1798.82 also requires businesses maintaining computerized data for others to notify the owners of that data should it be acquired by an unauthorized user.

Approved by former Governor Gray Davis in 2002, the law has sweeping implications for a wide range of businesses located both inside and outside of California. Experts estimate that nearly 100,000 security breaches occur every year.[1] Many of these breaches affect California residents. Companies that encrypt all personal data in their databases are exempt from the new law's disclosure requirements. Those that do not must fully comply with the new law, as the penalties for its violation include both monetary damages and injunctive relief.

The New Law

California Civil Code Section 1798.82 provides: "Any person or business that conducts business in California, and that owns or licenses computerized data that includes personal information, shall disclose any breach of the security of the system following discovery or notification of the breach in the security of the data to any resident of California whose unencrypted personal information was, or is reasonably believed to have been, acquired by an unauthorized person."

The law also provides: "Any person or business that maintains computerized data that includes personal information that the person or business does not own shall notify the owner or licensee of the information of any breach of the security of the data immediately following discovery, if the personal information was, or is reasonably believed to have been, acquired by an unauthorized person."

The notification requirements apply to any disclosure of "personal information." In order to be considered "personal information" under the new law, the information stored must include information from each of two categories. The first, or "name" category, is the California resident's first name or initial and last name. The second, or "information" portion, is either a social security number, a driver's license number, a California identification card number, or a credit or debit card account number plus any related information necessary to utilize the account.[2]

Businesses are required to notify California residents of any breach in "the most expedient time possible and without unreasonable delay." Businesses may meet this requirement with a written notice, or they may send an electronic notice,[3] provided that they receive an individual's valid consent to electronic notification.[4]

1. "California Sleeper" *Daily Deal*, April 7, 2003.

2. Information made public by local, state, or federal governments does not constitute personal information for purposes of the new law.

3. The new law provides that disclosure by electronic notice is permissible if it complies with the provisions regarding electronic records and signatures set forth in the federal law known as the Electronic Signatures in Global and National Commerce Act (15 USC § 7001 et seq.).

4. If a law enforcement agency believes that the notification would hinder an investigation, it can waive the notice requirement for a period of time.

A business whose notification is targeted at more than half a million people or would cost in excess of $250,000 is eligible to make a different type of notification. In that case, the law requires the use of email notification,[5] conspicuous posting on the company's Web site, and notification of the statewide media.

What Type of Breach Requires Notification

Requiring public notification of security breaches will be a sensitive matter for most companies. It is therefore important to understand the law, how it is implemented and enforced, and how to comply with it.

The legislature made clear that this is an act targeted primarily at reducing exposure to identity theft. According to its proponents, the notification required by the new law will provide the victims of identity theft with more time to mitigate the damages that can result from an unauthorized acquisition of their personal information.

However, the statute only vaguely defines what type of security breach triggers the notification requirement. The statute defines a "breach of the security of the system" as an "unauthorized acquisition of computerized data that compromises the security, confidentiality, or integrity of personal information maintained by the person or business." The business's duty to notify California residents is triggered upon the discovery of the breach.

Note that the statute requires notification based not only on the event of compromised "confidentiality," but also when "security" or "integrity" is compromised. Courts may give independent meaning to the terms "security" and "integrity," or they may view the whole phrase as a term of art.[6]

Importantly, the law does not require notification when either the name portion or the information portion of the personal information has been encrypted. But businesses seeking to take advantage of this may be surprised to find that the statute does not define what standard of encryption is sufficient to exempt them from the notification requirement.

Also, the statute does not require notification if the unauthorized person who acquires a California resident's personal information is an agent or employee of the information-owning business, the acquisition was in good faith, and the information is not further disclosed.

Extra-Territorial Application of Section 1798.82

The new law applies to a company if it conducts business in California. The law leaves to the courts the determination, on a case-by-case basis, of whether a given company located outside of California is conducting sufficient business in California that the notification and disclosure requirements will apply. The lack of guidance in the statute makes it impossible for a company to know in advance whether it must comply with the California law.

The jurisdiction of the California courts extends as far as allowed under the Due Process clause of the federal Constitution. It is clear that California courts have jurisdiction over all companies whose principal place of business or headquarters is located in California. Likewise, California courts have jurisdiction over companies located outside of California whose contacts within California are "systematic" and "continuous" enough that the defendant might anticipate litigating any claim in the state.

5. The new law intentionally uses the term "electronic notice" in one section and "e-mail notice" in another. To be effective and compliant, "electronic notices" must comply with the Electronic Signatures in Global and National Commerce Act, whereas "e-mail notices" (as part of the substitute notice provisions) apparently need not.

6. Furthermore, in its original form, Section 1798.82 stated that mere unauthorized access would constitute a breach that would trigger the notice and disclosure requirements. Amended before passage, the statute now provides that unauthorized acquisition, not access, triggers the requirements. It will be up to courts to decide whether there is a significant difference between these two terms.

The new law also applies to companies located elsewhere that engage in even minimum marketing or sales transactions with California residents.

However, jurisdiction generally does not apply to businesses that have no property in California, that have not sought to enter the California marketplace, and that have no telephone listings in California or any other contacts with California.

Companies headquartered and maintaining their principal places of business outside of California, but having business relations with California, may or may not be subject to California's jurisdiction for purposes of enforcing their compliance with the new law. The inquiry is a fact-intensive one. Courts look to whether the company "purposefully availed" itself by directing its actions at the state, so that it enjoys the benefits and protections of the state's laws. The claim must arise out of the company's actions that are directed at the state, and the jurisdiction must comport with the interests of "fair play and substantial justice."

Typically, the requirement that a company purposefully avail itself is met by demonstrating that it conducts continuing business relationships with citizens of the state. Even a single contact may be enough, depending on the nature and consequences of the contacts. Moreover, courts generally view a company's contacts as cumulative, so minimal contacts over a period of time may bring the company within the jurisdiction of California law and the California courts.

Internet Contacts

It is difficult to predict how a company's contacts will be viewed when those contacts with California are solely via the Internet. Internet Web pages are viewable anywhere, and while the Internet allows buyers to choose among more sellers, it is difficult for a seller to define where its customers come from. Courts tend to look at the nature of the contact and how the Internet Web page functions. The more interactive an Internet Web page, the more compelling the basis for asserting jurisdiction over those responsible for it. Other important factors include whether the initial contact is directed to the buyer (as in directed email) or is merely a passive advertisement.

Conclusion

It is likely that the new law will have a material impact on all companies that maintain data about California residents in their computerized databases. Companies that have offices, assets, or employees in California certainly have to comply with the new notification and disclosure requirements. But the new law also applies to companies located elsewhere that engage in even minimum marketing or sales transactions with California residents.

Despite the many uncertainties surrounding the new law, businesses should plan conservatively in order to comply with the fair meaning of the statute. This means that businesses should consider either immediately encrypting computerized personal information or develop strategies in order to meet their statutory notification requirements in the event of a security breach. Businesses may also want to take steps to decrease their potential costs of complying with the new law's notification requirements by adjusting their current intake forms to include a provision where the customer can consent to electronic notification in the event of a security breach.

talking to the walls

by Mark Burgess

Mark is an associate professor at Oslo College.

Mark.Burgess@iu.hio.no

A Meeting with Medusa

Throughout the passage of time we have talked to walls, in special rooms, and in private spaces, communing with deities and seeking guidance from spiritual powers. Today something else is happening: Our need for solace and comfort is more readily at hand in technological form. Our need for connection has become more rooted in the physical but has also expanded to become an addiction that veils a paradox. Are we all becoming possessed by distant voices – and thereby remote from our surroundings?

Imagine a chilly autumn day in downtown Oslo around 1999. The trams roll through the center of town; there are small flakes of snow in the air, and I am heading toward the Ibsen car park together with a visiting colleague at Oslo University College. (In Norway, we are careful to honor our important writers by naming parking lots after them.) The car park lies in the center of town, on the edge of the edge. It's a part of town called Grensen, literally "the edge," and it is populated by some of Oslo's more fantastic mythological beasts. In Oslo, as in most capital cities, a league of solvent abusers populates the city center, staggering around collecting coins and muttering to themselves in their own private reality. In this environment, it becomes natural to associate anyone muttering to themselves with some form of chemical escapism.

As we enter the high-tech lobby to the car park, I hear a voice talking frantically to someone, as if face to face. It is a figure in a black Armani, with expensive attaché case, standing next to the pay machines, stepping back and forth, staring into thin air, and facing the wall. A little wire hangs from his ear, but at the time I don't understand the significance of it. As he sees us, he seems shocked as though we have invaded his personal bubble. Right here in the most public place imaginable. I realize that there is something going on here that is of the greatest importance to society.

Today we don't think twice about hands-free mobile telephones. As we walk about, people are talking to themselves all the time, usually with a hand glued to their head. But all this has a deeper meaning – not just for us, but for system administration. Let's backtrack a little.

Getting Rid of the Keyboard . . .

Soon computers will be everywhere: in the walls, in our domestic appliances, and even in our clothing. Mark Weiser, former chief of technology at Xerox PARC, said, "The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it."

The keyboard is a potent symbol that this has not happened with computers yet. Computers are both conspicuous and unreliable, but there are several projects around the world to change this: the smart home of Hewlett Packard, Microsoft's tablet PCs, embedded Linux and Windows, etc. Still, given the advances in technology, it is reasonable to ask when this dream of disappearance might happen. The promise of a technological future has not yet caught up with science fiction. Technology for computation, multimedia, and communications has not yet disappeared from view: we cannot yet talk to our walls in a technological sense. A recent article in the IEEE computer magazine presented what it called the good news and the bad news about voice recognition. The bad news is that *Star Trek* has raised our expectations about voice

The location of individuals is unlikely to remain a real secret for much longer – the devices we carry will position us and cameras and sensors will recognize us.

recognition so high that it will be very hard to live up to them. The good news, on the other hand, is that we have until the 23rd century to sort it out.

We have had the promise of smart devices for many years, though they have been surprisingly slow in coming. The smart room that can detect your presence or switch on the lights and turn up the heating before you arrive by learning your patterns of behavior has not yet found widespread acceptance. The smart toilet that analyzes the colonies of bacteria that we donate to nature each day and finds out if we are sick or need dietary modifications has not yet materialized.

Kitchen computers were supposed to keep running inventories of supplies, be able to watch out for new recipes on the Net, order food when stocks got low, and so on. The kitchen cooker is supposed to be connected to your personal manager so that it starts warming up your dinner while you are on the way home (after all, everyone will be single in the future, so no one will have a partner to do this for them) – more on this later.

At the larger scale, smart cities will be able to route traffic automatically to avoid congestion and regulate resources such as lighting and heating. Buildings will control and reprocess their waste and be more resource efficient with regard to regulation of temperature and humidity. The location of individuals is unlikely to remain a real secret for much longer – the devices we carry will position us and cameras and sensors will recognize us. Cities will be able to share resources with other neighboring cities and organize common sharable resource pools – an automated city council, extra buses ordered when the demand increases. Local and global government will be replaced, slowly but surely, with automated cooperation and resource scheduling.

Embedded devices will eventually be found everywhere – and not just those left by the FBI! In restaurants we will have smart menus, with adaptive pricing and Amazon-style recommendations for your order based on what you ordered recently. Outside, attentive billboards will look back at you and gather information about sex, age, race, the clothes you wear, height, and weight. Walls will even monitor criminal activity for the police. Humans will be wearing the devices as they move around within this circus. The increase in surveillance devices has already been prolific in the last 10 years, especially in countries like the UK.

What Might It Mean?

What do these developments mean for those of us involved in the deployment and running of the technologies? We might expect to see tens of devices per room – a fairly complex network of devices linked probably by a Bluetooth type of wireless broadcast network.

There are management and security implications to living in such a density of information driven devices. The future of system management will not be a simple task like installing a package for Windows or GNU/Linux with some simple defaults, it will be a question of determining an increasingly complex policy that deals with how to exchange information with others, gives others access to our data, and protects ourselves from theirs. As we move from room to room in the house, the policy requirements will change. We will not want violent or explicit material transmitted to the children's den; we will not want telephone calls routed to the children after bedtime. Will we be able to cope with all of these constraints?

Today we use the term "trusted environment" quite often to describe a little island that we have made comfortable. But when computing becomes ubiquitous, the boundaries of our island have to break down, because we cannot sustain the illusion that we are all alone there. We cannot keep track of the pathways, the possibilities, or the interactions. There are people ballooning onto our island and digging tunnels to it. Others want to use it as a stepping-stone to get to somewhere else.

If computers are going to be running so-called intelligent software, then they cannot be isolated. How will they receive updates and instructions? We don't know exactly what operating systems embedded devices will use in the future, but they are bound to be complex adaptable operating systems (probably either Linux or Windows). If they are networked, it makes sense for them to receive updates and policy changes via the Net. But even after 10 years of developing management protocols for distributed devices, like SNMP, we are not much closer to finding a way to achieve this that is both efficient and non-intensive for humans.

Another problem is consistency and standardization. All of the pervasive devices listed above will eventually emerge, but not in any coordinated way. I am convinced that it is completely unrealistic to expect to be able to "manage" the resulting level of complexity using control protocols, as we shall see below.

The key to understanding pervasive computing lies very much in understanding people! We are the ones who will select or reject the technologies – by market forces. All we have to do today is to look around us. According to the dreams after the Second World War, everyone was going to be the proud owner of their own robot and personal spaceship by the year 2000. But, in reality, we were more interested in the immediate freedoms of cars and refrigerators.

Domestic Embedded Networks (DENs) will grow product by product, each with a different manufacturer using different standards. First it will be a Japanese or Korean microwave oven with an Internet connection. Then Microsoft will release the new X box that heats up a pizza while you're playing your favorite game so that you never have to remove the goggles and visit the real world. Then Sun will introduce a Java-enabled Open Sandwich toaster that produces more healthy food, and finally there will be a fight for standardization post-factum, and we will end up with the usual evolutionary gene pool of technologies that cannot be ignored. It won't be a neatly standardized set of controllable devices: After all, commerce is just warfare without politics.

If you are an evolutionist, then a broad technological gene pool is good for development. But if you are a system administrator control freak, or even the owner of one of these devices, then it is usually a nightmare. If technology is going to disappear, then it has to really disappear and not merely lurk in the shadows moaning for attention. All of this makes the problem of trust much harder – and therefore the problem of security a radically different one than before.

Eventually, simplicity tends to return as mass extinctions delete most of the competition and we learn to shift the boundaries of trust, and our little Cold War conspiracies dissolve toward more openness – if for no other reason than that it is really hard work distrusting people all the time. But before simplicity converges over this, we shall have to deal with the complexity of it. And here is a good reason why. Maybe smart rooms, smart walls, smart toilets are not what we want. What about smart people?

Maybe smart rooms, smart walls, smart toilets are not what we want. What about smart people?

Technology has never developed in the way we thought.

Mobility and Social Behavior

Steve Mann calls the smart room a "retrograde concept that empowers structure over the individual, imbuing our houses and public spaces with the right to constantly observe and monitor us." Mann wants us to be mobile devices – cyborgs. Others have argued that we already are! Take a look in the mirror.

The one aspect of ubiquitous computing that was never really envisioned (but which has flourished first) is mobile computing. Like the Internet, mobile services took off because they were at the root of a social phenomenon. In Japan, the under-25s call themselves the Thumb Generation, or the Thumb Tribe, because they live by their mobile phones, texting away with their thumbs – like touch typing.

Companies have tried several times to define Mobile Services for us, to sell us services that they dream up – like the 3G effort, with streaming video that would be used for business-like applications. But these have not taken off. Instead, cheap SMS messages have flourished and now camera still-pictures are taking off better than streaming video, because these are more "fun." They are not very useful for important communication, but they give pleasure to their users – perhaps because they retain a level of non-realism that still makes it seem like a game.

Technology has never developed in the way we thought. In the future visions that followed the Second World War (a time of aircraft and missiles), we imagined that every household would have its own spacecraft and that we would be traveling around the galaxy in a rich utopian marshaling of the galaxy. But when it came down to it, more domestic pursuits that empowered the individual over its civilization took precedence. The Italians bought motor scooters to be like the Americans and their cars, and the refrigerator allowed people to eat better. Society was formed from individual wishes, rather than having families fall into line with a greater vision.

Mobile technology is a freedom-giving device that has changed the way a society works where it has taken off. Particularly in Japan and here in Scandinavia, we see a generation of teenagers in constant contact with friends, no matter where they are. People no longer worry about being late for a meeting, because they can just send a text message to excuse themselves and reschedule. Time is now fluid; life is constantly being re-planned and rescheduled. With one foot in the future, people live by the moment and plans change in real time.

Social Changes

Social attitudes to one another have changed considerably. I was brought up to believe that a newspaper at the dinner table is the height of bad manners. Today, mobile phones are placed firmly between the starter and the fish knife, and conversations to the wall have equal if not higher priority than the face-to-face social graces. People will interrupt face-to-face contact for the immediately demanding mobile message.

This leads to cognitive confusion and social fragmentation. In Oslo, women get out their phones and talk loudly about nothing for the duration of their bus or tram journey – quite incapable of being "alone" in public. Perhaps they are so afraid of missing out on something in their remote social network that they have to exclude the possibility of enjoying their immediate environment. Humans are wired to relate in social ways, but if one loses respect for those in one's immediate environment, conflict rather than tolerance tends to arise.

Only a few years previously, the idea of revealing anything of oneself in public would have been a matter of considerable embarrassment in many countries. Today, people broadcast information and demand that others ignore it, as if emulating the very wireless protocols that are invading the electromagnetic airwaves with sound. Mobile users are constantly trading privacy for convenience – and struggling to renegotiate the bounds of privacy for increasingly selfish purposes. There are good users and bad users – those who respect each other's social spaces and those who do not. But they also use mobile communications as a shield to push others away.

Mobile users are constantly trading privacy for convenience.

Smoke Screen

Some would say that we are becoming more selfish, that our own microcosm is all that matters. It is our right to a kind of technological telepathy, or to spurn casual listeners for their impudence if we intrude into their space. In Scandinavia, the mobile phone has increasingly replaced the cigarette as a way of blowing smoke in faces at crowded places, or in an awkward situation like an elevator where normally one would be forced to communicate. Checking for messages is so much easier than making eye contact with someone. As soon as a situation becomes awkward (in an elevator, for instance), out with the phone. Many are literally dependent on their mobile phones now to run their lives and to keep others at arm's length. Clearly we shall all be single in the future.

Scandinavia has always had the stigma of having a difficult time with interpersonal relations. Now we have a way of avoiding them altogether. But this has various consequences. By placing virtual relationships above real ones, we distance ourselves even further from actual interaction. This affects our attitudes in social encounters (we are "cozy" on the phone, but hostile in public) and thus our formulations of acceptable policy in such cases. Whether we retreat or fight, adapt or conquer depends very much on our tolerance of others in society. Mobile, remote communication eliminates vulnerability and commitment. We risk nothing and gain little. Of course this is exactly the reason why we explore Mars with a remote probe – to avoid the possible risks associated with the reality of actual presence. With safe mobile communication, we never again have to reveal when we are having a bad hair day.

In Isaac Asimov's novel *The Naked Sun*, he describes a world called Solaria in which people never meet physically. They have retreated into a virtual world where they are safe from their neighbors and their attendant germs and smells. Today, we see people putting fences around their property, staking out their territory in terms of material wealth, and retreating from direct contact. It is perhaps no accident that these cultures are emerging most rapidly in Japan and Scandinavia, where – for opposite reasons – the population is insistent on distancing itself from its neighbors.

Why am I talking about sociology? I want to paint a picture of how humans behave, because it is humans who deploy technology and make the management decisions. Eventually, this will be a new battleground for conflict between opposing interests.

Modern Perseus?

Perseus was the warrior who slew the Gorgon Medusa, thanks mainly to some gadgets that he got from Hermes the Telecom provider and Athena his security advisor.

Modern society is increasingly based on toys for communication. By giving everyone these tools, our modern warrior is supposed to slay the ugly face of loneliness and rejection in society, bringing us all together. But how does it do it? By giving us so

much body armor that we are never comfortable without it again? By giving us the ability to avoid each other in reality, while clinging to one another's reflections?

Ad hoc encounters are what make life interesting, but how much do we want to reveal? History reveals an interesting dichotomy – we are getting less formal as time goes on (more ad hoc), but we are putting up more barriers in order to protect ourselves from risk. The barriers are getting closer to the core – personal firewalls, rather than building trust. There is an increasing spiral of distrust – which, for now, might excite the security industry, but which is not sustainable in the end.

Techno-Challenges of Pervasion

What does this have to do with us as system administrators? The answer is complicated, I believe, but it has to do with several things:

- Technology changes our behavior and our expectations. We torture-test it in ways that have more to do with sociology than technology.
- The boundaries of trust are the key to our deployment and expectations of technology. These boundaries are determined by human behavior.
- It is the interaction between humans and technology that is problematical for system administrators.
- The type of infrastructure that we will be expected to support in the future will be different and will be governed by personal freedom, selfish desire, habit, and pop culture rather than by the dictates of an IETF.

What are the main challenges of this pervasive computing for system administration, and how can we address them? First of all, we do not fully know the extent of the challenges yet – but, for the most part, I believe that they will not be radically different from what we see today, except that the arrival of smart devices will be nothing like what we imagine. However, the increased diversity will increase the magnitude of the problem and the rate at which the details of policy evolve.

- Diversity – we shall have an even more market-driven economy, fueled by whim rather than a desire for well-designed technology. This will lead to lots of conflicting coexisting technology. (This is normal and we have always experienced this in a smaller way.)
- We shall have to seek stability in the face of the much greater environmental noise from neighboring devices.
- Sociology of interaction will play a much greater role, because we cannot cordon off areas and isolate them any more. One organization flows into the next, and users roam around like cyber-tourists in foreign policy zones.

Most people want devices and technologies to be predictable. If they are not, then they cannot perform a useful function. In fact, since I have often spoken about the need to relax our strict ideas about frozen device configurations in order to allow some noise, I often hear from system administrators that they believe that every device has a correct configuration that should never change.

The kind of absolute stability that can be approached for immobile workstations is not really commensurate with the level of interaction that mobile or pervasive devices undergo. The idea of accepting any kind of uncertainty is more than many system administrators are willing to swallow. Yet this is precisely what we are going to have to accept if we employ increasing numbers of smart devices. The boundaries of trust will have to shift.

One area where things will change is in the level of exposure to environment. Environment means changing conditions and policy about right and wrong. Security consultants often posit that encryption is the solution to all security issues, but encryption is unlikely to help us here. The problem is not one of privacy, when individuals are being empowered with devices that allow them to expose themselves entirely and eagerly to a public audience.

Local regions are likely to demand their own rules, like micro-cultures. Both humans and devices will have to be aware of a much wider range of policies, rules, and standards of behavior that change as they move around. Some uniformity will no doubt emerge, but there will always be local features. Our ability to interact at a distance is leading us increasingly to draw boundaries around our property and shield our interests.

We might want to build our private island, but when we are in such a highly connected environment, the number of points of contact is too great to view isolation as a realistic possibility.

Where Lies the Authority?

In a world with fluid boundaries and increasing connectivity and blind trust in technology, we must work ever harder to define our own acceptable limits – our *policy*. To put it another way: If humans are constantly retreating from face-to-face confrontation with one another, then the rules of engagement must be ever clearer. In a human-computer collaboration, both humans and machine are supposed to obey policy. Who gets to decide on what policy says?

Smart devices are intrinsically bound to their environments. They must receive input and generate some output. If the exposure to environment increases, then a device will necessarily be more exposed to errors of configuration and random errors caused by misunderstandings and meddling.

I have claimed that we are becoming more mobile and connected, but also more suspicious of those who are not in our wired social networks. If we are roaming, do we have to adapt to the environment, or do we adapt the environment to us? Clearly the latter approach is a recipe for potential conflict. The likelihood for humans to cooperate is usually tied to the likelihood that they will see each other again. If we expect a long-term relationship in which reprisals for bad behavior are likely, then we are nice. All evidence shows that when humans believe that they will be long gone before anyone can catch them, they break rules and laws with alarming readiness.

Some imagine that mobile devices will always be rooted in a Virtual Private Network to home. How natural is it for a roaming device to maintain its ties to a home base? IPv6 allows and even encourages this, but I don't think that IETF has thought about an environment like Africa or Siberia, where connectivity will not be guaranteeable.

A more probable model will be for computing environments to supply cyber-tourists with services nearby. When the motor car was invented, it enabled freedom of movement because petrol/gas stations were available for refill wherever the individual decided to go. It was not necessary to stretch a cable from one's current location back to home base in order to fill up! This is why electric cars have had less success. Perhaps customers will be willing to pay the environment for a certain service (like a hotel) and guarantees on Quality of Environment will be the song of the day. Eventually, we will begin to accept local service provisions, because this is efficient. What this implies is

The likelihood for humans to cooperate is usually tied to the likelihood that they will see each other again.

that our environment is increasingly ad hoc. This has security as well as availability implications.

Trust in Clans and Societies

Who will make these decisions about what is acceptable? Will they occur top down or bottom up? By definition, administrators want to be on top, looking down. But down is not where users want to be. Clever users might resent this power structure and seek the freedom of their mobile phone or scooter to whisk them away from fascism.

We are increasingly empowering users toward autonomy. By giving them their own private communications bubble, we are also giving them the responsibility to find their own rules of engagement. Peer-to-peer networking shows this increasingly. It is an anti-authoritarian configuration. The only rule has to be mutual respect, or conflict. Human instincts will prevail here.

Perhaps a security policy based on mutual respect is more sustainable in the long term than one that is authoritarian. We shall have to discover the rules of society all over again. Mutual help and etiquette? Increased connectivity and mobility bring different cultures (social, racial, religious, or business), that is, different *policies*, closer together. Tolerance of others will be required.

In a ubiquitous computing environment everyone has roaming access to everything they need. That also means that the computers are exposed to a roaming environment — it usually works both ways: A greater contact area makes us more accessible and, therefore, more vulnerable. Even if we can apply access controls, there is a risk of configuration errors and possibly even the risk that persuasion might trick us into lowering defenses. Security does not depend only on technology.

The dynamics of cooperation and conflict are complex. Game theory is one way to analyze these issues. There are some basic results that characterize the interactions:

- The zero-sum game, where winner decides all.
- The prisoner's dilemma, bargaining for mutual gain, with tit-for-tat reprisals.
- Conditional consensus: I'll agree if everyone else agrees.

The results of games indicate that if we act in a purely selfish way, then a tit-for-tat strategy is best for protecting oneself from harm and for maximizing cooperation; that is, if one person is non-cooperative, non-cooperation is returned. If cooperation is offered, cooperation should be returned.

What about altruism and friendship (predictable agreement on policy)? What is it people get from investing in social relationships – even those with people they cannot see? We can call it social capital. Intimacy. A surrogate feeling of social acceptance that satisfies our genetic programming, like tofu for meat.

Game theory predicts that, if there is a reward from cooperation, then a reciprocal strategy is best. This forms a dynamical trust relationship, not merely a static one. Small groups are more likely to cooperate than large ones.

Cooperation takes us beyond zero-sum games, but when we have decided to cooperate conditionally, by voting, how do we arrive at consensus? In many cases, uncertainty leads to an overcautious strategy: we will vote if most other people do; we will flock with the others, if everyone is agreed. There is safety in numbers. These are the dynamics of consensus.

So, will there be discipline or anarchy in the world of pervasive computing? New alliances and allegiances are formed when roaming, but no stable consensus has to emerge. Humans make this even more difficult. In mathematics, if X=Y and X=Z, then Y=Z, but this is not true in human psychology. It is not impossible for X's policy to agree with Y's, but X and Y cannot agree, for other reasons. Humans are thus not easily predictable.

Swarm Intelligence: The Outcome of Weak Interaction

The new forms of pervasive computing and mobile communications lead to new social rules of engagement. If we do not understand those rules, some of us will disagree and the result will be conflict. Swarming or flocking is a way of capturing the equilibrium points of social conflicts and negotiations. On the one hand, isolationism creates little autonomous devices (insects), but mobile communications lead to involuntary clustering and flocking.

Swarms of insects and flocks of animals, for example, are assemblies of "devices" or "things" that communicate loosely but which spontaneously form quasi-stable structures that persist over long periods of time. Perhaps this is just what we are after for our devices (though perhaps not for our society).

The non-intelligent pieces have surprising properties when allowed to interact *weakly*. Do the pieces in a jigsaw puzzle know anything about the picture they form? Do any of the cells in our body have any idea about what they contribute to? These are emergent phenomena.

Swarm phenomena are already happening in humans as a result of mobile phone communication. Kids flock around like schools of fish with their mobile phones. They do not need to meet to be together, and final rendezvous can change even as they approach the moment! They are ad hoc social swarms – they change their behavior according to text messages and telephone conversations.

But can we harness swarming to secure a stable environment of pervasive devices? Conversely, once we release these devices, will be we able to prevent swarming phenomena from occurring? How do we guarantee that a swarm of ubiquitous computing devices will be a colony of helpful bacteria rather than a plague of harmful locusts?

One clue about the role of swarms is that socially developed swarms have many of the properties of social networks – quasi-hierarchies. Communication in swarms is by peer-to-peer transaction. This gives a robustness of form, combining trust in local neighbors with long-reaching connections ("strange connections") which occur in all social clusters. This is why no one on the planet is (on average) more than six degrees of separation from anyone else. Sometime, a central command might emerge spontaneously through centrality, but we should not be worried if it doesn't. Stability and security are not contingent on centralization or authoritarian control.

Conclusions

Do we want swarm behavior to emerge or not? In devices, in humans, or both? Can we stop it with judicious policies (i.e., "police" it away)? Sociology has a tendency to get its way; society has its own consciousness which usually trumps individuals. Does that mean that we are not safe? Security is about acceptable risk in relation to operating requirements. This should not be perceived as a problem, but we might need a change of philosophy in many system administrators.

Can we expect an ignorant tribe of technologically dependent, self-interested individuals to cooperate?

