;login:

THE MAGAZINE OF USENIX & SAGE
June 2002 volume 27 • number 3

inside:

USENIX & SAGE
The Advanced Computing Systems Association &
The System Administrators Guild

USENIX Upcoming Events

2ND JAVA™ VIRTUAL MACHINE RESEARCH AND TECHNOLOGY SYMPOSIUM (JVM '02)

AUGUST 1–2, 2002
SAN FRANCISCO, CALIFORNIA, USA
http://www.usenix.org/events/jvm02

11TH USENIX SECURITY SYMPOSIUM

AUGUST 5–9, 2002
SAN FRANCISCO, CALIFORNIA, USA
http://www.usenix.org/events/sec02

16TH SYSTEMS ADMINISTRATION CONFERENCE (LISA '02)

Sponsored by USENIX & SAGE

NOVEMBER 3–8, 2002
PHILADELPHIA, PENNSYLVANIA, USA
http://www.usenix.org/events/lisa02

INTERNET MEASUREMENT WORKSHOP 2002

Sponsored by ACM SIGCOMM and co-sponsored by ACM SIGMETRICS and USENIX

NOVEMBER 6-8, 2002
MARSEILLE, FRANCE
http://www.icir.org/vern/imw-2002/
Camera-ready copy due: August 9, 2002

5TH SMART CARD RESEARCH AND ADVANCED APPLICATION CONFERENCE (CARDIS '02)

Sponsored by USENIX and the IFIP Working Group 8.8 (Smart Cards)

NOVEMBER 20-22, 2002
SAN JOSE, CALIFONIA, USA
http://www.usenix.org/events/cardis02
Submissions due: June 24, 2002
Acceptance notification: August 12, 2002
Camera-ready final papers due: September 23, 2002

2ND WORKSHOP ON INDUSTRIAL EXPERIENCES WITH SYSTEMS SOFTWARE (WIESS '02)

Sponsored by USENIX
Co-sponsored by ACM SIGOPS & IEEE TCOS

DECEMBER 8, 2002
BOSTON, MASSACHUSETTS, USA
http://www.usenix.org/events/wiess02
Submissions due: July 15, 2002
Notification to authors: August 19, 2002
Camera-ready final papers due: September 30, 2002

5TH SYMPOSIUM ON OPERATING SYSTEMS DESIGN AND IMPLEMENTATION (OSDI '02)

Sponsored by USENIX
Co-sponsored by ACM SIGOPS & IEEE TCOS

DECEMBER 9–11, 2002
BOSTON, MASSACHUSETTS, USA
http://www.usenix.org/events/osdi02
Notification to authors: August 5, 2002
Camera-ready final papers due: October 7, 2002

4TH USENIX SYMPOSIUM ON INTERNET TECHNOLOGIES AND SYSTEMS (USITS '03)

MARCH 26-28, 2003
SEATTLE, WASHINGTON, USA
http://www.usenix.org/events/usits03
Submissions due: September 16, 2002
Notification of acceptance: November 8, 2002
Camera-ready final papers due: January 17, 2003

2ND CONFERENCE ON FILE AND STORAGE TECHNOLOGIES (FAST '03)

MARCH 1-APRIL 2, 2003
SAN JOSE, CALIFORNIA, USA
http://www.usenix.org/events/fast03
Submissions due: September 3, 2002
Notification of acceptance: November 1, 2002
Camera-ready final papers due: January 6, 2003

For a complete list of future USENIX events, see **http://www.usenix.org/events**

contents

;login: Vol. 27 #3, June 2002

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

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

$50 of each member's annual dues is for an annual subscription to *;login:*. Subscriptions for nonmembers are $60 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.

Cover: Norwegian UNIX User Group logo; see page 66

motd

by Rob Kolstad

Dr. Rob Kolstad has long served as editor of ;login:. He is also head coach of the USENIX-sponsored USA Computing Olympiad.

kolstad@usenix.org

Second Order Effects

I really enjoy observing the world in all its aspects, both human and otherwise, in order to try to understand why things happen the way they do – and to try to use that information to predict the future. If I ever get good enough at it, I figure I can exploit it in many different ways.

One of the subtleties that I truly enjoy is the notion of "second order effects." You hear these all the time, particularly from politicians. In one polarizing debate, you'll hear people say, "But if we tell our kids about [sex education, abortions, other religions, etc.] then they will just decide that it's OK or even go out and try [it] themselves." The knowledge won't just be transferred for its own sake (e.g., to prevent disease, unwanted pregnancy, immorality, etc.) but, a second order effect is inferred.

The biggest second order effect I'm observing right now is the aftermath of the tragedy on September 11, 2001. This is now the largest denial of service attack I've ever seen. It is natural to seek security and safety – and no culture exceeds ours in its ability to take things to excess. Airport waiting lines, enhanced personal identification schemes, and a general fear/paranoia about "leaving the nest" are just some of the offshoots of this second-order attack. How many hundreds of person-years have we now wasted standing in line for security checks at airports? How much commerce has been disrupted by the understandable reticence to travel?

On another front, I was privileged to spend a weekend visiting northern California and co-editor Tina Darmohray. She and her husband shuttled me around in their vehicle among various sites. On one multiple-mile excursion on a sunny afternoon, she remarked on the lack of children playing outside. Sure enough, nine miles later I had seen but one child outside in a yard. Where were they?

Apparently, the fear of abduction and other dangers has motivated parents in that community to keep their children inside. This is carried to the point of escorting children to the car, shuttling them to a friend's house, and making sure they get inside that house's door before moving on to another task. This is surely a denial of service attack – and what message do the children hear? I contemplate having been brought up that sort of fearful environment and shudder. How can happy, well-adjusted adults who can perform reasonable risk-assessment emerge from a such a milieu?

One of my friends has suggested that all this has to do with our culture's inability to assess risks. He asserts that the masses give far too much influence to isolated and unusual events. Of course, the media does little to help this. News is interesting because of its rarity, so rarer events (e.g., commercial airline crashes) get lots of play.

I wish that we can find a way through these denial of service attacks, a way to walk the streets and yards of our cities without fear and with a realistic approach to threats that really do exist within our society. I truly hope we can find a way for our children to be able to play in their yards. A permanent loss of that freedom is a huge one.

apropos

Don't Shoot the Messenger

by Tina Darmohray

Tina Darmohray, co-editor of ;login:, is a computer security and networking consultant. She was a founding member of SAGE.

<tmd@usenix.org>

If you've considered security at your site for very long, you've probably thought about what to do in an emergency. That is, if there is a computer security incident, what procedures are to be followed during the crisis, who will be in charge, who will make decisions to cut off services, who will talk to the media, etc. In fact, discussing and planning ahead for these crisis procedures is considered to be "industry best practices" and, as such, is becoming a pretty mundane topic, as it should be. I would have thought so, too, until just a few months ago, when I heard of a slight twist to this type of planning that should be considered by all organizations.

Has your organization considered what the procedure would be if the security threat is within your own walls? That is, do you know who to tell, who makes decisions to cut off services, etc.? Do any of the procedures differ from those designed for an attack from the outside? What happens if what has to be said or done is "unpopular" with the designated decision person? Has this potential conflict of interest been taken into consideration?

Recently, I participated in a security review of a site that had already considered this potential conflict of interest. In order to address it they have a security incident reporting structure that is different from that of the line-management of the security group. It's actually quite a clever design, which simultaneously incorporates varied user groups' computational needs and organization-wide security concerns. At this site, the responsibility for computer security comes through the CIO and then through the typical management chain-of-command to the computer security officer. What is unique in this organization is, in a parallel fashion, computer security issues are considered through a committee hierarchy: the senior manager's Committee on Computing, the Computer Coordinating Committee, and finally the Computer Security Committee, which is made up of computer and network security specialists and representatives of the organization's computer users groups. The computer security officer is a member of the Computer Security Committee. In the event of a security incident, the computer security officer is the one calling the shots. In the event of a computer security issue, the computer security officer has a committee, already in place, to report it to. In this way, security of the organization will be placed above the agenda or best interests of a single individual, or at least it will be openly discussed. Additionally, the computer security officer is protected from being "the messenger" if the security concern is about, or within, his or her chain of command.

So, if you haven't taken the time to create crisis-mode procedures for your site, do so now. If you haven't considered how such procedures might differ, depending on the source of the crisis, do that too.

;login:

EDITORIAL STAFF

EDITORS:
Tina Darmohray *tmd@usenix.org*
Rob Kolstad *kolstad@usenix.org*

STANDARDS REPORT EDITOR:
David Blackwood *dave@usenix.org*

MANAGING EDITOR:
Alain Hénon *ah@usenix.org*

COPY EDITOR:
Steve Gilmartin

TYPESETTER:
Festina Lente

PROOFREADER:
Lesley Kay

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*

letters to the editor

from Barb Dijker
barb@netrack.net

To the Editors:

I read with great interest the letter from the USENIX president in the February issue. I hope that the newly elected USENIX board will address first on their agenda measures to resolve the problems highlighted by Mr. Geer.

The first problem Mr. Geer highlights is the nominations process. Greg Rose participated in the process of nomination of candidates, which is a right of all members in the USENIX election policies. Any member can be nominated by getting the signatures of 5 other members and run in an election opposite those candidates nominated by the nominating committee. So if what Greg did is so deplorable, as to require public denouncement by the president, then the election policy should be changed to disallow it. Nominations should be allowed only through the nominating committee that is appointed by and serves at the pleasure of the sitting USENIX board.

The second problem is the elections themselves. Mr. Geer urged members "in the strongest way," even calling upon the Bible to support his argument, to vote only for only those candidates nominated by the nominating committee. Why should we leave that up to chance? Mr. Geer notes that every year many USENIX members are new to the organization, so they don't know they should be doing this. USENIX should set policy such that members can only vote for those nominated by the nominating committee. Then there is no need for the elections. USENIX should seat the new USENIX board with those nominated by the nominating committee. That will require the nominating committee trim their list a little, but that's easier than expanding it.

Those changes will fix the problems Mr. Geer identified with the current process. It will also change USENIX from a democracy to an oligarchy. While this change may be sub-optimal in an idealistic sense, it greatly simplifies things, saves money (elections cost about $10,000), and ensures that the problems associated with the 2002 election never happen again. An oligarchy is a perfectly valid, legitimate, and common form of governance for a membership organization. I'm a member of several organizations that have no member elections for their governing body. For example, the Front Range Unix Users' Group (a USENIX local group), the American Motorcyle Association, and the American Civil Liberties Union. When an organization governs member services rather than people, a democracy is not required and often not even desirable.

Barb Dijker
USENIX member since 1989

FREEBSD JAILS

from Poul-Henning Kamp
phk@FreeBSD.org

In the February issue Rik Farrow mentions the FreeBSD jail facility and states that the instructions call for a complete FreeBSD distribution to be installed inside a jail.

While it is correct that the jail(8) page shows this in an example, it is by no means required. The only file which must be inside a jail is the executable to be run there.

Many users of jail use it to isolate single processes and put no more than the required binaries inside the jail.

In the spirit of the "You must be this tall to attack this castle" cartoon, I will pass on a trick which will makes a FreeBSD jail immune to 99% of all known UNIX exploits in circulation: Do not put a /bin/sh in there unless you actually need it.

letters to the editor

RIK RESPONDS:
I understand that a chrooted environment doesn't require a complete install. That's just what the docs for jail() suggest, and most likely, what most people do.

Examples showing people how to set up Apache using Perl or mod_php in the jail() would help a lot more than the current docs do.

I like the jail() concept, but without proper configuration information, something that is easy for people to read and understand, jail() by itself does not do the job. It takes a crafty sysadmin who understands both chroot() and jail() to take full advantage of it.

Rik Farrow

[See page 48 for more details – Ed.]

OPENBEOS
From Stephen Schaff
webshifter@webshifter.com

Rik:

I stumbled across your article: *http://www.usenix.org/publications/login/1998-12/musings.html* and noted particularly your comments about BeOS: "BeOS is proprietary – no source code. I am looking for something like BeOS, but more open."

As a BeOS enthusiast, I'd like to bring to your attention that which you asked for in that comment above: *http://www.openbeos.org*

Although still in development, the momentum is astonishing, and the progress to date is very impressive.

Just thought I'd pass it on . . .

15 YEARS AGO . . .
from Ted Dolotta

Peter [Salus],

The reason for this message is your "Fifteen Years Ago in USENIX" item in the Feb. 2002 issue of *;login:*. In it, you mention – in addition to the 1987 USENIX meeting – the 1984 USENIX meeting in DC, snow storm and all. That started me reminiscing . . .

If my memory is correct, the January 1984 USENIX meeting was also marked by another "storm": There was a last-minute, surprise exhibitor at that show, namely Big Blue, which set up a dozen PC-AT's in a hotel suite (there was no more "real" exhibit space available?) running the Personal Computer Interactive Executive – PC/IX, a single-user UNIX for the PC-AT, and developed for IBM by my team at INTERACTIVE Systems. [I just looked at the User's Manual for PC/IX, and it says, smack in the middle of the title page, "by INTERACTIVE Systems Corporation," with the IBM logo relegated to the bottom of the page].

IBM invited all the attendees to come up to their hotel suite and play with the system at will – there were no canned demos, no presentations – just UNIX System III, and a bunch of IBM guys, and some of my guys – to answer questions. [Several of my guys had to buy – on short notice – their first adult suit; two among them actually asked me whether they could share a suit because their IBM-suite duty times did not overlap].

Suddenly UNIX was no longer a Bell Labs/Berkeley/university/ hacker/nerdy thing – it was in the mainstream, endorsed by the largest computer company in the world. A heady day, indeed (I know I'm biased).

PC/IX (which was not a commercial success; it went nowhere – the IBM sales force was scared of it and did not know how to sell it) was followed by VM/IX (UNIX as a guest on the VM/360 mainframe system); it flopped as well, as did IX/360 (native UNIX on a System/360 mainframe). And then came AIX – UNIX on the PC-RT (a RISC chip), which IBM sells to date, albeit on more modern hardware. VM/IX and AIX were done by my team at INTERACTIVE, and we were also involved with IX/360.

My AIX team consisted of 18 people, including the support staff – secretary, hardware guy, documentation people, administrator, etc. IBM had a team of 350+ testers in Austin testing the stuff my guys built; it was like having a firehose turned on you all the time. Each bug was reported dozens of times.

[I believe I wrote to you in a prior message about the AIX documentation (YACC stopping when it encountered a "workstation" symbol, etc.), and I also explained a while back about the various issues that arose in the context of creating UNIX manuals within the constraints of IBM's practices. But to give the devil his due, IBM graphic design folks did a *great* job of designing the binders for the PC/IX and VM/IX documentation (whose content was essentially identical except for the name of the system, and for some SysAdmin pages, PC/IX being a single-user, native-mode system for the PC-AT, while VM/IX was a multi-user system hosted on VM/360): The PC/IX binders were pin-striped, very dark charcoal gray, with white type, and a picture of a bud vase with a *single* red rose – harking back to the original IBM PC ad campaign featuring "The Little Tramp" (Charlie Chaplin look-alike with a red rose); the VM/IX binders were *identical*, except that the vase had a *bunch* of red roses. It was brilliant design.

Today, IBM is into UNIX in a big way, with Linux mainframes and AIX systems (the latter, I suspect, will be around alongside Linux systems for a good long time), and huge booths at Linux World.

But it was at the 1984 USENIX meeting that IBM first publicly put its big toe into the UNIX stream.

Best regards,
Ted

uncle sam as uncle spam

by Strata R. Chalup

President, VirtualNet; Starting as a Unisys 68K admin in 1983, Strata Chalup is now an IT project manager but allegedly has retained human qualities. Her mixed home network (Linux, Solaris, Windows) provides endless opportunties to stay current with hands-on tech.

strata@virtual.net

CORRECTION:

In my "Consulting Reflections" article (February *;login:*), I made a "braino," the advanced form of a typo, saying the US Social Security allocation is between 7 – 8% of your 1099 income. It's 7 – 8% of your W2 income, not your 1099. When you're doing 1099 work, you must provide the employer match yourself, as well as the Social Security. The 2001/2002 figure is 15.30%. That's actually combined Social Security and Medicare. The Social Security is only applied to the first $84,900 of your income (up from $80,400 in 2001), but the Medicare has no income cap. For a very lucid explanation of how to calculate these, try this "Ask Alice" column from the CCH Small Business Toolkit: *http://www.toolkit.cch.com/advice/01-070askalice.asp*. By the way, I don't recommend doing your own taxes if you are consulting, unless you're an accountant! Thanks to Toni Veglia for spotting the braino, and for letting me know that I got it right in a previous article. SRC.

Coming Soon to an Email Box Near You

Like many people, I have a habit of noticing some things only after it's too late to do anything about them. The issue raised in this article may belong to this category. It's been on my "looming over the horizon" list for quite a while.

As technologists, we can't afford our signature mistake of focusing on technology and ignoring policy. The ways in which technology is applied are almost always predetermined by existing policy. This truism scales from a small business to a huge government agency. It is the application of policy to technology in the latter that we are concerned with here.

The war against spam continues to escalate. Many of us see current legislation as one of the tools with which to mark spam for filtering ("ADV" in subject line) or to forbid its generation. Unfortunately, a powerful express train is in motion to head off anti-spam legislation and make it easier and easier for spam to reach you. The major player in this effort might come as a surprise to you – it's the US Postal Service.

Like Charity, Spam Begins at Home

I send in my little cards to the Direct Marketing Association annually so that our household's various US mail addresses and phone numbers are culled from the marketing lists of DMA-abiding organizations. Unfortunately, the United States Postal Service is not one of these organizations. I recently became aware of an annoying "feature" of US mail delivery, namely, that one cannot opt out of the neighborhood flyers, catalogs, grocery ads, etc. that come in the mailbox almost daily. They are addressed to "resident" at one's address, and the companies that put them together contract directly with the post office for delivery.

I pursued the query up to my local Postmaster here in Sunnyvale, CA, and was told that there is no mechanism whereby one can choose not to receive this material. I was told that it's too much work for the mail delivery workers to keep lists of who is and isn't getting it. What I was also told, which is much more important, is that there is no accounting mechanism to reflect that some customers might opt out. How convenient!

When I vented on the topic of physical spam to a local mailing list, one person suggested, "Mark it 'return to sender' and put it back. Make the post office deliver it twice, once to you, once to them." I appreciate the thought, but it seems an unwieldy solution for several reasons. First, it merely creates an extra obligation on my part, beyond simply recycling it or throwing it away. It also amounts to trying to start a denial-of-service attack on the USPS rather than to approach the problem constructively. This raises some ethical issues, to say the least.

The DMA substantially agrees with groups like the Coalition Against Unsolicited Commercial Email (CAUCE) about what constitutes spam. The USPS failure to adhere to DMA opt-out for its local physical-mail spam is all too likely a significant predictor of their stance on email spam. For many years, the US Postal Service and other groups have been kicking around various plans to allow so-called universal email delivery to US residential addresses. I find it very plausible that we could end up with non-opt-outable post office spam in our email boxes down the road. How real is this threat?

Let's take a look at the USPS policies and directions on physical spam mail, and try to make an educated guess.

A Disturbing Precedent

The USPS tried launching an electronic delivery service called PosteCS back in 2000. The service would deliver documents via a combination of email and hosted Web repositories, specifically by sending messages which contained embedded document retrieval URLs. If one goes to the PosteCS FAQ today, one sees that the service has been cancelled. The site says that PosteCS "did not meet the Postal Service's market expectations."

Pulling an older copy of the PosteCS page from Google's handy cache, one finds an FAQ question, "What if I want to send thousands of PosteCS packages per day and want the packages to be sent out automatically?" The answer was that "the PosteCS Team would be happy to speak with you regarding potential Application Programmer Interface (API) solutions for your automated sending needs," and included an 800 number with which to contact them.

What does the USPS itself have to say about it? This quote from the now-defunct PosteCS FAQ is fairly eloquent on the subject:

> Q. How does PosteCS fulfill the U.S. Postal Service's primary mission?
>
> A. PosteCS fulfills the Postal Service's mission to 'bind the nation together through its communications.' This includes providing universal service by delivering mail to every address in the country. In keeping with this mandate, USPS is offering PosteCS as a way to deliver electronic mail with the same security and privacy that customers have come to expect. The Postal Service recognizes the need to respond to new and emerging business needs and PosteCS helps to fulfill this need.

Alert readers will have immediately noticed that the focus is on business needs for delivery. As the US Postal Service is pressed more and more strongly to stand alone with minimal government funding, it is becoming more and more business oriented. Individual consumers simply do not use the system in sufficient volume to pay for the infrastructure – to survive, the USPS is going to have to continue to make business its number one priority. The issue for those of us with United States mailing addresses is that the USPS' status as a quasi-governmental agency may do an end-run around all the anti-spam efforts we can muster.

We're from the Government and We're Here to Help You (to Spam!)

It's not necessary to dig very deeply to find out how the USPS feels about spamming, excuse me, direct email solicitation. They currently offer a wide range of services for businesses, grouped conveniently on a page whose title is a blatant "Get More Customers." (*http://www.usps.com/smallbiz/smcust.htm*) Services include an online directory of Authorized Affiliate Merchants who sell direct mail services, including addresses, and NetPost(TM), a way to "create mailings on your computer and send them quickly and easily." The USPS even sponsors "Direct Mail Made Easy" seminars in cities all over the country, and licenses private companies to administer the National Change of Address program. When you file your change of address forms, the USPS bundles up all the info every two weeks and sends it to NCOA, where (for a fee, of course) firms can check their direct marketing lists for accuracy (*http://www.usps.com/directmail/faqs/lists.htm*).

"The truth is, 'junk mail' is nothing that can't be cured with a decent list." (USPS Web site).

Still not convinced? How about this?

> Q. How Can the Postal Service Help You with Your Direct Mail Campaign?
>
> A. Remember, the United States Postal Service is here to assist you in any way that we can. We believe that using Direct Mail will bring you business. And that's one of the things we're in business for." (*http://www.usps.com/directmail/faqs/working-with.htm*)

And one final, telling salvo:

> However, you could have the greatest product in the world, but, when the offer is mailed to the wrong target, it quickly becomes 'junk mail.' For example, hair products to a bald man. A dog catalog to a cat lover. Information about a preschool program to a senior citizen.
>
> Remember, if you haven't put your package into the right hands, you've wasted money getting it there.
>
> The truth is, 'junk mail' is nothing that can't be cured with a decent list." (*http://www.usps.com/directmail/dmguide/discoverdm/lists6.htm*)

There it is: "nothing that can't be cured with a decent list." As we will see, this takes on some rather interesting overtones when one considers the current planning underway for the continued modernizing of the US mail system.

The Door Swings Both Ways

It's not just that physical-world ideas end up implemented on the Internet. Internet-inspired ideas can translate into the physical world, especially if there is a sufficiently powerful and flexible interface.

Most government documents make dull reading. "Seizing Opportunity: The Report of the 2001 Mailing Industry Task Force" is quite the exception. In addition to providing a view into the cheerfully upbeat world of the marketing industry, it has some eye-opening findings and recommendations to make about USPS technology and future directions.

I noticed bar codes appearing on US mail many years ago. I kind of figured – when I thought about it, which was rarely – that they had the address info coded onto them. I was mostly right, but things have evolved considerably since last I looked.

The USPS now uses a system called PLANET to make sure each piece of US mail is uniquely identified. That makes sense, given added-value delivery services such as Return Receipt, Certified Mail, and the like. They need to track these things. Now comes the next phase, and here's where I get cold feet.

Speedy Delivery – Of a Cookie?

The USPS has been planning to move to using composite and two-dimensional bar codes which "allow companies to access information about a mail item wherever and whenever the information is needed. The mailing industry can utilize this information prior to induction into the mailstream, and after delivery by the Postal Service, without needing to access an online database, or open each mail piece." The reason for the more complex bar codes is to be able to attach more data to individual pieces of mail.

The report says that:

> Intelligent mail – the use of data-rich, machine-readable bar codes to make each mailing piece unique – will allow the mailing industry to compete by including data that 'lives' with the mail piece or package. By linking mail with complementary information channels, intelligent mail creates value for the consumer, sender, and the processor . . . Industries . . . are extending their services by implementing the next generation of barcoding . . . These codes allow tagging of each mail piece or parcel with an entire data file, not just a unique number.

The way I read this is that physical mail is generating the equivalent of cookies, and that the USPS is working directly to become another DoubleClick. This is pretty alarming. Equally alarming is the prospect that someone with access to my physical mail and a bar-code reader can suddenly get a lot more information about me than I might like. What data sets would be put on an individual piece of mail, and how widely would the data sets be shared? Who knows?

If the data is widely shared, someone scanning a random postcard coupon mailing could conceivably tie into targeted information that includes things like financials, family members, and the like. If the data is closely held and varies from direct mail firm to firm, we could see the emergence of a nice little cottage industry in mail pilfering or at-mailbox scanning to create aggregate databases. Though it would be tough for these hypothetical cottage industry players to compete with the USPS, given that every piece of mail will be scanned through their system. Did I already mention DoubleClick?

In this entire report, one lone sidebar mentioned in passing that "Other information lodged within the bar codes of catalog labels will allow the same consumers to tell mailers to remove them from the retailers' lists." That's *one* mention of opt-out so far in this entire document. Ouch! Disturbingly, on the same page as our tiny ray of light, we find that the Task Force recommends that "consideration be given to amendments to current privacy legislation to allow proper use of credit header and driver's license information for address quality – not marketing – improvement."

Rabbit Committee Announces Lettuce Patch Initiative

So, who exactly is the "Mailing Industry Task Force"? They spent six months producing a report released in October 2001 and are continuing with Phase Two. The Deputy Postmaster General and "other senior postal executives" are part of the Task Force. So are "eleven chief executives and leaders of the world's largest mail-focused corporations," ranging from Pitney-Bowes to Wunderman. The latter, not exactly a household word, was described in the report as "the world's largest direct-to-customer marketing solutions company."

The Task Force unsurprisingly recommends the formation of an Industry Council as the "logical and seamless extension of the work already undertaken by the Task Force." The number one item on the agenda of the Industry Council, right after starting a positive publicity storm about use of US mail? Tackling privacy legislation, of course!

> Current and pending information usage law and regulation will significantly impact the mailing industry and consumers. Much of the debate surrounds information access to interested third parties. To date, proponents of restricting information access have carried the day [*author: oh, really?*], without adequately considering economic and social consequences . . . [U]nyielding legislation that . . .

restricts the availability of lists and information used to target mail will cause mail volumes to decline.

Fortunately for the continued good health of the warp and woof of the social fabric, as well as the USPS, the Task Force realizes that "A partnership between the industry and the government is a necessity to ensure that consumers are protected and appropriate legislation is written."

And Then I Woke Up – Not!

Pretty scary stuff, but comfortably remote and in the future, right? Wrong! The report, issued in October 2001, goes on to state that the USPS has "assigned sponsors to each recommendation, is pursuing implementation options where it has the legal authority to do so, and is coordinating the recommendations with existing programs and its business plan."

Status reports and additional recommendations will be issued by the time of the National Postal Forum in late April 2002. It's probably far too late to derail this train, but we may be able to mitigate its effects with a little publicity. If some of the things described in this article sound like a bad idea, let your elected representatives know what you think. Keep an eye on the THOMAS database for specific legislation and on some of the direct mail industry Web sites. Industry alerts and news updates can also alert you to work against them. Another good resource is SpamLaws, which tracks anti-spam legislation around the globe.

Speaking of legislation, what about it? Wouldn't it stop things like USPS delivery of email spam? Basically, no. Take a look at the text of H.R. 1017 for an example of the limitations of current legislation. Its provisions are specifically concerned with spam that "falsifies an Internet domain, header information, date or time stamp, originating e-mail address, or other identifier." It doesn't actually prohibit spam itself, just certain anonymous or misleading ways of delivering it.

What about S. 630? It says that if at any time within the five years prior to receiving a spam there has been a transaction between the sender and recipient, and the recipient has been provided with an "opportunity" to opt-out and hasn't done so, it's not spam. A transaction is defined as "involving the provision, free of charge, of information, goods, or services requested by the recipient." This definition easily includes viewing a Web page and having your email address harvested by a spam script. As long as you were presented with an opt-out link somewhere on the page, it's not actionable as spam. How many of us scan every link on every page we surf to see if it includes an opt-out link?

S. 630 also seems to say that an email is not actionable spam if it has clear, untampered headers (routing, they call it), includes an opt-out link, and has a physical address for the spammer. I am not a lawyer, but to me these two anti-spam laws have very little in the way of teeth.

The Big Picture

I hate to say it, but to me the big picture here is looking pretty bleak. I'm interested in what you think we can do to head this one off at the pass . . . or why you think it's a Good Thing and that we shouldn't try to do so.

REFERENCES

"The Posts in an Interactive World," The Institute for the Future: *http://www.usps.com/strategicdirection/_pdf/postsint.pdf*

"Seizing Opportunity: The Report of the 2001 Mailing Industry Task Force": *http://www.usps.com/strategicdirection/_pdf/seizeopp.pdf*

United States Postal Service Direct Mail Web site: *http://www.usps.com/directmail/*

PosteCS (Then) *http://216.239.35.100/search?q=cache:uO6e43J0GmQC:www.usps.com/postecs/faq.htm+USPS+universal+email+&hl=en*

PosteCS (Now): *http://www.usps.com/postecs/faq.htm*

Direct Marketing Association white papers on the effect of information flow on marketing costs: *http://www.the-dma.org/isec/whitepapers.shtml*

DMA telephone solicitation opt-out: *http://www.the-dma.org/consumers/offtelephonelist.html*

DMA mailing list opt-out: *http://www.the-dma.org/consumers/offmailinglist.html*

DMA email list opt-out: *http://www.the-dma.org/consumers/optoutform_emps.shtml*

2002 National Postal Forum site: *http://www.npf.org/NPFSanDiego2002.htm*

Association for Postal Commerce: *http://postcom.org/*

THOMAS Legislative Database: *http://thomas.loc.gov/*

Coalition Against Unsolicited Commercial Email: *http://www.cauce.org/*

SpamCon: *http://www.spamcon.org/*

SpamLaws (international resource): *http://www.spamlaws.com/*

evolving behavioral boundaries in cyberspace

Lowering the Barrier to Hacking Charges

by Erin Kenneally

Erin Kenneally is a Forensic Analyst with the Pacific Institute for Computer Security (PICS), San Diego Supercomputer Center. She is a licensed Attorney who holds Juris Doctorate and Master of Forensic Sciences degrees.

erin@sdsc.edu

The most widely known and applied federal law enacted to prevent abuse to computer systems is the Computer Fraud and Abuse Act (CFAA), referred to in legal source code as 18 U.S.C. § 1030. Although branded as the nation's primary anti-hacking law, the CFAA is quietly drawing the boundaries of acceptable behavior for IT professionals engaged in business activities. That is to say, the law is being applied to punish more than the hacker miscreants who break into machines and damage networks or steal intellectual property. The CFAA's prohibition on unauthorized access to computer systems is being interpreted by courts to govern the actions of Jane and Joe Employee – which hold nontrivial implications for computer security and infosec professionals.

This article examines trends in recent judicial applications of the CFAA as they may affect business cyber-risk exposure and remediation efforts. On a macro level, this analysis helps illustrate how the CFAA is shaping social expectations and notions of "reasonable" behavior in this neoteric cybersociety within which the legality of our actions are increasingly being judged.