Society will not be threatened by its tendency to self-organize, but there are deeper ethical implications for society's use of technologies. We are constantly dumbing down human technology, taking responsibility away from the individual while simultaneously arming individuals with devices that allow them to be increasingly selfish. Soon we will have no burden of responsibility to learn about technology and we shall end up slaves to it – unable to understand, repair, or master it. As Arthur C. Clarke said, "Any sufficiently advanced technology is indistinguishable from magic." When it starts to seem like magic to us or when it truly disappears into the walls – out of sight and out of mind – we have a genuine cause for concern.

Can we expect an ignorant tribe of technologically dependent, self-interested individuals to cooperate? What kind of policy would they write? Is this a circuitous route back to nomadic anti-social behavior, in which individuals do battle rather than cooperate in meaningful society? Cold War isolationism is a slippery slope that leads to a downward spiral of trust.

To have our private boundaries penetrated is bad enough for our feeling of safety and well-being. To have our homes completely permeable to the outside world would be, for most of us, the ultimate breach of trust. Yet this is the potential vision we are concocting – will we fight it, or will we learn to embrace it? Not only in our homes, but in our clothes and in every aspect of our being. Pervasive computing is not only about making true cyborgs of us, but about weaving society together into a super swarm. How shall we behave then?

Society will always have a face that it cannot bear to look at. Our Medusa, the terrible face of loneliness, will probably always remain unbeheaded, but we must not be seduced into isolation-confrontation mode. Better to talk to smart neighbors than to end up talking only to our smart walls. Communication and cooperation are too complex to be entrusted to blunt electronic instruments. The way to solve our management and security problems is not by fueling an arms race, but by diplomatic conversation. That means that we must deploy technology along with education about the workings of both humans and machines, and we must preserve genuine close encounters between friends.

password protection for modern operating systems

The purpose of this paper is to help readers understand the security of the password encryption methods used in various operating systems and to establish some best practices for password management without requiring a background in cryptography. The article covers most pertinent background material in the first three sections.

by Steven Alexander

Steven is a programmer at Merced College. He programs for the Student System and manages the college's intrusion detection.

alexander.s@mccd.edu

Introduction

Early computer systems offered little in the way of password protection. The earliest designs stored a user's actual password along with his or her identifying information (username and/or user ID) in a central password file. Such schemes suffer from the obvious problem that any user who either legitimately has or surreptitiously gains access to the password file knows the password to every account on the system. Later designs avoided this problem by storing only an encrypted or hashed value in the password database.

Such systems are still vulnerable. Any attacker with access to a list of encrypted passwords can guess passwords by encrypting words from the dictionary or at random and comparing the encrypted results of his or her guesses against the encrypted passwords in the database. The designers of the UNIX operating system improved on this method by using a random value called a "salt." A salt value ensures that the same password will encrypt differently when used by different users. This method offers the advantage that an attacker must encrypt the same word multiple times (once for each salt or user) in order to mount a successful password-guessing attack.

Cryptography: The Basics

Encryption preserves the confidentiality of data. A user encrypts a message using a secret key so that others who do not know the secret key cannot recover the message. The message is called plaintext before encryption and ciphertext after encryption. Password protection schemes also use encryption, typically as a secure hash function.

Many modern encryption algorithms are iterated block ciphers that break data into blocks of a specified size and encrypt them by iterating an encryption function several times; each of these iterations is a round. One such cipher, Blowfish, is discussed later.

Rather than use the whole key to perform encryption in each round, most iterated block ciphers derive a subkey for each round using a "key schedule." A simple key schedule may select certain bits of the key for use in each round, whereas a more complicated key schedule, such as in RC5, may use mathematical means to generate the subkeys from the key.[1]

A hash function takes an input and applies a set of operations to reduce the input to a numeric value of a fixed size (usually 128, 192, or 256 bits). Outside of cryptography, programmers use non-secure hash functions to sort and search data. Secure hash functions need to have the special properties that it is computationally infeasible to determine a message from its hash value and that it is difficult to find another message with the same hash value.

1. Alfred Menezes et al., *Handbook of Applied Cryptography.* CRC Press, 1997.

Because it is difficult to find two messages with the same hash value, hash values are used to verify the integrity of messages. Users can independently compute the hash value of a message and compare it to a known source to verify that the message has not changed. Operating systems often apply a hash function to a password and store the encrypted result instead of the plaintext password. Because of the properties described, it is hard to determine the original password from the hash value and it is difficult to determine another password that has the same hash value (though one or more might exist). A user who needs to authenticate to the operating system tells the system his or her password, and the system hashes it and then compares the resulting value with the value stored in the password hash database.

Some operating systems use a dedicated hash function such as MD4 or MD5 to protect passwords; others use an encryption algorithm such as DES or Blowfish. When an encryption algorithm is used, the system uses each password as the secret key to encrypt a known message. In this case, the message is not secret, the systems designers only wish to prevent an attacker from guessing the key used to encrypt a message. The reader of this article should be aware that what is often referred to as password encryption is really password hashing. To add to the confusion, encryption algorithms are sometimes used as hash functions (albeit with some modification in use or design).

Early UNIX

The standard password hashing algorithm in UNIX, crypt, is a modification of the DES encryption algorithm;[2] it is often referred to by its man page entry, crypt(3). The system hashes each password after combining it with a 12-bit (4096 possible combinations) salt value. UNIX uses the salt value to modify one of the properties of the DES algorithm; this means that the same password will hash to a different result after merging each with a different salt value.

The salt value is not secret; it is stored with the hashed password. When a user presents a password to UNIX for authentication, the operating system looks up his or her salt value and hashed password. UNIX uses the user's stored salt value to modify the DES algorithm when it hashes the presented password; the system then compares the result with the value stored in the password database. Changing the DES algorithm with a salt value has the added benefit that an attacker cannot use off-the-shelf DES encryption hardware to brute force passwords.

Whenever a user chooses a new password, the system generates a new salt at random. Historically, UNIX generates the salt from the time of day or some other weak source of entropy (randomness). Because of this, it is common for two users at a site to have the same salt value. Still, the complexity of an offline password-cracking attack greatly increases because each word must be re-hashed for each salt value.

The designers of UNIX also introduced another important idea in the design of the UNIX crypt algorithm: They increased the time necessary to create or verify a password in order to further thwart offline password-cracking attacks while still keeping system response to an acceptable level. UNIX iterates the DES algorithm 25 times to create the crypt algorithm. Provos and Mazières estimated that a Digital Equipment Corporation VAX-11/780 contemporary to the design of crypt would be able to try only 3.6 possible passwords per second per salt.[3] If a system had 36 users, each with a different salt, it would take 10 seconds to try each possible password. Of course, modern equipment is able to fare much better; improved algorithms have replaced crypt in several modern UNIX variants.

2. Menezes et al., *Handbook*; Robert Morris and Ken Thompson, "Password Security: A Case History," *Communications of the ACM*, vol. 22, no. 11 (November 1979), pp. 594–97.

3. Niels Provos and David Mazières, "A Future-Adaptable Password Scheme," *Proceedings of the 1999 USENIX Annual Technical Conference* (June 1999), *http://www.usenix.org/events/usenix99/provos.html*.

Recently, researchers at the San Diego Supercomputer Center instituted a project to store the hashes of over 50 million common passwords computed once with each of the 4096 possible hashes used by the DES crypt mechanism.[4] Their results would be less worrisome if they had not shown that the requirements for such an attack are also within reach of a group of attackers using distributed storage.

Windows NT/2000/XP

The Windows NT line of Microsoft operating systems stores two password hashes: the LanMan hash and the NT hash.[5]

LanMan

The LanMan hash is used for backwards compatibility with Windows 95/98 and is the less secure of the two Windows hashes. Windows limits passwords to 14 characters for the LanMan hash. The system computes the LanMan hash by splitting the password into two seven-character halves and converting all characters to uppercase; if the password is less than 14 characters, the system pads it with null bytes. The system uses each half as a secret DES key to encrypt a fixed string of plaintext. The resulting hashes are concatenated.

Programs such as L0phtCrack have exploited the weaknesses in the LanMan hash. LanMan passwords are not case-sensitive. In addition, the system treats a 14-character password as two seven-character passwords. To illustrate the problem, let us consider the possible number of combinations for passwords of a given length and character set.

If x is the size of the character set used for a group of passwords (for instance, 26 if the passwords are alphabetic and not case-sensitive), then there are x^y possible passwords of length y with that character set. So a seven-character password that contains only numbers and uppercase letters has $36^7 = 78{,}364{,}164{,}096$ possible values.

An alphanumeric 14-character password should have $36^{14} = 78{,}364{,}164{,}096 \times 78{,}364{,}164{,}096$ possible values. Because Windows treats all passwords as two separate seven-character passwords for the purpose of computing the LanMan hash, an attacker only has to try $2 \times 36^7 = 2 \times 78{,}364{,}164{,}096$ possible values. If we assume case insensitivity, it is approximately 39 billion times easier to break two alphanumeric seven-character passwords than it is to break an alphanumeric 14-character password. If you had a computer that was capable of breaking any alphanumeric seven-character password in one second (quite a feat!), you would have to wait up to 1200 years to break an alphanumeric 14-character password.

Case sensitivity is a problem. A seven-character alphanumeric LanMan password has $36^7 = 78{,}364{,}164{,}096$ possible values, since Windows converts all letters to uppercase before it computes the hash. However, if the algorithm were case-sensitive, there would be 62^7 possible values, a total of 3,521,614,606,208. There are 45 times more seven-character case-sensitive alphanumeric passwords than seven-character case-insensitive alphanumeric passwords.

NT Hash

Windows computes NT hashes by applying the MD4 hash algorithm, invented by Ron Rivest, to the entire password; there is a 14-character maximum on Windows NT but not on Windows 2000 or XP. The NT hash does not suffer from the same problems as the LanMan hash. The NT dialect hash is case-sensitive and computed against the

4. Tom Perrine, "The End of crypt() Passwords Please?" *;login:*, vol. 28, no. 6 (December 2003), pp. 6-12.

5. Bruce Schneier and Mudge, "Cryptanalysis of Microsoft's Point-to-Point Tunneling Protocol (PPTP)," *Proceedings of the 5th ACM Conference on Communications and Computer Security* (November 1998), *http://www.schneier.com/paper-pptp.html*.

entire password. Administrators who do not need to support Windows 95, 98, or Millennium Edition users are encouraged to disable the storage of the weaker LanMan password.[6]

6. SANS, "SANS Top 20 Vulnerabilities," *http://www.sans.org/top20/*.

The MD4 algorithm consists of 48 steps which turn a 512-bit input into a 128-bit output. MD4 pads all inputs to a multiple of 512 bits, though it can iterate over several 512-bit blocks if necessary. When passwords are 13 characters long or less, the inputs to the last three steps are null. An attacker can use this information to reverse the last three steps of any password hash he or she is trying to crack. Subsequently, the attacker only needs to compute the first 45 steps for each password tried. This results in a speedup of about 6%.

Further improvements to this optimization are possible. The 128-bit output of MD4 is really four separate 32-bit values. Only one of these values changes in each step. The fourth of these 32-bit values changes last in steps 41 and 45. Since step 45 is reversed by the previous optimization, an attacker can compare the fourth part of the hash value after step 41 to the calculated value and stop computing if the values do not match. An attacker will only have to compute beyond step 41 once in every 4 billion tries. This improves the speedup by about 15%.

Common Problems

The Windows password schemes suffer from some common problems. First, there is no salt value applied to the passwords. Because of this, attackers can hash a large dictionary of possible passwords in advance to speed up their attacks. Using efficient sorting and searching methods, it is a trivial matter to determine whether a user's password hash corresponds to one of the hashes in the attacker's dictionary. An attacker who is actively guessing passwords against a large list of users can save an enormous amount of computation time by only having to encrypt each possible password once, unlike the case with UNIX passwords.

Sort-and-search algorithms make an attacker's job even easier. The naive method for finding a Windows password is to hash a possible password, then compare the result against the password hash of every user whose password you wish to recover. This is akin to reading every entry in the phone book until you find the person you're looking for. By sorting the password hashes, an attacker only needs to compare the hash of each possible password against a small number of password hashes.

UNIX crypt()	249467 hashes/second
BSDI DES (x725)	9618 hashes/second
FreeBSD MD5	4452 hashes/second
OpenBSD Blowfish	335 hashes/second
Kerberos AFS DES (short)	244907 hashes/second
Kerberos AFS DES (long)	435745 hashes/second
Windows NT LanMan DES	628234 hashes/second

Table 1

Another drawback to the password hashing schemes used in Windows is, unfortunately, efficiency. Faster password hashing algorithms allow an attacker to guess passwords more quickly. Table 1 offers a comparison of the speed of different password hashing algorithms as implemented in the popular password cracker "John the Ripper." The benchmarks in Table 1 are from the program's own benchmarking feature as it performed on a 2.4GHz Pentium 4 with 512MB RAM. I have noticed that John performs about 40–50% faster in practice (on my reference machine) than the benchmarks show; this is possibly a cache issue. L0phtcrack 4 performs significantly better than John and is able to compute about 5.3 million LanMan hashes per second; it computes about 126,000 NT hashes per second. The difference is very likely rooted in machine-specific optimizations, so your mileage may vary.

Obviously, it would do no good to slow down a password hashing routine to the point that it bottlenecks a system; however, other modern operating systems have slowed the process of password hashing in order to hinder would-be attackers. The key here is to balance security and efficiency.

New and Improved Problems

Recently, Philippe Oechslin, improving on a technique developed by Martin Hellman in 1980, developed a cryptanalytic attack that has consequences for Windows passwords.[7] Readers interested in the details of the attack should read Oechslin's paper. This article will only cover the attack in a basic Windows-specific manner.

The attack is a time-memory tradeoff that requires an attacker to store up to several gigabytes of data. If storage were of no consequence, an attacker could pre-compute the hashes of every imaginable password up to some reasonable length and store them along with the accompanying password. Attackers would then only need to look up a given hash in their database to discover the password. Oechslin's attack is able to achieve nearly the same convenience with thousands of times less storage.

The attack uses large tables of "rainbow chains." They are computed as follows:

1. A random word is generated and stored in the first column of the current row in the table.
2. The word is encrypted and the resulting ciphertext is "reduced" so that it represents another possible password (each character is converted into a printable ASCII character).
3. Step 2 is repeated many times over (usually a few thousand). The length of the chain dictates the number of times that step 2 is repeated. A part of Oechslin's improvement on Hellman's method is to slightly change how the reduction is computed after each repetition in the chain. The details and consequences require a paper of their own.[8]
4. The result of the final reduced value is stored in the second column of the current row.

Once an attacker generates sufficiently large tables, he or she will be able to recover a significant number of passwords in a small amount of time. An attacker first tries to determine if the password used to compute a given hash is the same as one used to compute one of the chains in his or her tables. This is determined by reducing and hashing the password hash in a manner similar to the method listed above for creating the tables. Each time the password hash is reduced, it is compared to the entries in column two of the table. If a match is found, the chain is recomputed (beginning with the value in column 1). The attacker will have to weed through some false-positives to find the target password. If one of the passwords hashed during the creation of the chain produces the password currently under attack, the attacker is successful. It takes several seconds to find a password using typical current workstations. The search takes longer (and fails) if the password is not a part of any of the stored chains.

This attack is statistical in nature. For alphanumeric LanMan passwords, a 2.8 gigabyte table will include about 99.9% of the possible passwords. An attacker actually has to perform about 10 times as much computation to generate such a table of rainbow chains as would be needed to compute and store every possible password and hash. This is not as bad for the attacker as it seems. Using a 2.4GHz Pentium IV with 512 megabytes of RAM, I was able to generate five rainbow tables, each with 35 million

7. Philippe Oechslin, "Making a Faster Cryptanalytic Time-Memory Trade-Off," *Crypto* 2003 (forthcoming). Douglas Stinson, *Cryptography: Theory and Practice.* CRC Press, 1995.

8. Oechslin, "Making a Faster Cryptanalytic."

chains of length 4666, in about 215 hours (nine days). These five tables are equivalent to the single 2.8 gigabyte table mentioned above.

Oechslin applied his technique to the Windows LanMan password hash; the attack is applicable to the Windows NT Hash with more effort. The attack is more difficult to use against the NT Hash because it is case-sensitive and the passwords are not limited to seven characters in length. Attackers could, of course, simplify their attacks by only considering passwords of seven characters or fewer. Still, an attacker would have to consider that there are a far greater number of possible case-sensitive passwords than case-insensitive ones. The attacker would probably consider a lesser set of passwords, such as those where all characters happen to be lowercase or where only the first letter is in uppercase.

FreeBSD

By default, the FreeBSD operating system uses a crypt mechanism based on the MD5 hash algorithm. MD5 is the successor to the MD4 algorithm used for password hashing in the Windows NT line of operating systems.

Poul-Henning Kamp developed the MD5 crypt routine based on Rivest's MD5 hash algorithm. MD5 crypt uses a salt of up to 48 bits and effectively has no limitation on password length. It is also far slower than either DES crypt or the Windows password hashing methods. To achieve this, MD5 crypt uses an inner loop with 1,000 iterations to continuously remix data into the hash calculation. FreeBSD also supports the traditional DES-based crypt and a Blowfish-based crypt mechanism. FreeBSD distinguishes MD5 and Blowfish hashes from DES crypt hashes by adding a prefix to the hash entries.

Provos and Mazières raised questions about the design of MD5 crypt;[9] however, the algorithm currently looks to be far more secure than the DES crypt mechanism or either of the Windows password hashing schemes. There is a better alternative to all of these.

OpenBSD

Niels Provos and David Mazières have designed a crypt mechanism based on the Blowfish encryption algorithm.[10] Blowfish is a block cipher encryption algorithm designed by Bruce Schneier.[11] Provos and Mazières actually designed two algorithms, eksblowfish and bcrypt. Eksblowfish is derived from Blowfish and has a purposefully slow key schedule. Bcrypt is a hash algorithm based on eksblowfish. Bcrypt is the most secure password hashing algorithm in common use at the time of this writing.

Bcrypt allows passwords to be up to 55 characters in length. Note that while MD5 allows longer passwords than bcrypt, this does not increase its security, because its 128-bit output is the limiting factor. It would be easier to find an alternate password with the same hash as a given password than to find a specific password in excess of 55 characters. A random password consisting of only printable ASCII characters only needs to be 20 characters long before a hash function output of 128 bits is the limiting factor. A hash function with a 192-bit output limits the security of passwords of 30 characters or more. Passwords that are not completely random would need to be longer to provide the same security; however, the limit of current supercomputing technology is close to 70 bits and will not approach 128 or 192 bits anytime soon.

Bcrypt requires a random salt value of 128 bits, which is large enough that no two accounts on the same system are ever likely to have the same salt. In fact, an attacker

9. Provos and Mazières, "A Future-Adaptable Password Scheme."

10. Provos and Mazières, "A Future-Adaptable Password Scheme."

11. Bruce Schneier, "Description of a New Variable Length Key, 64-Bit Block Cipher (Blowfish)," Fast Software Encryption, *Cambridge Security Workshop Proceedings.* Springer Verlag, December 1993, pp. 191–204.

would need to have the hashes of about 16 quadrillion users before it is more likely than not that two hashes are alike.

Bcrypt also uses a cost variable; an increase in the cost variable causes a likewise increase in the time required to perform a bcrypt hash. The cost assigned to new passwords is configurable using a systemwide configuration file. In OpenBSD, administrators can assign different cost values for normal users and the superuser.

Protecting Password Hashes

Storage

Password hashes need protection regardless of the security of the hashing mechanism. An attacker lacking the password hashes for a system cannot attempt any offline attacks.

Most UNIX systems offer password shadowing. When password shadowing is used, user information is stored in the /etc/passwd file but the password hashes are stored in another file, usually /etc/shadow or /etc/master.passwd. Many UNIX systems automatically use password shadowing; others (HP-UX for instance) require an administrator to configure password shadowing. All system administrators are encouraged to familiarize themselves with the pertinent areas of their system documentation.

When possible, UNIX administrators should use MD5 crypt or Blowfish instead of the traditional DES crypt. Both of these alternatives are available on Linux, Solaris, and the BSD systems. For information about Blowfish on Linux, please visit OpenWall (*http://www.openwall.com/crypt/*). Sun Microsystems recently introduced their own crypt mechanism based on MD5. The new mechanism is meant as a more secure replacement for the MD5 crypt mechanism introduced for FreeBSD. Sun's new algorithm uses a configurable number of iterations for its inner loop. The default value is currently 4096. I do not currently know if the inner loop is the same as FreeBSD's but, with the high number of iterations, it looks to be much slower (which is good!). To my knowledge, these alternative crypt routines are not currently available on AIX, IRIX, or HP-UX. Replacing DES crypt may break some upper-level applications, particularly those run from UNIX operating systems that do not support the new methods; consider yourself warned.

Microsoft introduced SysKey, with the release of Windows NT Service Pack 3, to encrypt password hashes stored in the Windows registry. Attackers can bypass SysKey protection using pwdump2. For more information on SysKey use, Windows administrators should refer to Microsoft's "knowledge base" articles.[12]

Transmission

Sending unencrypted passwords across a network is an activity best reserved for those who like to live dangerously. Many administrators erroneously think that switched networks will prevent an attacker from sniffing passwords as they travel across a network; this is not true.[13] Switches enter a "learning" mode after they start up. While in this learning mode, a switch will broadcast traffic in the same manner as a hub. The MAC tables on a switch can also be selectively poisoned, which allows traffic interception, as is done by programs like Dsniff and Ettercap, or they can be overloaded so that legitimate entries are flushed from the table, which will force the switch to broadcast traffic destined for those addresses.

12. Microsoft, "Windows NT System Key Permits Strong Encryption of the SAM," Microsoft Knowledge Base Article 143475; Microsoft, "How to Use the SysKey Utility to Secure the Windows 2000 Security Accounts Manager Database," Microsoft Knowledge Base Article 310105, *http://support.microsoft.com/*.

13. Abe Singer, "No Plaintext Passwords," *;login:*, vol. 26, no. 7 (November 2001), pp. 83–91.

The security personnel at the San Diego Supercomputer Center have eliminated the transmission of unencrypted passwords on their protected network using solutions such as SSH and SSL.[14] System administrators are encouraged to read Abe Singer's paper and consider what they can do for their own networks.

Some protocols that enable the elimination of plaintext password transmission have other drawbacks. For instance, the Windows NTLM protocols use a challenge response mechanism.[15] In the NTLM protocols, a server sends a random challenge to a client that has requested authentication. The client encrypts the challenge using a user's password hash and sends it back to the server. The server attempts to decrypt the client's response using the copy of the user's password hash stored on the server.

This scheme suffers from the obvious problem that a user only needs the password hash to authenticate; an attacker able to recover the hashes from the server can operate just as well as if he or she had the actual passwords. This version of the protocol has numerous other problems as well. Version 2 of the NTLM protocol has better security properties than its predecessor; administrators are encouraged to upgrade.[16] The Kerberos protocol also suffers from the problem that all of the information needed to authenticate is stored on the server.[17]

Password Policy

Password Handling

Often, the users of a system will bypass all of the carefully designed and maintained security mechanisms of that system to convenience themselves. It doesn't matter how securely passwords are chosen or stored in a system if users have those passwords stuck to their monitor or keyboard on a sticky note. There are two things to consider here: policy needs to strictly forbid such activity, and users need to have passwords that they can remember. Unfortunately, effectively enforcing policy requires the involvement of management. Getting all of the management in an organization to enforce rules against post-it notes may require divine intervention. However, other approaches may have better success.

One approach is for system policy to specifically authorize IT staff to confiscate post-it notes with password information and disable the related account. This should eliminate visible notes but probably won't prevent users from sticking notes under their keyboard or in the top drawer of their desk. User education can help increase adherence to policy.

Writing a password down is not always bad. It is quite reasonable to have system administrative passwords written down in a secure location. Of course, proper procedures must be maintained for handling the passwords. Sometimes, an administrator may need to keep a written copy of passwords temporarily. Consider, for instance, the case that a successful intrusion has happened and the administrator must change several different passwords at once. Some administrators solve this problem by choosing multiple passwords that are variations of each other. This method suffers from the problem that password cracking software might generate those same variations if one of the passwords is cracked successfully and added to an attacker's dictionary. If administrators (or users) must write down password(s), I recommend that they are stored in a semi-secure location, such as a locked cabinet or drawer (in a non-public area) or a wallet (if kept on the administrator's person). Change the passwords immediately if the written copy is lost.

14. Singer, "No Plaintext Passwords."

15. Hobbit, "CIFS: Common Insecurities Fail Scrutiny" (January 1997), *http://www.insecure.org/stf/cifs.txt*; Schneier and Mudge, "Cryptanalysis of Microsoft's PPTP"; Bruce Schneier et al., "Cryptanalysis of Microsoft's PPTP Authentication Extensions (MS-CHAPv2)," *CQRE '99* (Springer Verlag, 1999), pp. 192–203, *http://www.schneier.com/paper-pptpv2.html*.

16. Schneier et al., "Cryptanalysis of Microsoft's PPTP Authentication Extensions"; Microsoft, "How to Enable NTLM 2 Authentication," Microsoft Knowledge Base Article 239869 (2004), *http://support.microsoft.com/*.

17. Rik Farrow, "Network Defense: Kerberos for Net Authentication," *http://www.spirit.com/Network/net0902.html*; Simson Garfinkel and Gene Spafford, P*ractical UNIX and Internet Security*. O'Reilly, 1996.

Many organizations have periodic staff development activities; if management can be convinced to sponsor a workshop that includes basic security information and the reasoning behind policy, users may adhere to it more closely. It may also be effective for IT staff to periodically share information about attacks on the network. I am not suggesting that anybody should try to convince their system's users of an impending electronic doomsday, only that some awareness is helpful.

Another problem in most organizations is password sharing. Eliminating this problem involves both technical and policy measures. Restricting users to a single logon session can help to alleviate the problem. If possible, policy should authorize the IT department to disable accounts that it knows a user has shared. It is also important that it be as simple as possible for users to gain access to objects to which a user should legitimately have access. If it takes days for a user to get access to the resources that they need, they are much more likely to try to share another's account. In addition, in such an environment, users are more likely to be sympathetic to other users and share their password when asked.

PASSWORD SELECTION

The weakest passwords are short or are words, names, or derivations of words or names. Simply changing an "A" to a "4" or an "s" to a "$" does not significantly increase the security of a password. The password cracker John the Ripper has a customizable set of rules that it uses to try various permutations on supplied dictionary words and user information as possible passwords.

The three properties that define the security of a password are: length, character set, and randomization. Strength in one of these properties can make up for weaknesses in the others (to a point).

When passwords are not limited to seven or eight characters, as in Windows NT or some UNIX systems, length is the easiest way to increase the security of a password. It is easier to choose a very long password than a very random password. A 40-character password that uses only lowercase letters and spaces will be extremely difficult to break. Using current computing technology, such a password would be impossible to break without good language analysis or luck. Still, it is not wise to pick passwords from a popular text such as one of Shakespeare's plays or sonnets.

The characters used in a password have a major effect on the security of the password, especially when the length of a password is limited. An eight-character password with lowercase letters and punctuation is over 800 times harder to break than a password of the same length with just lowercase letters. When considering the characters used in a password it is useful to break the character set into groups: uppercase and lowercase letters each account for 26 characters, numbers account for only 10, and special characters account for 34. Strong passwords should have at least one character from each of three different groups.

Truly random passwords are hard to remember. It usually suffices to generate a password from a pattern that is meaningful only to you. "TfcIdwaT" is meaningful only if you know that it stands for "The first car I drove was a Toyota." Adding or prepending a number to this password would make it even better (if it wasn't in print).

On Microsoft Windows 2000 and later, there is an option in the Local Security Policy, accessed through Administrative Tools in the Control Panel, that "Password must meet complexity requirements," which can be enabled to force users to use strong pass-

The passwords used for Windows accounts should not match those used for UNIX accounts.

words. It requires that users not base passwords on their username (in whole or in part), that passwords be at least six characters long, and that the password contain characters from at least three of the four character groups mentioned above. I think that this forms the basis for a good password selection policy on any system. My only complaint is that the length requirement is too low. Thankfully, Windows also allows a minimum password length to be set; I recommend 14 characters if storage of the LanMan password has not been disabled; otherwise, depending on the security requirements of the system, the value can be set as low as 10.

Various methods exist in other operating systems for enforcing password policy. OpenWall has made a PAM module available that allows flexible policy configuration (*http://www.openwall.com/passwdqc/*). Among other things, the module can modify password length requirements based on the character groups present in a given password. Many systems allow an administrator to configure a minimum password length even if PAM is not available. Recall that only the first eight characters matter if the traditional DES crypt is used.

Password Aging

Password aging is configurable on most systems and is important for a good security policy. I don't recommend expiring passwords more often than every 30 days; users are more liable to forget their passwords, reuse passwords, or write them down. Passwords for accounts with administrative privileges should expire every 30 to 90 days, whereas it may be acceptable to force the expiration of user passwords as infrequently as every 120 to 180 days. If possible, use password history to prevent users from reusing old passwords. My recommendation, if the option is configurable, is to remember one to two years' worth of passwords.

Administrators must consider several factors when deciding how often to change passwords. If the security requirements of the network are high, passwords should expire more frequently. If the password hashing mechanism is weak, DES crypt, or a Windows method, the passwords should again expire more frequently. If an attacker gains administrative access, expire all passwords immediately (aside from other measures). If passwords are not transmitted across the network in the clear, passwords can be changed less frequently. Systems that force users to choose strong passwords of 10 characters or more, except systems using LanMan or DES crypt, can afford to allow passwords to expire a little less often.

It is important to remember that, regardless of their cryptographic strength, passwords can be captured by sniffing, keystroke logging, or other methods. A major point to consider is the willingness of your user population to comply with the requirements and not break security by other means (such as sticky notes).

Passwords on Multiple Systems

To whatever extent possible, administrators and users should use different passwords for different systems. The passwords used for personal accounts should never be the same as those used in the workplace. I would also recommend that passwords be different for different classes of systems in the workplace.

To clarify this statement the passwords used for Windows accounts should not match those used for UNIX accounts. One can consider network devices such as routers and switches a single class, but they should be separate from security-critical devices such as intrusion detection systems and firewalls.