Recent CFAA cases are both shaping and reflecting judgments of acceptable cyber-behavior. By ordering criminal penalties or civil relief for computer-related misbehavior, courts are transitioning standards of right and wrong behavior from the physical to the digital society. Whereas judges and juries can draw upon personal experiences when they adjudge the reasonableness of someone's actions – that it is unreasonably dangerous to drive drunk, for example – the context to make value determinations in the cyberworld is immature.

The CFAA has been applied relatively freely in recent cases, thereby expanding the scope of what constitutes criminal behavior as well as lowering the threshold of damages needed to raise a claim. Specifically, the elements of "exceed[ing] authorization" and "loss" have been interpreted rather broadly. Significant precedent was set by the US Court of Appeals in *EF Cultural v. Explorica* (9 I.L.R. (P&F) 3040 (1st Cir., 2001)), which agreed with the lower court that (1) the defendant's use of a scraper program to access information from EF's Web site could be construed as unauthorized access; and (2) money spent by the plaintiff-business to assess whether the software robot had caused damage to its systems was enough to satisfy the "loss" requirement under CFAA.

By What Standard Does Computer Access "Exceed Authorization"?

In a nutshell, the section of the CFAA used by *EF Cultural* prohibits the knowing access of a protected computer without authorization (or in excess of authorization) with the intent to defraud, and the value of the thing obtained must exceed $5,000 in

Realizing that IT professionals need to advance their knowledge in step with technology, it may be a challenge to sanitize the technical, business, or financial information that they take from job to job.

any one-year period. Whereas this is the black-and-white law, some actions may fall into a gray area where its illegality is questionable. Here the defense argued that its use of the robot software to parse through the data on EF's Web site and extract information did not violate the CFAA because it was not unauthorized.

This court defined the contours of unauthorized access by referencing the "reasonable expectations" standard to judge Explorica's "gray" actions. In other words, Explorica violated the CFAA when it used EF's Web site in a manner outside the "reasonable expectations" of both EF and its ordinary users. The court reasoned that because of a confidentiality agreement between the defendant-employee and EF Cultural (one of the defendant employees who helped design the scraper had formerly worked for EF), the defendant exceeded authorization by abusing proprietary information needed to create the scraper.

This carries significance to both the individual and the business enterprise in the face of current business climate – where non-competition and non-disclosure agreements are passed like currency, the IT workforce is increasingly job-mobile, and the web of outsourced partners and third-party affiliates are ever important. What's more, competition is forcing businesses to find new ways to extract value from data that openly resides throughout the Internet. Software technologies offer the capability to identify, collect, and contextualize this data more efficiently and at a competitive advantage.

Furthermore, realizing that IT professionals need to advance their knowledge in step with technology, it may be a challenge to sanitize the technical, business, or financial information that they take from job to job. Even assuming Pat Sysadmin can subjectively segregate this "proprietary" information, the slope is slippery, nonetheless. What is to prevent a Web-based company from alleging CFAA violations in light of the default rule that "conduct is without authorization if it is not in line with the reasonable expectations of the Web site owner and its users"? This may be an instance of the cart driving the horse, thereby enticing a competitively disadvantaged company to "rethink" how its reasonable expectations can lead to civil compensation under the CFAA.

This raises perhaps the most underestimated aspect of this case, which lies in the arguments that the court sidestepped. The travel codes and corresponding tour price data were all publicly accessible through normal browsing of the Web site. The court even admitted that the tour codes could be correlated to actual tours and cost data by manually searching and deciphering the URLs to extract pricing information. However, the scraper program automated this search to allow the pricing information to be extracted quickly, and this was then utilized by the defendant (a competitor of EF Cultural) to set competitive prices. The real question becomes: would the use of the scraper alone render access unauthorized under the CFAA?

Although the court found the access to be unauthorized based on the confidentiality agreement, the existence of Webreaper-like programs and Web-page monitoring agents that contextualize data and use it for various e-commerce applications ensures that the courts will have to face the aforementioned issue in the future.

Interestingly, the lower court in *EF Cultural* found that the scraper circumvented technical restraints in the Web site "by operating at a warp speed that the Web site was not normally intended to accommodate." So, despite the fact that this software did not use a back door to access information or crack into a password-protected area, the district court appeared willing to label the use of a program that captures and data mines dis-

parate data as exceeding authorization. Indeed, this conjures serious issues for a technology-driven society where capabilities outpace intentions and automation is proliferating.

Another notable case illustrating actions "without authorization" has bearing on disloyal employees who access their employer's computers to communicate proprietary information. In *Shurgard Storage Centers, Inc. v. Safeguard Self Storage, Inc.* (119 F. Supp. 2d 1121 (W.D. Wa. 2000)), an employee of Shurgard sent an email containing trade-secret information to the defendant-competitor. Relying on principles of agency law, the court found that the employee's authority ended when he started acting as an agent of the defendant. In other words, there was an implicit revocation of authority, regardless of whether the employer had knowledge of the improper communications. This was the hook that allowed his accessing the employer's computers to be criminalized under CFAA. In short, the employee effectively met with the same treatment as a random hacker who may have compromised the company's network. So, "transitioning" employees and their future employers should pay attention to how their access and use of proprietary data may create CFAA exposures.

Defining "Loss" Absent Physical Damages

Recall that unauthorized access is actionable under the CFAA if damages are shown. "Damage" is defined as "any impairment to the integrity or availability of data, a program, a system, or information that . . . causes loss aggregating at least $5,000 in value during any one-year period to one or more individuals" (18 U.S.C. § 1030 (g)). Pretty cut-and-dried, right? Well, the gray area that *EF Cultural* addressed was the contours of "loss," which are not defined in the CFAA.

Whereas it would be rational to assume that EF Cultural was stymied on this element, the court rejected the defendant's argument that only the use of the scraper program qualified as damage. Instead, the cost of diagnostic measures to assess the damage of the scraper on EF's Web site satisfied the damage threshold.

This rationale was made on the shoulders of other cases that wrestled with damage disputes. For instance, Shurgard construed damages to result from impairment to the "integrity" of Shurgard's computers. This was the case when trade-secret data was merely copied and disseminated, adding that physical modification was not necessary for integrity to be called into question.

The other referenced case stated that Congress intended "loss" to cover remedial measures borne by victims that could not be considered direct damage by a computer hacker (*In re Doubleclick, Inc. Privacy Litig.*, 154 F. Supp. 2d 497, 521 (S.D.N.Y. 2001)).

Another case that is influential in ascribing the contours of "damage" is *U.S. v. Middleton* (35 F. Supp. 2d 1189 (N.D. Ca. 1999)). It allowed damages based on salaries paid to, and hours worked by, in-house employees who repaired the damage done by an unauthorized intruder. "As we move into an increasingly electronic world," the *EF* court reasoned, "the instances of physical damage will likely be fewer while the value to the victim of what has been stolen and the victim's costs in shoring up its security features undoubtedly will loom ever-larger."

Although *EF Cultural* permits consultant fees, recovery costs, and remediation expenses to satisfy the meaning of "loss," other courts have concluded that lost business or goodwill, by itself, could not constitute loss (*Register.com, Inc. v. Verio, Inc.*, 126 F. Supp. 2d 238, 252 (S.D.N.Y. 2000)); and loss for purposes of calculating damages

U.S. v. Middleton . . . allowed damages based on salaries paid to, and hours worked by, in-house employees who repaired the damage done by an unauthorized intruder.

The threshold to satisfy damage requirements has been lowered and liberalized.

means "irreparable damage" (*In re Intuit Privacy Litig.*, 138 F. Supp. 2d 1272, 1281 (C.D. Ca. 2001)).

Although these narrow rulings bolster the conclusion that courts are unlikely to allow claimants to throw in the kitchen sink, the threshold to satisfy damage requirements has been lowered and liberalized. In light of this, the legal system may find CFAA opening a floodgate of potential claims.

As with the unauthorized access arguments, the *EF Cultural* court also declined to consider whether the claimed expenses related to boosting Web server security could count toward the damage tally.

This is an issue, however, whose time is drawing near. Combining the ease with which a company can cry foul that its data integrity has been compromised, along with the plethora of security consultants hit by the economic downturn, the potential for abuse is almost as strong as the likelihood of gaining relief. It is not difficult to imagine instances where a shoddily secured e-business might invoke the CFAA for less than earnest purposes, and seek reimbursement for adding state-of-the-art security that puts it in a better position than before the intrusion.

In conclusion, whether IT professionals or businesses are at the giving or receiving end of a CFAA claim, they will do well to understand how courts are interpreting cyber-behavior under the umbrella of the CFAA. Another take-away lesson is that regardless of how broad or narrow courts may construe the CFAA, the ultimate success of a claim or a defense will hinge on the evidentiary proof of wrongdoing and damages. This is where courts will undoubtedly insist on the production of reliable electronic audit trails and logs that reconstruct cyber-behavior.

practical perl

Cleaning Up With Dispatch Tables

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

Editors' Note: Please join us in welcoming Adam Turoff as our new "Practical Perl" columnist.

In this column, we'll be exploring tips and techniques for writing better Perl programs. This month, we examine a very powerful and elegant technique – using dispatch tables to simplify your code.

Recovering from Bad Code

Over time, programmers will inevitably come across a spot of bad code. Perhaps the last rough patch of code was left behind by a former colleague. Maybe it was something that you yourself wrote a few months or years ago. In many ways, the provenance of a piece of ugly code doesn't matter too much – especially when it works in spite of its ugliness. But bad code does make maintenance more difficult, and significantly more frustrating.

This pattern appears to be especially true with Perl programs. Lots of factors contribute to this perception, but the most important one is that Perl is designed to let you, the programmer, solve a problem in the way you feel most comfortable. This means that you are not only expected, but also encouraged to start out writing "baby talk Perl" as a beginner, progressing at your own pace through some of the more advanced features and idioms in the language.

The advantage here is that if you can quickly visualize a quick-and-dirty solution to a problem, very little stands in your way. The disadvantage is that though your brute-force solution may get the job done before the boss fires you, it may cause problems a few months down the road when it is time to extend your program.

Here's an example. The program fragment below needs to perform one of a set of possible operations on a database. The operation is specified through the $mode variable, passed in through the command line, a CGI program, or some other mechanism. The easy way to choose the proper operation is through a cascade of if / elsif / else statements:

```
if ($mode eq "insert") {
    ## insert a record into a database
} elsif ($mode eq "update") {
    ## update an existing record in a database
} elsif ($mode eq "delete") {
    ## delete a record in a database
} elsif ($mode eq "display" or $mode eq undef) {
    ## display the contents of the database
    ## Note: this is the default operation
} else {
    ## Error: no valid operation specified
}
```

Nested Conditionals

For a small series of conditions, this kind of coding is simple, easy to write, and easy to understand and extend. Sometimes, the number of cases to be handled runs into the dozens, making it difficult to keep all of the possibilities in your head. And frequently, each block within this cascade contains dozens of lines of code, making the entire construct take hundreds of lines.

Suppose the requirements for the program change and we need to handle user authentication. If this code were part of a Web log system, we might want a single user to be able to insert and update messages into his own Web log but not other users' Web logs. Similarly, we would want to allow site maintainers to delete inappropriate messages but disallow a regular user from deleting messages. The code might be extended to look something like this:

```
if ($mode eq "insert") {
    if (is_current_user($user)) {

        ## a user is posting into his own Web log

    } else {

        ## Error: attempt to post into someone else's Web log
    }
} elsif ($mode eq "delete") {
    if (is_maintainer($user)) {

        ## site maintainer is deleting an inappropriate posting

    } else {

        ## Error: not a site maintainer; cannot delete messages

    }
}
```

As requirements mount, the simple brute-force design leaves an ever increasing pile of ugly (and difficult to maintain) code in its wake.

Using Dispatch Tables

These types of if / elsif / else cascades tend to inspect the value of a single variable. In the first example code above, we're choosing one of many possible execution paths by inspecting the $mode variable. Because we're using a series of if / elsif statements, only one of statements in this block will execute.

If we turn this problem on its side, then we see that we're associating a set of operations with each possible value for the $mode variable. This behavior should sound familiar, because it describes the behavior of one of Perl's three fundamental datatypes: the associative array, more commonly known as the hash. This insight is the key to using dispatch tables: associating data with code, using references to Perl subroutines (similar to function pointers in C).

A simple dispatch table looks like this:

```
my %dispatch = (
    insert  =>  \&do_insert,
    update  =>  \&do_update,
    delete  =>  \&do_delete,
    display =>  \&do_display,
);
```

Elsewhere in our program, we would find definitions for the do_insert, do_update, do_delete, and do_display subroutines. The bodies of these subs are simply the statements that were previously found within the if / elsif statements.

Next, we refer to the dispatch table to find the piece of code we need to execute when we want to perform one of these operations:

```
my $sub = $dispatch{$mode};

if (defined($sub)) {       # we've found a sub; call it
    $sub->($param1, $param2, ...);
} else {

    ## Error: no sub found; this isn't a known mode

}
```

And that's it. We've clarified our code in a number of important ways:

- Each chunk of code within the (possibly lengthy) if / elsif / else cascade now has a name: do_insert, do_update, etc.
- The dispatch table concisely associates when we want to perform an operation (hash keys) with what operation we want to perform (hash values).
- Extending the dispatch table is as simple as defining a new sub and adding an entry in the dispatch table.
- If necessary, new behaviors can be added or deleted from the dispatch table at runtime.

More Advanced Dispatch Tables

If you look closely, you'll see that we haven't quite replaced our original code yet. Our first dispatch table handles the case where the mode is explicitly specified, but does not handle the default mode, where the value of $mode is undefined. This is actually quite simple to fix with a few lines of setup code:

```
my $key = $mode;

## handle the default mode
$key = "display" unless defined $key;

my $sub = $dispatch{$key};

## continue as before
```

At this point, you may be thinking that dispatch tables are sufficient for replacing simple cascading if / elsif, but not more complex structures like those found in the second code sample above. More complex code blocks can be handled with dispatch tables, but they require a little more ingenuity.

Recall that two embedded conditionals need to be handled: "insert" operations being performed by the "current user" and "delete" operations being performed by a maintainer. We can update our dispatch table to look like this:

```
my %dispatch = (
    insert-user    =>   \&do_insert,
    delete-maint   =>   \&do_delete,
    display        =>   \&do_display,
       ## ...
    );
```

and can update the key for our dispatch tables with some more setup code.

```
my $key = $mode;

$key = "dispatch" unless defined $key;
$key .= "-user"
    if (is_current_user($user) and $key eq "insert");
$key .= "-maint"
    if (is_maintainer($user) and $key eq "delete");

## ...
```

Note that the dispatch table contains no generic "insert" or "delete" action. This is because our setup code filters out inserts that are not being performed by the current user and deletions that are not being performed by the maintainer. As a result, these invalid errors don't map to a valid key in the dispatch table, and raise an error.

Synthetic Keys

This technique for supplying synthetic values can be quite powerful. We started out with simple string equality comparisons, but dispatch tables can be used with more complicated comparisons, including regular expressions. Again, all we need to do is add a little more setup code before evaluating the dispatch table:

```
$key = "music" if $input =~ m/jazz|blues|ragtime/i;
$key = "sport" if substr($game, -4) =~ m/ball/i;
$key = "pair" if @terms == 2;
```

Simplifying Dispatch Tables

At this point, we've seen how to convert simple cascading if / elsif blocks into dispatch tables, as well as how to convert some nested conditionals into dispatch tables. But we haven't seen how to convert a series of equivalent values into keys in a dispatch table. For example, take this test:

```
if ($input eq "baseball"
    or $input eq "football"
    or $input eq "basketball"
    or $input eq "volleyball"
    or $input eq "racquetball"
    or $input eq "squash"
    or $input eq "tennis"
    or $input eq "golf") {
    ## do something with sports
}
```

The simple and obvious solution is to put one entry in the dispatch table for each sport we're trying to match:

```
my %dispatch = (
    baseball    =>  \&do_sports,
    football    =>  \&do_sports,
    basketball  =>  \&do_sports,
    volleyball  =>  \&do_sports,
    racquetball =>  \&do_sports,
    squash      =>  \&do_sports,
    tennis      =>  \&do_sports,
    golf        =>  \&do_sports,
    ## ....
);
```

Alternatively, we could have one entry called "sport", and use some setup code to translate each of these sports to the value "sport" just before we inspect our dispatch table. That would have the advantage of having one definition in the dispatch table that matches "sport".