The key is that there should be some separation. If passwords are the same or similar across multiple systems, an attacker is more likely to leverage access on one machine to gain access to the rest of the network. This is especially true of passwords used to access third-party server software; several years back I discovered that the password encryption used in a popular email server for Windows was just a simple substitution cipher. Six months later, a security company discovered that the company had replaced the password encryption algorithm with another equally simple cipher. Be careful and use your own judgment when reusing passwords.

Acknowledgments

I extend my thanks to Keith Simonsen for his suggestions and Casper Dik for answering my last-minute questions.

reflections on witty

by Nicholas Weaver

Nicholas Weaver is a postdoctoral researcher at the International Computer Science Institute in Berkeley. His research focuses on modeling existing and future Internet-scale attacks, and automatic defense systems.

nweaver@icsi.berkeley.edu

and Dan Ellis

Dan Ellis is a Ph.D. student at George Mason University and a researcher at MITRE. His interests are information security and intrusion detection, and malicious code in particular.

ellisd@mitre.org

Analyzing the Attacker

On March 20, 2004, an attacker released a single-packet UDP worm, Witty into the wild. Although only infecting roughly 12,000 machines, and less than 700 bytes long, this worm represents a dangerous trend in malicious code. The attack is well understood: There have been several analyses[1] of the worm itself, and an excellent analysis by Shannon and Moore on the network propagation,[2] including the presence of seeding or hitlisting (starting the worm on a group of systems to speed the initial propagation). But what can we learn about the attacker?

Examining the timeline of events, the worm itself, its malicious payload, and the skills required all point to an author who was motivated, sophisticated, skilled, and malicious. Although there have been previous well-engineered worms (notably the Morris worm and Nimda), Witty represents a dangerous new trend, combining both skill and malice.

It's actually unfortunate that Witty hasn't gotten the amount of attention lavished on previous worms, as it was a very significant attack. This worm contained a payload malicious to the host computer and was released with almost no time to patch systems. The worm contained no significant bugs and was written by a malicious author deeply familiar with the theoretical and practical state of the art in worm construction and computer security.

The Timeline and the Witty Worm

On March 8, eEye security discovered a stack overflow in the BlackICE/RealSecure products of ISS (Internet Security Systems). These end-host IDS products don't just restrict access based on port numbers and sender addresses like a conventional personal firewall, but include analyzers that evaluate traffic for particular applications, looking for anomalies or known vulnerabilities. One such module, the analyzer for ICQ instant messages, contained a series of stack overflows.

These vulnerabilities were relatively easy to exploit. The ICQ analyzer is triggered whenever the host computer receives a UDP packet whose source port is 4000, regardless of the destination port or the presence of a listener. Since there was no use of StackGuard,[3] /GS,[4] nonexecutable stacks, or other protections, a single-packet UDP-based overflow simply overwrites the return address, pointing to the injected code contained within the packet.

Stack overflows that can be exploited with a single UDP packet are among the most dangerous vulnerabilities, as they naturally support worms like Slammer.[5] Turning a normal single-packet stack overflow exploit into a Slammer-class worm requires a small amount of additional logic. Instead of shellcode, the attack initializes a random-number generator, gets a pointer to the start of the overflowed buffer, and goes into an infinite loop to send the packet to random addresses.

eEye quickly notified ISS, which released a patched version of the RealSecure and BlackICE products on March 9. eEye then published their advisory on March 18,[6] giving a high-level description of the vulnerability. On the evening of March 19, the Witty worm was released into the wild, less than 48 hours after eEye's public disclosure.

1. LURHQ Threat Intelligence Group, "Witty Worm Analysis," *http://www.lurhq.com/witty.html*; Kostya Kortchinsky, "Witty Worm Disassembly," *http://www.caida.org/analysis/security/witty/BlackIceWorm.html*.

2. Colleen Shannon and David Moore, "The Spread of the Witty Worm," *http://www.caida.org/analysis/security/witty/*.

3. Crispin Cowan et al., "StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks," *Proceedings of the 7th USENIX Security Conference*, *http://www.usenix.org/publications/library/proceedings/sec98/cowan.html*, January 1998, pp. 63–78.

4. Brandon Bray, "Compiler Security Checks in Depth," *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_vstechart/html/vctchcompilersecuritychecksindepth.asp*.

5. David Moore et al., "Inside the Slammer Worm," *IEEE Magazine of Security and Privacy*, July/August 2003, pp. 33–39.

6. eEye, "Internet Security Systems PAM ICQ Server Response Processing Vulnerability," *http://www.eeye.com/html/Research/Advisories/AD20040318.html*.

Witty (named for the portion in the stack overflow stating "Insert witty comment here") was an architecturally simple worm. It began by cleaning up from the buffer overflow, including obtaining API pointers from the overflowed DLL. Because of this reliance on magic numbers, Witty could only infect systems running version 3.6.16 of iss-pam1.dll, although it might crash older versions.

After cleaning up the stack overflow, the worm's code executed the main-body routines. The worm first seeded the pRNG (pseudo-random number generator) with gettickcount(), a recording of the number of milliseconds since the machine reset. It then sent out 20,000 copies of itself to random addresses with random destination ports. As an additional obfuscation, it also randomly padded the packet size. After completing this loop, it attempted to open a random physical disk and, if successful, to overwrite a random 64KB block of data. Finally, it jumped back to the seeding of the pRNG, repeating the process until the host machine finally crashed.

Although simple, the execution was superb. There were no significant bugs, especially in the pRNG (a common error in previous worms). Reseeding the pRNG each time through the main loop may have been accidental, but it was a useful flourish, papering over many possible flaws in the pRNG. The author also avoided common mistakes by using the system-provided pRNG instead of coding his own pseudo-random generator. The malicious payload, slowly corrupting the drive, causes immediate damage but does not significantly slow the worm's spread, while randomizing the destination port makes the worm more likely to penetrate firewalls.

Finally, the author seeded the worm. Rather than just starting at a single location, the worm started out on over 110 different victims.[7] Although previously a theoretical technique,[8] this represents the first occurrence of significant seeding in an autonomous worm. However, this was almost superfluous. Because Witty's single-packet nature is naturally fast, an unseeded Witty would have still spread worldwide in under two hours.

Analyzing the Attacker

Although it may not seem like much information, we can actually develop several insights into the worm author or authors. Not only was the author a skilled programmer, he (she, or they) was familiar with the lore, motivated, and malicious. He avoided the common mistakes and had access to a small, geographically distributed network of compromised systems.

The attack demonstrated considerable skill and knowledge. The attacker understood how to program in x86 assembly language and access Windows API functions. He was able to implement a stack-overflow attack. As importantly, the attacker understood worm-lore: Not only did the attacker seed the worm, he constructed a payload that was malicious to the host yet did not slow the worm's spread. Due to the short time frame, the attacker most likely knew all this information in advance rather than learning on the fly.

The attacker wrote compact code. Witty's body consists of just 177 x86 instructions in 474 bytes (the rest of the worm is the buffer overflow and padding).[9] In this small space, the author constructed routines to clean up from the overflow attack, seed the random-number generator, propagate the worm, and execute the malicious payload, demonstrating the attacker's skill at writing x86 assembly.

7. Shannon and Moore, "The Spread of the Witty Worm."

8. Stuart Staniford, Vern Paxson, and Nicholas Weaver, "How to 0wn the Internet in Your Spare Time," *Proceedings of the 11th USENIX Security Symposium*, USENIX, August 2002, *http://www.usenix.org/publications/library/proceedings/sec02/staniford.html*.

9. Kortchinsky, "Witty Worm Disassembly."

The lack of major bugs suggests that the attacker tested the worm before release. Although it would be possible for someone to write a worm without errors, it seems more likely that the worm was tested. The testing would only require a couple of systems, if the attacker monitored the network traffic on the test systems and knew the common errors made by worm authors.

But the most substantial implications arise from the short time between disclosure and the worm's release. Either the attacker discovered the vulnerability independently, reverse-engineered the patch, obtained an advance copy of the disclosure, developed the bulk of the worm in advance, or simply worked quickly.

The first option is that the attacker had developed his exploit independently of eEye's disclosure. In that case, why did he wait to release the worm? Why not release the worm after the disclosure, rather than just after a patch was released to block his exploit? Although this is still a possibility, the timing of the worm's release argues against the attacker independently discovering the ISS security flaw.

The second possibility, reverse engineering the patch before the vulnerability was disclosed, also seems unlikely. The version release notes for BlackICE (*http://blackice.iss.net/update_center/readme_pcp.txt, version 3.6.ccg*) did not mention a vulnerability fix among the changes. If the attacker was using or developed an automatic analysis tool, he might have noticed the changes and used them as a guide to creating an exploit, but he had no public indication of a vulnerability.

The third option has the attacker knowing the vulnerability before public disclosure, giving him more time to work. Thus, rather than 48 hours, the attacker would have had over a week to develop and test his code. In this case, the number of possible suspects is considerably smaller, as such information requires that the author be an insider or someone who has compromised ISS's or eEye's communication systems.

The attacker could have constructed the framework in advance, waiting for single-packet vulnerabilities to insert into his worm. If the attacker developed a generic framework, this implies that he was both malicious and premeditated, but wasn't targeting ISS or ISS users. But if the attacker was opportunistic, why use a payload that damages the host? Given the bulk of a worm, created in advance, there are many more attractive payloads (such as deploying a control network) which a generic attacker might employ. Such payloads would grant the attacker control of all the victims instead of simply corrupting them.

The final possibility is that the attacker simply worked fast. For an attacker with the required skills, it is definitely plausible that he constructed, tested, and released Witty in under two days, although the time window is relatively short.

The attacker apparently used a network of compromised machines obtained using a different mechanism, rather than a hit list of probable victims, to seed his worm. This vulnerability resists scanning (unless actually exploited, all systems react the same way), making the creation of a list of probable victims difficult. The very short timeline between disclosure and release, with no reported scanning before the worm's release, further suggests that a list of targets wasn't created in advance. Finally, the initial machines did not stop scanning in the same manner as later infections, suggesting that they were running a program to propagate the worm rather than the worm itself.

The use of previously compromised machines requires that the attacker either obtained access to 110 machines using a different tool, already had access to 110

machines, or took control of these machines from a third party. Thus Witty's author probably possessed some ties to the attacker underground, to gain access to these machines in the short time frame.

The use of a directly malicious payload also suggests the attacker had a motive. One possibility is that he was experimenting with malicious payloads. Yet since the attacker probably tested the worm, he could have verified that the payload wouldn't restrict propagation. Thus, although Witty's author could have been conducting an experiment, it seems unlikely.

Another possibility is that the attacker was targeting a particular ISS customer or group of customers without caring for collateral damage, or that the attack was an attempt to blind ISS sensors from a different, targeted attack.

The final possibility was that Witty was a direct attack against ISS by deliberately damaging ISS's customer base, or an opportunistic attack on ISS simply because ISS is one of many security companies. How many users will have second thoughts about purchasing ISS's systems? How many customers will change to a competitor? Especially when the vulnerability exploited was a stack overflow, the simplest and most easily prevented C programming error.

Conclusion

Witty represents a new generation of malcode: written by a motivated, skilled, and malicious individual. Witty's author is the first to combine both skill and substantial malice. Witty's author had some motive that led him (or her or them) to desire a destructive effect. And Witty was written by an expert who, unless caught, could do it again.

ACKNOWLEDGMENTS

This paper was a product of group effort. Thanks to Stuart Staniford, Vern Paxson, Colleen Shannon, David Moore, Stefan Savage, Scott Tenaglia, and Jack Aiken for fruitful and interesting discussions. Nicholas Weaver receives support from NSF/DHS grant NRT-0335290. Dan Ellis receives support from the Active Worm Detection & Response MITRE Sponsored Research Project.

Witty represents a new generation of malcode: written by a motivated, skilled, and malicious individual.

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

The more I learn about security, the more paranoid I get. This won't surprise too many of you, because you very likely feel the same. When you know that kernel-level rootkits can hide anything the attacker desires from you (short of rebooting your system from a CD and running a thorough check), then I believe you have reason to be paranoid.

And are attackers out to get you? Really, in most cases, it's nothing personal. You just happened to be running some vulnerable bit of code, and the latest automated attack rooted your system, installed itself, installed the rootkit, and modified your system so that the tools will restart after any reboots. In a way, you are lucky if your system does start misbehaving right away, scanning Class B-sized blocks of IP addresses, as such activity should make you notice that something bad has happened. And if you have configured your firewall to block unusual outgoing traffic, no one else will even notice before you have time to boot from a CD and see what has happened to your system.

It would be nice if things like this just didn't happen. But even if you secured your system, installed the latest releases or patch levels, your system may still be exploited. There are zero-day exploits, ones that have received no publicity and for which no patches yet exist. Software is complex, and complexity guarantees there will be mistakes in coding or in implementation. Even the famed Wietze Venema expects that his Postfix code has at least one bug per thousand lines of code. Thirty bugs in Postfix, and we can expect that as Postfix continues to improve, the odds of more bugs being discovered will increase.

Paranoia

I could try the minimalist approach. Run a pared-down BSD system, with no X windows, no servers of any kind. Get my email from some other server, use only ed to read that mail on a VT100 terminal. By reducing my attack surface to the bare minimum, I can considerably reduce the chance of a successful attack.

But suppose I want to browse the Internet? I could use Lynx, but how will I display the latest satellite weather photos? Before you know it, I have millions of lines of code running X Windows, and millions more to provide a Web browser. Even if I avoid Windows and its considerably more complex IE browser, I still have increased my attack surface considerably.

Sandboxing

What I want to do, but have not yet done, is to sandbox my browser. Actually, I want to sandbox my several servers as well, but the browser is certainly important. "Sandboxing" means to isolate one application from the rest of the system; there are several ways to go about doing that.

One way would be to run a virtual machine that contains the browser. User-mode Linux (UML) makes it possible to run a virtual machine that is distinct from the actual system. If the virtual machine gets owned, I only lose what was on that system, which is essentially a large file that contains enough of a Linux install to run a browser (or mail server). Careful firewalling would prevent the virtual machine from doing anything that the browser or mail server would not have been doing. You might immediately figure out that this is not a perfect solution: A mail server exploit could still infect the virtual machine, and turn it into an attack engine that probed the same port, 25/TCP, that the firewall allowed the mail server to visit.

BSD offers its jail approach, but that may fall prey to the same problem, that is, an exploit which installs an attack tool that could, in turn, begin attacking from a BSD jail. Chroot by itself provides part of what the BSD jail does: It limits a process to a subtree of the file system. Jail does more by controlling access to network system calls as well.

There has been some research into yet another approach, one that I believe will eventually come into widespread use. That approach involves using the operating system itself to sandbox applications. Modifications have been made to both the Linux and BSD kernels to make this possible.

The Linux kernel modifications came about because several organizations, including the NSA, wanted to be able to enforce Mandatory Access Controls (MAC). Mandatory means just that. Even if you are root, you can't override these controls. You can gain authorization to configure these controls, but, theoretically at least, no exploit should be able to change these controls. Rather than modify the kernel for a single approach, 2.6.0-test2 and later kernels include hooks that support different approaches to MAC, including using the NSA SELinux approach (*http://www.nsa.gov/selinux*). *Lsm.immunix.org* has kernel patches for adding these hooks into older (2.4.20) Linux kernels.

MAC has been around for a while, with systems using MAC, including UNIX systems, having been built all through the '80s. The big problem with any system that included MAC was that ease-of-use went flying out the window. You had pain-of-use instead. But this is where the notion of sandboxing comes in. Instead of chaining down the entire system, you focus on sandboxing particular applications. You sandbox your most dangerous apps, such as any network server, your Web browser, email client, and anything like chat or IM. You can also get rid of set-user-ID programs and use MAC to control access to privileged operations, such as access to a raw network socket or writing the password file.

And how do you configure your sandbox? Fortunately, there has been research into how to do that as well. One group that has worked on configuring sandboxing called it a computer immune system. You can read some of their papers at *http://www.cs.unm.edu/~immsec/begin.html*. The basic notion is simple enough. You run your application with your sandbox in learning mode. You will need to exercise the application so that all of the important execution paths have been taken in order to create an accurate profile of normal system-call activity. Systrace, Niels Provos's approach to this in OpenBSD (*http://niels.xtdnet.nl/systrace/*) has been ported to FreeBSD and Linux, although work on the Linux port has apparently ceased.

Systrace uses execution profiling to handle the configuration. And you can run a front end to systrace (with or without X) so that it will report exceptions, allowing you to update the profile interactively.

There are a couple of approaches to doing this in Linuxland. Immunix has been around for a while, having started with buffer overflow busting approaches (StackGuard), and now sells a Linux distribution with MAC features (*http://www.immunix.org*). Grsecurity resembles the better known systrace in some ways. It, too, has a learning mode, but the latest version comes with a default least-privilege configuration.

The University of New Mexico papers (referenced above at *www.cs.unm.edu*) do suggest one feature that both systrace and grsecurity appear to lack. Sanasecurity, the

The big problem with any system that included MAC was that ease-of-use went flying out the window.

company where one of the authors of the UNM papers now works, is building a product that takes a step beyond security policies that specify permitted system calls and arguments by adding in the notion of grouping system calls temporally. In other words, a policy that sandboxes an application should include not just system calls, but some context. The context chosen includes the order of recently used system calls. This appears to nail down the definition of policy a bit more than either systrace or grsecurity.

The only way that MAC versions of operating systems will succeed is if they can overcome the ease-of-use problems found in older MAC operating systems. AT&T MLS (Multi-Level Security) added over a hundred new command-line tools for routine administration back in the late '80s, which made it difficult to use. Well, impossible for most people would be a better description. While this might be great for security, it is not going to help encourage people, even paranoid ones like myself, to move to a MAC-based system.

Still, I am worried. I just might go there . . .

KNOPPIX

Earlier in this column, I mentioned using a CD to check out a system where a kernel rootkit might be installed. One approach is to use the install CD that is available for just about any version of UNIX (hard to imagine that such a CD wouldn't exist). You boot the install CD, then escape to a shell. Often, the menu system that comes with most install CDs will make this easy to do. Then you mount your hard drive partitions and start looking for evidence of a rootkit.

If you are working with Intel or PowerPC-based systems, there is something even better to work with. KNOPPIX (*knoppix.org*) is a single-boot CD that contains a huge collection of GNU software, running on top of a Linux kernel. You might be tempted to ignore KNOPPIX, because you already have bootable Linux install CDs. But what makes KNOPPIX special is the care taken by its author, Klaus Knopper, in writing setup scripts. I found that KNOPPIX worked better than many Linux distros (most!) at finding key devices, like your monitor and graphics card, and setting up a useful environment. I plan on using KNOPPIX in my Boston USENIX class.

KNOPPIX also probes hard drive partitions and sets up an `/etc/fstab` for read-only mounting of the partitions it finds. You can even examine Windows NTFS partitions. I had my only Win2K box bluescreen, all by itself, for no good reason – but with an unbacked-up copy of my 2003 tax data. The Win2K install disk couldn't mount this partition, but KNOPPIX could, allowing me to recover key files.

I highly recommend that you make certain you have bootable CDs handy for any Linux or UNIX system you manage. In times of crises, you don't want to waste time searching for one – you want to have what you need at hand, and know how to use it.

ISPadmin

by Robert Haskins

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

rhaskins@usenix.org

In this edition of ISPadmin, I look at how ISPs monitor the systems and networks that provide services to their customers. Let me start off by stating that I am employed by Renesys Corporation, which has a product described in this article.

Why Monitor?

There are many reasons for an ISP to monitor systems and networks:

- Troubleshooting: Monitoring systems is very helpful in troubleshooting problems quickly.
- Capacity planning: Historical graphs make capacity planning easier.
- Product differentiation: More information helps customers, making them more likely to use your services.
- Security: Monitoring helps spot potential security problems sooner rather than later.
- Documentation: In order to monitor, you must know what systems and networks are in place.

Background

Even for the smallest service providers, monitoring tends to be a highly customized activity. By that I mean each service provider's situation is unique, and each monitoring system must meet a diverse set of circumstances. Some of these parameters include:

- Number and type of services offered
- Number and type of customers
- Service level agreements in effect
- Complexity of back-end systems and networks
- Internal processes and procedures
- Reporting requirements

Each one of these parameters is covered in the following sections.

Services Offered

The more types of services that are offered, the more complex the monitoring system needs to be. Each service will require some level of monitoring, though the level can vary widely depending upon the complexity.

Customers

If a service provider sells mostly to commercial entities, it is possible their customers *might* directly monitor (a subset of) the provider's services. If the customer base is largely residential/individuals, then monitoring will be more focused on providing tools to an NOC or customer/technical support call center.

Service Level Agreements

If a service level agreement (SLA) is in effect, it might specify exactly how the provider is to monitor their network. Often SLAs specify what level of quality of service (QoS) a provider must achieve. The only way to measure the QoS is to have some sort of monitoring. SLAs also might stipulate that a customer may directly monitor a provider's services. Examples of such monitoring would be SNMP-based monitoring of equipment/systems.

Complexity

This is probably the biggest variable that dictates how a monitoring system is deployed. The more intricate the system/network is, the more complex the monitoring system will be.

Internal Processes

Internal processes will to some degree dictate the monitoring required. For example, if management wants to be made aware when a certain event or problem occurs, then monitoring systems must be in place to handle these occurrences.

Reporting Requirements

Reporting requirements will also necessitate that certain monitoring occurs. For example, if a dial-up provider wants to determine the need to add lines at each point of presence (POP), then the number of calls at each POP must be tracked over time. Similarly, the utilization of all high-speed lines on a provider's network should be tracked, so that additional capacity can be added when needed. In some cases, the information used for monitoring can also drive the service provider's billing systems. For example, Cisco's Netflow application built into their IOS device software can be used for both monitoring and billing.

Wes Cottrell of the Stanford Linear Accelerator Center (SLAC) has an outstanding page listing a wide array of network-monitoring tools. The URL is in the Resources section at the end of this article.

Protocols

With monitoring, the services being monitored must send their results across the network. There are a number of ways to accomplish this transfer. Some of the more common mechanisms/protocols used in monitoring are Simple Network Management Protocol (SNMP), finger, ICMP (ping and traceroute), Remote Monitoring (RMON, a subset of SNMP), and Cisco Netflow.

Of course, many tools have their own customary way of reporting results. For example, Nagios uses the Nagios Remote Plugin Executor to perform certain system checks and to allow administrators to write their own custom plugins.

Software

Both commercial software and good open source software are available for monitoring networks. For the purposes of this article, the available monitoring software is broken down into two categories: network-monitoring platforms and stand-alone products.

Network-monitoring platforms are frameworks that enable management of many different types of devices. Often, they are described as the "manager of managers." Perhaps the most well known of these types of applications is HP Openview. These systems, all by themselves, have little functionality. Applications like Openview are very useful in managing agents specifically designed to run with them or with agents based on open protocols such as SNMP.

Unless it is a service provider with very deep pockets, most will not be able to afford a commercial network-monitoring platform. These systems cost thousands to millions of dollars to acquire and deploy. If a provider does need functionality like Openview but doesn't have a ton of money, they might use the open source product OpenNMS,

which is a freely available network-monitoring platform. It is written in Java, and can be quite daunting to set up and run successfully.

Some of the better-known commercial products in the network-monitoring platform category include SNMPc by Castle Rock, IBM's Tivoli Netview, HP Openview, and Micromuse NetCool.

A common lower-end commercial tool used by ISPs is Ipswitch's WhatsUp Gold. This does an acceptable job of monitoring smaller- and medium-sized networks, up to about 1000 devices, though there is no set maximum. One downside is that WhatsUp Gold runs only under Microsoft Windows.

In the stand-alone area, many tools are available. These range from the likes of ICMP ping and traceroute, all the way up to products like Gradus from Renesys, which is useful in managing exterior border gateway protocol (BGP)-based networks.

Open Source Software

The vast majority of providers utilize free monitoring tools, which might include monitoring, graphing, and traffic-analysis packages.

Monitoring

Of the large number of open source monitoring packages, the most widely deployed is Nagios, but some other packages that could be used include SNIPS (formerly NOCOL), MON, Big Brother, and MIDAS NMS.

These systems all do basic ICMP ping monitoring and save some sort of history. Some also do graphing and SNMP monitoring, which might save an ISP from having to deploy a separate graphing system such as Cricket.

Nagios

Since Nagios is one of the most commonly used monitoring packages out there, it is useful to look at it in detail. I use Nagios 1.1 for the purposes of this article, but 1.2 is now available. Besides being Web-based, Nagios's features include:

- Ability to monitor both network attributes and system services
- User-definable system and service checks
- Escalation
- Acknowledgment of problems via Web interface
- Redundant configuration
- History of events
- Reporting

It would be useful to define the following terms in Nagios's nomenclature before covering the details of what Nagios can do.

- Hosts – any device Nagios monitors (router, switch, server, etc.)
- Hostgroups – collections of hosts that are related in some way
- Contacts – people who get notified when an event happens
- Contactgroups – sets of people who are collectively responsible for services
- Services – individual applications monitored on hosts (Web, DNS, FTP, etc.)

Nagios will monitor the services you define, and keep a history to enable reporting service level agreements and other benchmarking. It has the ability to call programs on remote machines via the Nagios Remote Plugin Executor. This functionality is very

RESOURCES

SLAC's Network Monitoring Tools by Les Cottrell
http://www.slac.stanford.edu/xorg/nmtf/nmtf-tools.html

SNMP starting point
http://www.snmplink.org/

Another good SNMP reference
http://www.simpleweb.org/

Cisco Netflow
http://www.cisco.com/warp/public/732/Tech/nmp/netflow/netflow_nms_apps_part.shtml

HP Openview
http://www.openview.hp.com/

Sun Solstice Domain Manager (formerly Sun Net Manager)
http://wwws.sun.com/software/solstice/dm/

IBM Tivoli Netview
http://www-306.ibm.com/software/tivoli/products/netview/

WhatsUp Gold
http://www.ipswitch.com/products/whatsup/index.html

SNMPc from Castle Rock
http://www.castlerock.com/

NetCool from Micromuse
http://www.micromuse.com/products_sols/index.html

OpenNMS
http://www.opennms.org/

Nagios
http://www.nagios.org/

CAIDA's cflowd
http://www.caida.org/tools/measurement/cflowd/

SNIPS
http://www.navya.com/snips/

MON
http://www.kernel.org/software/mon/

MIDAS NMS
http://midas-nms.sourceforge.net/

useful, as Nagios can easily be customized to monitor things that are not built into it by default.

Nagios Options

The left-hand side of all Nagios screens always displays the options available. This section goes over some of the more useful Nagios options available.

The "Tactical Overview" screen shows the overall status of Nagios itself, as well as the hosts and services it is monitoring. It is a nice summary of everything within Nagios. I personally find myself viewing the "Service Detail" screen the most. This screen is one big table with the hosts in alphabetic order, with each host listing services, one per line. If you want to see the status of every service you are monitoring, use this.

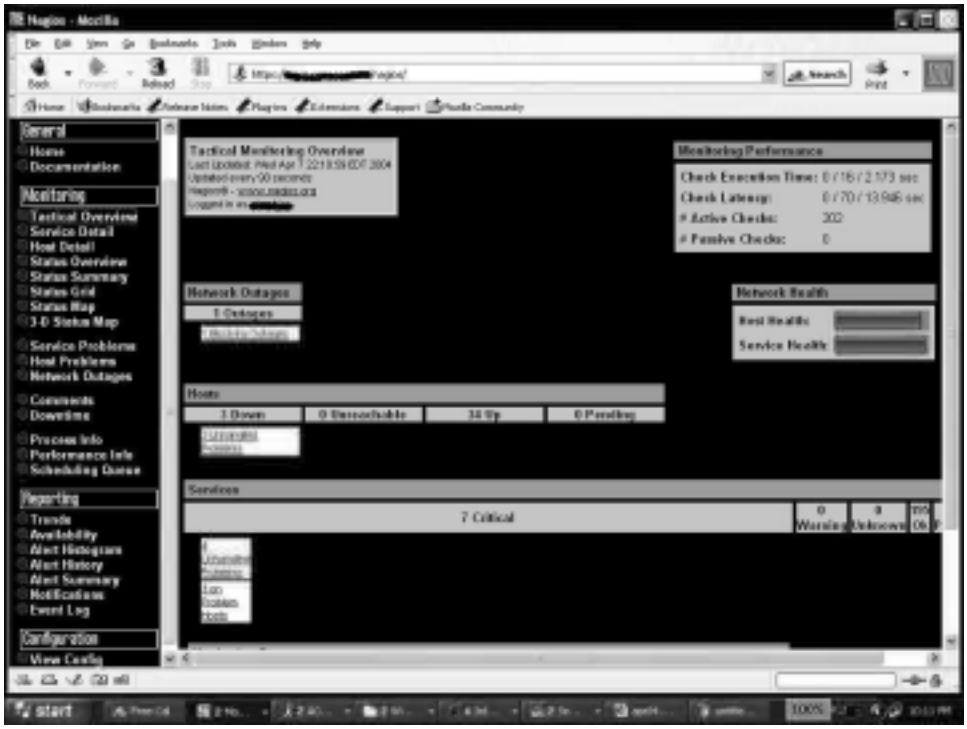
"Host Detail" lists every host's ping status. The "Status Overview" shows the various host groups defined. Each host's set of service-status details can be displayed by clicking on the individual hostname. "Status Map" can be used to graphically show the dependencies between various hosts and is probably most useful for displaying large networks of routers and similar devices. "Service Problems," "Host Problems," and "Network Outages" show the various classifications of items that are currently in alarm.

Nagios allows users to enter comments regarding services. These are used to communicate information about services. "Comments" displays the comments that have been entered for a given service. Nagios also allows users to put services into downtime. "Downtime" lists the services that have been put into downtime and which are not being currently monitored.