If all we want to do is remove seven extraneous declarations in our dispatch table, we don't necessarily need to jump through such hoops. We could simply declare one particular sport in the dispatch table, and declare later on that the other seven sports have the same behavior as our first sport:

```
my %dispatch = (
    baseball   => \&do_sports,
    ## ...
);

## These values are equivalent to $dispatch{baseball},
## whatever it may be
my @equivalent = qw (
    football     squash
    basketball   tennis
    volleyball   golf
    racquetball
);

foreach (@equivalent) {
    $dispatch{$_} = $dispatch{baseball};
}
```

Using Closures

Finally, there's one last issue to discuss: why do we need to create subroutines just to take their references? If we have a complicated set of statements, creating a new subroutine and giving it a descriptive name helps clarify the intention of our code. On the other hand, for dispatch behaviors that are just very short subroutines – one or two statements – we can create an anonymous subroutine, also known as a closure.

As the name implies, an anonymous subroutine is a subroutine that simply has no name:

```
my %dispatch = (
    coffee   =>   sub {print "Turning on the coffee maker.\n"},
    tea      =>   \&make_tea,
);
```

In this example, both values in the dispatch table are subroutine references. There is no difference in the way they are invoked. As a result, our code that grabs a value from the dispatch table doesn't care if a value is a reference to a named subroutine or an anonymous subroutine. The difference here is one of describing intent – the layout of this code tells other programmers that the process to make tea is more complicated than the process to make coffee.

Conclusion

Dispatch tables are an excellent way to bring order to disorderly code. Their primary benefit comes from separating when actions take place from what actions take place. While this technique can be used to simplify and refactor poorly written code, it can also be used as a design tool to create maintainable software simply and easily.

Dispatch tables are an excellent way to bring order to disorderly code.

python or perl: which is better?

by Kragen Sitaker

Kragen Sitaker is a multilingual hacker who's used UNIX since 1992, presently consulting on server-side Web software development in San Francisco. See *http://pobox.com/~kragen/* for more.

kragen@pobox.com

[Editors' Note: Kragen Sitaker wrote this reply to the standard question, "Which is better, Python or Perl? And why?" He has graciously granted us permission to reprint it.]

Python has a read-eval-print loop, with history and command-line editing, which Perl doesn't.

Python has a bigger standard library, which tends to be better designed. For example,

- it's possible to write signal handlers that work reliably without a deep knowledge of the implementation (and I'm not convinced that a deep knowledge of the implementation is sufficient in Perl).
- time.time() and time.sleep() speak floating-point.
- open() raises exceptions when it fails. Every day, people post on *comp.lang.perl.misc* asking why their programs are failing, and it's because some file is missing or unreadable, and they can't tell what's wrong because they've forgotten to check the return code from open(). This doesn't happen in Python.
- os.listdir() omits "." and ".."; they're always there, so you can include them by writing [".", ".."] + os.listdir(), but you almost never want them there. Perl's readdir doesn't omit them, which is a very frequent source of bugs in programs that use readdir.

On the other hand, Perl has a much bigger nonstandard library – CPAN – and some of its standard library is better designed: system() and popen() can take a list of arguments to avoid invoking the shell, meaning all characters are safe.

Python makes it sort of a pain to build types that act like built-in lists or strings or dictionaries or files, and (as of 2.0, with "x in y" now having a meaning when y is a dict) it's impossible to build something that acts both like a dictionary and like a list. It's relatively easy to build something that acts like a function.

But Perl makes it a pain to *use* types that act like those things, and it's impossible to build types that act like functions. But you don't need to build types that act like functions, because you have Scheme-style closures.

Python's syntax is far, far better.

Except that it's indentation-sensitive, which makes it slightly harder to cut-and-paste code and offends the kind of people who unthinkingly adhere to stupid traditions for no good reason. (It might offend other kinds of people, too, but I know it offends this kind of people.)

Perl has a C-like plethora of ways to refer to data types, which brings with it lots of confusion and dumb bugs. It also makes things like arrays of arrays and dicts of dicts slightly more confusing.

Perl has lots of implicit conversions, which hide typing errors and silently give incorrect results. Python has almost none, which leads to slightly more verbose code (for the explicit conversions) and occasional fatal exceptions (when you forgot to convert). (Unfortunately, Python has some implicit conversions and is getting more.)

On the other hand, Python overloads + and * to do different, though vaguely related, things for strings and numbers; Perl makes the distinction explicit, calling Python's string + and * as . and x instead. Also, there is a certain variety of implicit conversion – namely, from fixnums to bignums – that Python doesn't yet do, but should. (Python 2.2 does it.)

Python's variables are local by default; Perl's are global by default. Perl's policy is unbelievably stupid. Worse, in Perl, the normal ways to make variables local don't apply to filehandles and dirhandles, so you have to use special tricks for them.

Perl can be (and, for me, always is) configured to require that all variables be declared and local. Python has no way to require variable declarations, although reading an uninitialized variable is a runtime error, not a runtime warning.

Perl implicitly returns the value of the last expression in a routine. Python doesn't. This is a point in Python's favor most of the time, although it makes very short routines verbose. (Although Python has lambda, which lets you write those very short routines in the Perl style.

Perl has nested lexical scopes, which means that occasionally your variables disappear when you don't want them to, but more often, they aren't around to cause trouble when you don't want them to. They also make it really easy to write functions that return functions as Scheme-style lexical closures. In Python, writing functions that return functions is painful; you must explicitly list all of the values you want to close the inner function over, and if you want to keep callers from accidentally blowing your closure data by passing too many arguments or keyword arguments with the wrong names, you need to write a class with `__init__` and `__call__`. Also, in Python, if you want statements in your closure, you can't write it inline – you have to write `def foo()` and then refer to `foo` later.

Also, Python 2.x has list comprehensions, which reduce the need for really simple inline functions (for map and filter), and also are a very nice language feature in their own right.

The Perl parser gives better error messages for syntax errors.

Perl optimizes better.

Python has an event loop in the standard library. Perl has POE, which isn't in the standard library.

Perl has while (<>). Python doesn't, although it has `fileinput.input()`, which seems to be broken for interactive use. (It doesn't hand the lines to the loop until it's read 8K, and it requires you to hit ^D twice to convince it to stop reading and once more to end the loop.)

Strings in Python are immutable and pass-by-reference, which means that passing large strings around is fast, but appending to them is slow, and it's possible to intern so that string compares are blazingly fast. Strings in Perl are mutable and pass-by-value, which means that passing large strings around is slow, but appending to them is fast, and comparing them is slow.

Python lists don't auto-extend when you try to assign to indices off the end. Perl lists do. This is generally a point in Python's favor.

Perl autovivifies things, so you can say things like $x->{$y}->[$z]++, which will make a hash for $x if there isn't one already, an array for $x->{$y} if there isn't one already, and an element for $x->{$y}->[$z] if there isn't one already, before incrementing it from its default value of zero. Doing this in Python is painful. However, Python allows tuples as hash/dict keys, which lessens the need for this; you can write:

```
if not x.has_key((y, z)): x[y, z] = 0
x[y, z] = x[y, z] + 1
```

Python's variables are local by default; Perl's are global by default. Perl's policy is unbelievably stupid.

Python has this icky (x,) syntax to create a tuple of one item. Perl doesn't have this problem.

Python treats strings as sequences, so most of the list and tuple methods work on them, which makes some code much terser. You have to use substr() or split() in Perl.

Python requires you to use triple-quoted strings to have multi-line strings. Perl has here-docs, but it also lets ordinary strings cross line endings.

Perl indicates the ends of ranges in two ways: the index one past the end of the range, and the index of the last element in the range. The ".." notation uses the second; @foo in scalar context uses the first; etc. Python consistently uses the index one past the end of the range, which is confusing for new users.

Python has this icky (x,) syntax to create a tuple of one item. Perl doesn't have this problem.

On the other hand, Perl has cryptocontext bugs: expressions evaluate to different, and possibly unrelated, things in scalar and list context. This rarely bites me any more, but it used to.

Perl's context-dependency in function evaluation leads to brittleness problems in ways that are difficult to explain; it leads to difficulties in wrapping functions, à la Emacs advice.

Python has reasonable exception handling; you can catch just the exceptions you expect to have happen. Perl has "die" and if you want, you can eval {} and then regex-match $@ to see if it was the exception you wanted, and if not, die $@. The usual upshot is that Perl programs that catch some exceptions usually end up catching all exceptions and continuing in the face of exceptions that should be fatal.

On the other hand, it's still too easy to write a program that does that in Python, too, so people do.

Python gives you backtraces when there are exceptions, from which you are more likely to be able to find the error than from Perl die messages, because they have more information; but Perl die messages are likely to tell you where the error is more quickly, for the same reason. Perl has "croak" which lets you decide which level of the call stack to accuse of causing the error, and can give you backtraces if you want them.

The Python syntax for referring to things in another module is terse enough that people actually use it. Perl's syntax for the same thing is uglier ($math::pi instead of math.pi), and Perl module names are longer, so people tend to import things from the other modules into their own namespace. This makes Perl programs harder to understand.

However, in both Perl and Python you can specify which names can be imported from your module into someone else's namespace, but you can't specify which names can be referred to from another module (e.g., as math.pi, $math::pi). The consequence is that in Perl programs you can usually tell which names are internal to the module and which ones are used from other modules, and in Python programs you usually can't. (Without looking at the other modules, that is.)

Perl lets you trivially build dicts out of lists, which is good, because lists are easy to compute. Python doesn't, although you can write imperative loops to do the same thing.

Perl lets you easily splice lists into other lists (functionally, in list literals).

You can't slice dicts in Python, although you can use 2.x listcomps to get almost the same effect; Perl's @thing{qw(foo bar baz)} becomes [thing[k] for k in 'foo bar baz'.split()]. I'm not sure whether this is better or worse; I think they're both pretty unreadable.

Perl has "last LABEL" and "next LABEL"; Python doesn't. This is stupid of Python, although I don't need multi-level break often. I can get two-level break by moving the nested loops into a new function and using "return" and two-level continue by one-level break.

When Perl converts aggregate data types into strings (e.g., for printing), it turns any references into ugly strings. When Python does, it recursively prints what is pointed to (which fails if the structure is cyclic).

In both Perl and Python, the rules for what counts as true and what counts as false in conditional expressions are needlessly complicated.

In Python, loop conditions that have side effects end up needing to be hidden in functions, or you have to write an N-and-a-half-times loop — which there's no syntactic construct for, so you have to kludge it with "while 1:" and "break".

In Python, you can iterate over multiple sets of items at once:

```
for number, name in [(0, 'zero'), (1, 'one'), (2, 'two')]: pass
```

You can't do that in Perl, although you can do the equivalent if you have an iterator function which returns the tuples one at a time:

```
while (($number, $name) = next_num_name) { }
```

Python also has the zip/map(None,...) function to make this easier, and map() can take multiple sequences, which it iterates over in parallel.

Python has built-in arbitrary precision arithmetic. Perl has it in a nonstandard library.

Python has built-in named, default, and variadic parameters; Perl lets you do all those things yourself, which means that every Perl library that uses named parameters does it differently, and none of them has syntax as nice as Python's f(x=3, y=5) syntax.

Perl has class methods; Python doesn't, although, unfortunately, it is adding them in 2.2.

Perl unifies classes with modules; Python doesn't. So in Python, you can't import a class directly, the way you can in Perl; you can import it from the module it lives in, or you can import its module and get it from there. In Perl, the module is the class. On the other hand, in Python, modules are unified with files, and in Perl they aren't; this usually results in more verbosity in Perl.

Python lets you create classes at runtime with the same ease, or lack thereof, that you can create functions. Perl doesn't allow you to create classes at runtime as easily. This is arguably excessive flexibility that leads to excessive cleverness and unmaintainable code.

Perl will destruct any objects left around at program exit, possibly resulting in destructing objects that hold pointers to already destructed objects; Python doesn't destruct them at all. Both of these approaches suck.

In both Perl and Python, the rules for what counts as true and what counts as false in conditional expressions are needlessly complicated.

Python's sort() and reverse() are in-place only, which means they don't work on immutable sequences, and often makes your code more complicated; this is moronic. Perl did the right thing here.

Perl's split uses a regex; Python's standard split doesn't. Perl is better here. However, Python's standard split defaults to the right separator (whitespace) when you just specify a string, and Perl's split, by default, discards trailing empty fields, which Python's doesn't.

Python's built-in comparison routines do recursive lexical comparison of similar data structures, so you can sort a list of lists or tuples straightforwardly; and you can sort records by some computed key by forming tuples of the key and the record, then sorting the tuples. If you try to sort complex data types in Perl, it will sort them by memory address.

Python's regular expression library is easier to understand than Perl's and uses mostly compatible syntax (although Perl keeps adding features). Perl's regular expressions return subexpressions by mutating global variables $1, $2, etc., which have their state saved and restored in hard-to-understand ways, and they don't mutate those variables if they don't match. Python's regular expression match operator returns a "match object," which, if the regex failed to match, is None; or, if the match succeeded, has a method to fetch numbered subexpressions of the matched text.

See also *http://mail.python.org/pipermail/python-list/1999-August/009693.html* and *http://www.amk.ca/python/writing/warts.html*.

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 for Real Programmers* and the *TclTutor* instruction package. He has been programming computers since 1970 and a Tcl advocate since 1994.

clif@cflynt.com

There seems to be a universal rule that no matter how large a container is, what you need to put in it is larger.

This applies to screens and graphic displays just as much as it applies to my bookshelves. No matter how large the display, there will be an application that needs to display more information than you can fit on it.

The GUI solution to this problem is the scrollbar, which lets us create an image that's larger than the viewing area, and then move the viewable window to the subset of the image we're interested in. (If someone develops a scrollbar to put on bookcases they'll make a fortune.)

Syntax: scrollbar *scrollbarName ?options?*

scrollbar		Create a scrollbar widget.
scrollbarName		The name for this scrollbar.
options		This widget supports several options. The -command option is required.
	-command *"procName ?args?"*	This defines the command to change the state of the scrollbar. Arguments that define the changed state will be appended to the arguments defined in this option
	-orient *direction*	Defines the orientation for the scrollbar. The *direction* may be horizontal or vertical. Defaults to vertical.
	troughcolor *color*	Defines the color for the trough below the slider. Defaults to the default background color of the frames.

A scrollbar interacts with a Tk widget via callback procedures registered with the scrollbar and the associated widget. When the widget changes configuration (for example, more text is added to a text widget), it evaluates a script to update the scrollbar. When the scrollbar is modified (a user moves a slider), it evaluates a script that will update the appropriate widget.

Several Tk widgets (listbox, entry, text, and canvas) have built in support for a scrollbar. Each of these widgets supports a yview and/or xview widget command that moves the viewable window, and can be invoked by a scrollbar.

The scrollbar supports a set widget command that changes the size and location of the slider and can be invoked by the widget associated with the scrollbar.

To make a canvas and scrollbar combination you'd use the -xscrollcommand option to register the appropriate scrollbar's set command to the canvas, and the scrollbar -command option to register the canvas's xview command to the scrollbar.

This code will create a small (50x50) window into a larger (200x50) canvas with a horizontal scrollbar to reposition the displayed window, looking a lot like figure 1:

```
canvas .c -height 50 -width 50 -scrollregion {0 0 200 50} \
    -xscrollcommand {.xsb set}
scrollbar .xsb -orient horizontal -command {.c xview}
```

Figure 1

```
grid .c -row 0 -column 0
grid .xsb -row 1 -column 0 -sticky ew

.c create rectangle 100 20 120 40
```

This is OK, but the viewable window in this example is a fixed size. It would be nice to enable the user to resize the window. The previous Tclsh Spot article described techniques for making resizable windows. This example uses the grid columnconfigure command to allow a widget to be resized.

Figure 2

```
canvas .c -height 50 -width 50 -scrollregion {0 0 200 50} \
      -xscrollcommand {.xsb set}
scrollbar .xsb -orient horizontal -command {.c xview}
grid .c -row 0 -column 0 -sticky nsew
grid .xsb -row 1 -column 0 -sticky ew

grid columnconfigure . 0 -weight 1

.c create rectangle 100 20 120 40
```

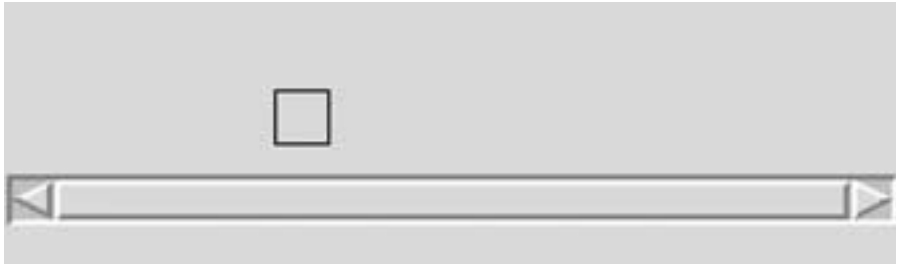

Figure 3

This makes a window that we can expand to look like figure 2, or even stretch to display the entire viewable area of the canvas (figure 3).

Of course, when we can see the entire canvas we don't need the scrollbar. It would be nice to remove the scrollbar when it's not needed.

The most common way to use a scrollbar/widget combination is to invoke a scrollbar directly from the widget and vice versa using the xview and set methods, as we've done here. However, Tcl does not require this style. You can easily write your own scripts to be invoked by the scrollbar and widgets.

When a widget evaluates the script to modify a scrollbar it appends two values for the start and end position of the slider. Both of these are numeric fractions between 0 and 1. If the start position is 0, and the end position is 1, that indicates that the entire viewable area of the widget is displayed, and we don't need a scrollbar.

We can use those values in a procedure that would check to see if the scrollbar were still needed, and remove it when the associated widget is scaled to be fully viewable. This procedure is derived from one of the Tk Tips on the Tcler's Wiki (*http://mini.net/tcl/*). The idea is credited to Brent Welch.

```
# Define a grid command to be sent to the modifyScrollbar proc.
set cmd [list grid .xsb -row 1 -column 0 -sticky ew]

# Create and grid a canvas using the modifyScrollbar procedure to
#  control the scrollbar.
canvas .c -height 50 -width 50 -scrollregion {0 0 200 50} \
      -xscrollcommand [list modifyScrollbar .xsb $cmd]

grid .c -row 0 -column 0 -sticky nsew

# Create the scrollbar. Do not grid. That will be done in
# modifyScrollbar when necessary.

scrollbar .xsb -orient horizontal -command {.c xview}
```

```
# Configure the column to expand when possible.
grid columnconfigure . 0 -weight 1

# Put something in the canvas. Just to add interest.
.c create rectangle 100 20 120 40
```

Figure 4


```
#################################################################
#    proc modifyScrollbar {scrollbar cmd startFract endFract}—
#    Control the scrollbar appearance.
# Arguments
#    scrollbar :  The name of the scrollbar widget.
#    cmd       :  The command to display the scrollbar (grid, pack, place).
#    startFract:  Fraction for the start edge of the slider. PROVIDED BY TCL.
#    endFract:    Fraction for the end edge of the slider. PROVIDED BY TCL.
#
# Results
#    If necessary, the scrollbar is displayed in (or removed from) the
#    parent frame.
#    The slider is modified to reflect new values.

proc modifyScrollbar {scrollbar cmd startFract endFract} {
    if {($endFract < 1.0) || ($startFract > 0)} {
        eval $cmd
        $scrollbar set $startFract $endFract
    } else {
        eval [lindex $cmd 0] forget $scrollbar
    }
}
```

You could hardcode a grid command in the modifyScrollbar procedure, but since the place, pack, and grid window managers all support a forget subcommand, this trick of passing the command to eval allows the modifyScrollbar procedure to be used with any geometry manager.

This technique is fine for a single canvas, text, or listbox widget, but the more common problem is just plain having too many widgets to fit on the screen. Having a scrollable frame would make constructing a scrollable display easy, but frame widget doesn't support the necessary xview and yview widget commands.

However, what we can do is place a frame inside a canvas, and then scale and scroll the canvas as necessary.

Mark Harris and Michael McClennan describe making a scrollable canvas and frame in Effective Tcl/Tk. The big trick is that the frame should be a child window of the canvas you are going to place it into.

```
# Define the grid commands for X and Y scrollbars.

set xCmd [list grid .xsb -row 1 -column 0 -sticky ew]
set yCmd [list grid .ysb -row 0 -column 1 -sticky ns]

# Create a canvas and grid it.
canvas .c -height 50 -width 50 -scrollregion {0 0 200 50} \
    -xscrollcommand [list modifyScrollbar .xsb $xCmd] \
    -yscrollcommand [list modifyScrollbar .ysb $yCmd]
```

```
grid .c -row 0 -column 0 -sticky nsew

# Create the scrollbars; they'll be gridded when needed.
scrollbar .xsb -orient horizontal -command {.c xview}
scrollbar .ysb -orient vertical -command {.c yview}

# Allow the canvas to resize when the parent resizes.
grid columnconfigure . 0 -weight 1
grid rowconfigure . 0 -weight 1

# Create a frame as a child of the canvas, place it in the
# canvas, and bind resizing the canvas to frame size changes.
frame .c.f
.c create window 0 0 -window .c.f -anchor nw
bind .c.f <Configure> {.c configure -scrollregion [.c bbox all]}
```

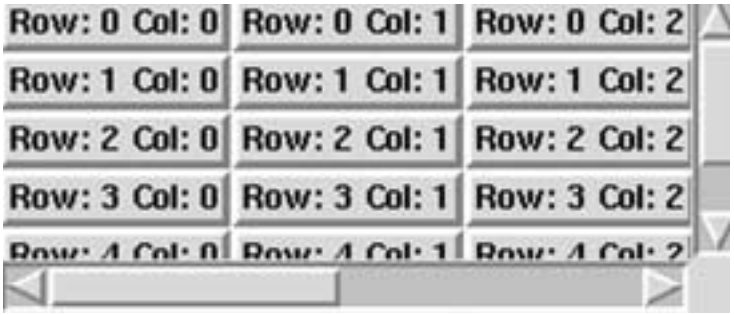

Figure 5

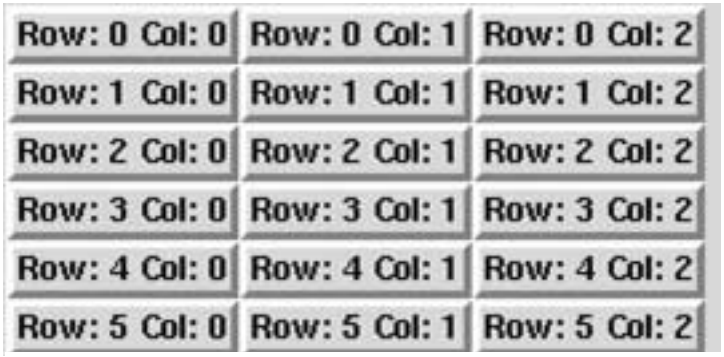

Figure 6

```
# Build some label widgets in the frame for a demo.

for {set i 0} {$i < 3} {incr i} {
 for {set j 0} {$j < 6} {incr j} {
    label .c.f.l_$i,$j -text "Row: $j Col: $i" \
-relief raised -borderwidth 3
    grid .c.f.l_$i,$j -row $j -column $i
}
}
```

This set of code will create a window that looks like figure 5.

If the main window is expanded, the scrollbars go away, as in figure 6.

There are a couple of lines to take note of in this code:

```
.c create window 0 0 -window .c.f -anchor nw
```

This places the frame inside the canvas, with the upper left corner of the frame at the top left corner of the canvas.

```
bind .c.f <Configure> {.c configure -scrollregion
 [.c bbox all]}
```

This line causes the canvas scrollregion to be updated whenever the size of the frame is modified. The frame size will be modified when a user resizes the window or a new widget is added to the frame.

The bind command binds an event/window pair to a script.

Syntax: bind *window event script*

Causes script to be evaluated if event occurs while window has focus.
window The name of the window to which this script will be bound.
event The event to use as a trigger for this script.
script The script to evaluate when the event occurs.

The .c bbox all command is a canvas command that returns the bounding rectangle for a set of canvas objects. The all option tells the canvas to select all objects it is displaying.

The bounding rectangle for all the objects in the canvas (in this case, just the one frame) is the total displayable area of the canvas. We can then configure the -scrollregion to that area, to allow all the objects in the canvas to be scrolled to.

When the canvas is resized by the configure command, it automatically invokes its -xscrollcommand and -yscrollcommand scripts.

This is a useful set of code, but it would be more useful to be able to create scrollable frames when needed.

This code pulls together these ideas into a pair of procedures to create scrollable frames with scrollbars that appear and vanish as needed.

```
package provide scrollFrame 1.1

#################################################################
# proc scrollingFrame {name args}—
#    scrollingFrame - Returns the name of a frame within a canvas
#    attached to vanishing scrollbars.
# Arguments
#    name        The name of the parent frame.
#      NOTE  ::   NON-CONVENTIONAL RETURN – RETURNS THE INTERNAL
#                 NAME, NOT THE NAME OF THE PARENT WINDOW!!!
#    args        Arguments to be passed to frame and canvas
#
# Results
#    Creates 2 frames, a canvas and a two scrollbars.
#    |-------------|   <- Outer holding frame
#    | ccccccccc ^|   Canvas within outer frame
#    | cfffffffc ||   Frame within canvas
#    | cf       fc ||
#    | cfffffffc ||   <— Vertical Scrollbar within outer frame
#    | ccccccccc v|
#    | <--------> |   Horizontal Scrollbar within outer frame
#    |-------------|
#
proc scrollingFrame {outerFrame args} {
    if {[string first "." $outerFrame] != 0} {
        error "$outerFrame is not a legitimate window name -
            must start with '.'"
    }

    # Create the outer frame (or not if it already exists).
    catch {frame $outerFrame}

    # Build the scrollbar commands for the X and Y scrollbar.

    set cmdy [list modifyScrollBar $outerFrame.sby \
        [list grid $outerFrame.sby -row 0 -column 1 -sticky ns]]

    set cmdx [list modifyScrollBar $outerFrame.sbx \
        [list grid $outerFrame.sbx -row 1 -column 0 -sticky ew]]

    # Create and grid the canvas.
    set cvs [canvas $outerFrame.c -yscrollcommand $cmdy
          -xscrollcommand $cmdx]
    grid $outerFrame.c -row 0 -column 0 -sticky news

    # Create the scrollbars. Do not grid. They'll be gridded when
    # needed.
        scrollbar $outerFrame.sby -orient vertical -command "$cvs yview"
        scrollbar $outerFrame.sbx -orient horizontal -command "$cvs xview"
```

```
    # Configure the canvas to expand with its holding frame.
    grid rowconfigure       $outerFrame 0 -weight 1
    grid columnconfigure  $outerFrame 0 -weight 1

    # Create a frame to go within the canvas. The various frame
    #  options are applied here.
    eval frame $cvs.f $args

    # Place the new frame within the canvas.
    $cvs create window 0 0 -window $cvs.f -anchor nw

    # Bind frame changes to modify the canvas scrollregion.
    bind $cvs.f <Configure> "$cvs configure -scrollregion \[$cvs bbox all\]"

    return $cvs.f
}

############################################################
#
#    proc modifyScrollbar {scrollbar cmd startFract endFract}—
#        Control the scrollbar appearance.
# Arguments
#    scrollbar    :   The name of the scrollbar widget.
#    cmd          :   The command to display the scrollbar (grid, pack, place).
#    startFract   :   Fraction for the start edge of the slider. PROVIDED BY TCL.
#    endFract     :   Fraction for the end edge of the slider.  PROVIDED BY TCL.
#
# Results
#    If necessary, the scrollbar is displayed in (or removed from) the
#    parent frame.
#    The slider is modified to reflect new values.

proc modifyScrollBar {scrollbar cmd startFract endFract} {
    if {($startFract > 0) || ($endFract < 1.0)} {
        eval $cmd
        $scrollbar set $startFract $endFract
    } else {
        eval [lindex $cmd 0] forget $scrollbar
    }
}
```

The frame returned by the scrollingFrame procedure can be used just like any other frame; you can place new widgets into it using pack, place, or grid; set the relief or background color; and so on. However, if the frame is too small to display all the widgets, it will suddenly acquire a set of scrollbars.

Here's a short example that uses the scrollingFrame procedure to create a frame and then populates that frame with a bunch of labels.

```
    set ff [scrollingFrame .f2 -background yellow -height 30 -width 30]
    grid .f2 -row 0 -column 0 -sticky news

    for {set i 0} {$i < 200} {incr i} {
        label $ff.l_$i -text $i
        grid $ff.l_$i -row [expr $i / 10] -column [expr $i % 10]
    }
    grid rowconfigure . 0 -weight 1
    grid columnconfigure . 0 -weight 1
```

As usual, this code is available at *http://www.noucorp.com*.

new I/O features in C9X

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

We've been presenting some of the new features in C9X, the standards update to C. In this column we'll discuss I/O features added to the library. We'll start by looking at printf specifiers, and then go on to consider several new I/O functions.

Printf and New Types

C9X adds four types to C: _Bool, wchar_t, long long, and _Complex. How do you print values of these types? _Bool has no printf specifier, and so to print a value of the type, you need to say:

```
#include <stdio.h>
#include <stdbool.h>

int main()
{
    _Bool b = true;

    printf("%s\n", b == true ? "true" : "false");
}
```

Alternatively, you can treat a _Bool as an integer, with values 0/1.

The wide character type, wchar_t, is output using the printf %lc specifier or functions like fputwc. Here's an example:

```
#include <stdio.h>
#include <wchar.h>

int main()
{
    wchar_t c1 = L'\u1234';

    FILE* fp = fopen("test", "wb");
    fprintf(fp, "%lc", c1);
    fclose(fp);

    fp = fopen("test", "rb");
    wchar_t c2 = fgetwc(fp);
    fclose(fp);

    if (c1 != c2)
        printf("c1 != c2\n");

    fp = fopen("test", "rb");
    int c;
    while ((c = getc(fp)) != EOF)
        printf("%x ", c);
    printf("\n");
    fclose(fp);
}
```

Wide characters have an encoding, used to convert them to or from a sequence of bytes. For example, the wide character L'\u1234' is encoded as the three bytes:

```
e1 88 b4
```

The long long type is formatted using the %lld specifier, like this:

```
#include <stdio.h>
#include <limits.h>
```

```
int main()
{
      long long x = LLONG_MIN;

      printf("%lld\n", x);
}
```

The _Complex type has no specifier. Instead, you use the creall and cimagl functions to extract the real and imaginary parts of the complex number. An example:

```
#include <stdio.h>
#include <complex.h>

int main()
{
      _Complex long double c = 37.0L + 47.0L * I;

      printf("%Lg + %Lg*I\n", creall(c), cimagl(c));
}
```

The output is:

```
37 + 47*I
```

and the %Lg specifier is used to format long doubles.