Reports

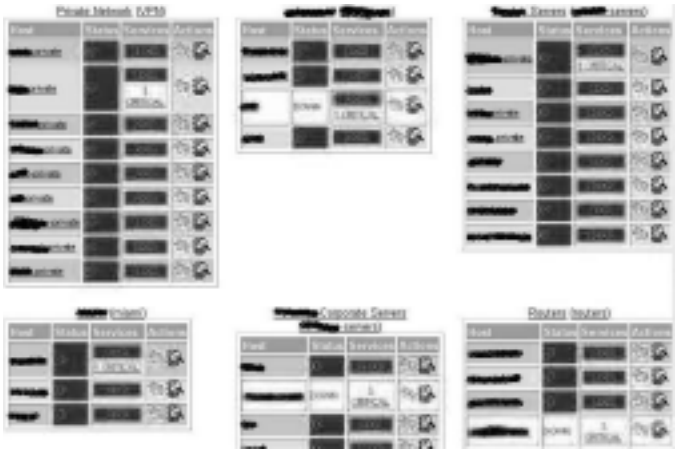
Nagios keeps a history of all services it has monitored. This history can be used to generate reports and statistics for things like mean time between failures and service level agreements. Nagios can graph data over time as well: for example, ping times and alerts via the "Trends," "Availability," and "Alert Histogram" functions. The configuration can also be viewed by clicking on the "View Config" option. Note that the configuration options must be adjusted by editing files (located by default in /etc/nagios) and cannot be managed through the Web GUI.

Next time, I will look at an rrdtool-based graphing package, as well as a couple of network-specific monitoring tools.



Tactical Overview screen

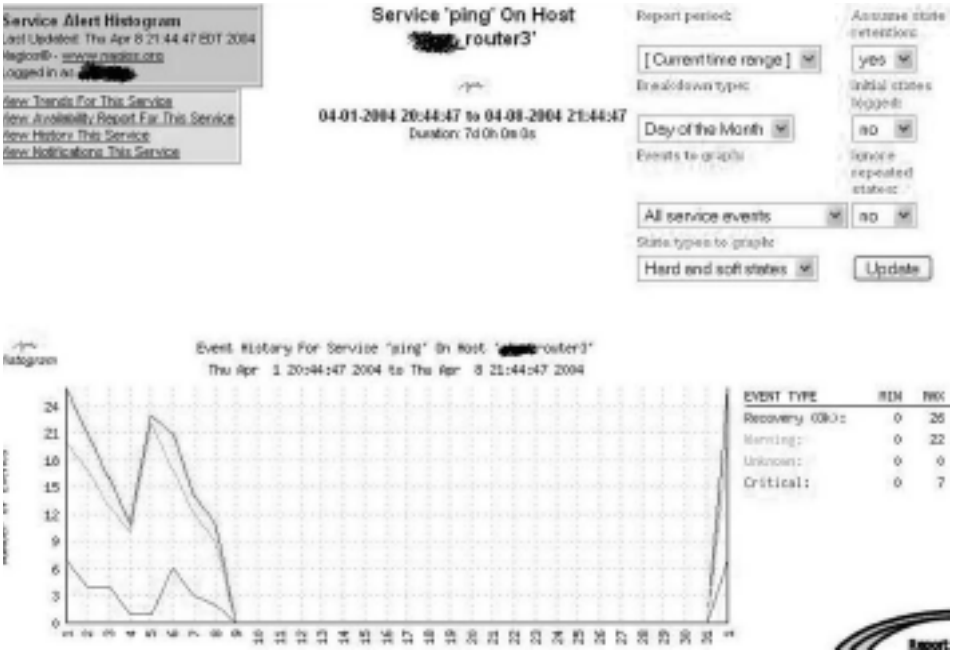
Service Details screen (partial)



Host Details screen (partial)

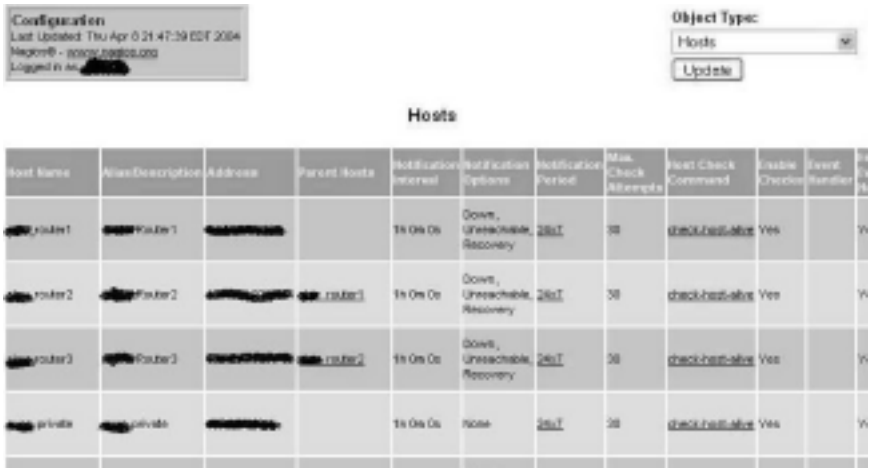
Status Map screen (partial)

Service Overview screen (partial)



Service Problems screen (partial)

Hosts Problems screen (partial)



Network Outages screen (partial)

C# overloaded operators

by Glen McCluskey

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

glenm@glenmccl.com

In our examination of the C# programming language thus far, we've seen that classes are a basic design and structuring tool. For example, you might have an application that uses a lot of X,Y points, and you could implement a Point class using C# language features. Instances (objects) of this class would represent specific points like 123,456.

Classes bring together both data (such as a pair of integers to represent points) and operations on that data (e.g., comparing one point to another). The operations are called methods, and in this column we'll look at how methods can be specified using operator names.

The idea is that a method's name can be something like == instead of Equals, or + instead of add, and using such names leads to a natural way of expressing operations on objects.

An Example

Let's look at an example, using a Point class to illustrate the idea of overloading:

```
using System;

public class Point {
    public readonly int x;
    public readonly int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public static bool operator==(Point p1, Point p2) {
        return p1.x == p2.x && p1.y == p2.y;
    }

    public static bool operator!=(Point p1, Point p2) {
        return !(p1 == p2);
    }
}
```

```
public class Driver {
    public static void Main(string[] args) {
        Point point1 = new Point(10, 20);
        Point point2 = new Point(20, 30);
        Point point3 = new Point(10, 20);

        if (point1 == point2)
            Console.WriteLine("point1 == point2");

        if (point1 == point3)
            Console.WriteLine("point1 == point3");

        if (point1 != point2)
            Console.WriteLine("point1 != point2");
    }
}
```

This class defines a couple of public readonly fields, used to represent particular X,Y values. Readonly means that the fields are initialized in the constructor, and can then be read but not written by users of the class's objects. C# properties can also be used to achieve a similar end; they are a hybrid of data fields and methods.

The class also defines two operator methods, operator== and operator!=. These methods take two objects of Point type, and thus it is possible to say things like:

```
if (point1 == point2)
    ...

if (point3 != point4)
    ...
```

and provide a customized definition of what == and != mean for a given class. In the Point example, two points are equal if their X,Y values are the same, and inequality is defined simply as not being equal.

Defining == in this way might seem pretty obvious. But in real-world situations, equality can be defined in many ways. For example, if the X,Y points are represented as double values instead of integers, it might make sense to consider two points equal if the values are close to each other, say within 0.001%, rather than exactly the same.

We can define operator== with any semantics we like. For example, we can swap the bodies of the operator== and operator!= methods, and thereby invert the semantics. But doing so violates a fundamental rule of operator overloading – using such operators in a confusing way can make programs impossible to read and comprehend. It's possible to create your own reality using overloaded operators, a reality incomprehensible to others.

C# requires that the == and != operators be overloaded as a unit. If one is overloaded, the other must be as well.

A C# compiler may give a warning for the code above, saying that operator== is overloaded, but there is no Equals method specified. What does this mean? Equals is a method in the root class (System.Object), and typically you want to override it to provide class-specific behavior. Since C# is designed to interoperate with other languages, and those languages may not have operator overloading but may wish to call a C# Equals method, it's a good idea also to define Equals if operator== is defined.

This can be done by adding some additional lines of code:

```
public override bool Equals(object obj) {
    if (!(obj is Point))
        return false;
    return this == (Point)obj;
}

public override int GetHashCode() {
    return x ^ y;
}

public override string ToString() {
    return String.Format("({0},{1})", x, y);
}
```

We have defined Equals in terms of the == operator already specified above. GetHashCode and ToString are two other System.Object methods that are typically overridden as well, and we have also provided implementations of them.

Conversion Operators

You can also specify conversion operators in the C# classes you design. Such operators are used when converting from one data type to another.

For example, suppose that we define another Point class, one that represents X,Y values using unsigned 32-bit integers. An alternate representation of such points might be a single unsigned 64-bit integer, with the two 32-bit values stored in the two halves of the larger integer. Here's some code that shows how this idea can be implemented:

```
using System;

public class Point {
    public readonly uint x;
    public readonly uint y;

    public Point(uint x, uint y) {
        this.x = x;
        this.y = y;
    }

    public Point(ulong val) {
        x = (uint)(val >> 32);
        y = (uint)(val & 0xffffffffUL);
    }

    public static implicit operator ulong(Point p) {
        return ((ulong)p.x << 32) | p.y;
    }
}

public class Driver {
    public static void Main(string[] args) {
        Point p1 = new Point(123456, 234567);

        ulong pt = p1;

        Point p2 = new Point(pt);

        Console.WriteLine(p2.x + " " + p2.y);
    }
}
```

This class defines the usual constructor that takes an X,Y pair of values, along with a constructor that takes a single 64-bit value. The class also defines a method:

```
implicit operator ulong(Point p)
```

Such a method supports operations such as:

```
Point point1 = new Point(123, 456);

ulong p = point1;
```

that is, automatic conversion from a Point object to an unsigned long value.

The implicit specifier is used in declaring the method. If we'd used the explicit specifier instead, we would then need to say:

```
ulong p = (ulong)point1;
```

This situation is analogous to converting a long primitive value to a short value; some languages require an explicit cast, because the conversion may not be possible without data loss.

Indexers

A final example of C# operator overloading illustrates what is called an indexer. The idea is that you might have a class whose objects represent databases or tables or something, and it would be natural to overload the [] operator to represent lookup in the database or table.

Here's an example of how indexers are used:

```
using System;
using System.Collections;

public class Index {
    private string[,] list = new string[,] {
        {"jane jones",  "123-4567"},
        {"tom garcia",  "234-5678"},
        {"mildred smither", "345-6789"}
    };
```

```
    public string this[string index] {
        get {
            int listlen = list.GetUpperBound(0);

            for (int i = 0; i <= listlen; i++) {
                if (list[i,0] == index)
                    return list[i,1];
            }

            return null;
        }
    }
}

public class Driver {
    public static void Main(string[] args) {
        Index phonelist = new Index();

        string num = phonelist["tom garcia"];

        Console.WriteLine(num);
    }
}
```

This demo implements a simple phone list lookup scheme.

The key line of code in this example is:

```
public string this[string index] { ... }
```

This says that when [] is applied to objects of the Index class, the get and set code should be executed to actually do the indexing. In this example, we specify a string argument to [] that is used as the key for lookup, but any type of argument is allowed, and you can even use multiple arguments – for example, to implement virtual two-dimensional arrays.

The syntax is very similar to what is used for C# properties. The get code is invoked when obj[index] is used in an rvalue context, and the set logic (which we do not define) is executed when obj[index] is used as an lvalue.

Operator overloading is a powerful technique that you might want to use in your C# programs.

practical perl

Web Automation

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

Introduction

Web service protocols like XML-RPC and SOAP are great for automating common tasks on the Web. But these protocols aren't always available. Sometimes interacting with HTML-based interfaces is still necessary. Thankfully, Perl has the tools to help you get your job done.

In my last column, I introduced Web services using XML-RPC. Web services are commonly used as a high-level RPC (remote procedure call) mechanism to allow two programs to share data. They enable programs to exchange information with each other by sending XML documents over HTTP.

There are many advantages to using Web service tools like XML-RPC and its cousin, SOAP. First, all of the low-level details of writing a client and server programs are handled by reusable libraries. No longer is it necessary to master the arcana of socket programming and protocol design to implement or use a new service or daemon. Because information is exchanged as text, Web services are programming-language agnostic. You could write a service in Perl to deliver weather information, and access it with clients written in Python, Tcl, Java, or C#. Or vice versa.

Yet for all of the benefits Web services bring, they are hardly a panacea. Protocols like SOAP and XML-RPC focus on how programs interact with each other, not on how people interact with programs. For example, I cannot scribble down the address of an XML-RPC service on a napkin and expect someone to use that service easily. Nor can I send a link to an XML-RPC service in the body of an email message.

Generally speaking, in order to use a Web service, I need to write some code, and have an understanding of how to use that particular service. This is why after about five years, Web services are still a niche technology. They work great if you want to offer programmatic access to a service like, say, eBay, Google, or Amazon.com. But if you want to publish information or offer a service to the widest possible audience, you still need to build a Web site.

The HTML Problem

Before Web services, automatic processing of data from the Web usually involved fetching HTML documents and scanning them to find new or interesting bits of data. Web services offer a more robust alternative, but do not eliminate the need to sift through HTML documents and "screen scrape" data off a Web page.

Processing HTML is the worst possible solution, but it is often the only solution available. HTML is a difficult format to parse. Many documents contain invalid or otherwise broken formatting. Using regular expressions to extract information from HTML documents is a common coping strategy, but it is quite error-prone and notoriously brittle.

Nevertheless, HTML is the universal format for data on the Web. Programmers who are building systems may consider alternatives like XML-RPC or SOAP Web services. But publishers and service providers are still focused on HTML, because it is the one format that everyone with a Web browser can always use.

Automating the Web

Since the early days of the Web, people have used programs that automatically scan, monitor, mirror, and fetch information from the Web. These programs are generally called robots or spiders. Today, other kinds of programs traverse the Web, too. Spammers use email harvesters to scour Web pages for email addresses they can spam. In the semantics of the Web community, "scutters" follow links to metadata files to build up databases of information about who's who and what's what on the Web.

There are many other mundane uses for Web automation programs. Link checkers rigorously fetch all the resources on a Web site to find and report broken links. With software development moving to the Web, testers use scripts to simulate a user session to make sure Web applications behave properly.

Fortunately, there are a great many Perl modules on CPAN to help with all of these tasks.

Most Web automation programs in Perl start with libwww-perl, more commonly known as LWP. This library of modules is Gisle Aas's Swiss Army knife for interacting with the Web. The easiest way to get started with LWP is with the LWP::Simple module, which provides a simple interface to fetch Web resources:

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

use strict;
use LWP::Simple;

## Grab a Web page, and throw the content in a Perl variable.
my $content = get("http://www.usenix.org/publications/login/");

## Grab a Web page, and write the content to disk.
getstore("http://www.usenix.org/publications/login/", "login.html");

## Grab a Web page, and write the content to disk if it has changed.
mirror("http://www.usenix.org/publications/login/", "login.html");
```

LWP has other interfaces that enable you to customize exactly how your program will interact with the Web sites it visits. For more details about LWP's capabilities, check out the documentation that comes with the module, including the lwpcook and lwptut man pages. Sean Burke's book *Perl & LWP* also provides an introduction to and overview of LWP.

Screen Scraping

Retrieving Web resources is the easy part of automating Web access. Once HTML files have been fetched, they need to be examined. Simple Web tools like link checkers only care about the URLs for the clickable links, images, and other files embedded in a Web page. One easy way to find these pieces of data is to use the HTML::LinkExtor module to parse an HTML document and extract only these links. HTML::LinkExtor is another of one of Gisle's modules that can be found in his HTML::Parser distribution.

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

use strict;
use LWP::Simple;
use HTML::LinkExtor;

my $content = get("http://www.usenix.org/publications/login/");

my $extractor = new HTML::LinkExtor;

$extractor->parse($content);

my @links = $extractor->links();

foreach my $link (@links) {
    ## $link is a 3-element array reference containing
    ## element name, attribute name, and URL:
    ##
    ## 0 1 2
    ## <a href="http://....">
    ## <img src="http://....">
print "$link->[2]\n";
}
```

Most modern Web sites have common user interface elements that appear on every page. These are elements like page headers, page footers, and navigation columns. The actual content of a page is embedded inside these repeating interface elements that appear on every page of a Web site. Sometimes, a screen scraper will want to ignore all of the repeatable elements and focus instead on the page-specific content for each HTML page it examines.

For example, the O'Reilly book catalog (*http://www.oreilly.com/catalog/*) has each of these three common interface elements. The header, footer, and navigation column on this page all contain links to ads and to other parts of the O'Reilly Web site. A program that monitors the book links on this page is only concerned with a small portion of this Web page, the actual list of book titles.

One way to focus on the meaningful content is to examine the structure of the URLs on this page, and create a regular expression that matches only the URLs on the list of titles. But when the URLs change, your program breaks. Another way to solve this problem is to write a regular expression that matches the HTML content of the *entire* book list, and throw out the extraneous parts of this

Web page. Both of these approaches can work, but they are error-prone. Both will fail if the page design changes in a subtle or a significant manner.

Of course, this is Perl, so there's more than one way to do it. Many Web page designs are built using a series of HTML tables. A better way to find the relevant content on this Web page is to parse the HTML and focus on the portion of the page that contains what we want to examine. This approach isn't foolproof, but it is more robust than using a regular expression to match portions of a Web page and fixing your program each time the Web page you are analyzing changes.

There are a few modules on CPAN that handle parsing HTML content. While HTML::Parser can provide a good general-purpose solution, I prefer Simon Drabble's HTML::TableContentParser, which focuses on extracting the HTML tables found in a Web page. This technique will break if the HTML layout changes drastically, but at least it is less likely to break when insignificant changes to the HTML structure appear.

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

use strict;
use LWP::Simple;
use HTML::TableContentParser;

my $content = get("http://www.oreilly.com/catalog/");
my $parser = new HTML::TableContentParser;
my $tables = $parser->parse($content);

## $tables is an array reference. Select to the specific table
## content and process it directly.
```

Interacting with the Web

Most Web automation techniques, like the ones described above, focus on fetching a page and processing the result. This kind of shallow interaction is sufficient for simple automation tasks, like link checking or mirroring. For more complicated automation, scripts need to be able to do all the things a person could do with a Web browser. This means entering data into forms, clicking on specific links in a specific order, and using the back and reload buttons.

This is where Andy Lester's WWW::Mechanize comes in. Mechanize provides a simple programmatic interface to script a virtual user navigating through a Web site or using a Web application.

Consider a shopping cart application. A user starts by browsing or searching for products, and periodically clicks on "Add to Shopping Cart." On the shopping cart page, the user can click on the "Continue shopping" button, click on the back button, browse elsewhere on the Web site, or search for products.

If you were developing this application, how would you test it? Would you write down detailed instructions for the people on your test team to repeat by rote? Or would you write a program to simulate a user, checking each and every intermediate result along the way? Mechanize is the tool you need to write your simulated user scripts. That user script might look something like this:

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

use strict;
use WWW::Mechanize;

my $mech = new WWW::Mechanize;

## Start with the homepage.
$mech->get("http://localhost/myshop.cgi");

## Browse for a book.
$mech->follow_link( text => "Books" );
$mech->follow_link( text_regex => qr/Computers/ );
$mech->follow_link( text_regex => qr/Perl/ );

## Put "Programming Perl" in the shopping cart.
```

```
$mech->follow_link( text_regex => qr/Programming Perl/);

## Add this to the shopping cart.
$mech->click_button( name => "AddToCart");

## Click the "back button."
$mech->back();

## Check out.
$mech->click_button( name => "Checkout");

## Fill in the shipping and billing information.
....
```

Mechanize is also an excellent module for scripting common actions. Every other week, I need to use a Web-based time-tracking application to tally up how much time I've worked in the current pay period. I could fire up a browser and type in the same thing I typed in two weeks ago. Or I could use Mechanize:

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

use strict;
use WWW::Mechanize;

my $mech = new WWW::Mechanize;

$mech->get('...');

## Log in.
$mech->set_fields(
    user => "my_username",
    pass => "my_password",
);
$mech->submit();

## Put in a standard work week.
## Log in manually later if this needs to be adjusted.
## (Timesheet is the 2nd form. Skip the calendar.)

$mech->submit_form (
    form_number => 1,
    fields => {
        0 => 7.5,
        1 => 7.5,
        ...
        9 => 7.5,
    },
    button => "Save",
);

## That's it. Run this again in two weeks.
```

Mechanize is also a great module for writing simple Web automation. Scripts that rely on HTML layout or specific textual artifacts in HTML documents are prone to breaking whenever a page layout changes. For example, whenever I am reading a multi-page article on the Web, I invariably click on the "Print" link to read the article all at once.

I could use regular expressions, or modules like HTML::LinkExtor or HTML::TableContentParser, to examine the content of a Web page to find the printable version of an article. But these techniques are both site-specific and prone to breakage. With Mechanize, I can analyze the text of a link — the stuff that appears underlined in blue in my Web browser. Using Mechanize, I can look for the "Print" link and just follow it:

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

use strict;
use WWW::Mechanize;

my $mech = new WWW::Mechanize;

my $url = shift(@ARGV)
$mech->get();

if ($mech->find_link(text_regex => qr/^2|Next$/i)) {
      ## This is a multipage document.
      ## Open the "print" version instead.
      $url = $mech->find_link(text_regex => qr/Print/);
}

## Open the file in a browser (on MacOS X).
system("open '$url'");
```

Conclusion

Perl is well known for automating the drudgery out of system administration. But Perl is also very capable of automating Web-based interactions. Whether you are using Web service interfaces like XML-RPC and SOAP or interacting with standard HTML-based interfaces, Perl has the tools to help you automate frequent, repetitive tasks.

Perl programmers have a host of tools available to help them automate the Web. Simple automation can be accomplished quickly and easily with LWP::Simple and a couple of regular expressions. More intensive HTML analysis can be done using modules like HTML::LinkExtor, HTML::Parser, HTML::TableContentParser, or WWW::Mechanize, to name a few. Whatever you need to automate on the Web, there's probably a Perl module ready to help you quickly write a robust tool to solve your problem.

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

By the mid '80s, it was generally recognized that FORTRAN was a dead language, and all that was left was to rework a few calculation libraries in C and bury the corpse.

This turns out to not quite be the case. FORTRAN has evolved and is still being used. Or, to quote an anonymous source: "We don't know what language engineers will be coding in in the year 2100. However, we do know that it will be called FORTRAN."

In the last month, I've seen three different projects add new functionality (GUI or network support) to a FORTRAN program.

Thanks to Arjen Markus, of WL Delft Hydraulics (a big engineering and FORTRAN shop), it's easy to embed the Tcl/Tk interpreter into the FORTRAN main code, providing yet another face-lift to an old compiler.

So, a brief digression from the articles about firewall validation to discuss embedding the Tcl interpreter into FORTRAN applications.

The first question is probably, "Why?" After all, most scripting applications are written using the pattern of extending the interpreter so you can write your application in the scripting language and call into a compiled library to do the heavy lifting, not embedding an interpreter into a compiled application.

The big reason is that the old code works. It may not do everything we'd like it to do, it may be cranky about input format, and it may have no GUI, but any conceptual flaws were revealed and fixed decades ago. Unfortunately, the act of making the old code work usually means that any pretense of architectural purity has long since been lost, and refactoring it into a library won't be fast or easy.

By adding the Tcl interpreter to a functional FORTRAN program, we can easily extend the program in ways that FORTRAN doesn't normally support. Tcl's clean socket support makes it easy to add client-server support to the program, and Tk makes it simple to add a GUI.

The first FORTRAN program I ever used was the Lunar Lander program, run from cards on the printing console of an IBM 1130. So, when I needed a project for relearning FORTRAN and playing with Markus's package, I decided to reimplement that program with a nice interactive GUI.

Markus has developed a FORTRAN->Tcl interface library that exposes the minimal subset of the Tcl "C" API to FORTRAN and allows the Tcl interpreter to be embedded in a FORTRAN program. It provides entry points to start the Tcl interpreter, evaluate a set of Tcl code, and exchange data between the compiled FORTRAN and interpreted Tcl sections of an application.

The body of the Lunar Lander code loops until the rocket hits the surface. Within that loop, it queries the user for the amount of fuel to burn and calculates the current height and speed. It looks like this:

```
      DO WHILE (fheight .GT. 0)
         IF (fuel .GT. 0) THEN
            WRITE(*,*) 'Enter burn: '
            READ(*,*) burn
         ELSE
            fuel = 0
            burn = 0
         ENDIF
         CALL calcspeed (speed, fuel, gross, burn, impulse, speed)
         i = i + 1
         fuel = fuel - burn
         fheight = fheight - speed
         WRITE(*,100) i, speed, fuel, fheight
  100    FORMAT('TIME: ', I3, ' SPEED: ', F8.2, ' FUEL: ', F8.2, ' HEIGHT: ', F8.2)
      ENDDO
```

A few runs looks something like this:

```
      Enter burn: 0.0
TIME:    1 SPEED:    101.70 FUEL:   1000.00 HEIGHT:       9898.30
      Enter burn: 0.0
TIME:    2 SPEED:    103.40 FUEL:   1000.00 HEIGHT:       9794.90
      Enter burn: 10.0
TIME:    3 SPEED:    103.31 FUEL:    990.00 HEIGHT:       9691.59
      Enter burn: 10.0
TIME:    4 SPEED:    103.20 FUEL:    980.00 HEIGHT:       9588.39
```

The first step in merging the Tcl interpreter into this code is to compile the FORTRAN-Tcl source files and create a library. This requires editing the makefile to reflect your environment, and then make ftcl.a. There are configuration options in ftcl_mod.c to support the C interface of different FORTRAN compilers.

Now that we've got a compiled library, we can start modifying the FORTRAN code.

The ftcl_start subroutine initializes the Tcl interpreter. This merges the Tcl C library Tcl_CreateInterp, Tcl_Init, and Tcl_EvalFile commands into a single subroutine.

The ftcl_start subroutine expects to be able to find the Tcl (and possibly Tk) libraries installed at runtime. The program will run if the libraries aren't installed, but only the core Tcl commands will be available, not the Tk graphics or other extensions.

Syntax: ftcl_start scriptname

scriptname The name of a Tcl script to load into the interpreter.
This line will initialize the Tcl interpreter.

```
CALL ftcl_start('config.tcl')
```

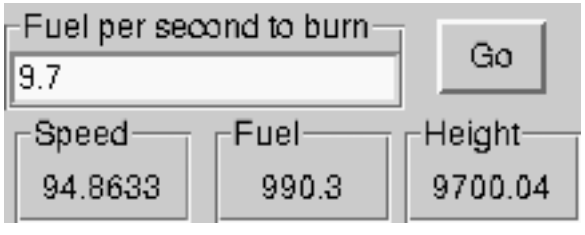
At runtime the config.tcl file is loaded and evaluated. This config.tcl script loads Tk and the GUI code and then builds the GUI to start the application.

```
package require Tk
source GUI-1.tcl
buildGUI 10000
```

The buildGUI procedure can generate any Tk GUI we'd like.

A GUI to duplicate the original Lunar Lander might have an entry widget to accept the amount of fuel to burn, a button to let the program know when we're ready to proceed, and a few labels to display the current height, speed, and remaining fuel.

It would look like this:



The entry and label widgets are contained within labelframe widgets. The labelframe widget is a container widget that can draw an outline around its edge and place a label on that outline. A labelframe can be used to group a related set of widgets, or identify a single widget. The labelframe creates a cleaner-looking GUI than the older technique of putting a label widget next to an associated data widget.

Syntax: labelframe .widgetName ?-option value?

widgetName The name for this labelframe.

-option value An option/value pair to configure the labelframe. Common options include:

-text *label text*
The text to display as a label for this frame.

-labelanchor *anchor*
Defines how to place the label. May be one or two of the letters n, s, e, w.

A single letter describes which side to put the label on, with the top being n, the right being e, etc.

If the anchor is two letters, the first defines the side of the labelframe for the text, while the second letter defines which side to anchor the label to, with the default being to center the text.

The default value for this is nw.

The first labelwidget contains a Tk entry widget. The entry widget is used to allow a user to enter any sort of textual information, such as login ID, password, or the amount of fuel to burn.

Syntax: entry .entryName ?-option value?

entryName The name for this entry.

-option value An option/value pair to configure the entry. Common options include:

-textvariable *variableName*
The contents of the entry widget will be automatically placed in the variable variableName.

-width *number*
Set the entry widget to be number characters wide. The default value is 20.

The -textvariable option simplifies creating a GUI. You don't need to write code to query the widgets; just use the variable name.

Like all Tk widgets, the labelframe command returns the name of the widget that was created. Using this name with commands related to this widget (grid, or creating child widgets) makes your code more robust if the GUI needs to be modified. For instance, in the code below, the frame and entry widget can be moved to another frame or top-level window by changing only the labelframe command.

The code to create the label frame and entry widget looks like this:

```
set w [labelframe .lfb -text "Fuel per second to burn"]
grid $w -row 1 -column 1 -sticky ew -columnspan 2
entry $w.burn -textvariable burn
grid $w.burn
```

The modern GUI could not exist without the button. The Tk button command creates a button that can contain an image, text, or both, and will invoke a command when the button is clicked.

Syntax: button .buttonName ?arguments?

A couple of commonly used arguments are:

-text string The text to display on the button.

-command body The body of a command to evaluate when the button is activated.

The code that defines this button is:

```
button .b -text "Go" -command "set ready 1"
grid .b -row 1 -column 3
```

Finally, there are the three labels showing the current height, speed, and remaining fuel.

The Tk label widget displays a string. Similar to the entry widget, a Tk label can be linked to a variable and will automatically update itself to display the contents of that variable when the variable is modified.

Syntax: label .labelName ?-option value?

-textvariable variableName This label will display the contents of the named variable.

-text string This label will display a particular string.

These widgets can be created and displayed in a foreach loop:

```
set col 0

foreach txt {Speed Fuel Height} var {speed fuel ht}  {
    set w [labelframe .lf$var -text $txt]
    grid $w -row 2 -column [incr col]
    set w [label $w.l-$var -textvariable $var -width 8]
    grid $w
}
```

Now, we need to move data to and from this GUI from the FORTRAN code.

The ftcl package supports a family of subroutines to copy data between the Tcl interpreter and the FORTRAN variables. These include:

ftcl_get_int(TclVariableName, FortranVariableName)
Copy an integer value from a Tcl variable to a FORTRAN variable.

ftcl_get_real(TclVariableName, FortranVariableName)
Copy a real value from a Tcl variable to a FORTRAN variable.

ftcl_get_double(TclVariableName, FortranVariableName)
Copy a double value from a Tcl variable to a FORTRAN variable.

ftcl_get_string(TclVariableName, FortranVariableName)
Copy a string value from a Tcl variable to a FORTRAN variable.

ftcl_put_int(TclVariableName, FortranVariableName)
Copy an integer value from a FORTRAN variable to a Tcl variable.

ftcl_put_real(TclVariableName, FortranVariableName)
Copy a real value from a FORTRAN variable to a Tcl variable.

ftcl_put_double(TclVariableName, FortranVariableName)
Copy a double value from a FORTRAN variable to a Tcl variable.

ftcl_put_string(TclVariableName, FortranVariableName)
Copy a string value from a FORTRAN variable to a Tcl variable.

In each case, the TclVariableName is passed as a string, while the FortranVariableName is just the name of the FORTRAN variable.

The command associated with the Go button sets the variable ready to 1 when the user clicks it. We can set and read that variable into a FORTRAN variable named irdy with these lines:

```
irdy = 0
CALL ftcl_put_int('ready', irdy)
! Pass control to the Tcl event loop
CALL ftcl_get_int('ready', irdy)
```

Like other interactive systems, Tcl has an event loop that collects inputs from the outside world (timer events, mouse movements, keyboard events, etc.) and processes them. While the FORTRAN code is being executed, the event loop isn't being processed, and the Tk GUI is inactive. The next trick is to transfer control to the Tcl interpreter to run the event loop and make the GUI active.

The ftcl_script subroutine passes a string to be evaluated to the Tcl interpreter. This string can be a single command or a longer script. We can process the event loop with the update command, which causes the Tcl interpreter to run a single pass through the event loop, and process a single event. This code is a round-robin polling loop that waits for the user to press the Go button:

```
irdy = 0
CALL ftcl_put_int('ready', irdy)
DO WHILE (irdy == 0)
   CALL ftcl_script('update')
   CALL ftcl_get_int('ready', irdy)
ENDDO
```

Polling loops are simple, but they eat up all available CPU cycles.

The vwait command described a few "Tclsh Spot" articles ago will cause a script to pause until a variable changes value. The interpreter stops at the vwait command and enters the event loop, processing events until the variable is assigned a new value. After this, the interpreter continues evaluating the commands in the script.

Internally, Tcl uses the select system library call to wait for events, rather than polling. Using the vwait command reduced the CPU usage of the Lander program from 50–80% to under 1%.

Syntax: vwait varName

varName The variable name to watch. The script following the vwait command will be evaluated after the variable's value is modified.

A simple procedure to wait for the button to be pressed looks like this:

```
proc wait4click {} {
    global ready
    vwait ready
}
```

And the FORTRAN main loop code looks like this:

```
DO WHILE (fheight .GT. 0)
    irdy = 0
    CALL ftcl_put_int('ready', irdy)
    CALL ftcl_script('wait4click')

    ! The Tcl burn variable now contains the amount of
    ! fuel to burn.

    CALL ftcl_get_real('burn', burn)


    ! If we're out of fuel, no burn.
    IF (fuel .LE. 0) THEN
       fuel = 0
       burn = 0
       CALL ftcl_put_real('burn', burn)
    ENDIF

    ! Calculate the speed.
    CALL CALCSPEED (speed, fuel, gross, burn,
          impulse, speed)

    ! Update the FORTRAN variables.
    i = i + 1
    fuel = fuel - burn
    fheight = fheight - speed

    ! Update the Tcl variables.
    CALL ftcl_put_int('time', i)
    CALL ftcl_put_real('speed', speed)
    CALL ftcl_put_real('fuel', fuel)
    CALL ftcl_put_real('ht', fheight)
ENDDO
```

The last step is to compile the new FORTRAN code and link with the ftcl and Tcl libraries. Using the GNU FORTRAN compiler, it looks like this:

```
g95 -o lander lander.f90 ftcl.a -L/usr/local/lib -ltcl8.4
          -ltk8.4 -lm
```

At this point, we've used 21st-century technology to duplicate the behavior of a 1960s program. It feels like it should have exposed rivets and be the size of a walk-in freezer.

The next "Tclsh Spot" article will look at using a Tk GUI to display the information graphically and run in realtime.

As usual, this code (including a version of Arjen Marcus's ftcl library) is available at *http://www.noucorp.com*.

the bookworm

BOOKS REVIEWED IN THIS COLUMN

SECURITY WARRIOR
CYRUS PEIKARI AND ANTON CHUVAKIN
Sebastopol, CA: O'Reilly, 2004. Pp. 531. ISBN 0-596-00545-8.

BIOMETRICS FOR NETWORK SECURITY
PAUL REID
Upper Saddle River, NJ: Prentice Hall, 2004. Pp. 252.ISBN 0-13-101549-4.

ESSENTIAL CHECK POINT FIREWALL-1 NG
DAMEON D. WELCH-ABERNATHY [AKA PHONEBOY]
Boston: Addison-Wesley, 2004. Pp. 612. ISBN 0-321-18061-5.

EXPLOITING SOFTWARE
GREG HOGLUND AND GARY MCGRAW
Boston: Addison-Wesley, 2004. Pp. 471. ISBN 0-201-78695-8.

CODE READING
DIOMIDIS SPINELLIS
Boston: Addison-Wesley, 2003. Pp. 495 + CD-ROM. ISBN 0-201-79940-5.

LINUX PROGRAMMING BY EXAMPLE
ARNOLD ROBBINS
Upper Saddle River, NJ: Prentice Hall, 2004. Pp. 492. ISBN 0-13-142964-7.

LINUX POCKET GUIDE
DANIEL J. BARRETT
Sebastopol, CA: O'Reilly, 2004. Pp. 191. ISBN 0-596-00628-4.

HARDWARE HACKING PROJECTS FOR GEEKS
SCOTT FULLHAM
Sebastopol, CA: O'Reilly, 2004. Pp. 331. ISBN 0-596-00314-5.

BEOWULF CLUSTER COMPUTING WITH LINUX, 2ND ED.
WILLIAM GROPP, EWING LUSK, AND THOMAS STERLING
Cambridge, MA: MIT Press, 2004. Pp. 617. ISBN 0-262-69292-9.

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@netpedant.com

Rik Farrow wrote a review of *Security Warrior* in the February *;login:*. I was going to review it at length, but decided that just a few words will suffice. (I've written a longer review for *http://www.UnixReview.com*.) I just wanted to say that I liked Peikari and Chuvakin's book more than Rik seems to.

Peikari and Chuvakin have written a valuable book which will soon find its way onto the shelf of everyone involved in network and machine security. I think of it as a supplement to Cheswick, Bellovin, and Rubin on firewalls and Schneier on cryptography, and a number of other works.

There are parts of Peikari and Chuvakin's book that are quite frightening. But war is frightening and computer/information war is no exception to this.

Further on this topic . . .

There are over a thousand books on computer security listed at Amazon. About a dozen of them are really worthwhile. That short list has just grown to include Peikari and Chuvakin's volume. Also quite informative is Reid's *Biometrics for Network Security*.

Fingerprints, footprints, hand geometry, iris and retina scans, voice, face, handwriting – they're all used. Reid's book is a first-rate summary of methods as well as a guide for system and network engineers.

Biometrics ends up with a very useful glossary and an extensive bibliography.

Essential Check Point FireWall-1 NG is by "PhoneBoy," who knows more about installing and maintaining FireWall-1 than anyone else. My problem is that FireWall-1 is proprietary. But if you're going to use it, PhoneBoy has produced the ultimate installation and configuration guide.

Hoglund and McGraw have devoted themselves to informing the good guys about what the black hats already know about how to take advantage (exploit) cracks and weak spots in the software we use. I see their *Exploiting Software* as a follow-up to Viega and McGraw's *Building Secure Software* of a few years ago. It's a worthwhile addition to the security bookcase (one shelf will no longer do).

Looking at Code

About 20 years ago, Marc Donner pointed out to me the importance of reading code carefully. He later taught a course at NYU on code reading. Spinellis has turned out a fine book on *Code Reading*, accompanied by a CD full of source and examples. He makes the same point that Donner did: You will write better code if you make it a habit to read good code.

Two Stray Penguins

Linux Programming by Example is a very fine book. I happen to be an admirer of Robbins' *Effective AWK Programming* (which lives next to my desk) and his book on vi (which I recommend frequently). But *Linux Programming* is exemplary. Interestingly, Robbins begins with, "One of the best ways to learn about programming is to read well-written programs." Donner and Spinellis would agree.

Robbins supplies the reader with a vast number of programs and a lot of elucidation. This is a primer in Linux pro-

gramming, but also serves as a tract on UNIX programming. Most of the illustrations are from actual GNU and UNIX V7 code. With ever more companies converting to Linux, this will be an invaluable resource for those converting from another [which one?] system.

I keep *Essential System Administration* and *Essential CVS* nearby for emergencies. Another valuable addition to O'Reilly's pocket guides is *Linux*. Barrett includes all the commands and flags I looked for and is so up-to-date that Fedora is covered.

Tinkering

If you believe that taking apart alarm clocks and building tuners or receivers is the right way to gain a technical education, *Hardware Hacking Projects for Geeks* is for you! "How to Hack 802.11b Antennas" and "How to Build an Internet Toaster" may be my favorite chapters, but "How to Hack a Furby" is useful, too. All of us who used to read *Popular Electronics* or *Popular Mechanics* or who still have a copy of an AARL handbook (my hand's up) will really love this book.

Beowulf Redux

The second edition of Gropp, Lusk, and Sterling's *Beowulf Cluster Computing with Linux* is over double the size of Sterling's volume on Beowulf of five years ago. But if you're into building a Linux cluster, you need it.

book reviews

SOFTWARE ARCHITECT BOOTCAMP, 2ND ED.

RAPHAEL C. MALVEAU AND THOMAS J. MOWBRAY

Upper Saddle River, NJ: Prentice Hall, 2003.
Pp. 400. ISBN 0-13-027407-0.

Reviewed by Harry DeLano
hdelano@adelphia.net

OVERALL IMPRESSIONS

Reviewing books isn't as much fun as writing software, but it can be pretty satisfying. I was a bit skeptical when I began reading this book, mostly because of a personal aversion to things military and bureaucratic. The bootcamp analogy kind of bothered me because, as I see it, the primary purpose of military training is to turn a human being into a mindless, obedient drone. My attitude softened as I realized that there is a lot of common sense here that would help one to understand the culture if one happens to get involved in a large software project. Actually, this book is more like Officer Candidate School than bootcamp and seems designed to help someone's career develop from "just" programming into leading a team of developers to successfully deliver a system.

If that's what you're interested in, this book might be for you. It reviews the techniques and tools used to help design a system at very high levels of abstraction, taking into account a wide range of points of view. It also covers many of the software development environments currently available. The reader is also offered tips on how to mold oneself professionally, including advice on how to improve communication skills and a discussion of various aspects of the psychology of software development.

In spite of my background in engineering application development, system administration, and systems programming on various platforms, as well as participation as a systems analyst on large software development projects, I found the perspective taken in this book was pretty much new territory. The overview to the use of components here is enlightening.

OVERVIEW OF BOOK

Bootcamp is about the need for software architects and the role they should play in large commercial software development projects. "The era of limitless demand for IT talent is over." This is partly due to the methods described here to build these systems. Techniques and technology in this field have evolved to allow more regimented and controlled development. (This, by the way, has made outsourcing easier.) What are these techniques and technologies? A set of formal models has evolved for specifying (1) what the problem is, (2) who cares, and (3) how that system will be built. The new technologies include the use of components (as opposed to just objects) in building large, distributed software systems.

The book is organized by chapters corresponding to some aspect of military training (e.g., "Jump School," "Military Intelligence"), with a final chapter containing advice on how to design your career. Again, the military analogy used to explain how one might be a software architect fits well with the authors' rigid and hierarchical view of how teams might best work.

THE DETAILS

What follows is a collection of what seemed to be the salient points from each of the first five chapters of the book.

CHAPTER ONE: INTRODUCTION

This chapter includes some interesting (unverified) facts worth mentioning: "Corporate America spends more than $275 billion each year on approximately 200,000 application software development projects"; application development

success rates were only 16% in 1994 vs. 26% by 1998; the cost of failed projects went from $80 billion in 1995 to $75 billion in 1998; cost overruns dropped from "$59 billion in 1005 [sic, probably 1995] to . . . $22 billion in 1998."

Chapter Two: Military History

This is an overview of the field of software architecture. It seemed a little incoherent to me in that it introduces five "schools of thought" on the subject and then goes on to describe only a couple of them thoroughly, interspersed with garbled references to others. For example, they throw in a mention of "Enterprise Architecture" without providing previous noticeable context to help understand how it fits into the discussion.

An attempt is made to justify the need for architects and to describe how one operates, including what tools they use to build system specifications. In designing a system, the architect needs to consider multiple viewpoints to achieve simplicity, maintain strict consistency in terminology to achieve system understanding, and use the notion of "complete models," describing multiple phases of development while taking care not to get too detailed. There are techniques available to allow the consideration of several points of view of the project. One of the approaches to defining a model (the Zachman Framework) provides for 30(!) viewpoints.

Information systems have evolved from static and local to dynamic and global, so that we now have distributed multi-organizational systems with heterogeneous hardware and software configurations. The architect needs to be able to separate concerns about business application functionality from concerns about distributed-system complexity. Requirements change frequently and account for the majority of system software costs of the life cycle, so there's a need to "future proof" the architecture.

Once careful consideration has been given to what problem the system will solve, there needs to be a way of describing the solution in fairly general terms. We used to write systems in which the code was all in one place and data was in another, but both were in the same neighborhood. Then the object-oriented approach came along and software is now written so that the code and the data it is associated with are encapsulated into elements (objects). These objects work together through message-passing mechanisms (fairly local and fairly primitive (sockets, RMI, etc.). Lately, it has been found that these object-oriented elements need to work with each other in very heterogeneous, widely distributed environments. There are many common approaches used in the various communication environments ("idioms") that could be abstracted and used in specifying how the software project might solve a problem.

The drift of this chapter seems to be that there is a need for a good set of architectural tools to describe how to have a successful software development project. One reason is that there is a need for a system to be resistant to requirements and context changes. In the past, writing specifications and using object-oriented development techniques were sufficient. However, that was before the need for globally distributed enterprise requirements. Now the problem needs to be described from various stakeholders' points of view. These techniques allow one to propose solutions that are abstract enough that the underlying technology can be ignored above a certain level in the hierarchy of system stakeholders (investor, architect, development manager, developer). Development can then proceed by taking advantage of component-based tools to bring the higher-level vision to fruition. Development and implementation technologies can be left as details.

The authors make the point that 70% of the code in a typical application is infrastructure. Component technology supposedly means that reinventing the wheel is no longer necessary. That was the promise of OO programming, but now the "idioms" we see over and over again that are used to have objects play together are defined as abstract elements of models and have been made part of the infrastructures that are now available for component-based development.

The authors review various approaches to this need and seem to settle on the OSI's X.900 Reference Model for Open Distributed Processing as about the best currently available. There are other choices (e.g., IBM's 4+1 View Model), but most are variations on the theme of RM ODP.

These tools force one to take several points of view in describing a solution to the problem: enterprise (what the system's purchaser needs it to do); information (what data will flow and how); engineering (familiarity with the guts of the infrastructure, similar to an OS engineer); computation (partition of the system into components that can interoperate in a distributed fashion and definition of the boundaries between components, and use of CORBA IDL [see below]); technology (component interoperability concerns).

RM ODP is described over the course of about 200 pages in a set of four documents that are concise but "relatively inscrutable." This includes conformance assessment criteria that can be used to decide whether or not the project is going well.

The notion of "design patterns" is introduced, which has been used to codify and document a lot of software knowledge. The authors state that patterns represent a rejection of originality as a technical goal (so leave your imagination at home). A very formal mechanism exists for documenting patterns, and

book reviews

these are collected in catalogs you can buy and which architects should study to be pattern literate. Design patterns are derived as follows: a single design occurrence is an event; two occurrences are a coincidence; three constitute a pattern.

The formality referred to as "anti-patterns" are patterns, enhanced with annotations, known to have failed in their attempt to solve a problem. Included is a description of how a new version might be derived from that original ill-used pattern to better solve the original problem (a sort of tale of woe with a happy ending). There is a class of patterns called idioms; these are programming-language specific (think cookbooks).

Chapter Three: Basic Training

Here the authors go over the tools available to a software architect for doing the actual development. It reviews a history of software environments, starting with procedural technology, in which program code exists separate from the data it deals with. This is OK, but if data representation is modified, there can be a large impact on the program. OO technology (pieces of data and program elements to access and manipulate that data all together – OK but weak for distributed processing since language-specific encapsulation is insufficient to support software reuse and distributed systems); objects communicating with each other via messages, devoid of software-architecture approach; specification objects representing modules; rapid iterative prototyping, with ruthless disregard for architectural principles (bad).

They describe the evolution of distributed technology, from file servers through database servers and transaction processing monitors through distributed objects to *N*-tier componentware. Object-oriented middleware is an outgrowth of procedural predecessors, including RPC, socket-based Open Network Computing, and Distributed Computing Environment. Later, Microsoft's Distributed Common Object Model attempted to add another layer of abstraction, but, according to the authors, it still exposed too much of its underlying distribution mechanism.

CORBA (Common Object Request Broker Architecture) attempts to provide a standard interface to services used by applications, no matter what platform they run on. Using CORBA Interface Definition Language is a way to standardize on APIs throughout a system. Vendors usually provide hundreds of APIs, and developers pick and choose from that list as they see fit, causing there to be many more used in the project than really need be. Providing a layer between the application and the OS services needed by the application that is tailored to the needs of the organization simplifies the use of those OS services, making them more manageable and maintainable.

In component technology, specification objects represent constraints rather than programming objects. It emphasizes larger-grained software interfaces and modules. Component infrastructures include MS .Net and Sun Java Enterprise Java Beans, including CORBA. Software architecture for componentware allows parallel, independent development of the system or its parts (good for outsourcing). It tries to standardize means of component interaction so that custom interfaces between individual components are minimized. Distributed components can communicate using standard interfaces by way of CORBA and its IDL, which is centered around the Object Request Broker; CORBA Services provide a way to implement CORBA on particular OS platforms, including Netscape Communicator(!); XML allows for universal data interchange.

A comparison of J2EE and .Net shows that .Net is easier to use but J2EE is more robust and might be preferred by experienced developers, since it allows greater programmer flexibility (hey, I thought that was bad!).

Chapter Four: Software Architecture: Going to War

Here the authors go over some ways that large software projects happen and then lay out steps for an architectural approach. Most large software is very fragile (sucks) except for "telecommunication systems, video games, mainframe operating systems, or rigorously inspected systems (e.g., CMM Level 5)." (Hmm, how about the Linux kernel?) They point out that traditional system assumptions are local and assume that the system will be stable, but in a distributed system one needs to assume that things will be global and unstable. This is like comparing Newtonian mechanics to quantum mechanics. Also, distributed systems typically involve more than one organization to deal with. They recommend the following approach: proactive thinking (actively anticipate problems); use design patterns and anti-patterns to avoid redesigning the wheel; use the Universal Modeling Language to lay out and describe the design.

Traditionally, large systems are pulled off by "heroic programmers" coming in to rescue a flagging project. They make it work under extreme time pressure but typically leave a fragile and undocumented system. Architects must avoid this by initially laying out the project very carefully (using the latest tools like CORBA IDL and UML), making sure that appropriate development tools are used (componentware), and staying on top of the development process to make sure that no heroes are needed.

The architecture-centered development process should include the following steps: system envisioning, requirements analysis, mock-up prototype, architecture planning, architecture prototype,

project planning, parallel development (of components), system transition (quality assurance), operations and maintenance (rolling it out), and system migration (moving the organization over to the new system). What's new about all of this? The process has been very much more formalized as the componentware approach has evolved. For example, the architecture-planning step involves documenting these architectures using OSI's ODP: enterprise (how the business works), logical information architecture (what objects are needed to represent the business), computational interface (what will flow between these objects and how), distributed engineering (allocation of responsibilities), and technical selection (component mechanisms).

Chapter Five: Software Architecture: Drill School

Here we are introduced to the idea of using "design levels" to lay out a model. Design levels have been applied to hardware design for decades and are used to simplify by separating concerns. Software may be looked at as having various levels of granularity (objects and classes [the finest, defined by programming language], configurations of objects [the next level up], micro-architecture, frameworks, applications, systems [the coarsest]).

The last five chapters of the book cover how best to provide leadership in a software development project. Topics include how to be a good leader, project management basics, the architect's role vis-à-vis the project manager, various roles in the software design process, teamwork communication skills, using UML, architectural mining, and the psychology of software development. Some highlights follow.

The architect acts as an assistant to the project manager.

Many experienced programmers will not be able make the shift from the procedural to the object paradigm.

Component architectures can be looked at as having four layers: foundation (classes to manage basic object services), domain (business entities), application (specialized domain classes for particular views), and user interface (tailored application classes for various types of user).

There's a "process for creating processes" (sub-projects?). The software architect must be an expert in teamwork and make sure that the team works well together. Also, he or she has responsibility for explaining incremental development to upper management, justifying required shifts in architecture.

It's often necessary to force the "lone wolf" developer into constructive interaction.

There are lot of issues with regard to communications, including running good brainstorming sessions (and being able to decide when they're needed), keeping good records of meetings, making sure consistent terminology and modeling notation is used, and listening well.

Use Universal Modeling Language to document the meta-model of applications and systems. "Design a thing considering its next larger context – a chair in a room."

Architectural mining is extracting from preexisting designs information on how those designs could be applied to the problem at hand.

Polya's paradox: It's often easier to solve a general problem than a specific one (which may have an overwhelming amount of detail).

The traditional approach of "analyze requirements, design, code, test" is being replaced by a more iterative approach wherein this cycle is done repeatedly over the life of the project for a particular piece of the system. This provides for more feedback opportunities.

Psychological techniques include being able to propagandize without seeming to do so. Taking advice is difficult by nature. Architects need to avoid situations where positive feedback causes developers to get carried away in the next phase by trying to top themselves. This usually leads to a system that's too complex.

Signs of egomania in software architects (sampling): Forgetting that the job is about communicating, not winning arguments; use of the royal "we"; referring to developers as "grunts"; believing these signs are about someone else.

"A typical adult gets angry about ten times every day." Not me.

Career advice is offered, including how architecture isn't taught much in schools yet; so, for now, you'll have to develop your own curriculum.

CONCLUSION

As I said earlier, I was skeptical when I first got this book, but I'm now glad I had a chance to spend time with it. It covers a fast-developing field and provides lots of detail that might be handy to delve into if ever one finds oneself working on a large enterprise software project. In fact, it might convince you that you want to be a software architect. Further information can be found at the Worldwide Institute of Software Architects (*http://www.wwisa.org*).

USENIX notes

USENIX MEMBER BENEFITS

As a member of the USENIX Association, you receive the following benefits

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

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

ACCESS TO PAPERS from the USENIX Conferences online starting with 1993: *www.usenix.org/publications/library/proceedings/*

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

DISCOUNTS on registration fees for all USENIX conferences.

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

SPECIAL DISCOUNTS on a variety of products, books, software, and periodicals. See *<http://www.usenix.org/membership/specialdisc.html>* for details.

FOR MORE INFORMATION REGARDING MEMBERSHIP OR BENEFITS, PLEASE SEE

http://www.usenix.org/membership/

OR CONTACT

office@usenix.org

Phone: 510 528 8649

Election Results

The results of the elections for Board of Directors of the USENIX Association for the 2004–2006 term are as follows :

PRESIDENT
Mike Jones, *Microsoft Research*

VICE-PRESIDENT
Clem Cole, *Ammasso*

SECRETARY:
Alva Couch, *Tufts University*

TREASURER:
Theodore Ts'o, *IBM*

AT LARGE
Matt Blaze, *University of Pennsylvania*
Jon "maddog" Hall, *Linux International*
Geoff Halprin, *The SysAdmin Group*
Marshall Kirk McKusick, *Author and Consultant*

NOT ELECTED
John Nicholson, *Shaw Pittman LLP*
Brian Noble, *University of Michigan*

Newly elected directors will take office at the conclusion of the next regularly scheduled board meeting, which will be held June 27, 2004, in Boston, MA.

Summary of the USENIX Board of Directors Meetings

by Tara Mulligan
tara@usenix.org

The following is a summary of the actions taken by the USENIX Board of Directors from December 4, 2003, through March 2, 2004.

FINANCES

The final budget for fiscal year 2004 was approved.

CONFERENCES

The Board approved a proposal from David Culler to sponsor the First International Workshop on Real Large Distributed Systems (WORLDS), which will precede OSDI in 2004.

Registration fees for the 2004 USENIX Annual Technical Conference were approved as shown in the table on page 66.

ADVOCACY

In February 2004, the USENIX Board drafted a response letter to claims by the SCO Group, Inc. (who have initiated legal action against technology researchers

Event Type	Member	Nonmember	Student member	Student nonmember
Annual Technical Conference	1 day: $250 2 days: $450 3 days: $600 4 days: $700 5 days: $775	1 day: $350 2 days: $550 3 days: $700 4 days: $800 5 days: $875	1 day: $75 2 days: $145 3 days: $210 4 days: $270 5 days: $325	1 day: $105 2 days: $175 3 days: $240 4 days: $300 5 days: $355
Annual Technical Conference Training	1 day: $ 600 2 days: $1100 3 days: $1550 4 days: $1950 5 days: $2300 6 days: $2600	1 day: $600 2 days: $1100 3 days: $1550 4 days: $1950 5 days: $2300 6 days: $2600	1 day: $150 2 days: $300 3 days: $450 4 days: $600 5 days: $750 6 days: $900	1 day: $150 2 days: $300 3 days: $450 4 days: $600 5 days: $750 6 days: $900

and developers using open source code) that challenge the legality of General Public Licenses. In early 2004, SCO sent a position letter to members of the U.S. Congress to promote their case, which is currently in federal courts. The USENIX response letter refuting SCO's claims was sent to USENIX members, members of Congress, and the media. The text of the letter was included in the April 2004 issue of *;login:* and on the USENIX Web site.

FUTURE BOARD MEETINGS

The next scheduled Board meeting will be Sunday, June 27, 2004, at the Annual Technical Conference in Boston, MA. The newly elected Board members will be invited to attend, and they will take office at the conclusion of the meeting.

USENIX STRATEGY MEETING

On March 1, 2004, the Board of Directors met with several long-time members, staff, and past Board members to discuss future direction for USENIX. The discussion focused on who the USENIX constituency is, what the organization has done successfully in the past, and areas that may be worth investigating. The following actions were decided upon:

- USENIX will research co-hosting conferences with groups from other disciplines, such as bioinformatics, law, and physics.
- USENIX will investigate holding regional training events.
- The USENIX Awards Committee will look into offering a "Most Influential Presentation" award to honor those who've introduced groundbreaking technology at USENIX conferences.
- USENIX will look into how best to approach advocacy on issues of relevance to the membership.

Influencing the Field

by Peter Salus

USENIX Historian

peter@pedant.com

Remember that class or book or paper that changed the way you looked at things?

The USENIX Board has asked me to organize a new award. This would be for the Most Influential Paper of Ten Years Ago.

Not the "best" paper, necessarily. But the one you feel has had the widest and most profound influence.

This first time, all papers offered in 1993 or 1994 are eligible. (Most of them are on the USENIX Web site.)

If it was delivered at a workshop, a symposium, a LISA, or an annual Technical Meeting, it's eligible.

Send me your nominations by 1 August. I'll put the short list in the October *;login:*.

The winner will be announced at a USENIX conference later in 2004.

Write to: *best10@usenix.org*

conference reports

First Symposium on Networked Systems Design and Implementation (NSDI '04)

San Francisco, California March 29-31, 2004

TECHNICAL SESSIONS

SENSOR SYSTEMS SESSION

Summarized by Ramakrishna Kotla, Xun Luo, and Vinay Mallikarjun

The Emergence of Networking Abstractions and Techniques in TinyOS

Philip Levis, Joseph Polastre, Robert Szewczyk, Alec Woo, Eric Brewer, and David Culler, University of California, Berkeley; Sam Madden, MIT Computer Science and Artificial Intelligence Laboratory, and Intel Research Berkeley; David Gay, Intel Research Berkeley

Is sensor system design any different from conventional mobile system design? Sam Madden explained that these two kinds of systems differ in that sensor systems are limited by resources like memory (512–4K bytes), battery power, etc. "Bluetooth is totally wrong for sensor systems, as power management is totally integrated into the protocol and does not expose it to the application." He explained the design of the TinyOS operating system, used in sensor systems that they built at Intel Labs, and how it was different from traditional OSes. TinyOS uses component-based design with no "kernel" and a single application running at a time, which chooses its own set of OS services. He then explained the software abstractions of the resources (energy, memory, etc.), the various applications built on top of TinyOS (localization, habitat monitoring, etc.), the services provided in TinyOS (network stack, radio stack, mica stack), and the issues involved in building these services. He concluded the talk by making the observation that software abstractions in sensor systems are dictated by limited memory constraints, energy concerns, flexibility for application to choose policies, and the noninteractive nature of applications, which makes it different from traditional mobile systems.

Q: The first generation of sensor systems are not powerful, but with powerful future sensor systems will the abstractions change?
A: Some of the constraints and issues with sensor systems are fundamental, like energy concerns, which will hold in future, and these abstractions hold good for them as well.

Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks

Philip Levis and David Culler, University of California, Berkeley, and Intel Research Berkeley; Neil Patel, University of California, Berkeley; Scott Shenker, University of California, Berkeley and ICSI

This won the Best Paper award at the conference. Re-tasking nodes in the sensor systems requires efficient dissemination of data. Maintenance of the system could be costly since it calls for transmitting 20–400 bytes of data for each update. Philip Levis presented a gossip-based algorithm called Trickle for propagating and maintaining code in sensor systems. The algorithm assumes the presence of a broadcast medium and that nodes can verify metadata. This

Philip Levis

This issue reports on the First Symposium on Networked Systems Design and Implementation (NSDI '04) and on the 3rd USENIX Conference on File and Storage Technologies (FAST '04).