Other Printf Specifiers

Another group of printf specifiers is used to handle situations where an integral type is expressed as a typedef, and the underlying type could be signed or unsigned int, long, or long long; size_t is an example. The %u notation specifies an unsigned type, and the z modifier (i.e., %zu) indicates that the type has size_t width, based on local system settings. Here's how you print a size_t value:

```
#include <stdio.h>
#include <stddef.h>

int main()
{
      size_t x = ~0u;
      printf("%zu\n", x);
}
```

A similar approach is used for the intmax_t types defined in <stdint.h> with the j modifier for %d:

```
#include <stdio.h>
#include <stdint.h>

int main()
{
      intmax_t x = INTMAX_MAX;
      printf("%jd\n", x);
}
```

The output on my Linux system is:

```
9223372036854775807
```

A third example is the t modifier for the ptrdiff_t type:

```
#include <stdio.h>
#include <stddef.h>
```

```
int main()
{
      char a;
      char b;
      char c;
      char d;
      ptrdiff_t x = &d - &a;

      printf("%td\n", x);
}
```

Here are a couple of other examples of new specifiers. %hh converts the corresponding printf argument to character width, and then formats the value as an integer. For example, the output of this program:

```
#include <stdio.h>

typedef unsigned char UINT8;

int main()
{
      UINT8 a = 100;
      UINT8 b = 200;

      printf("%u\n", a + b);
      printf("%hhu\n", a + b);
}
```

is:

```
300

44
```

In both cases, a + b has a value of 300, passed to printf as an argument. But in the second case, the argument is converted to an unsigned character, and thus has the value 44 (300 mod 256). The %hh specifier is useful for working with short integers, for example types like int8_t defined in <stdint.h>:

```
typedef signed char int8_t;
```

A final example uses the %a specifier to format hexadecimal floating constants:

```
#include <stdio.h>

int main()
{
      float f = 16320;

      printf("%a\n", f);
}
```

The output of this program is:

```
0xf.fp+10
```

In other words:

```
(15 + 15/16) * 2^10 = 16320
```

Scanf **Specifiers**

Many of the same specifiers used in printf are available in scanf. For example, this program is the inverse of the one just above:

```
#include <stdio.h>

int main()
{
      double d;

      sscanf("0xf.fp+10", "%la", &d);

      printf("%g\n", d);
}
```

The output of the program is:

```
16320
```

The Snprintf **Function**

Snprintf is a function much like sprintf, but with the ability to specify a maximum buffer width. Here's an example of snprintf:

```
#include <stdio.h>

void f()
{
      char buf[8];

      //sprintf(buf, "testing %d", 1234);
      //printf("%s\n", buf);

      snprintf(buf, sizeof buf, "testing %d", 1234);
      printf("%s\n", buf);
}
int main()
{
      f();
}
```

When I run this program with the sprintf call uncommented, the result is a segmentation violation, due to buffer overflow. snprintf avoids this problem by allowing you to specify the buffer width.

This particular problem is a major source of security holes: for example, manipulating the amount of buffer overflow such that a stack frame gets overwritten.

Vfprintf

vfprintf, and the related functions vfscanf, vsnprintf, vsprintf, and vsscanf, allow you to pass a variable argument list to the function. Here's an example that defines an error-reporting mechanism:

```
#include <stdio.h>
#include <stdarg.h>

void report_error(const char* file, int line, char* format, ...)
{
      va_list args;
```

```
    va_start(args, format);
    fprintf(stderr, "Error at file %s, line %d: ", file, line);
    vfprintf(stderr, format, args);
    va_end(args);
}
int main()
{
    int x = 37;
    int y = 47;
    if (x < y) {
        report_error(__FILE__, __LINE__,
            "x < y (x=%d y=%d)\n", x, y);
    }
}
```

In this example, I have a report_error function, and I want to pass it a file and line, and also a printf format and a variable number of arguments to be used with the format. Inside report_error, I can set up a variable argument list, and further pass it to the vfprintf function.

The result of running this program is:

```
Error at file vf1.c, line 23: x < y (x=37 y=47)
```

The features we've described above are all useful in writing more portable and secure programs, and in working with new C9X types.

protowrap

A Generic and Protocol-Specific Wrapper

by Gunnar Wolf

Gunnar Wolf is the systems administrator for a campus of UNAM, Mexico's largest University. He has been a strong promotor of both Computer Security awareness and the Free Software movement in his community, and likes doing small, useful tools in Perl.

gwolf@campus.iztacala.unam.mx

As system administrators, all of us should be very concerned, even paranoid, about security. There are simply too many threats out there waiting for us to become distracted in order to exploit a vulnerability in our systems. New vulnerabilities are found day to day, and exploits are very quickly crafted for each of them. We want to trust the programs we run at our server, but we know that a secure server is utopian. There are many ways to harden our servers against unknown attacks, but these often require heavy tinkering with the system. In this article, I will propose another alternative, much less invasive and more flexible than what I have found up to now: generic and protocol-specific wrappers, implemented through a couple of Perl modules called ProtoWrap.

A word of warning to the faint of heart: although ProtoWrap does work, it is a work-in-progress, and some important aspects (such as the interface to the operator) have been relegated to focus development on correct and full operation, not on ease of use. I am the first to recognize that I am by no means an excellent programmer, and I know that a better implementation can be made. If you think you can help make ProtoWrap work better, please do. In fact, if you want to develop a similar project sharing the ideas provided here, I would be delighted to hear about it.

General Philosophy

As you know by now (unless you like skipping introductions), ProtoWrap is a series of wrappers. Assuming we are in a hostile environment, we cannot trust a network client to be a good citizen. All incoming connections are, thus, potentially hostile and must pass through a defensive layer before reaching our real server.

I decided to focus my work on a very specific kind of protocol: line-oriented TCP protocols. By "line-oriented" I mean that all commands sent from the client to the server are one or more lines of text, delimited by a new-line character. Such protocols include HTTP, FTP, SMTP, IMAP, POP3, finger, ident, and many others. They do not, however, include Telnet (character-oriented; lines may be assembled by the server, but they simply are not relevant at the protocol level), DNS and NFS (UDP-based), SSH (traffic is encrypted, we can not guess where new lines are), and many others.

Wrapping a program should happen as transparently as possible. Many network services do not involve – on regular operation – a human at either end, and we should alter regular operation as little as possible in order not to trigger a denial of service.

ProtoWrap was designed to be useful even without knowing which protocol it would be wrapping (hence I describe it as a generic wrapper). We will see later how it can be invoked to protect almost every line-oriented protocol. It was very important for me, however, to make it easy to teach it how to intelligently wrap a specific protocol.

Another very important point is that ProtoWrap should be easy to deploy. It should not depend on a particular system configuration, and it should be able to scale. This will also be further explained later on.

Finally, ProtoWrap was designed to be able to protect heterogeneous networks. If ProtoWrap is unable to run on a particular system setup, it should be able to protect it from the outside, running on a different computer or even a different network.

In the Beginning

ProtoWrap began as my answer to stopping spam with different mail transport agents, on different architectures, with a minimum of work. I soon realized the solution I proposed could very easily be generalized, and become much more useful, to many different protocols.

I presented ProtoWrap as my final paper for graduation in computer science at Kennedy Western University. If you want to get full details on how and why ProtoWrap exists and works, I invite you to visit *http://www.gwolf.cx/seguridad/wrap/*.

I chose to develop ProtoWrap using Perl because of Perl's rich pattern-matching capabilities and because of the availability of just about any needed function in the CPAN (Comprehensive Perl Archive Network, *http://www.cpan.org*). Perl also provided me with a clean and easy-to-understand way of dealing with network sockets, an absolute requirement for the wrappers to exist.

A number of problems arose while developing ProtoWrap, as happens in any software project. Among the most challenging was the dual-input problem: data can come, at any moment, from either the client or the server. How can you make a Perl program listen to two different data sources at the same time, and react to the one that provides the whole line first? I owe the answer to Salvador Ortiz, with whom I spent several hours hacking and testing possible alternatives, until we decided to go to a lower level, using Perl's IO::Handle and IO::Select modules. Once again, for more details on the possibilities we studied and why this one was chosen, please visit *http://www.gwolf.cx/seguridad/wrap/node62.html*.

ProtoWrap is able to listen for clients and talk to its server in different ways. Some users might require that a daemon always be running, directly listening to its port, to save resources on multiple invocations and reduce startup time. Others will prefer running the wrapper from inetd, handing it the connection in the form of a STDIN/STDOUT file descriptor pair. As for the server, in some cases it will be started from ProtoWrap, also via a mechanism similar to inetd's, and in others it will always be running – maybe even on a different machine – and the connection will be done by TCP/IP sockets.

The Generic Wrapper Behavior

Using ProtoWrap in the most basic way is very straightforward. Of course, the protection it offers will not be as complete as if we were using a protocol-specific module. The protection that ProtoWrap will give to an unknown protocol is, nevertheless, very important: buffer overflows can be easily prevented altogether. Calling ProtoWrap as shown in Listing 1 results in having the IMAP server in the same system (as 127.0.0.1 is the localhost address) protected by ProtoWrap. The real server is running on port 10143, and should be protected by packet filtering or TCPWrapper rules so that it only accepts connections from the same machine. The result of this setup is shown in Figure 1.

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

my $wrapper = ProtoWrap->new('standalone' => 1,
                'listenPort' => 143,
                'destType' => 'ip',
                'destAddr' => '127.0.0.1',
                'destPort' => 10143,
                'maxLineLength' => 25,
                'logLevel' => 0
                );
$wrapper->startServer() or die 'Can\'t start wrapper!';
sleep;
```

Listing 1

This wrapper will limit every line coming from the client to 25 characters, more than enough to send IMAP commands, and will effec-

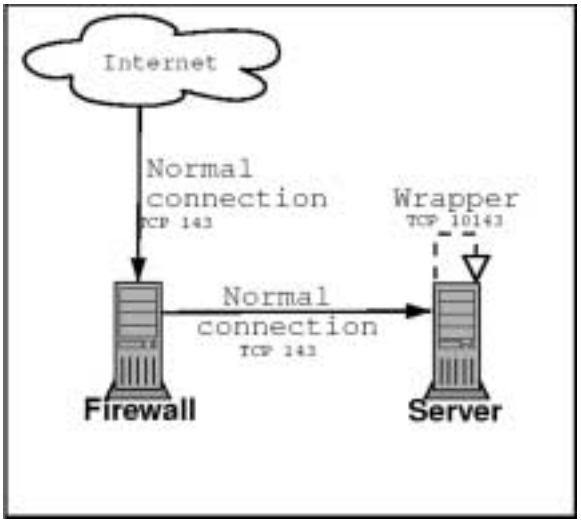

Figure 1

```
#!/usr/bin/perl -w
use ProtoWrap::POP3;
use strict;

my $wrapper = ProtoWrap::POP3->new('standalone' => 0,
                'destType' => 'ip',
                'destAddr' => '127.0.0.1',
                'destPort' => 10143,
                'logLevel' => 0,
                'maxLoginAttempts' => 3
                );
$wrapper->startServer() or die 'Can\'t start wrapper!';
sleep;
```

Listing 2

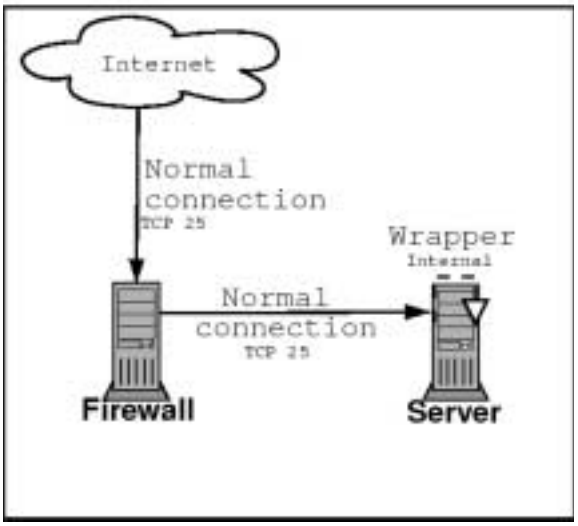

Figure 2

tively avoid any buffer overflow attack. Further, ProtoWrap will alert the system administrator via a syslog entry that an illegal line was sent to the server, reporting what the line's contents were.

Protocol-Specific Extensions

Of course, ProtoWrap can be easily extended well beyond this simple behavior. All protocols implement a specific set of instructions, and deviations from it can be easily marked as misuse and discarded with no remorse. In most protocols, it is also easy to define stages of operation – different subsets of commands will be valid or invalid at different times during the session. By making each line's validation by using the wrapper's testLine method, it is very easy to call a protocol-specific validation function. In order to demonstrate this, I wrote two protocol-specific wrappers: for POP3 and SMTP services.

A wrapper for POP3 can be called with Listing 2; maxLineLength was now omitted, as we will validate each line separately. Here, instead of running in stand-alone mode, the wrapper will now be invoked from inetd or a similar daemon, which will determine which port it will listen on. Conceptually, this setup still resembles Figure 1. We also have a new entry: maxLoginAttempts. If someone tries to log on with an incorrect password more than the specified number of attempts, the connection will be dropped.

The wrapper for SMTP is much more elaborate and able to do more. A typical startup configuration for SMTP can be seen in Listing 3. Here, instead of running SMTP at a different port and connecting to it via regular sockets, we do not run the SMTP server until a connection is received. The server will then be spawned, and when the connection is closed, only the wrapper will continue running (see Figure 2). Though it looks very similar to Figure 1, not having the server listening on a different port can make a huge difference, both from security and performance standpoints.

We see many new parameters here. Most of them were introduced to help stop spam. They are:

- blockAddrList – Addresses we do not wish to receive mail from (anchored to end of string). They can be specific mailboxes (as hahaha@sexyfun.net, a well know worm) or whole domains (everything coming from spammer.org).
- blockBodyList – Every line of the incoming message will be tested against the lines provided here, and if a line matches, the message will be discarded. In this example, most attachment viruses will be avoided, as the most common executable attachment types for Windows systems are disallowed.
- maxMsgSize – The maximum message size (in bytes). In the example, messages over a megabyte will not be allowed.
- maxRcpt – The maximum number of recipients for a message in a single SMTP session. Spammers usually send hundreds of messages at a time using open relays. If a spammer is able to use our machine as a relay, this will drastically cut its effectiveness as a spam relay. In the example, this number is set to zero, allowing for any number of recipients. This machine may be a mailing list server.