For NSDI '04: Our thanks to Amin Vadhat, who shepherded the following summarizers:

Magdalena Balazinska
Laura Grit
Vinay M. Igure
Suchita Kaundin
Chip Killian
Ramakrishna Kotla
Xun Luo
Vinay Mallikarjun
Piyush Shivam
Chunqiang Tang
Praveen Yalagandula
Aydan Yumerefendi

For FAST '04: Thanks to Ismail Ari and his summarizers:

Michael Abd-el-Malek
Nitin Agrawal
Akshat Aranya
Dean Hildebrand
Andrew Klosterman
Xun Luo
Kiran-Kumar Muniswamy-Reddy
Steve Schlosser
Shafeeq Sinnamohideen
Deepa Tuteja
Wenguang Wang
Lan Xue
Aydan Yumerefendi

Note: The reports on BSDCon '03, held in San Mateo, California, September 8–12, 2003, can be found at *http://www.usenix.org/events/bsdcon03/confrpts.pdf*

algorithm uses a "polite gossip" policy in which the nodes periodically broadcast a code summary to neighbors but stay quiet if they hear a summary identical to theirs. It uses a sender-side rate-controlling mechanism to avoid flooding the network: nodes hear a small trickle of packets that can keep them up-to-date. Philip went on to explain how they evaluated the system using a TOSSIM simulator and the real system. He concluded the talk by presenting results, which show that Trickle scales logarithmically with respect to the density of the nodes and can achieve rapid propagation with low maintenance.

Q: How does it scale with a large amount of data propagation?
A: For large data, a hierarchy of metadata suppression is used to scale, where the only differences in data are sent.
Q: Lossy links create heavy tail distribution in the results. Can we increase the load probabilistically so that there are no bottlenecked links?
A: We can identify critical links and retransmit more often on those links.

Programming Sensor Networks Using Abstract Regions

Matt Welsh and Geoff Mainland, Harvard University

Programming sensor networks is difficult in that data needs to be pushed into the network, as opposed to pulling out the data and processing it at a centralized node. Matt addressed the question, "How do we develop a programming model for sensor networks as a whole?" He raised issues that the programming model has to take care of, like allowing accuracy/overhead tradeoff, flexible communication primitives, exposing certain resource parameters to the application while hiding others, etc. He defined "abstract regions" as a group of nodes with some geographic or topological relationship that capture common idioms in sensor networks. Various operations can be performed on abstract regions, like neighbor discovery, accessing shared variables, and reductions to support aggregation of shared variables. He then explained the applications he built using these primitives: contour finding, directed diffusion, and object tracking. Region operations are inherently statistical, and programming abstractions allow tuning these variables in order to trade resource usage for accuracy. He concluded the talk by making a case for a spatial programming language using abstract regions to program sensor networks that are inherently resource constrained, volatile, and distributed. For further details on this work, please visit *http://www.eecs.harvard.edu/~mdw/proj/mp*.

Q: Why can't we use a distributed programming paradigm for sensor networks?
A: Nodes in sensor networks are resource constrained, which requires a different programming methodology.
Q: Ensemble programming is a good thing irrespective of resource constraints.
A: I agree.
Q: Why is exposing certain parameters necessary in sensor systems, unlike traditional systems?
A: Programmers think at a high level; however, they need knobs to adapt to application requirements like the tradeoff between accuracy and lifetime.

NETWORKING SESSION

Summarized by Laura Grit and Xun Luo

Design, Implementation, and Evaluation of Duplicate Transfer Detection in HTTP

Jeffrey C. Mogul, Terence Kelly, HP Labs; Yee Man Chan, Stanford Human Genome Center

Terence Kelly presented this paper addressing the problem of duplicate transfers in HTTP. Redundancy can be foiled due to aliasing, rotation, and faulty metadata. To demonstrate that this problem exists, the authors ran two large traces on Compaq and WebTV and found that one in five transfers is redundant. The authors' solution is to create a digest of payloads (the entire message body of the HTTP response) and to cache the payload in a digest in a method they call Duplicate Transfer Detection (DTD). Their algorithm involves the client asking the digest for a requested URL. If a match is in the cache, the server delivers the cached payload; if it is not in the cache, the client needs the full response from the URL. They argue that DTD eliminates all redundant transfers.

Their talk focused on the analytic and benchmark results of their work. The authors found their method reduces latency when response length is sufficiently long. The overheads of their implementation were under 2%. When asked if they used different benchmarks, Terence referred the audience to the paper to see results with different Web service payload models. With regard to benchmarking, their response time and time to first byte were best with small payloads, like that of cell phones. The cost of a DTD miss is constant except for large body sizes, and they pay an extra round trip for misses. However, the benefits of DTD increase with body size.

Code is available for their DTD implementation over Squid at *http://devel.squid-cache.org/dtd*, but they warn that the code is currently too buggy for production use.

OSPF Monitoring: Architecture, Design, and Deployment Experience

Aman Shaikh and Albert Greenberg, AT&T Labs–Research

Aman Shaikh presented their experiences with OSPF (Open Shortest Path First) monitoring. Their objective was to perform both real-time and offline analysis of OSPF behavior. They created a monitor that collects OSPF Link State

Advertisements (LSAs) passively from the network. There are three ways their monitor can attach itself to the network: multicast group, full adjacency mode, or partial adjacency mode. In particular, they are looking for topology changes, node flops, LSA storms, and anomalous behavior. They have deployed their monitor in both an ISP and an enterprise network.

In implementation, the monitor consists of three components: LSA Reflector, to capture LSAs from the network; LSA aGregator, to perform real-time analysis; and OSPF Scan, for offline analysis. The monitor infers what the network looks like from the messages being sent. If it detects anomalies, it sends alarms to a higher-level network manager. By doing this, they can detect things that other management systems cannot find. For example, they have caught equipment problems, configuration problems, and even an OSPF implementation bug. Since they keep all information, their offline performance evaluator can make their offline simulator think there is an actual large topology and can accurately determine what is happening on the network.

When asked about the difference between partial and full adjacency modes, they said that, in their full mode, LSA Reflector is more unstable. Aman was also asked to explain what OSPF monitoring does not do. He said that OSPF monitoring does not do everything, but it can tell people if the problem is in OSPF and whether it may need other monitoring when OSPF is not running.

OverQoS: An Overlay-Based Architecture for Enhancing Internet QoS

Lakshminarayanan Subramanian, Ion Stoica, and Randy Katz, University of California, Berkeley; Hari Balakrishnan, MIT

Lakshminarayanan argued that the current best effort is not sufficient for QoS applications. Deployment of techniques is difficult because the IP layer is hard to change – you need everyone to agree on an implementation – and requires end-to-end deployment. They believe that overlay networks can do the same thing for QoS as they did for Multicast.

In this overlay environment, there is no control on the cross traffic or on node placement. Their implementation must be fair to cross traffic and keep network stability when there are multiple QoS overlays. Their technique is to trade resources between overlay flows to manipulate QoS parameters. For example, they would sacrifice throughput for better loss characteristics. Their design ensures that network overlay traffic looks like TCP traffic and follows TCP guarantees. To do this they add controlled-loss virtual links to characterize the service for the overlay and to bind observed loss rate. They keep normal flows the same across the link, but use the rest of the space on the link for service. These overlays can be used with multi-player games, streaming media, or leasing overlay networks. In their experimentation, they found OverQoS can provide statistical loss and bandwidth guarantees, it is fair and stable to coexist with TCP, and cost overheads are only between 0.5 and 6%. With 99% probability, available bandwidths were at least the QoS value.

When asked what is the additional overhead of OverQoS, they said it is the bandwidth added to the stream to reduce loss rates. The follow-up question was, What if everyone starts sending multiple packets? In OverQoS, the net aggregate rate looks like TCP. If they were just looking at the flow and appending 30% more packets, there would be a problem; however, they are concerned with the aggregate rate.

DISTRIBUTED HASH TABLES SESSION

Summarized by Magdalena Balazinska, Chip Killian, and Praveen Yalagandula

Designing a DHT for Low Latency and High Throughput

Frank Dabek, Jinyang Li, Emil Sit, James Robertson, M. Frans Kaashoek, and Robert Morris, MIT Computer Science and Artificial Intelligence Laboratory

Frank Dabek presented the need for a DHT with low latency and high throughput by citing the infeasibility of realizing a backup system on a PlanetLab type environment, where bulk reads and writes experience a low throughput of 10Kbps and any operation suffers from a high latency of about 450ms.

This work has two main contributions. The first is a series of modifications made to the original Chord design that reduce the lookup latency. The second is a new transport protocol called STP, which makes use of per-hop ACKs and a sliding window for RPCs to achieve good throughput as well as good latency.

The first set of performance changes is the switch from iterative to recursive routing, using proximity neighbor selection, striping blocks across nodes, and using replication and neighbor selection to choose the closest nodes that also contain the interesting data. Low latency, however, is not sufficient, which is why they also consider high throughput.

To achieve high throughput, there is a need to efficiently manage the many connections needed by the parallel downloads. Opening many TCP connections is not efficient, because each one imposes a start-up latency (slow start), takes time to acquire a good congestion window estimate, and uses resources. In

contrast, using a few persistent connections constrains communication to the overlay links only. The fundamental problem is that there is a need to limit the number of parallel connections currently maintained (and for those connections to not use TCP). For this purpose, STP keeps a sliding window of RPC connections to control the number of outstanding calls. Combined, these two techniques can both decrease latency and increase throughput.

More information is available at *http://pdos.lcs.mit.edu/chord*.

Q: Amin Vahdat, UC San Diego. Regarding performance results of STP vs. TCP. Is it inherent that blocks have to be sent through overlay links when using TCP connections?
A: Yes, because you don't want to open connections to all nodes.
Q: Terence Kelly, HP. Are any methods from the Microsoft USITS paper on Skipnet applicable?
A: In Skipnets, nodes with the same geographic location have the same name, so lookups stay local when possible. No such change in namespace is needed in the Chord approach.
Q: Nick Murphy, Microsoft. What about insert and update traffic?
A: Coding is a good example of the tradeoff between read and write performance. All lookup optimizations would be the same.
Q: Nick Murphy, Microsoft. Are you storing to stable storage or in memory?
A: We use a database, so we do store to stable storage.
Q: Achim, UC Berkeley. Can we use DHTs as a backup store? How much data did you store in the system and per node? What about reliability?
A: The cost of keeping nodes in sync is future work.

Beehive: O(1) Lookup Performance for Power-Law Query Distributions in Peer-to-Peer Overlays

Venugopalan Ramasubramanian, Emin Gun Sirer, Cornell University

Passive caching as done by current DHT algorithms suffers from two drawbacks: (1) The heavy tail of the zipf type access distributions implies that the caching cannot reap many benefits and (2) mutable objects give rise to coherency problems, and the caching can only provide a weak consistency model. Rama described Beehive, a general proactive replication framework for DHTs that automatically replicates and places the objects on several nodes in the DHT, satisfying the application-specified latency requirements while optimizing the maintenance bandwidth. Beehive achieves this through an analytical model based on a closed-form optimal solution that fulfills the application's requirements with the minimum number of replicas and that approximates that closed-form solution with nodes deciding locally how much to replicate objects based on estimated popularity.

Rama also presented CoDoNS, a cooperative domain name service, built on Beehive, to replace the traditional DNS. Through experiments on PlanetLab with MIT DNS traces as workload, the authors observed that Beehive (1) has very low latency (7ms vs. 39ms in legacy DNS), (2) is resilient against denial-of-service attacks, and (3) propagates updates quickly.

More information is at *http://www.cs.cornell.edu/people/egs/beehive*.

Q: Matt Welsh, Harvard: The goal of replication is failure protection, but you did not evaluate failures.
A: Yes, we did evaluate for some churn rate. The key point of the talk, however, is to decrease latency in lookup.
Q: Rob Szewczyk, UC Berkeley. Caching the Web would require too much space. Is this approach useful for large workloads?
A: Better than passive caching, but we didn't try large workloads.
Q: Mike Walfish, MIT. What happens when alpha is greater than 1?
A: Optimality factor is not valid for alpha greater than 1.

Efficient Routing for Peer-to-Peer Overlays

Anjali Gupta, Barbara Liskov, and Rodrigo Rodrigues, MIT Computer Science and Artificial Intelligence Laboratory

An important challenge in structured peer-to-peer systems such as Chord is to achieve efficient routing in spite of frequent changes in membership. Currently, most peer-to-peer systems perform lookups in time logarithmic to the size of the system, because nodes know only about log *N* other nodes. This large number of hops causes high latency.

The main thesis of the work that Anjali presented is that it is "possible to keep full routing state on every node and route in one overlay hop." Two approaches are proposed. The first scales up to order 100K nodes and routes in one hop. The second routes in two hops, but it scales to significantly larger networks (>1M).

To achieve one-hop routing, every node keeps a complete routing table. To keep these routing tables up-to-date, nodes organize themselves into a hierarchy over which all membership-change events propagate. The hierarchy is as follows. The ring is divided into a configurable number of slices. The node with an identifier closest to the middle of the slice is the slice leader. Each slice is subdivided into units. The node in the middle of the unit is the unit leader. All membership changes propagate up from nodes that detect the event to slice leaders. Periodically, slice leaders exchange information about changes in their

respective slices and push these changes down to nodes within their slice.

Slice leaders are the most loaded nodes because they send all membership updates to all other slice leaders. The load is low for systems less than a million nodes in size, but at greater sizes we have to either reduce the load or switch to a super-node scheme. One solution to alleviate the load is for slice leaders to send only membership updates about selected representative nodes within their slices. This approach improves scalability but increases lookup time by one hop since lookups now go through representative nodes before getting to the final destination. The first hop, however, can take advantage of proximity routing.

Q: Sean Rhea, UC Berkeley. Is the gain of the one-hop routing significant, since Frank Dabek argues that using proximity routing reduces latency to only two hops across the network?
A: This scheme has a latency of 1.5 hops across the network, and the first hop can benefit from proximity routing.
Q: Franklin, Northwestern. In the experiments, you assume that lifetime is exponentially distributed, but in real networks it appears that time is Pareto distributed instead.
A: Exponential distribution is more extreme, so the results would be even better if we assumed a Pareto distribution.
Q: Chunqiang Tang, University of Rochester. There are two types of traffic in the system: maintenance traffic and lookup traffic. Your scheme reduces lookup traffic but increases maintenance traffic. Shouldn't you be optimizing for the sum of the two instead?
A: Our goal is to reduce lookup latency, not traffic. Also, we expect lookup traffic to be significantly higher than maintenance traffic.
Q: Jonathan, University of Utah. What if slice leader crashes?
A: If a lookup fails, a node generates an event, and all routing tables will get updated so a new node will become the slice leader.

Stefan Savage, Co-Chair, presenting the best student paper award

SECURITY AND BUGS SESSION

Summarized by Vinay M. Igure and Chunqiang Tang

LISTEN AND WHISPER: SECURITY MECHANISMS FOR BGP

Lakshminarayanan Subramanian, Ion Stoica, Randy H. Katz, University of California, Berkeley; Scott Shenker, University of California, Berkeley and ICSI; Volker Roth, Fraunhofer Institute, Germany

This was awarded Best Student Paper. BGP is a path vector protocol that has policy-based routing. It has a control plane and a data plane, and invalid routes in either of these planes could cause widespread damage across the Internet. In the control plane, a router can propagate spurious routes, and in the data plane, routers may inconsistently forward packets. Some of the possible attacks could be black hole attacks, impersonation, and eavesdropping. Lakshmi also provided some real-world examples of attacks that targeted the BGP. In order to deal with these threats, this paper presented a combination of two mechanisms called Listen and Whisper. The approach uses Whisper to deal with the attack at the control plane and Listen to deal with it at the data plane.

Lakshmi compared this new approach with other previously proposed mechanisms, such as a key-based mechanism, centralized databases, configuration-checking tools, and data-plane route-probing tools. Verification using PKI can pinpoint the source of the problem, but this is not possible with Listen and Whisper. Whisper is based on the Chinese whisper game, which has two versions: split-whisper and loop-whisper. A demo of the game was given to easily explain the Whisper route consistency checking mechanism. Due to time limitations, only a brief summary of Listen was presented. The basic idea behind Listen is that if TCP data contains a SYN and DATA packet, it implies that an ACK has been previously received and a valid route exists. The biggest challenge with this approach was to deal with the false positives and false negatives. Listen reduces both of these to less than 1%.

Although these two techniques can be used individually, they offer best performance when used jointly and the results of the two mechanisms are correlated. Lakshmi presented the results of testing the effects of colluding adversaries on the current Internet and compared it with the effects on Whisper implementations. It shows that Whisper policies reduce the percentage of affected ASes.

Measurement and Analysis of Spyware in a University Environment

Stefan Saroiu, Steven D. Gribble, and Henry M. Levy, University of Washington

Steven Gribble presented the results of the study done at the University of Washington to investigate the problem of spyware. The popular media coverage of spyware generally scares people. Politicians have also stepped into the field and are trying to create laws to deal with this problem. Given the privacy and security risks, it is essential to conduct rigorous research into this problem. The biggest problem is defining spyware. For the purpose of this study, spyware was defined as any software that collects personal information and relays it to a third party.

The work focused on studying the prevalence of four major spyware products: Gator, Cydoor, eZula, and SaveNow. The authors created network signatures for each of these by monitoring the HTTP traffic. The university network traffic was sniffed for seven days and then analyzed to find these signatures. It was found that 5.1% of machines were infected with spyware. There were many interesting correlations, but not necessarily causations, between the presence of spyware and the number of Web servers contacted, Web objects downloaded, executables downloaded, and presence of P2P file sharing software. Most P2P software comes with spyware. The study also found that of those PCs infected with spyware, many contained more than one spyware program and most spyware lingered in the machines for extended periods. The work also found security flaws in the "auto-update" feature of eZula and Gator that could be exploited to launch attacks. Given the widespread deployment of spyware, an attack exploiting these bugs could cause enormous damage.

Steven also noted that it is difficult to implement technical solutions to deal with spyware, primarily because most users choose to install these programs and because it is very hard to distinguish between spyware and non-spyware programs. Legal solutions are also difficult for these reasons. However, detecting the presence of spyware is easy and hence the focus should be on containment and removal.

Model Checking Large Network Protocol Implementations

Madanlal Musuvathi, Dawson R. Engler, Stanford University

Madan introduced the general problems associated with checking network protocols. Conventional testing of protocols is usually inadequate. The approach taken in this paper is to model check the entire protocol implementation. Model checking explores all possible states of the protocol and checks for vulnerabilities at each state. Since the state space is usually large, the checking is done until all the resources are exhausted. Madan used CMC, a C Model Checker, that checks code directly, to test protocols. CMC is similar to a network simulator but provides the ability to checkpoint implementation states. CMC also computes the signature of each state, which can then be compared to determine if they are similar.

The first test of CMC was done on AODV, a routing protocol for mobile ad hoc networks, and they were successful at finding bugs in the implementation code. The next step was to scale CMC to test a large protocol such as the TCP protocol in a Linux implementation. TCP was chosen because it is a hard protocol and widely tested. Madan highlighted the contributions of the work, which included demonstrating that CMC is applicable for TCP and that the techniques used to check TCP can be used to scale CMC to test other large systems. The work also provides a general infrastructure that can be used for testing large systems. The biggest problem in testing TCP was in isolating the TCP code from the Linux kernel. Since that was not possible, the entire kernel was ported into CMC with well-defined interfaces to run TCP code.

The techniques that were used to deal with exploring large state spaces – handling large states and incremental state transitions – were discussed. The boot-up state was the initial state, with single-step transitions between states. Since the states were large and transitions slow, a "copy-on-write" mechanism was used. In order to deal with the large states and their increments, states were split into objects and made to share unmodified objects. To handle heap canonicalization, Madan presented an incremental heap canonicalization algorithm, discussing the performance improvements achieved. The CMC checked for generic correctness properties such as memory leaks and for protocol conformance. The TCP RFC was translated to C. They found four bugs in the code. Some of the future work includes using this approach to test the entire Linux kernel.

RESOURCE MANAGEMENT SESSION

Summarized by Laura Grit and Piyush Shivam

Constructing Services with Interposable Virtual Hardware

Andrew Whitaker, Richard S. Cox, Marianne Shaw, and Steven D. Gribble, University of Washington

Andrew Whitaker presented mu-Denali, an extensible design for constructing virtual machine (VM) services. The authors argue that although VMs are powerful platforms for allowing services, developing services is poor since VMs are closed systems and are difficult to tinker with. They believe that the correct VM service development platform is one with extensible design and a clean pro-

gramming API. Virtual machine monitors have a "whole-system perspective," which makes possible a variety of interesting services like migration, intrusion detection, and replay logging. The key ideas in their mu-Denali architecture are the interposition of events generated in one VM's virtual hardware device by another, the extension of the virtual architecture of the VM, and, finally, a clean programmable API that lets one build the VM services.

The experimental evaluation of the system addressed the overhead of interposition and mu-Denali's ability to support the development of VM services. It was observed that system call and packet-intensive operations suffer degradation in performance because of extra trapping, copying, and so on. However, the overall application throughput numbers were acceptable and the development effort minimal.

There were questions related to Apache migration, its network connections, and the benefits of restarting VMs instead of application. Andrew acknowledged the limitation of the LAN-based implementation (no IP routing) and mentioned that restarting VMs was a part of completeness argument. Jay from Sun Microsystems pointed to some related work in this area, to which Andrew acknowledged that a lot of interposition work is existing in the same space with some differences. The mu-Denali approach gains a lot by using a generic OS. Someone asked if anyone outside of the authors' group had tried to build the services using the primitives, to which Andrew replied no.

SWAP: A Scheduler with Automatic Process Dependency Detection

Haoqiang Zheng and Jason Nieh, Columbia University

Haoqiang stated the motivation for his work by emphasizing the large number of dependencies that exist between the many components in creating tasks. Existing solutions have limitations since they do not address the dependencies not caused by mutexes such as signals, dynamic priorities, or priority inheritance.

Their solution, SWAP, is a simple model that detects dependencies based on resource access history and uses a feedback loop to improve accuracy. In their model, which is transparent to users, everything is treated as a resource (e.g., sockets, file locks, and semaphores) and all requests use system calls, which they use to see what resources have been requested. Past resource providers are predicted as potential current resource providers. Each provider is assigned a confidence level based on system feedback of how they have performed as a resource provider in the past.

The SWAP system uses existing schedulers as black boxes. After the existing scheduler runs, they use the SWAP scheduler with dependency detection to determine what was scheduled. Their system was implemented with a minimal kernel-level change and by adding dependency detection to the system call layer.

In their experiments, they showed how high-priority tasks would benefit from SWAP. When system load was low, SWAP had no effect and did not impose any system overheads. However, when system load was high, SWAP performed much better and increased throughput compared to that in naive Linux.

In response to Steve Gribble's question whether they can catch all dependencies, Haoqiang said that they found all system calls on Linux. For more information, see *http://www.ncl.cs.columbia.edu*.

Contract-Based Load Management in Federated Distributed Systems

Magdalena Balazinska, Hari Balakrishnan, and Mike Stonebraker, MIT Computer Science and Artificial Intelligence Laboratory

Magdalena presented her load management system to provide mechanisms for handling peak system load without overprovisioning and with wide area distribution of load in such federated systems. The system tries to achieve the challenges of having sufficient incentives to participate, efficiency of mechanisms, and customization options for participants. In this system, pair-wise contracts are negotiated between participants offline to specify a fixed price per unit of load. The system goal is an acceptable (not optimal) allocation of resources: Participants compute marginal costs for current loads and compare this to contract prices to determine if it is better to transfer or keep load. The fixed prices are chosen around the gradient of the cost function. Magdalena pointed out that fixed price constraints can cause unacceptable allocations, and this can be rectified by using a small price range, which allows load to propagate through a chain of identical contracts. Moreover, since the price contracts are done offline, they can be greatly customized, which enables service discrimination. The implementation of these mechanisms was presented in Medusa to show that this approach is simple, feasible, efficient, and customizable.

When asked how this approach can gain widespread use, she responded that the system is user-friendly and that by having contracts offline, payments can be specifically customized to fit the participant's needs. Simulations included a thousand participants, each with only a few contracts, and they found acceptable allocations, so Magdalena felt it was a scalable approach. She also felt that although oscillations of load were rare,

this technique was designed to deal with occasional load spikes. For more information, see *http://nms.csail.mit.edu/projects/medusa*.

DHT APPLICATIONS

Summarized by Magdalena Balazinska, Chip Killian, and Praveen Yalagandula

Hybrid Global-Local Indexing for Efficient Peer-to-Peer Information Retrieval

Chunqiang Tang and Sandhya Dwarkadas, University of Rochester

This paper works to solve the problem of full-text search in peer-to-peer networks, in contrast to the traditional keyword search. The approach is to leverage DHTs to help deal with the information explosion problem. This is done by creating inverted lists by keyword and storing both the inverted lists and the complete word lists on each node that the keyword maps to.

By storing the index in this fashion, you can locate any document by keyword by searching a single node in the DHT. And if you are performing a query that deals with more than a single-word search, you can still perform the query with just one computer, because you can use the word list to complete the query.

To improve upon these results, however, and to reduce the storage cost, they further focused on weighted keywords. In this fashion, they extracted the top X keywords from each document, where added weight is given to keywords that appear frequently in this document, but seldom in other documents. Two other optimizations include rebalancing the load in the system by mapping a keyword to a range of hash addresses instead of a single address, and using automatic query expansion, which maps terms to other likely interesting candidates to return additional relevant pages.

Q: David Oppenheimer, UC Berkeley. New nodes join a system at appropriate places in the ID space. Why did you make this assumption? How often do you assume that nodes are joining?
A: We only assume a computer joins once.
Q: Matt Welsh, Harvard. Why does your approach require so much space per node compared to Google?
A: We replicate word lists, too. Compared to Google, we don't have to search every computer.
Q: L., UC Berkeley. What about ranking query results?
A: We only search for computers responsible for keywords.

Untangling the Web from DNS

Michael Walfish, Hari Balakrishnan, MIT Computer Science and Artificial Intelligence Laboratory; Scott Shenker, ICSI

This paper had two fundamental tenets. First, that we should restructure the Web such that Web links contain no identifiers as to whom they can be associated with or where they might be located. This would then make links stable, and when they change physical location or ownership, their URL would not change. The second tenet for the talk was the design of an appropriate lookup service based on the non-identifying IDs.

The first part of the talk argued that Web links as currently implemented suffer from problems when changing locations or owners, from the point of view of technical challenges as well as the idea of vanity domains. The main conclusion is that, no matter what the design of user-friendly-identifiers is, humans will always fight over them, making these systems unsuitable for stable URLs. Instead, by using a flat-space identifier with an update scheme, these two identifiers can be completely decoupled. By doing so, we get: (1) stability, where references remain invariant when objects change; (2) support for object replication, where one identifier can map to a list of replicas; and (3) automatic management of namespace.

The design in the second part of the talk was rather straightforward. The authors used a DHT. To allow updates and manage changes safely, the authors used self-certifying records.

Q: Doug, University of Wisconsin. If I establish a DNS server, it is my responsibility to serve the names. With your scheme, everyone is responsible to serve names.
A: We envision a management model for the infrastructure, but it could be at the level of the individual doing puts and gets.
Q: Jonathan, University of Utah. One way DNS is used is to locally resolve suffixes, and no one knows about these in the outside world.
A: Details about write locality are in the paper.

Democratizing Content Publication with Coral

Michael Freedman, Eric Freudenthal, and David Mazières, New York University

It is well known that flash crowds bring down small Web sites. Existing approaches that deal with flash crowds are too expensive for these small Web sites (content distribution networks, load balanced servers, etc.), while client-side proxying does not provide 100% coverage. Instead, CoralCDN proposes to use volunteers to pool resources and dissipate flash crowds.

CoralCDN is a decentralized, self-organizing, peer-to-peer Web-content distribution network. To use CoralCDN, a content provider must rewrite its URLs into "Coralized" URLs (e.g., *www.x.com* becomes *www.x.com.nyud.net:8090*). This directs unmodified browsers to Coral, which absorbs load.

To provide such service, CoralCDN has two key components. First, Coral DNS servers map clients to nearby Web proxies, by probing clients to determine any nearby nodes, based on either network

topology hints or proximity measurements. Second, Coral Web proxies enable clients to retrieve a locally cached copy of the data or find a nearby neighbor that has the data, without needing to contact the origin Web server.

To enable the above functionality, Coral provides the abstraction of a distributed index built on a key-based routing layer. Coral nodes organize themselves into a hierarchy of clusters based on RTT distances between nodes, so that requests remain local when possible in order to minimize latency. Values are stored once in each level cluster. To always store a value at the closest node in the identifier space, however, would cause hotspots. Instead, Coral introduces a rate-limiting mechanism, by halting the publishing ("put") operation as soon as it finds a full and loaded node.

More information is available at *http://www.scs.cs.nyu.edu/coral*.

Q: Student from UC San Diego. What would be the motivation for people to install Coral?
A There are already volunteers that mirror Slashdot. Organizations can benefit by improving local client latency, reducing downstream bandwidth usage, and offering locally trusted proxies to provide security benefits to their clients.
Q: Steve, University of Washington. What are possible abuses of Coral?
A: There are several challenges. One is that nodes could overwhelm each other with requests. We could employ rate-limiting mechanisms at the application layer. Additionally, a node could donate only upstream bandwidth to prevent distant clients from replacing their local cache. Lastly, there are integrity concerns, but we could leverage content-hashes in URLs that can be verified by the client's proxy, which is the incentive for running local proxies.
Q: Rama, Cornell. What is the latency you get when a flash crowd occurs?
A: Please refer to the graph in the paper. Median latency is approximately 100ms for a hierarchical system on PlanetLab.

OVERLAY NETWORKS SESSION

Summarized by Suchita Kaundin and Aydan Yumerefendi

Operating System Support for Planetary Scale Network Services

Andy Bavier, Scott Karlin, Steve Muir, Larry Peterson, Tammo Spalink, and Mike Wawrzoniak, Princeton University; Mic Bowman, Brent Chun, and Timothy Roscoe, Intel Research; David Culler, University of California, Berkeley

Timothy Roscoe described the main design principles and organization of PlanetLab – an open, globally distributed platform for developing, deploying, and accessing planetary-scale network services. The challenge behind PlanetLab is to build a new distributed system with the existing technologies, which should be able to evolve under the simultaneous development of a large research community. The main design goal is to implement a minimal set of abstractions and interfaces to enable service builders to construct powerful applications, yet protect and isolate competing and co-existing services. PlanetLab also supports unbundled management – independent evolution of its abstractions from the operating system running on each node.

PlanetLab consists of a collection of geographically distributed physical nodes. Each node can host a number of virtual machines (VM) that interface with the underlying hardware using a common VM monitor. A set of distributed VMs forms a slice, which is the basic unit of resource allocation and isolation; each service in PlanetLab runs within a slice. Within each system node, two important VMs provide a minimal set of administrative privileges, the node manager manages and monitors the VMs running on the node, and the local manager is a privileged VM that allocates local resources to a VM.

PlanetLab is a work in progress, and only time will tell which infrastructure will evolve to give it fuller definition. The design allows network services to run in a slice of PlanetLab global resources with the PlanetLab OS, providing only local abstractions and as much global functionality as possible. In the future, its designers expect to limit the functionality implemented in privileged services and allow service builders to implement as much low-level functionality as possible.

MACEDON: Methodology for Automatically Creating, Evaluating, and Designing Overlay Networks

Adolfo Rodriguez, Charles Killian, Sooraj Bhat, and Dejan Kostic, Duke University; Amin Vahdat, University of California, San Diego

MACEDON is an infrastructure for building, deploying, and evaluating large-scale distributed systems. It automates a large number of commonly performed routine tasks in the process of developing and evaluating large-scale systems. MACEDON is general, and shows high performance, and its building blocks can be easily replaced. For novices, the infrastructure is easy to use, yet it does not limit experts from customizing the system to their specific needs.

The main idea of MACEDON is to separate specification from implementation. To develop a system using MACEDON, system developers use a C++-like domain-specific language to define the specific algorithm used in their system and make use of the API exported by MACEDON to automatically perform tasks related to failure detection, timers, transports, bloom filters, etc. MACEDON parses the specification and translates it into executable code that uses library functions and the MACEDON

code engine. The engine and code libraries are common to all systems and thereby increase evaluation consistency and code reuse. The final code can be either emulated or run live on the Internet.

MACEDON uses a finite state machine approach to represent the state of each node in a distributed system. In this model, events such as message reception, scheduled completion of timers, and application commands trigger actions. Actions include setting local node state, transmitting new messages, scheduling timers, and delivering data. In addition, events may cause the protocol to move from one system state to another.

MACEDON's approach has been validated by implementing a large number of overlay protocols and comparing their performance to the official published results. Each implementation is smaller than 600 lines of MACEDON code and shows performance comparable to its original custom implementation containing thousands of lines. One question left unanswered by the current implementation is about abstraction fairness, i.e., whether some algorithms get better treatment than others.

Structure Management for Scalable Overlay Service Construction

Kai Shen, University of Rochester

Kai Shen described Saxons, a service-independent distributed software layer that dynamically maintains a selected set of overlay links for a group of nodes. Saxons is designed to provide efficient overlay connectivity support that can assist the construction of large-scale Internet overlay services. This new software layer maintains high-quality overlay structures with three performance objectives: low path latency, low hop-count distance, and high path bandwidth. At the same time, it is scalable, introduces low overhead, and demonstrates stable behavior.

Saxons achieves its design goals by controlling the management overhead introduced by probing and maintaining the structure. In this system the per-node management cost depends only on the number of attached links, not on the overlay size. Saxons uses a randomized membership protocol to avoid maintaining and exchanging complete membership information. Randomization is used to maintain and exchange membership information as well as to discover nearby hosts. Each node in the overlay has a limit on the number of overlay links it can establish and accept.

To maintain the quality of the structure, Saxons can make use of three algorithms that either minimize latency, minimize latency on half of the links or choose the remaining half at random, and minimize latency to half and choose the remaining half at random from nodes with high bandwidth. To avoid structural oscillation, Saxons performs link adjustment only if the new topology persists for a period longer than a given threshold.

Saxons was evaluated using simulation and live PlanetLab deployment. The simulations and experiments on 55 PlanetLab sites demonstrate Saxons' structural quality and the performance of Saxons-based service construction. It is believed that it is more feasible for sharing low-level activities such as link property measurements. Further investigation on this issue is needed.

RELIABILITY SESSION

Summarized by Aydan Yumerefendi and Suchita Kaundin

Session State: Beyond Soft State

Benjamin C. Ling, Emre Kiciman, and Armando Fox, Stanford University

Session state is an essential part of any dynamic Web application: It stores important per client information for the duration of a client's session. Benjamin presented SSM, a system that decouples the management of session state into a separate dynamic layer. As a result of its design, it decreases management overhead and provides efficient recovery and load balancing.

SSM makes use of a secondary storage hashtable. The table consists of a number of bricks, which store per-client session data. Web servers use a client-side library to communicate with the storage bricks. Writes in SSM avoid two-phase commit by selecting a random subset of bricks to write to and waiting until only a fraction of them acknowledge the write. SSM does not make use of an index to locate data. Instead, the Web server prepares an HTTP cookie with information about the location of the client's session state and sends it back to the client. When reading data, the Web server makes use of this cookie to request information from the specified bricks.

SSM tolerates failures gracefully: it preserves the availability of data during failure and recovery. Brick recovery is achieved for free by restarting the brick. Client usage patterns cause data to be rewritten and ensure sufficient availability even in the presence of failures. SSM makes use of admission control to balance the load. Failure detection in SSM is based on Pinpoint, a framework for detecting likely failures. Pinpoint monitors a set of brick parameters and considers as failed any brick with parameters that deviate from the statistical averages.

The evaluation of SSM shows that it preserves throughput, improves latency, and enforces high availability. Benjamin also mentioned that there has been interest in SSM from the industry.

Path-Based Failure and Evolution Management

Mike Y. Chen, Dave Patterson, and Eric Brewer, University of California, Berkeley; Anthony Accardi, Tellme; Emre Kiciman, Armando Fox, Stanford University

Chen started by emphasizing the importance of fast recovery and rapid online evolution. Fast recovery is important in preserving availability, and rapid online evolution is necessary for the continuous software updates of large-scale systems. To facilitate both tasks, Mike presented an approach that monitors the path of each request in a system, aggregates information, analyzes the behavior of individual components, and quickly detects faulty nodes and recovers them to a stable state.

The key idea of this approach is to separate observation from analysis and use statistical techniques to achieve its goals. Each request in the system carries a unique identifier and a timestamp. The system records the components that the request visits, aggregates the different observations, and tries to draw statistical conclusions using machine learning techniques. The C4.5 decision tree learning algorithm used in the project can tolerate noisy data and helps reliably identify problematic components and system states. It also supports application evolution management by tracking component dependencies.

There are three path-based macro-analysis implementations of the above ideas: Pinpoint, ObsLogs (used by Tellme Networks), and SuperCAL (used by eBay). They all show good performance and help achieve fast recovery and rapid evolution management. The total overhead of monitoring and analysis is within 1 to 3% on tests performed on eBay. The future plans for the project involve extending its approach to wide area systems and multiple administrative domains.

Consistent and Automatic Replica Regeneration

Haifeng Yu, Intel Research Pittsburgh and Carnegie Mellon University; Amin Vahdat, University of California, San Diego

Replication improves availability of large-scale systems but also presents significant management overhead due to the time and effort spent to preserve the consistency of a replicated system. Haifeng Yu presented Om, the first peer-to-peer wide-area read/write storage system that improves the manageability of large-scale distributed systems through online automated replica regeneration while preserving system consistency. These properties are achieved through three novel techniques: (1) single-replica regeneration helps achieve high availability with a small number of replicas; (2) failure-free reconfigurations allow fast reconfiguration within a single round; and (3) a lease graph and two-phase write protocol serve to avoid expensive consensus.

Om uses Pastry, a distributed hashtable, to construct an overlay network and route messages. Om stores each data object on a number of replica nodes. Reads for an object go to any of the replicas, while writes involve the whole replica group. When failures occur, it is important that all nodes have the same system configuration. Instead of using distributed majority quorum to achieve this goal, Om uses the witness model. The witness model allows quorums as small as a single node, by choosing random system nodes and using their view of the system and the intuitive fact that the different views do not diverge significantly. Om organizes witnesses into a matrix, and each replica tries to coordinate with one witness from each row. The witness model requires at least one common witness for a replica group to achieve unique configuration. Multiple rounds can help address the problem of non-intersection; the expected number of such rounds is 3.1.

The evaluation of Om involved LAN emulation and live PlanetLab deployment. In all experiments the system showed high performance and availability. Regeneration completes in approximately 20 seconds, availability can be as high as 99.9999% with only four replicas, and the probability of incorrect configuration is 0.000001 and is independent of system size.

STORAGE SYSTEMS SESSION

Summarized by Piyush Shivam and Ramakrishna Kotla

Total Recall: System Support for Automated Availability Management

Ranjita Bhagwan, Kiran Tati, Yu-Chung Cheng, Stefan Savage, and Geoffrey M. Voelker, University of California, San Diego

Ranjita began by presenting the definition of availability and indicating that redundancy provides availability. She then outlined two fundamental questions: (1) How can one quantify the availability of systems? (2) What is the exact relationship between redundancy and availability? Total Recall tries to address these questions by providing redundancy management knobs for automatic maintenance of availability metrics. This is done using mechanisms such as availability prediction, redundancy management, and dynamic repair to meet required availability goals.

Ranjita went on to describe in more detail what these mechanisms are and how they interact with each other. Given the target levels of availability and current system availability, a policy module

decides whether to use replication or erasure coding for redundancy and whether to use eager or lazy repair for dynamic repair. This information is then fed to the redundancy management module, which predicts the degree of redundancy as a function of the availability, coding, and repair scheme. Ranjita then gave an experimental evaluation of the system for which she used a prototype implementation incorporating the ideas outlined earlier.

During the discussion, someone asked if the availability of the master node is higher than other nodes. Ranjita replied that availability of all nodes is the same and if the master fails, another node takes over as master. Concerns over the impact of extra redundancy over consistency were raised, and she acknowledged that as an important issue and mentioned it as something for future work. She was asked why the bandwidth numbers in the graph were so high and replied that it is because of the way the simulation was done: they are just the upper bounds; the actual bandwidth is much less. Finally, she was asked if it really makes sense to store data on unreliable components. She answered yes, it does make sense by using the mechanisms outlined in the talk. For further information, please refer to *http://ramp.ucsd.edu/projects/recall/*.

TimeLine: A High Performance Archive for a Distributed Object Store

Chuang-Hue Moh and Barbara Liskov, MIT Computer Science and Artificial Intelligence Laboratory

Barbara began by introducing the idea of a timeline service and what it does. The key idea is to take on-demand snapshots of data at a user-specified time and run computations on data sometime in past. This enables time travel for computation. However, their systems allow read-only computations on the snapshots and do not allow modifications to the state stored in the snapshot.

This system is built in a client-server setup using the Thor environment, which provides support for persistent object store. The users run computations at the client machines, and the persistent state resides at the servers.

She described several approaches for building such a system, with their problems, and then described the TimeLine approach, in which each snapshot has a time which tells what modifications have been made up to that point. The association of time with the modification is done using the idea of sender time, receiver time and Lamport's logical clocks. Finally, she presented the implementation of this system, which met the goals of not disrupting the rest of system while taking snapshots and of avoiding penalty for taking snapshots.

In the question-answer session, someone asked if it is possible to provide read and write computations on the snapshots. Barbara replied that it is possible but one will get many versions of system state at a given time and the overall system will become very complex. The second question was whether clients have to participate in the snapshots, to which Barbara replied yes, and mentioned that in general things need to move through the client. Another question was what happens to writes in the system when Thor gets replaced by a general file server? Barbara replied that the kind of optimization with respect to writes (e.g., delayed writes or write gathering) depends on the semantics of storage system and file system. The current implementation takes advantages of semantics provided by Thor.

Explicit Control in the Batch-Aware Distributed File System

John Bent, Douglas Thain, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, and Miron Livny, University of Wisconsin, Madison

John Bent began by motivating the central question behind this work – harnessing remote storage for batch workloads. The key questions addressed in this work are: (1) Do the batch computing workloads offer new challenges to the distributed computing infrastructure? (2) Is the current infrastructure okay for such a need, and, if not, what is the appropriate architecture? Having motivated the work, he went on to describe the batch computing environment, batch workloads, and the I/O characteristics of the same. In such environments the key challenge is to manage the flow of data into, within, and out of the system. The current distributed file systems are not practical for such applications, because they make uninformed decisions about caching, consistency, and replication. On the other hand, the Batch-Aware Distributed File System (BAD-FS) exposes this knowledge and removes the guesswork. In addition, it is practical and deployable, because it is packaged as a generic batch system.

BAD-FS makes intelligent scheduling decisions by using the remote cluster knowledge of storage availability and failure rates, and workload knowledge regarding data type, data quality, and job dependencies. John went on to explain the performance enhancement techniques used by BAD-FS, like I/O scoping (eliminating unnecessary I/O) and capacity-aware scheduling, which makes the system appealing for batch workloads. He went on to describe a simplified implementation of this system and the scientific workloads that were analyzed/run under BAD-FS.

In the discussion, someone asked if BAD-FS is burdening the user to find

out the extra knowledge which the BAD-FS scheduler uses to make intelligent decisions. John replied that the ultimate goal is to automate this process, and in fact the BAD-FS architecture reduces user burden for organizing scattered data. Another question was whether the intermediate data between different stages of a job needs to be maintained, since it is uncommonly an input to the next stage. John answered that they have not looked at that, but there are certain applications that overwrite the output data, and hence it becomes important to maintain it. For more information on this work, please refer to *http://www.cs.wisc.edu/adsl*.

This issue reports on the First Symposium on Networked Systems Design and Implementation (NSDI '04) and on the 3rd USENIX Conference on File and Storage Technologies (FAST '04).

For NSDI '04: Our thanks to Amin Vadhat, who shepherded the following summarizers:

Magdalena Balazinska
Laura Grit
Vinay M. Igure
Suchita Kaundin
Chip Killian
Ramakrishna Kotla
Xun Luo
Vinay Mallikarjun
Piyush Shivam
Chunqiang Tang
Praveen Yalagandula
Aydan Yumerefendi

For FAST '04: Thanks to Ismail Ari and his summarizers:

Michael Abd-el-Malek
Nitin Agrawal
Akshat Aranya
Dean Hildebrand
Andrew Klosterman
Xun Luo
Kiran-Kumar Muniswamy-Reddy
Steve Schlosser
Shafeeq Sinnamohideen
Deepa Tuteja
Wenguang Wang
Lan Xue
Aydan Yumerefendi

Note: The reports on BSDCon '03, held in San Mateo, California, September 8–12, 2003, can be found at *http://www.usenix.org/events/bsdcon03/confrpts.pdf*

3rd USENIX Conference on File and Storage Technologies (FAST '04)

March 31–April 3, 2004
San Francisco, California

TECHNICAL SESSIONS

RELIABILITY & AVAILABILITY

Summarized by Kiran-Kumar Muniswamy-Reddy

Best Paper award

Row-Diagonal Parity for Double Disk Failure Correction

Peter Corbett, Bob English, Atul Goel, Tomislav Grcanac, Steven Kleiman, James Leong, and Sunitha Sankar, Network Appliance, Inc.

Peter Corbett described a new algorithm, Row-Diagonal Parity (RDP), for protection against double failures and described its application to RAID.

The RDP algorithm uses a simple parity scheme based on EX-OR operations. Each data block belongs to one row-parity set and one diagonal parity set. In a simple RDP array, there are p + 1 disks. The stripes across the array consist of one block from each disk. In each stripe, one block holds diagonal parity, one block holds row parity, and p – 1 blocks hold data. Every row parity block has an even parity of the data blocks in the row, excluding the diagonal parity block. Every diagonal parity block has an even parity of the data and row parity blocks in the same diagonal. In case of double disk failure, each diagonal misses one disk, and there are two diagonal parity sets that miss only one block. Once we recover one block, we can recover a complete row. Using this, we recover another diagonal, and so on.

In the Q&A session, someone asked why weren't triple (or more) failures considered. The speaker said that double failures are the more common case and that this could be extended for triple failures. Another asked whether the algorithm could really be added to an existing RAID, as the paper claims. The speaker replied that it could be added to an existing RAID.

Improving Storage System Availability with D-GRAID

Muthian Sivathanu, Vijayan Prabhakaran, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, University of Wisconsin, Madison

This paper won one of the two Best Student Paper award.

The talk was given by the primary author, Muthian Sivathanu. Normal RAID works on a simple failure model. The basic premise of this work is that if D or fewer disks fail, RAID or D-GRAID continues to operate with some degraded performance. The disk array becomes completely unavailable once more than D disks fail. D-GRAID degrades gracefully and also recovers quickly in the event of a disk failure.

For graceful degradation, D-GRAID uses two techniques: selective metadata replication and fault-isolated data placement. In selective metadata replication, the naming and system metadata structures of the file system are highly replicated. Thus, failures of a few disks do not render the entire array unavailable. In fault-isolated data placement, semantically related blocks are placed within the array's unit of fault-containment. For example, all the blocks of a file are placed within a disk. This way, semantically meaningful data units are available under failure. Since fault-isolated data placement reduces the parallelism inherent in RAID, they make copies of blocks of "hot" files across the drives of the system.

MEASUREMENT, MODELING, AND MANAGEMENT

POLUS: GROWING STORAGE QOS MANAGEMENT BEYOND A "FOUR-YEAR-OLD KID"

Sandeep Uttamchandani, Kaladhar Voruganti, John Palmer, and David Pease, IBM Almaden Research Center; Sudarshan Srinivasan, University of Illinios at Urbana-Champaign

Summarized by Nitin Agrawal

Sandeep Uttamchandani described the Polus framework, which addresses the QoS goal transformation problem in the context of policy-based storage management.

In spite of prior research and standardization, the problem of mapping high-level QoS goals to low-level storage device actions still yields complex and error-prone processes. Polus generates this mapping by using a combination of rule-of-thumb specifications, a reasoning engine, and a learning engine. Currently, policies are specified as a collection of rules in <event, condition, action> format, and the management module simply invokes the appropriate rule. The problems with this approach are complexity (the level of detail required in generating the rules) and brittleness with respect to changing system configurations and workloads.

Polus performs the balancing act by requiring only high-level specifications from the administrator and using machine learning and pre-packaged procedures for the rest. The paradigm essentially consists of three parts: taking input from toolkit user, quantifying the relations using learning functions, and base strategies.

The work illustrates how Polus can provide storage management guidance to a simulated Storage Area Network file system. A quantitative comparison of Polus with rule-based systems is done for different system states. A future direction of work is to handle incomplete and even incorrect specifications.

During the Q&A, someone pointed out that the decision of when to perform prefetching, which was described as a candidate for a Polus-based learning framework, would be good only if it is taken quickly. Sandeep's response was that they used the prefetching example only as an illustration. Managing a real system using the Polus paradigm is currently under implementation. In summary, the paper proposes Polus as a conceptual stepping stone in the direction of "non-rule"-based approaches for automated system management.

In normal RAID, recovery is slow, as all the blocks in a disk must be recovered. D-GRAID makes recovery fast by recovering only the "live" file system data, i.e., data that is currently in use by processes.

BUTTRESS: A TOOLKIT FOR FLEXIBLE AND HIGH FIDELITY I/O BENCHMARKING

Eric Anderson, Mahesh Kallahalla, Mustafa Uysal, and Ram Swaminathan, Hewlett-Packard Laboratories

Summarized by Steve Schlosser

Mustafa Uysal presented Buttress, a flexible, highly accurate I/O generation tool. Buttress can be configured both as a synthetic workload generator and as a trace replayer. Reproducing very heavy I/O workloads with high fidelity requires a great deal of care, especially to correctly handle bursts of traffic, which exist in many workloads. The authors contend that getting this right requires I/Os to be issued within 100 microseconds of their intended arrival time. Another goal of Buttress is to be as portable as possible and not to require modifications to the host operating system. With this goal in mind, Buttress is implemented as a user-level program, working with standard pthreads on both Linux and HP-UX. It requires a multiprocessor machine to produce the desired I/O loads of up to 100,000 I/Os per second.

In order to generate a workload with very high fidelity, Buttress has to deal with many issues. First, Buttress has to minimize latency to shared data structures by reducing contention at high load. It does this by minimizing the number of lock operations, minimizing critical section time, and using bypass locking. Second, Buttress must minimize the impact of unpredictable OS behavior owing to unpredictable system-call latencies and preemption due to interrupts. It must deal with multiprocessor clock skew by recalibrating the clock when a thread switches CPUs. To handle timing variations on thread wakeup, threads are allowed to pre-spin before they are to issue events. Lastly, it uses a single low-priority spinning thread to keep track of time.

To evaluate Buttress, they compared its performance to two simple trace replay methods. The first uses existing OS syscalls like Select and Sleep to handle timing, and the second uses a dedicated thread to spin and keep track of time. Using a range of workload traces, they showed that Buttress issues the majority of requests within 100 microseconds of their intended issue time and handles bursts better than the simple schemes.

One questioner in the Q&A session wondered if Buttress verified data that was written to the disks using checksums. Mustafa answered that in this methodology the data that is actually stored is not important, just the I/O requests themselves, and that Buttress does not write any real data to the disks. Another asked whether there is value in extending Buttress to operate at the file system level rather than the block level. Another asked if using realtime extensions to Linux or HP-UX would help the simple replay methods that just use system calls. Mustafa said they tried that

with realtime HP-UX but that the results still weren't as good as with Buttress. Another asked about the intuition behind why such accuracy is necessary. Mustafa said there were two factors: naive approaches end up issuing I/Os out of order, and handling bursts correctly requires higher accuracy than one would think.

Designing for Disasters

Kimberley Keeton, Cipriano Santos, Dirk Beyer, Jeffery Chase, and John Wilkes, Hewlett-Packard Laboratories

Summarized by Steve Schlosser

Keeton presented a solver to automatically design basic dependability solutions for large storage systems based on a customer's workload, recovery requirements, and the penalties of outages in terms of dollars lost. Using these parameters, and a few assumptions about the types of recovery mechanisms that are available, the system formulates a mixed integer program and uses an existing solver, CPLEX, to find a solution.

The system uses simple models of the available hardware (i.e., hosts, storage area networks, and disk arrays) and of the available data protection mechanisms (i.e., remote mirroring, tape backup). Remote mirrors can use synchronous or asynchronous updates, with or without batching. Tape backup solutions can be kept locally or off-site. A system can use the secondary copy either for failover or to reconstruct the primary copy. Lastly, the secondary copy can be kept hot or unconfigured, and can be dedicated or shared.

Given all of these alternative configurations and the customer's requirements, the goal of the system is to find the best configuration to meet the customer's needs.

The key is to express the user's dependability requirements in terms of monetary penalties (dollars per hour). First is the data-outage penalty rate, which is the penalty until the system comes back up. Second is the data-loss penalty rate, which is how much recently written data is lost when the system goes down. Given these penalties and the customer's workload requirements, the system can find the optimal solution.

They evaluated the system using the Cello 2002 file system trace from HP Labs, looking at the average bandwidth multiplier for bursts of activity and at the batch update rate. They used a number of different scenarios ranging from a university IT department backing up student directories to a large financial institution with regulatory requirements for backup and availability. These scenarios ranged from cases in which data loss was not so important to those for which data loss would be a business catastrophe. The examples illustrated the choices of dependability solutions that were appropriate for those scenarios.

During Q&A, a questioner wanted to know if the models took into account the solution's performance requirements as well as dependability requirements. Keeton said that they only consider update rates as an element of the solver, but that other issues of performance are an area of future work. She also mentioned that outlay costs are annualized and depreciated over three years with one failure per year. Another questioner mentioned that financial penalty is not linear and should be asymptotic. Keeton agreed that this is an important extension to this work. The last questioner asked if this method can be used to extend existing systems. Keeton said, yes, the system can do this.

KEYNOTE ADDRESS

Scaling File Service Up and Out

Garth Gibson, Panasas, Inc., and Carnegie Mellon University

Summarized by Dean Hildebrand

Garth Gibson discussed the need for improvements in file access and several current methods for doing so. He believes that quality expectation drives technology, with the highest level of quality being demanded by the Tri-labs and the oil and gas industry, requiring throughput of 1GB/s and 1.2GB/s, respectively. For technology to improve and breach the chasm, the need for such performance must be demanded by the masses. This includes the need for both file and file aggregate bandwidth improvements.

Current improvements started with the evolution from monolithic supercomputers to Linux clusters, giving a cost improvement from $100 million/Tflop to $1 million/Tflop. Garth then discussed several phases of file access improvements that have occurred since this development:

- Scaling up the NAS File Server
  To eliminate the single file-server bottleneck, the file system is partitioned among several file servers. Limitations: Cannot increase the aggregate throughput of a single file/directory.
- Partitioning is a manual process and quickly becomes non-optimal. Backup consistency issues.
- Scaling out phase 1: client data cache. Exploits client data cache, i.e., NFSv4 delegations. Limitations: Workloads can exceed client cache size.
- Scaling out phase 2: forwarding servers. Binds many file servers into a single system image. Forwards requests from accessed file server to the data's home file server. File system still partitioned, with each file

server providing global data access. Limitations: Data not directly connected to accessed file server must travel through two file servers.

- Scaling out phase 3: any server will do. All file servers have access to all storage (not indirectly through another file server). Limitations: Throughput for a single file is not increased.
- Scaling out pPhase 4a: asymmetric out-of-band. Clients obtain file layout map from a metadata server and access storage devices directly. Limited by network instead of CPU. Examples: SanFS, EMC High Road, SGI CXFS, Panasas, etc.
- Scaling out phase 4b: symmetric out-of-band. Client cluster is file server. Examples: RedHat GFS, IBM GPFS, Sun QFS, etc.

The description of file service scaling improvements concluded with a statement that Phase 4 is the most advanced architecture currently available. Panasas has combined the architecture of Phase 4a with object-based storage, an evolutionary improvement to standard SCSI. Each object is a container of related data, offloading most data path work from server to an intelligent device. Garth explained that the features of object storage he likes the most are metadata encapsulation, extensible attributes, security at device through digital signatures, and a smaller file layout map allowing more caching while providing performance and scalability for a variety of workloads.

Garth concluded with a statement that the most important aspect in improving throughput is utilizing out-of-band file access. Therefore, a pNFS (Parallel NFS) initiative has started to provide a single standard framework for data access, whether it is based upon blocks, objects, or files. This effort will continue to reduce file access cost by sharing client support among all vendors, enabling the file service to scale up and out for use by the general population.

GRABBAG

Diamond: A Storage Architecture for Early Discard in Interactive Search

Larry Huston, Rajiv Wickremesinghe, Intel Research Pittsburgh; Rahul Sukthankar, M. Satyanarayanan, Gregory R. Ganger, and Anastassia Ailamaki, Carnegie Mellon University; Erik Riedel, Seagate Research

Summarized by Aydan Yumerefendi

Larry Huston presented Diamond, an attempt to provide efficient searches over large volumes of data. The key insight behind the system is to move some of the search functionality to the storage servers and to discard irrelevant data as soon as possible. By distributing the search over a number of servers, Diamond speeds up query execution. It also limits the network traffic by sending only relevant data.

Diamond uses Active Storage to execute client-specified queries wrapped in "searchlets." Each searchlet consists of configuration code and a number of predicate filters, which are applied in sequence and determine the useful data. The system also dynamically manages the load among the storage servers and the client machine by using two different algorithms: CPU partitioning delegates work based on the CPU power of each machine, and queue backpressure considers each system stage as a queue and maintains a certain load on each stage.

The implementation of Diamond runs on RedHat Linux 9. Tests of the system use SnapFind, a custom application for performing searches in digital images. The evaluation results show that Diamond performs well and correctly balances the load among all participants. An audience member pointed out that searchlets do not maintain state and, as a result, they limit the system expressiveness. Larry answered the question by saying that lack of state allows for better load distribution, yet gives the system enough expressive power.

More information about this project can be found at *http://info.pittsburgh.intel-research.net/project/diamond/*.

MEMS-Based Storage Devices and Standard Disk Interfaces: A Square Peg in a Round Hole?

Steven W. Schlosser and Gregory R. Ganger, Carnegie Mellon University

Summarized by Andrew Klosterman

Steven Schlosser introduced a novel storage device, the MEMStore, a magnetic storage device that uses X- and Y-axis motion to access desired data. He explained how the settling time of the seeks is dependent on the maximum of settle time in both the X- and Y-axis, and that the X-axis settling time dominates.

In evaluating this new storage technology, Mr. Schlosser compared the interface of current storage technologies (logical block addressing) with various alternatives for the MEMStore that might exploit its unique characteristics. He examined roles (uses of MEMStores in systems) and policies (ways to change systems to specifically use MEMStores). Two tests were used to evaluate the fitness of MEMStores for a given role or policy: a specificity test and a merit test. The specificity test evaluated the fitness of a MEMStore for a particular role or policy. The merit test evaluated whether a change in the abstraction for use of the storage device was warranted.

After presenting some results of tests from his paper, Mr. Schlosser concluded by mentioning that MEMStores can be used as fast disks in systems with programmers using the familiar LBN addressing scheme.

During the Q&A session, Dan Ellard of Harvard asked whether or not the

"busiest disk" from the EMC trace saw the most I/Os or had the worst seeks, because (Dan claimed) sometimes lots of small I/Os tend to be sequential. Mr. Schlosser did not have an answer to the question, but posited that by replacing that disk with a MEMStore, the trace replay improved, thus validating the use of MEMStores in that role. Another question was raised as to the actual performance of the MEMStore research prototypes, but Mr. Schlosser had to inform the audience that no instances of the device he modeled yet exist as fully implemented storage systems.

Additional information may be obtained from *http://www.pdl.cmu.edu/MEMS*.