```
#!/usr/bin/perl -w
use ProtoWrap::SMTP;
use strict;
my $wrapper = ProtoWrap::SMTP->new('standalone' => 0,
                'destType' => 'pipe',
                'pipeCmd' => '/usr/sbin/sendmail -bs',
                'logLevel' => 3,
                'maxMsgSize' => 1048576,
                'blockAddrList' => ['hahaha@sexyfun.net','@spammer.org'],
                'blockBodyList' => ['^Content-Type: application.+\.(PIF|EXE|VBS|COM|BAT|LNK|SCR)\"'],
                'relayIpList' => ['192.168.150.','192.168.160.'],
                'relayDomainList' => ['mydomain.org','gwolf.cx'],
                'maxRcpt' => 0
                );

$wrapper->startServer() or die 'Can\'t start wrapper!';
sleep;
```

Listing 3

- relayDomainList – Domains for which we allow relay, when they appear either as senders or as recipients of a message (anchored to end of string).
- relayIpList – IP ranges or specific addresses for which we allow relay (anchored to the beginning of the string).

Some of these functions are already handled by most SMTP servers – why am I reimplementing them with ProtoWrap? First, most SMTP servers allow only for specific text matching. With ProtoWrap, we have access to the whole Perl regular expression engine, which gives us much more flexibility and ease of use. Second, if we have our wrappers at a central site such as a firewall, with a setup similar to Figure 3, configuration will be much easier to mantain than if we have them spread on each of our servers.

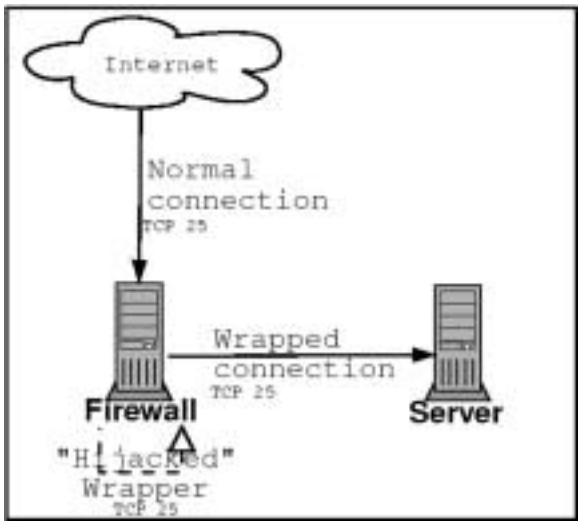

Figure 3

Wrapping Up

ProtoWrap has changed a lot as I have found and incorporated new ideas into it. I am sure it can be a very useful security tool to system administrators with almost every kind of setup. I am also sure that the ideas I have shown here are just the beginning of what can be achieved by such a wrapper.

I have been using ProtoWrap for almost a year on my production servers, since I first labeled it as usable. There are still many features pending, and I sincerely hope to have some of them done by the time this article reaches you. The actions I wish to take before labeling ProtoWrap as stable, and will be done before this goes to print, are:

- Correct Perl module packing – ProtoWrap should be installed as Perl modules. Right now, installation must be done by hand. Soon, I hope to have ProtoWrap ready to be set up as most Perl modules are.
- .rpm, .deb, .tgz packages – Most operating systems are provided with package management systems. Linux distributions handle usually either .rpm or .deb format packages; most other UNIX systems use the simpler .tgz format. These packages allow installation, deinstallation, version management and dependencies. ProtoWrap should then also be available in packaged format.

For more details on other various interesting points that ProtoWrap can be extended to cover, please visit *http://www.gwolf.cx/seguridad/wrap/node72.html*.

To sum up, ProtoWrap is just a proposal, a proof of concept, and I am more than sure it is not the ultimate security solution. It is, however, a valuable addition to most sites' overall security strategy.

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

Like every other USENIX member, I am always learning. The resources appear endless: new books, classes, online Web pages, mailing lists, magazine articles, and questions that people send me.

Just the other day somebody emailed me hoping I could tell her how to change the admin password on the used notebook running XP she had just acquired. I checked out my old favorite, a Linux boot floppy that enables you to change any password on a Windows NT system, and discovered that it won't work for Win2K. At least the site with the bootdisk is still around (*http://home.eunet.no/~pnordahl/ntpasswd/bootdisk.html*).

The buzz for a while now has been attacks on Cisco routers. Now, you probably all remember that Cisco has had its share of security woes (not an unreasonable burden, but still there). What has changed has more to do with rumors than reality – at least so far.

One rumor is that the source code for IOS, Cisco's Internetworking Operating System, has been stolen. That rumor dovetails nicely with a second rumor, that a rootkit for Cisco routers is in the wild. Rootkits for UNIX systems have been around since at least 1994. The "original" rootkit ran on SunOS, included trojaned commands that hid the existence of a sniffer and its logfile, and made it easy for the installer to return and upload the logfile. Some people really appreciated rootkits, as they were busy installing them on every open system they could find – particularly at ISPs.

ISPs made dandy places to install rootkits, especially in the mid-'90s. Small ISPs would install a UNIX mail/Web server, and the attacker would load the rootkit on it. The UNIX server also would sit on a broadcast network, so any transit traffic would be sniffed as well. Of course, the accepted practice today is to put servers on their own subnets and to use switches instead of hubs. Not that hubs are a proven way to prevent sniffing. Check out angst (*http://angst.sourceforge.net/*) if you don't believe me.

The notion of a Cisco rootkit disturbed me at first. I guess I just didn't like to think of a router as something running a vulnerable OS with vulnerable services. But, of course, routers run operating systems. Cisco has written their own. Juniper Networks uses a modified version of BSD.

O'Reilly keeps publishing books, and occasionally sends me a copy, which I much appreciate. *Hardening Cisco Routers*, by Thomas Akin, seems very appropriate for these days. And, Akin's tome secretly pleased me as well, because it covers much of the same turf that I once did in a router security class – but in more detail. For example, I didn't realize that the difference between logging into a Cisco router and what you can do after entering the enable password is based on privilege levels. You can actually set up user accounts (if you are using TACACS or RADIUS) with different privilege levels, then configure the router to provide access to sets of commands at any of the 16 different privilege levels. I had heard that IOS runs at a single hardware privilege level, and the notion of software configurable access to commands agrees with this. IOS is its own "secure" OS, although without the usual aid of hardware support. Like running a shell within the kernel.

Akin takes you through the hardening process succinctly, starting with a description of the issues, going into access control, passwords, remote authentication servers, logging, disabling dangerous protocols/services, controlling routing protocols, and even physical security. I had hoped there would be more on BGP4 filtering, but the focus was

more on not accepting or distributing routes via Interior Gateway Protocols (IGPs), and setting up connection authentication with BGP4. I have always wanted to have all of the security documentation for Cisco routers in one place. While *Hardening* does not include the firewall features of Cisco routers (other than rate limiting for DDoS attacks and which ICMP packets you can consider dropping), it admirably covers its topic area. And at 172 pages, it's a quick read, too.

The other oft-rumored "big attack" on routers involves BGP4. BGP, Border Gateway Protocol, is the glue that holds the Internet together. Unlike IGPs, BGP uses Autonomous System (AS) numbers to describe routes. An autonomous system is a collection of networks under the technical control of a single agency. The way I think about how BGP works is this. Each AS has routes to many networks that belong within that AS, their netblocks (see arin.net, ripe.net, apnic.net for the various AS and netblock registries). An AS advertises routes to their many networks via their AS number, rather than as specific routes through a list of routers. That makes routing within an AS transparent to sites outside of the AS. You just get the packets to the border of the AS, and the AS handles routing the packets to their destinations.

Of course, this setup implies that each AS must be using an IGP internally, so that its own routers know the actual routes to each supported network. What BGP4 does is takes the information from the IGP routing advertisements, converts it to BGP4 advertisements, and shares this with the BGP-speaking neighbors. Only updates are distributed, as every update must be exchanged with every BGP4 speaker. Unstable networks result in frequent changes, or route flapping, wasting not so much network bandwidth as router CPU cycles.

Okay, so BGP4 is the glue and seems to be working just fine. What's the problem? A nice answer to that is AS7001, in April 1997. AS7001 was the AS number for a small ISP in Florida, a Sprint customer. This ISP made a mistake in configuring BGP advertisements so that all the routes that were being advertised internally were forwarded to Sprint using BGP4. As I understand it, this little ISP began advertising itself as the best route for many Class C networks, and as soon as this route spread, the link between Sprint and this little ISP became flooded. Imagine, if you will, the US airline system of spokes and hubs, and now Santa Rosa, California, has announced it has taken the place of San Francisco International, Atlanta Hart, Washington Dulles, Chicago O'Hare, etc., and all the traffic heads there. It was not a pretty picture.

Cooler heads prevailed. By examining the BGP routing updates, someone noticed that AS7001 was declaring itself the best route for networks having nothing to do with it, and filtered all updates coming from AS7001. The problem stopped once people started filtering (blocking) updates from AS7001, and gave Sprint a chance to help the little ISP fix their problem.

The AS7001 incident helped make NSPs aware of how crucial BGP filtering is. Configuring BGP4 routing and filtering is an art form, and not practiced by many (compared to the number of network admins there are). We haven't had a similar problem in years. Also, it is standard practice today to either use dedicated links between BGP4 neighbors, or include an MD5 digital signature with each packet, to prevent spoofing, resetting, or hijacking of the connection between BGP4 neighbors, which stays up as long as the link and routers are up.

This brings me back to where I started: potential, wide-scale attacks on routers. If many routers can be penetrated and rootkits installed, then these routers become simi-

Configuring BGP4 routing and filtering is an art form, and not practiced by many.

As a rumor-monger, I am strongly suggesting that you see to the security of any routers under your control.

lar to the agents used in DDoS attacks. If these routers begin sending incorrect BGP4 updates on command (from authenticated routers, mind you), then considerable disruption of the Internet will occur. With no one suspecting that their router has been corrupted, and general Internet connectivity being disturbed, well, things could get messy for a day or so. Just remember the original Internet Worm. If you want to read more on this, check out "Origins of Internet Routing Instability," by Craig Labovitz et al. (Arbor Networks): *http://www.comsoc.org/confs/ieee-infocom/1999/papers/*. BBN and others have suggested using digital signatures on every update, but if the routers are subverted, the digital signatures will authenticate the phony advertisements as well. You can learn more about the BBN solution, secure BGP, by visiting their Web page: *http://www.ir.bbn.com/projects/s-bgp*.

Note that this is not just a problem for router vendors. You can run BGP4 on Linux and BSD systems as well (MRTD, *http://www.mrtd.net*, and Zebra, *http://www.zebra.org*). And we know that these systems are always totally secure.

Of course, all this is fantasy and rumors right now. As a rumor-monger, I am strongly suggesting that you see to the security of any routers under your control. The little whispers I have been hearing remind me a lot of what was being said before the DDoS attacks of February 2000 occurred, and I really thought I should mention this.

USENIX Needs You

People often ask how they can contribute to the USENIX organization. Here is a list of needs for which USENIX hopes to find volunteers (some contributions reap not only the rewards of fame and the good feeling of having helped the community, but authors also receive a small honorarium). Each issue we hope to have a list of openings and opportunities.

- The *;login:* staff seeks good writers (and readers!) who would like to write reviews of books on topics of interest to our membership. Write to peter@matrix.net.
- The *;login:* editors seek interesting individuals for interviews. Please submit your ideas to *login@usenix.org*.
- *;login:* is seeking attendees of non-USENIX conferences who can write lucid conference summaries. Contact Tina Darmohray, <*tmd@usenix.org*> for eligibility and remuneration info. Conferences of interest include (but are not limited to): Interop, Internet World, Comdex, CES, SOSP, Ottawa Linux Symposium, O'Reilly Open Source Conference, Blackhat (multiple venues), SANS, and IEEE networking conferences. Contact *login@usenix.org*.
- *;login:* always needs conference summarizers for USENIX conferences too! Contact Alain Hénon *ah@usenix.org* if you'd like to help.
- The *;login:* staff seeks columnists for:
  - Large site issues (Giga-LISA),
  - Hardware technology (e.g., the future of rotating storage)
  - General technology (e.g., the new triple-wide plasma screens, quantum computing, printing, portable computing)
  - Paradigms that work for you (PDAs, RCS vs. CVS, using laptops during commutes, how you store voluminous mail, file organization, policies of all sorts)

  Contact *login@usenix.org.*

ISPadmin

Public Internet Access

by Robert Haskins

Robert Haskins is currently employed by WorldNET Internet Services, an ISP based in Norwood, MA. After many years of saying he wouldn't work for a telephone company, he is now affiliated with one.

rhaskins@usenix.org

Introduction

In this edition of ISPadmin, methods of providing public Internet access are covered. The first area examined is the wired access one might see at hotels, Internet cafes and similar venues. Next, 802.11b fixed public access wireless points are covered. Finally, miscellaneous topics such as access point manufacturers, community networks, and software will be considered.

What exactly is public Internet access? As the name implies, it is allowing Internet access in public or quasi-public locations. Some examples of this would be building lobbies (hotels, airports), hotel rooms, Internet cafes, libraries, and similar locations. It can take the form of wired access (usually indoor locations, such as Internet cafes and hotel rooms) or wireless access (any indoor or outdoor area). The most common form of this type of wireless access is based upon the IEEE 802.11b specification, though other methods/protocols exist.

Public Access (Wired)

Figure 1 illustrates how a provider could deploy a wired public access net in a hotel, for example. The boxes to the left represent subscriber client machines, which could be located in hotel rooms or Internet cafe workstations. These machines would connect to switches (or other aggregation equipment) marked "Switch" via 10Mb or 100Mb Ethernet links. These switches would in turn be connected via Ethernet to a firewall. This firewall would house the appropriate authentication and billing interface to enable access to the Internet, after the subscriber has provided the "go ahead" and/or entered credit card billing information.

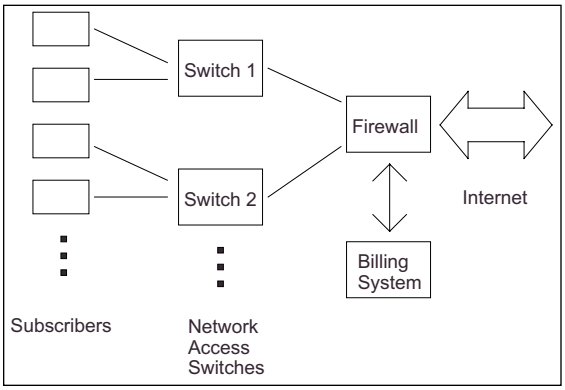

Figure 1

802.11x Background

802.11b is a wireless access standard adopted by the IEEE in 1999. It utilizes the 2.4GHz spread spectrum (unlicensed) to offer 11 megabits per second (Mbps) of bandwidth between two end points. The wireless access point (WAP) will have at least one upstream "wired" port (usually 100Mbps Ethernet) so data not destined for a machine on the WAP network can be delivered. As usual for any evolving technology, WAPs are being integrated into similar products (as well as seeing their price drop). For example, one can purchase a WAP with integrated firewall and 4-port switch for around $150 from Linksys, among other vendors.

There seems to be a lot of confusion between 802.11b and another wireless LAN standard called Bluetooth. Figure 2 illustrates the differences between the two similar technologies: 802.11b is designed for high-speed Internet access with higher radio power and wider range. Bluetooth, on the other hand, is designed for communication between small devices (e.g., cell phones) with low radio power and more limited range.

	802.11b	Bluetooth
Power Consumption	High*	Low
Effective Range	High	Low
Cost	High	Low
Highest ISO Layer**	2	5

*New power-saving mode reduces this to "medium" with appropriate hardware

**ISO layer 2 means protocol requires higher-level software (for example, TCP/IP stack); ISO layer 5 means most functions implemented in protocol.

Figure 2: 802.11b vs. Bluetooth

802.11b wireless access can be used anywhere, indoors or outdoors. However, public access points have been largely deployed up to now in high population density areas (i.e., cities). It is costly

to deploy a wireless technology such as 802.11b in remote areas with limited demand. As deployment costs decline, it will become more cost effective for providers to enable more thorough coverage.

Wireless access is used for point-to-point as well as point-to-multipoint networks. (In this article, WAP will always refer to point-to-multipoint.) The big advantage (and, alternatively, problem) with deploying 802.11b vs. other licensed spectrum products is the fact that 802.11b uses unlicensed spectrum. Of course, the use of unlicensed spectrum may also cause interference problems (from microwave ovens, Bluetooth devices, and wireless phones among others) that have to be corrected. Multipoint to multipoint (or peer to peer) wireless networks exist, though they are not in wide use. Check the References for pointers to additional information on this topic.

There are other wireless standards and products arriving. One is 802.11a, which supports data rates up to 54Mbps in the 5GHz range. The 5GHz spectrum has much less interference than the 2.4GHz band, since it doesn't have nearly the number of uses the 2.4GHz band does. Equipment for 802.11a started hitting the market about January 2002.