A Performance Comparison of NFS and iSCSI for IP-Networked Storage

Peter Radkov, Prashant Shenoy, University of Massachusetts; Li Yin, University of California, Berkeley; Pawan Goyal and Prasenjit Sarkar, IBM Almaden Research Center

Summarized by Wenguang Wang and Lan Xue

This paper turned out to be the most controversial paper of the conference. Prasenjit Sarkar presented a performance comparison of two IP-networked storage protocols that allow remote data access. The basic idea is to measure how performance differs when clients access remote data via file system versus IP-based storage area networking (SAN). NFS and iSCSI were used as specific instantiations of file- and block-level access protocols in this experiment.

Using combinations of micro- and macro-benchmarks, this paper provides a thorough comparison between NFS versions 2, 3, and 4 and iSCSI. Individual file and directory operations and overall application performance are measured, with both data-intensive workloads and metadata-intensive workloads, under various scenarios. Also, a decent amount of work has been done to identify the bottleneck of NFS protocol by capturing and analyzing the network message overhead.

The experiment results show that for a single-client environment, NFS and iSCSI have under comparable performance data-intensive workloads, while iSCSI outperforms NFS by a factor of two for metadata-intensive workloads. This work also identifies lack of metadata caching and update aggregation as the two major reasons for NFS degradation.

However, the fairness of the performance comparison between NFS and iSCSI was questioned. The major concerns and comments raised by the audience can be summarized as follows: It's not an apple-to-apple comparison because block-level protocol accesses data directly from storage, which is straightforward and bypasses file system layers overhead, while NFS provides more complex functionality which incurs extra overhead as a consequence. In addition, NFS is being improved and optimized, and the performance of the NFS prototype can differ greatly from an optimized implementation of NFS. Thus, it's still a question whether the results and conclusions of this experiment can be generalized to every NFS implementation.

Optimizing Block Access

A Versatile and User-Oriented Versioning File System

Kiran-Kumar Muniswamy-Reddy, Charles P. Wright, Andrew Himmer, and Erez Zadok, Stony Brook University

Summarized by Michael Abd-el-Malek

Kiran-Kumar Muniswamy-Reddy described Versionfs, a versioning file system that differs from earlier versioning file systems such as Elephant and CVFS in that existing applications do not have to be modified in order to access previous versions. Additionally, multiple versioning policies provide flexibility, and current-version access is optimized.

Whole-file versioning is used, as opposed to block-level versioning, as that is viewed as more user-friendly. Versions are created on close, mmap, or metadata operations (e.g., chmod). Two types of per-file versioning policies are provided: retention (how many versions to keep – number, total space) and storage (how to store versions – full, compress, or sparse). Copy-on-change is used, which is different from copy-on-write because the new data is compared to the old data and a version is created only if they are different. Existing (unmodified) applications can work with versioned files using a preloaded library that overrides the various file system syscalls.

Using the Am-utils benchmarks, the authors show little (1–4%) time overhead when using their versioning file system. Approximately 20% more storage is required for the old versions. However, the Postmark benchmark runs two times slower.

Q: Wenguang Wang. Each directory has more files than before. What's the performance penalty?
A: In a directory with many small files (such as Postmark), we found a difference in chopping down the directory in system time. It depends on the lower-level file system, but, yes, we did find a difference.
Q: Ed Gould. Hard links were not versioned, why? If I have two symbolic links to a file, will operations on each of these symbolic links create a different version?
A: Yes.
Q: Daniel Ellard. You used open/close to create new versions. What about NFSv3 that does not have open/close? Alternatively, if you have more than one thread, how do you create different files?
A: Haven't done this yet; it's a direction for future work.

Q: When two processes with different files are open at the same time, how many versions are created?
A: One on every close; we keep a counter every time a file is opened.
Q: Brent Calaghan. Insertion/deletion of data into the middle of files, how does that work with sparse data mode, where all following blocks are "pushed down" and sparse mode won't be as useful?
A: Yes, that is an issue. There is no way for us to tell if data is pushed down.

Q: Val Henson. Did you have LD_PRELOAD set to anything else before you started this project? Because this is not transparent to the user, since they have to set LD_PRELOAD.
A: Yes that's true, the user has to do this.

Tracefs: A File System to Trace Them All

Akshat Aranya, Charles P. Wright, and Erez Zadok, Stony Brook University

Summarized by Deepa Tuteja

Akshat Aranya described the details of Tracefs, a stackable and portable file system developed for capturing file system traces. The motivation of this work is to evaluate file system performance and help in its development. Security applications, such as monitoring activities, can also be built based on this work. The background study or related work quoted is in the BSD, Sprite, and Roselli's FS tracing studies. The work is based on the design goals of flexibility, performance, convenience, security, privacy, and portability.

Architecture of the system is such that the Tracefs (consisting of a number of tracers) sits in between the VFS layer and the file system to be traced. The main components of the tracer are input filters, assembly drivers, output filters, and output drivers. The input filters determine what to trace based on the user input. The job of the assembly drivers is to convert the traced operations and its parameters into a traced stream format. The output filters perform a series of transformations like encryption/compression. The output is written to a stable storage by the output drivers. The stable storage can be a file or a socket and can be specified by the user. The traces generated are in binary format, which helps in saving space and in parsing.

Anonymization is required to deal with the concerns of security and privacy. But at the same time, some correlation is required for the traces. Here symmetric key encryption methodology is used for anonymization.

Evaluation of the system was done on different sets of input and output filters. The configuration of input filters used were:

- full – to trace all file system operations
- medium – tracing only 40–50% of the operations
- light – tracing approx. 10% of the operations

The various output filters used were none, compression, checksum, encryption, and all.

The Am-utils and the Postmark benchmarks, respectively CPU-intensive and I/O-intensive, were used to test the system. The system did not show much variation for the elapsed time when using different output filters on the Am-utils benchmark. For the Postmark benchmark, there is an overhead over ext3 for elapsed time and the system time.

Using compression filters reduces the size of the traces. The compression ratio achieved is in the range of 8–21. Postmark generates traces 2.5 times faster than Am-utils, which explains the difference in overheads. Also, the file size increases because of the encryption padding required for anonymization.

To conclude, Tracefs gives a systematic approach to tracing and is a general solution that can be applied to various situations. It can be configured and is extensible in the sense that any output or assembly filters can be used with it. It is mainly helpful in file system development and security applications.

Additional information is available at *http://www.fsl.cs.sunysb.edu*.

HyLog: A High-Performance Approach to Managing Disk Layout

Wenguang Wang, Yanping Zhao, and Rick Bunt, University of Saskatchewan

Summarized by Shafeeq Sinnamohideen

The objective of this work is to improve disk I/O performance for servers with multiple concurrent users. Most reads are absorbed by in-memory buffer caches, so writes dominate the disk workload. Write performance is determined by disk performance (seek time and bandwidth), strategy (overwrite or log-structuring), and scheduling. Overwrite overwrites a block's previous contents with its new contents, possibly requiring a seek from the last write. LFS (Log-structured File System) aggregates small writes together and writes them in one log segment, making use of the disk's sequential write bandwidth. LFS, however, was generally believed to perform poorly for random update workloads because of the overhead of cleaning partially used segments (those with data that has been superseded by more recent writes). The authors observe that advances in disk technology, particularly the increasing ratio of sequential bandwidth to positioning time, lead LFS to outperform Overwrite on modern disks, except when disk space utilization is high.

To overcome this, the authors propose HyLog, a hybrid scheme that offers the benefits of both LFS and Overwrite. They observe that most writes go to a

small number of hot pages, while most of the overhead in LFS comes from cleaning cold pages. Thus, HyLog separates the disk into an LFS portion for hot data and an overwrite portion for cold data and directs writes to the appropriate section based on the observed write frequency. On standard benchmarks, such as TPC-C, HyLog picks nearly the optimal hot page percentage, and thus performs similarly to the best of both LFS (at low utilization) and Overwrite (at high utilization)

Bill Morris from Sun asked why a random write, sequential read workload was not evaluated. The response was that it is not common, but represents a worst case.

OPTIMIZING BLOCK ACCESS

ATROPOS: A DISK ARRAY VOLUME MANAGER FOR ORCHESTRATED USE OF DISKS

Jiri Schindler, Steven W. Schlosser, Minglong Shao, Anastassia Ailamaki, and Gregory R. Ganger, Carnegie Mellon University

Summarized by Andrew Klosterman

Jiri Schindler presented a means for extending the logical volume abstraction of disk arrays to accommodate efficient access to two-dimensional data structures. Through an example, he showed how traditional data layouts lead to efficient (streaming sequential) access in only one dimension. With a modification to the data layout, using what he called "quadrangles," more efficient access can be obtained. With quadrangles, column-major access can be performed on entire disk tracks at one time and row-major access is semi-sequential by exploiting disk head, or track, switch times as the disk spins (eliminating rotational latency).

Graphical examples of the quadrangle data layout scheme were presented, along with a graph showing the increased efficiency for accessing two-dimensional data with quadrangles. Experiments showed the performance gains resulting from the more efficient data layout on TPC trace replays. Mr. Schindler concluded by pointing out that by exploiting the physical characteristics of disks and maintaining the LBN programming model, efficient access to two-dimensional data structures was possible.

During the Q&A session, Wenguang Wang of the University of Saskatchewan asked whether switching disk heads instead of disk tracks would improve accesses with quadrangles. Mr. Schindler's response was that the use of head switches or track switches depended on the manner in which the disk sequenced its logical block numbers. Val Henson of Sun Microsystems, asked how the disk and array parameters could be extracted such that quadrangles could be exploited. Mr. Schindler indicated that experimental extraction (e.g., at format time) suffices to obtain the necessary information.

Additional information may be obtained from *http://www.pdl.cmu.edu*, the Parallel Data Laboratory at Carnegie Mellon University.

C-MINER: MINING BLOCK CORRELATIONS IN STORAGE SYSTEMS

Zhenmin Li, Zhifeng Chen, Sudarshan M. Srinivasan, and Yuanyuan Zhou, University of Illinois at Urbana-Champaign

Summarized by Deepa Tuteja

Zhenmin Li described the use of block correlations for improving the effectiveness of storage systems. There have been previous approaches for exploiting file correlations, but these did not scale well enough to be used for block correlations.

There are three possible approaches to obtaining block correlations: the white box approach, used by NASD, the gray box approach, used by SDS, and the black box approach, used by probability graphs and C-Miner. The probability-graph approach works well for finding file correlations, but cannot be effectively used for block correlations due to a scalability problem and multi-block correlation problem. C-Miner gives a practical black box approach that uses data mining techniques. It is based on the "frequent sequence mining" algorithm called CloSpan. The main observation is that the correlated blocks are accessed together in a short period of time. The frequency sub-sequences are a good indication of block correlations.

For an evaluation of the system, a comparison was done among the following approaches: no prefetching, sequence prefetching, correlation-directed prefetching (CDP), and CDP with disk layout. CDP decreases the miss ratio by 24% and reduces average response time by 25%. Another test done to determine the stability of block correlations shows that they are very stable and effective for a long time. There are, however, reasonable time and space overheads associated with it due to data mining.

C-Miner can improve the performance of a storage system and involves reasonable overheads. It can also be used to solve other problems such as network traffic monitoring and intrusion detection.

Additional information is available at *http://carmen.cs.uiuc.edu*.

WORK-IN-PROGRESS REPORTS

Summarized by Andrew Klosterman

SELF-*

Andrew Klosterman, Carnegie Mellon University

The Self-* project is building a brick-based distributed-object store from the ground up with management in mind. Current figures place storage administrative load at about one administrator per 5–10 TB. This figure makes manage-

ment a large part of storage total cost of ownership.

The system is built around a human organizational analogy, with individual storage bricks being self-optimizing "workers," while a hierarchy of "supervisors" decide how to spread data and work across the worker nodes. In the process of building the system, the team plans to get real management experience in order to more effectively target the problem. More information: *http://www.pdl.cmu.edu/SelfStar*.

Spensa: An Adaptive Distributed File System

Douglas Santry, University of Cambridge

The Spensa system is a brick-based cluster system in which all bricks are peers. Each brick in the system will run the Xen hyper-visor, to allow bricks to both store data and do processing. The cluster will thus be able to run databases, genetic searches, Web servers, etc., in virtual machines inside the cluster.

The Xen hyper-visor allows live migration of virtual machines, which permits a variety of optimization decisions. For example, migration can be used to load-balance or to bring data close to the corresponding computation. In addition, they envision that the system will be able to snapshot virtual machines, allowing the system to restart a previous snapshot if a virtual machine crashes.

MRAMFS: A File System for Non-Volatile RAM Using Inode Compression

Nate Edel, University of California, Santa Cruz

Magnetic RAM (or MRAM) is a new type of non-volatile RAM. The project studies ways to include MRAM in file systems. Because MRAM is very expensive compared to disk space, they use compression to conserve space usage. The system puts small files in MRAM in order to improve performance. They compress inode data using Gamma compression and file data on a block-by-block basis. Gamma compression takes a 128-byte inode down to around 15–20 bytes.

They have a prototype implementation that only compresses inode data, but will eventually implement data compression as well. Evaluation shows that the current prototype performs comparably to ext2 running in a RAM disk. This work is the starting research for another project called the Linking File System.

AVFS: An On-Access Anti-Virus File System

Charles Wright, Stony Brook University

AVFS is a virus-scanning file system, which uses a stackable file system approach to layer virus checking onto an existing file system. Most current virus scanners operate on points such as close or exec. AVFS scans for virus signatures on every read and write.

The system operates in two modes. In immediate mode, when a virus is written the block is discarded; when it is read the file is quarantined. However, this allows a virus to evade detection by writing itself in several different sessions. To address this problem the system also has a forensic mode, in which it versions files on the first writes, and does not return an error until a virus signature is detected in the file. The system uses a variant of the ClamAV OSS scanner, which uses automation for pattern finding. However, ClamAV is slow in the number of patterns that are searched. Thus, the AVFS team modified ClamAV into Oyster, which is faster than the original scanner.

Limiting Liability in a Federally Compliant File System

Zachary Peterson, Johns Hopkins University

Congress has enacted a variety of data maintenance acts and regulations to enhance corporate accountability. Most of these acts require that storage keep an audit trail by keeping versions of data. However, companies want to limit their liability within compliance to these acts.

This work focuses on how to securely delete data in a file system that keeps such an audit trail. Current secure deletion technology either requires multiple overwrites to remove magnetic signatures, or keeping data encrypted and securely deleting the key. Both of these techniques are difficult to perform in a versioning file system.

This work proposes keeping data encrypted using a small (128-bit) key kept in the file inode. This allows the user to securely delete the file by securely deleting this key. The current progress in the project is a modified ext3 file system that performs versioning (but does not implement this secure deletion technology). The versioning ext3 file system can be found at *http://www.ext3cow.com*.

Modeling Stateful Network File System Protocols Using Hidden Markov Models

R.J. Honicky, UC Berkeley and Network Appliance

One difficulty in building synthetic file system workloads is that operation orders is very important. For example, read and write operations to a file must be performed after an open to that file.

This work proposes the use of hidden Markov models (HMMs) to model file system workloads. HMMs have been used extensively in fields such as speech recognition. The group uses a vanilla HMM learning algorithm to generate an HMM for a trace, along with some methods to try to avoid local minima.

They have performed some initial work at validating the results. The operation counts, and the operation pairing counts, are similar to the original trace,

and hand inspection seems to show the traces to be similar.

FileBench

Patrick McDougall, Sun Microsystems

The team asserts that file system benchmarks are used mainly in two ways. They are used by vendors to classify and characterize products, and they are used by designers to set design goals. For both of these tasks the benchmarks must represent applications and not simply I/O.

For example, in cases the group has encountered, a database ran much more slowly than corresponding benchmarks would have suggested. They determined that it was due to slow performance in a logging thread, which gated the performance of the whole system. None of the existing benchmarks revealed this critical dependency.

They are working on a new file system benchmark that incorporates a variety of improvements. For example, they hope to make it modular, to include file system aging, to make it scalable by throughput, users, data set, and clients.

There are a variety of techniques for benchmarks, each with a set of drawbacks. Micro-benchmarks have limited coverage, trace replay can lose dependencies, and model-based approaches can lose important details. They plan to make a model of I/O generation and dependencies for individual threads, to attempt to avoid these problems.

File Attribute-Based Predictions and Optimizations

Daniel Ellard, Harvard

This group's studies have found that there is a strong association between create time attributes of a file and the operations performed on the file during its lifetime. They have implemented a system that builds a model of these associations. It is small and can be put into the kernel.

They are currently exploring ways in which these associations can be used to improve file layout, caching, and replication policies. They also speculate that this information could be used to make global tuning decisions and identify common idioms (such as lock files).

More information on the work can be found at *http://www.eecs.harvard.edu/sos* or *http://www.pdl.cmu.edu*.

TBBT: A Trace-Based File System Benchmarking Tool

Ningning Zhu, Stony Brook University

This work focuses on building an NFS trace playback tool. It is very hard to capture the important attributes of a real workload in a model, which makes trace replay important in a variety of situations. A variety of traces are available to the research community, and the number will continue to increase. This makes a replayer an important tool.

The playback tool involves a variety of challenges. For example, creating the initial file system image is difficult, as is file system aging, concurrency, error handling, and disk and CPU perturbation by the tool itself.

The replayer should be available to the community in two to three months.

Reliable Parallel File Transfer

Lu Zhao, Bowling Green State University

A variety of methods exist to do file transfer, such as FTP, HTTP, bitTorrrent, grid FTP, and parallel FTP. This group hopes to build a protocol that captures both reliability and parallelism.

In the proposed protocol, clients request file info from a main server. This main server knows the location of other storage nodes containing pieces of the data. This node tells these other nodes to send data to the client. The data is protected with checksums; if a checksum fails, the client can re-request data from another server.

Can Replicas Converge Across Network Partitions

Brent Byuonghoon Kang, University of California, Berkeley

Many systems use optimistic replication, where updates go to a single replica and are lazily propagated to other replicas. Previous work uses a version vector with an entry for each node; when combined, the vector takes the highest values of the previous vectors, plus an update for the combining itself.

However, with a network partition, the vectors can diverge. Instead of using a version vector, this work uses an approach called a summary hash history, which they claim allows for convergence on network partition.

Collaborative Buffer Caches in Data Centers

Zhifeng Chen, University of Illinois at Urbana-Champaign

Many data centers contain heterogeneous systems with low latency connections and a large amount of cache in a variety of locations. This creates a problem in that caches often will contain the same data. This group proposes content-aware caching, where each cache has some knowledge of the other caches in the system and can use this knowledge to improve its caching policies. This knowledge can be obtained through message passing or by prediction of cache behavior.

Deep Store

Lawrence You, University of California, Santa Cruz

One cost problem in storage systems is the maintenance of the growing volumes of archival data. This group proposes a new archival storage system to ease the cost and difficulty of maintaining archival information. For example, they propose the removal of redundancy

using inter- and intra-file compression. They also hope to manage content; data lives and dies with applications and systems. Performance is also a challenge, since many users require near-line access to archival data.

CACHING & SCHEDULING

CAR: Clock with Adaptive Replacement

Sorav Bansal, Stanford University; Dharmendra S. Modha, IBM Almaden Research Center

Summarized by Xun Luo

Dharmendra presented CAR, an enhanced CLOCK-caching algorithm, which keeps conformation with CLOCK on the primitives and outperforms CLOCK across a wide range of cache sizes and workloads.

The idea of CAR is to maintain two clocks; one of them contains pages of "recency" while the other contains pages of "frequency." New pages are first inserted in the former and graduate to the latter upon passing a certain test of long-term utility. The researchers did extensive trace-driven tests on the CAR algorithm and observed that CAR is comparable to ARC and substantially outperformed LRU and CLOCK.

Circus: Opportunistic Block Reordering for Scalable Content Servers

Stergios V. Anastasiadis, Rajiv G. Wickremesinghe, and Jeffrey S. Chase, Duke University

Summarized by Shafeeq Sinnamohideen

The goal of this work is to improve the scalability of Internet content servers that perform whole-file transfers, such as those serving images or non-streaming video and music. In these cases, the client needs to receive the entire file before it can make progress. In common with other Internet services, the majority of accesses are to a small number of common files. Many clients may simultaneously be fetching the same file although they may have started at different times.

The traditional approach is to serve each transfer individually and let each client progress at its own rate. If the requested file is too large for the server's cache, however, the multiple request streams will thrash in the cache. In the worst case, the access pattern to the disk degenerates toward random.

The proposed solution is to use the client's memory as a reorder buffer and transfer each block read from the disk to each client that needs it in the order it is read from the disk. Since scheduling the optimal sequence of transfers is NP-hard, the following heuristic is used: First, blocks are read to satisfy the most demanding (fastest, furthest ahead) client. Other clients are sequentially served blocks from the cache as fast as they can be transferred. If a client falls far behind, it is moved ahead to the most recent block, leaving a gap of missing blocks. When a client reaches the end of the file, it sequentially reads any missing blocks.

The system was evaluated with a synthetic workload of requests arriving in a Poission distribution, uniformly distributed across files with an average size of 512MB. The server and network were configured so that the bottleneck to sequential transfers would be the server's disk. Both identical and varied client network links rates were considered. With identical links, the network becomes the bottleneck, and with varied links, the disk is the bottleneck, but at much higher performance than with sequential transfers. Circus is insensitive to changes in file size, unlike sequential, but degenerates to sequential if file sharing is low. The conclusion is that content servers don't benefit from sharing if the files they are serving are large, and block reordering helps if there is sharing, up to a 10x speedup at the sweet spot.

Erez Zadok from Stony Brook University questioned how clients that perform multiple fetches or partial fetches from mirrors affect Circus. Mirroring reduces sharing, and thus the benefit, but parallel fetches do not. Erik Reidel from Seagate Research wanted to know how more sophisticated sharing models such as 90/10 or zipf would affect results. The response is that the results are applicable to the shared portion of the workload. Yitzhak Birk from Technion questioned whether the use of digital fountain techniques wouldn't be a better solution to the problem. While fundamentally different, it shifts the cost into client CPU usage and disk capacity, which may be cheaper than improving disk seek time.

A Framework for Building Unobtrusive Disk Maintenance Applications

Eno Thereska, Jiri Schindler, John Bucy, Brandon Salmon, Christopher R. Lumb, and Gregory R. Ganger, Carnegie Mellon University

Summarized by Michael Abd-el-Malek

This was one of the two Best Student Paper awards. The talk described a mechanism for running low-priority applications that perform disk maintenance (e.g., backup, cleaning, cache writeback, etc.). Storage systems can use idle time or "wasted" disk head rotation time in order to handle disk requests for the background applications, without impacting foreground applications.

Idle-time detection is conceptually simple, and so the presenter concentrated more on how to make use of "wasted" disk head rotation time. For example, if you have two foreground disk requests that are spaced apart in their disk layout, then you can process a disk request that occurs in the middle of the rotation for free. This constitutes the freeblock scheduling subsystem.

An API is provided that lets an application register its intent to read/write blocks in the system in the background.

An asynchronous calling model is used – an application callback is called when the freeblock subsystem is about to handle a disk request. This model is useful because it gives freedom to the application to lazily allocate memory.

Evaluation using TCP-C, Postmark, and a synthetic benchmark demonstrates that free disk bandwidth is available, even if no idle time is detected (e.g., in the case of Postmark). Foreground performance is not impacted.

More information is available at *http://www.pdl.cmu.edu/*.

MOBILE STORAGE

Integrating Portable and Distributed Storage

Niraj Tolia, Jan Harkes, and M. Satyanarayanan, Carnegie Mellon University; Michael Kozuch, Intel Research

Summarized by Akshat Aranya

Niraj Tolia presented "lookaside caching," a technique that harnesses the convenience and speed of portable storage devices, such as USB memory keychains, to enhance performance and availability of distributed file systems.

Niraj started his talk by asking whether portable storage devices can be used in more meaningful ways or if they are just glorified floppy disks. He showed that portable devices and distributed file systems have certain complementary properties. For instance, portable devices provide high performance and availability, whereas distributed file systems provide robustness, consistency, and high capacity.

The basic idea behind lookaside caching is to use a portable device as a fast cache; a file server is still the authoritative source for a file. Whenever a file is opened, its 20-byte SHA-1 hash on the portable device is compared with that stored on the server. If the hashes match, then the file is provided from the portable store; otherwise it is fetched from the server. The benchmark results show significant performance improvement for slow links even when a small number of files are served by the cache.

The work was questioned for contrived benchmarks: Why would anyone carry around the Linux kernel source code on a portable device for compilation? Also, a member of the audience suggested work on policies to decide what to put in the lookaside cache.

Segank: A Distributed Mobile Storage System

Sumeet Sobti, Nitin Garg, Fengzhou Zheng, Junwen Lai, Yilei Shao, Chi Zhang, Elisha Ziskind, and Randolph Y. Wang, Princeton University; Arvind Krishnamurthy, Yale University

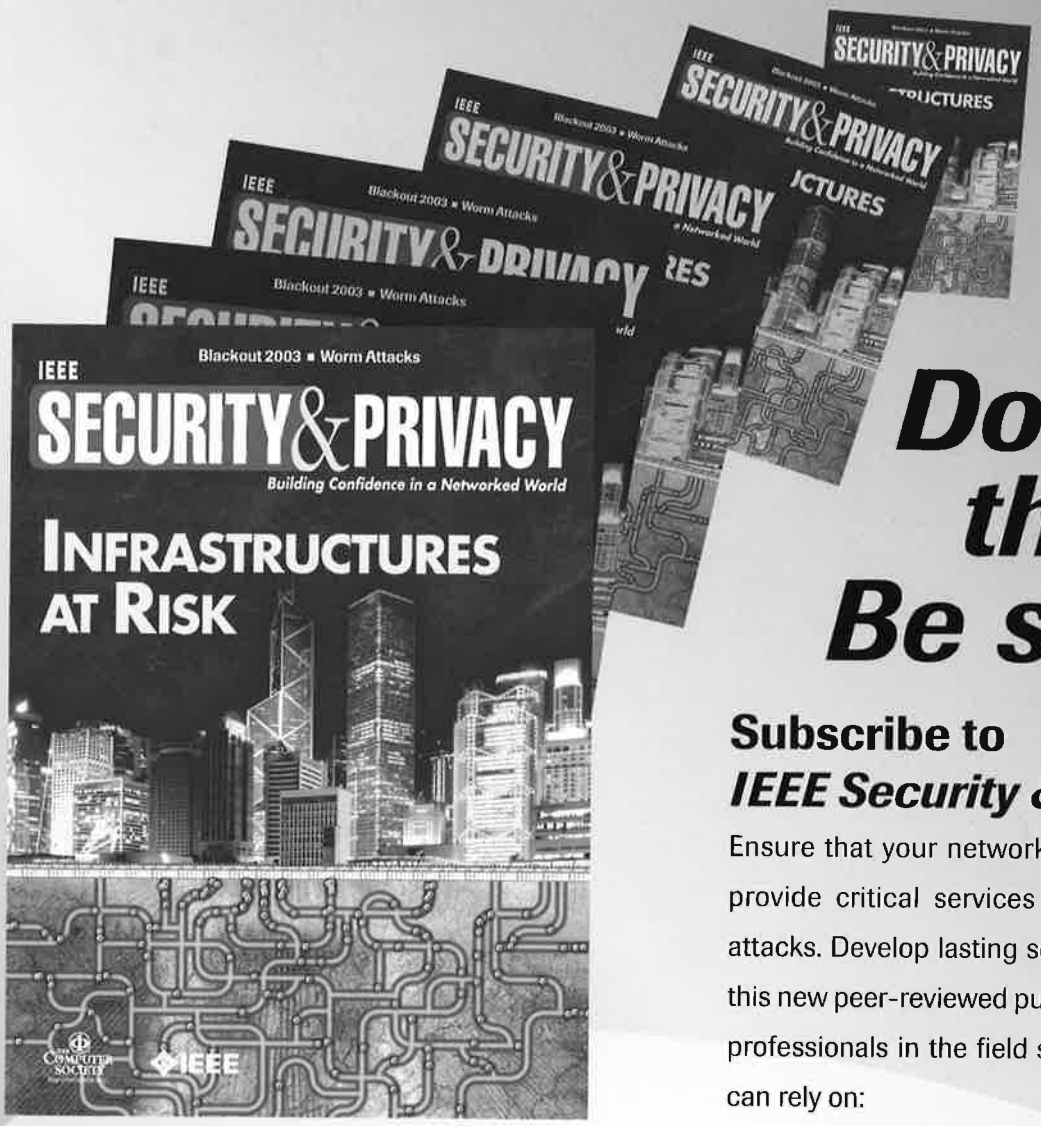
Summarized by Akshat Aranya

Sumeet described a system to manage data and mobile devices in a heterogeneous environment. The Segank system provides a uniform namespace and location independence of data without complete replication. The major emphasis of Segank is on awareness of network characteristics so as to minimize use of weak links.

Segank guarantees consistency by maintaining an invalidation log that maintains the timestamp for each update. The most recent invalidation log is maintained in a portable device that the writer carries. The log is propagated lazily to invalidate stale copies of the data. This approach decouples data propagation from log propagation.

Segank uses a multicast protocol, called Segankast, for location and access of replicated data. This provides network-aware reading by heuristically building a multicast tree. Data is shared using snapshots. This provides a trade-off between freshness and performance.

The presentation generated plenty of interest from the audience, who raised questions about concurrency and compared the system with alternative approaches like VNC. There were also some concerns about background propagation of data in insecure mobile environments, which, the speaker pointed out, were orthogonal to the problems addressed by the system.



Don't run the risk. Be secure.

Subscribe to *IEEE Security & Privacy*

Ensure that your networks operate safely and provide critical services even in the face of attacks. Develop lasting security solutions, with this new peer-reviewed publication. Top security professionals in the field share information you can rely on:

Wireless Security • Securing the Enterprise • Designing for Security Infrastructure
Security • Privacy Issues • Legal Issues • Cybercrime
Digital Rights Management • Intellectual Property Protection and Piracy •
The Security Profession • Education

IEEE

SECURITY&PRIVACY

Building Confidence in a Networked World

IEEE **http://www.computer.org/security/** IEEE COMPUTER SOCIETY www.computer.org

;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