Another standard is 802.11g, currently a draft standard that has been the subject of much heated debate. It is 54Mbps (like 802.11a) but is backwards compatible with 802.11b (utilizes the 2.4GHz spectrum) while having 30% greater range than 802.11a. Time will tell which standard "wins," but for now, 802.11b is way ahead of the others simply because it has been around longer and therefore has a much larger installed base. 802.11g chipsets are in the process of being developed, with large-scale shipments scheduled for the third quarter of 2002 (according to a 80211 Planet announcement) by Intersil, a wireless chipset manufacturer.

802.11b Technical Details

The range of 802.11b WAP varies greatly depending upon such factors as transmitter power, antenna type, and the topography between the WAP and client station. The greatest range at full power and clear line of sight with omnidirectional (point-to-multipoint) links is in the neighborhood of 300 meters. The directional antennas (point-to-point links) at full power can exceed 32 km (20 miles).

There are several parameters that can be changed on most WAP models. These include service set identifier (SSID), which associates a WAP with a client. If it is set incorrectly, the WAP will ignore the client packets. Setting this parameter on most client adapters is a manual process, although several aggregators are designing client software to make this transparent to the wireless roamer. Also, the channel (frequency) as well as transmit power and encryption (among other settings) can be adjusted to suit the needs of the WAP owner. Usually these needs are determined by coverage requirements and interference under "Part 15" of the FCC regulations.

Types of 802.11b Networks

The lines between 802.11b network operators are rapidly blurring. For the purposes of this article, wireless networks can be broken down into three types of operators: public, private, and cooperative/community.

Public networks are those installed by service providers for the express intent of reselling/providing access to the public (or quasi-public) user. Private networks are those operators whose primary intent is to run networks for a private entity rather

than provide public access. Finally, the cooperative networks are those operators who build networks in a nonprofit mode (e.g., Seattle Wireless and NYC Wireless).

PUBLIC ACCESS WIRELESS NETWORK

A public access network could be designed as illustrated in Figure 3. Attributes of public wireless 802.11b networks usually take the form of the following:

- Firewall
- RADIUS (Remote Authentication Dial-In User Services) back-end authentication
- No encryption

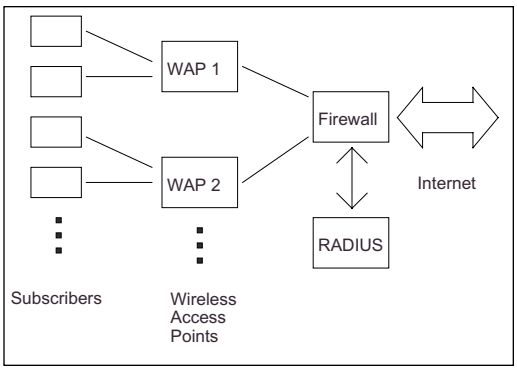

Figure 3

As one might notice, this diagram is very similar to Figure 1 (wired public access network). The only two differences are the wired aggregation points (switches) are replaced with wireless access points, and the billing system is replaced with a RADIUS server. Other than that, the functions remain the same.

PRIVATE WIRELESS NETWORKS

It is difficult to generalize private 802.11b networks. These networks reflect the needs of their owner/operators. They may or may not use firewalls and access control methods. They may or may not participate in a cooperative (Sputnik and Joltage are examples of two such commercial wireless cooperatives). They may or may not utilize encryption, back-end authentication (for example MySQL and/or RADIUS), and MAC address restrictions.

Discovering open wireless networks seems to be a hobby of choice lately (check out Netstumbler and bitshift.org to name two starting points for this activity). To this day, many "private" wireless network owners protect their networks with authentication of some sort (for example, allowing access from certain MAC addresses or authenticating via a database) or with encryption. Also, many opportunities exist for the casual 802.11b network owner to barter/resell access. Participating in such cooperatives may violate the terms of service for the upstream access provider (DSL, cable modem, etc.).

COOPERATIVE WIRELESS NETWORKS

As with private networks, cooperative (co-op) or community wireless networks are very difficult to simplify. There are many co-op wireless networks in operation. Two of the larger and better known ones are Seattle Wireless and NYC Wireless. They focus primarily on point-to-point, but have some point-to-multipoint (public) access as well.

In fact, the NoCatNet co-op group based in Sonoma County, CA, has written one of the few (if only) open source wireless authentication packages available. This code has been modified by Sputnik for use in their hybrid service. (For more on the topic of software, see "802.11b Authentication/Authorization Methods and Software," below.)

802.11b Wireless Access Point Vendors

There are currently a number of 802.11b wireless access point manufacturers. The Network Computing Buyers Guide of November 12, 2001, lists no fewer than 32 WAP models! They range in price from the mid $100s for a Linksys to several thousand dollars for models with integrated management and firewall, among other features. An ISP will likely purchase a less expensive model and attempt to add firewall and man-

REFERENCES

802.11b Networking News: *http://80211b.weblogger.com/*

802.11b vs. Bluetooth: *http://www.imparttech.com/802.11-bluetooth.htm*

bitshift.org: *http://www.bitshift.org/wardriving.shtml*

Bluetooth Special Interest Group: *http://www.bluetooth.com/*

Bluetooth Web log: *http://bluetooth.weblogs.com/*

Boingo: *http://www.boingo.com/*

Building Wireless Community Networks, by Rob Flickenger, O'Reilly, 2001, ISBN 0-596-00204-1

Earthlink: *http://www.earthlink.net/*

Exploiting and Protecting 802.11b Wireless Networks: *http://www.extremetech.com/article/0,3396,s%253D1024%2526a%253D13880,00.asp*

Extreme Tech article on deploying 802.11b access: *http://www.extremetech.com/article/0,3396,apn=5&s=1034&a=13521&app=3&ap=4,00.asp*

Gast, Matthew : 802.11 *Wireless Networks: The Definitive Guide* O'Reilly, 2002, ISBN 0-596-00183-5

GRiC: *http://www.gric.com/*

hereUare: *http://www.hereuare.com/*

IEEE 802.11b standard: *http://standards.ieee.org/reading/ieee/std/lanman/802.11b-1999.pdf*

Internet.com's 802.11 Planet: *http://www.80211-planet.com/*

Intersil 802.11g chipset announcement: *http://www.80211-planet.com/news/article/0,,1481_963341,00.html*

Intersil: *http://www.intersil.com/cda/home/*

IPASS: *http://www.ipass.com/*

Joltage: *http://www.joltage.com/jsp/home/home.jsp*

Linksys multi-function WAP: *http://www.linksys.com/Products/product.asp?grid=23&prid=173*

Linus Router Project: *http://www.linuxrouter.org/*

Mesh Networks: *http://www.meshnetworks.com/*

agement features in a separate firewall box rather than pay for such functionality in a multi-function access point.

One option is to build your own access point. NoCat has a package called WRP (Wireless Router Project, based upon the Linux Router Project). According to the NoCat page, it is "a linux distribution-on-a-floppy that provides wireless support." It appears to be an easy way to reuse old, slow, Pentium-based hardware as a combined wireless access point and chokepoint firewall.

Firewalls

Almost any configurable firewall can be utilized as a public access chokepoint for providers. Most firewalls do not ship with authentication software built in, so this must be developed or perhaps modified if something like NoCatAuth is used. If the site requires multiple WAPs or hardwired networks, all traffic is brought back to a single chokepoint firewall to reduce cost and operational headache. A relatively small box (Pentium 133-class machine) can easily handle the traffic from several active access points. Of course, for very large deployments the traffic would need to be partitioned, but this would not normally be required for a typical rollout involving up to about 250 subscribers.

There are too many firewall vendors to list here, both open source and commercial. An open source firewall can also be utilized and is what most service providers would use to reduce cost and give the ability to customize functionality.

802.11b Authentication/Authorization Methods and Software

As mentioned previously, NoCatAuth is a software package that allows wireless operators to control who accesses their network(s). It is meant for community/cooperative type networks but can be adapted for use in a service provider environment. Its back-end authentication mechanism (as written) can be either text file or a MySQL database. Most service providers require RADIUS authentication for the back end, as that is how existing retail customers usually authenticate. In order for a provider to use NoCatAuth with their existing RADIUS server(s), it must be modified to allow RADIUS authentication. Leveraging existing infrastructure is extremely important these days, with service providers going out of business every week it seems!

NoCatAuth works in conjunction with a firewall to block outside access (by allowing/disallowing MAC addresses through) until the user authenticates. Prior to authentication, "walled garden" access may be granted, which would give the wireless user access to a certain limited set of services. For example, a hotel might allow access to their Web site prior to authentication, but all other access is disallowed.

A version of the NoCatAuth software has been deployed by Sputnik for access to their wireless hot spots (network). See the Sputnik site for more information.

This author is not aware of any commercial off-the-shelf software for deploying WAP authentication mechanisms. However, some WAP manufacturers include authentication/firewall functionality in firmware as an integral part of their access point. This does increase the cost and complexity of the access points in addition to potentially causing interoperability problems with a provider's infrastructure.

Billing

For wired public access, the customer will usually pay up front or be redirected to a Web page that authorizes charges to a hotel room, credit card, or other similar entity.

This functionality can be implemented with most firewalls and an interface (albeit, expensive) to a hotel or credit card billing system.

For 802.11b public access, if RADIUS is utilized as the back-end authentication mechanism, all of the data required for billing should be contained in the RADIUS accounting data. The providers' existing billing system should easily be able to handle these records, once appropriate record filters and billing plans are created. If RADIUS is not utilized, then the process is more difficult and a customized process may be required.

802.11b Aggregators

The state of 802.11b wireless access is very similar to wholesale dial-up at the start of its large-scale deployment a few years ago. Wireless-only aggregators (such as Boingo and hereUare) are joining existing dial aggregators (such as GRiC and IPASS) in this arena. (In fact, the founder of Earthlink, one of the first aggregators of dial-up, is also a founder of Boingo.) Two other aggregators, Sputnik and Joltage, don't seem to fit easily into either category.

Traditional ISP aggregators utilize a settlement process where ISP-A tallies up the amount of usage on its network by ISP-B, and ISP-B adds up usage on its network by ISP-A. The appropriate rate(s) are applied to usage, and whoever ends up owing the other money sends a check. GRiC and IPASS are essentially commercial, third-party implementations of that process. Wireless settlement works in the same manner.

Many of the commercial aggregators develop their own client wireless access software. (In fact, GRiC's software can manage wireless as well as wired and dial connections!) This software manages many of the attributes of the card transparently (SSID being the most relevant) so the subscriber doesn't have to deal with changing them. As additional features are standardized and added to wireless provider networks, this software can be easily upgraded by the subscriber.

Security Considerations

For the end subscriber, security should be of the utmost concern. The fact that critical information (such as credit card data) is traversing open, public access networks and/or radio waves should make one stop and think. If a hardwired public access provider is utilizing hubs (and certain [misconfigured] switches as well), then all ports receive all data destined for one port. Needless to say, this could be hazardous to one's financial well being.

In a similar way, 802.11b access can be "sniffed" out of the air by rogue wireless clients. The encryption standard associated with 802.11b has been proven to be insecure (see the Exploiting and Protecting 802.11b Wireless Networks reference from extremetech.com for a full discussion of security problems and possible solutions). Also, if appropriate access controls aren't in place on each subscriber's machine, one subscriber can hack any other subscriber's machine on the wireless network. This is identical to a subscriber connected to a hardwired hub accessing other subscribers' machines on the same hub, without ever going through the firewall.

Hopefully, future versions of wireless standards and implementations will contain better security. Until then, tread carefully!

Next time, anti-spam mechanisms from a server perspective will be examined in detail. In the meantime, please send me your questions and comments!

Mitre's MobilMesh (Multipoint) project: *http://www.mitre.org/tech_transfer/mobilemesh/*

Multipoint to Multipoint Wiki Wiki Wan in Santa Cruz: *http://wiki.haven.sh/index.php/WikiWikiWan*

MySQL: *http://www.mysql.org/*

Netstumbler: *http://www.netstumbler.com/*

Network Computing article on 802.11a: *http://www.networkcomputing.com/1201/1201ws1.html*

Network Computing WAP Buyers Guide chart: *http://www.networkcomputing.com/ibg/Chart?guide_id=3484*

Network Computing WAP Buyers Guide: *http://www.networkcomputing.com/1223/1223buyers2.html*

NoCat WRP: *http://nocat.net/ezwrp.html*

NoCatAuth: *http://nocat.net/*

NYC Wireless: *http://www.nycwireless.net/*

O'Reilly's wireless starting point: *http://www.oreillynet.com/wireless/*

OpenAP project: *http://opensource.instant802.com/*

Personal Telco, a co-op based in Portland, OR: *http://www.personaltelco.net/*

RADIUS accounting standard: RFC2866

RADIUS authentication/authorization standard: RFC2865

Seattle Wireless: *http://www.seattlewireless.net*

Sputnik: *http://www.sputnik.com/*

Webopedia page for 802.11: *http://www.webopedia.com/TERM/8/802_11.html*

Wireless Anarchy: *http://wirelessanarchy.com/*

Wireless Ethernet Compatibility Alliance: *http://www.wirelessethernet.org/*

using jails in freeBSD for fun and profit

by Paco Hope

Paco Hope has a M.C.S. from the University of Virginia, where he worked as the head system administrator in the Department of Computer Science. Hope is a UNIX and information security consultant currently consulting with Halyard Systems.

paco@paco.to

Wouldn't it be great if you could create a little sandbox for a bunch of users where they could play to their hearts' content, even have root privileges, but they couldn't actually take the box down? What about isolating that buggy or vulnerable piece of legacy software that you just can't phase out, putting it in its own little world where even if it is hacked it creates a minimal liability on your network? How would you like to install one box in your collocation facility that takes up only two units of rack space but creates the feeling of 100 virtual servers inside?

"Jails" are a relatively recent development[1] in OS technology available in FreeBSD, and they offer the potential applications outlined above. They are similar to the genie's description of his magical servitude in Disney's *Aladdin*: "Phenomenal, cosmic power! . . . Itty bitty living space."

Over the years, various techniques have been created to try to isolate processes, partition resources, or otherwise control the interactions between processes and system resources. These techniques have been motivated by desires to conserve on hardware, consolidate management activities, or isolate risks of harmful interactions between applications. They are most interesting to apply to software that serves some public function, like FTP or DNS.

Near one end of the isolation spectrum are operating system calls, like "chroot," that cause the OS to restrict a regular process to see only a subset of the actual file system. Toward the other end of the spectrum are virtual machine environments. In these environments multiple instances of operating systems run on virtualized hardware. In between these two points is the "jail" system call. This article will describe the existing solutions and their limitations and then explain in some detail what jails can do. That foundation of capabilities will provide the basis for several example applications. A few limitations of jails will be discussed, and then some practical commands for how to actually set up and use jails will be provided.

chroot(2)

A chroot environment is one in which a process's view of the file system is restricted to a specific part of the hierarchy. The process's file system has a virtual root. Probably the most readily available example of a commonly chrooted process is the FTP daemon. Most ftpd programs use a small, partial copy of the file system rooted at the virtual root of the FTP hierarchy. The directory /var/ftp is actually the root for the FTP daemon. That is why /bin and /etc and /lib directories are often found on public FTP sites. There is actually a /bin/ls program, which corresponds to the real file /var/ftp/bin/ls, and it is executed when an FTP daemon services a client's ls request. FTP is not the only process commonly chrooted these days. IMAP software and DNS software are both commonly chrooted as well.

The chroot solution is attractive for protecting certain kinds of vulnerable processes. It is usually used to guard against software that can be coerced to read or write files that it should not touch in normal operations. Such vulnerabilities are limited in scope to just the chroot area of the file system. Thus the files available to such a vulnerable process are dramatically reduced.

There are significant limitations to chroot's applicability. Though a process may be chrooted, it is not restricted from opening network sockets, creating special device files, or seeing other processes. A process running as "root" in a virtual file system is still a process with full administrative privileges and the ability to interfere with other running processes on the system. Furthermore, exploits for chroot exist (see *http://openbsd.org.br/ouah/chroot-break.html*) that enable programs to break out of their chrooted directory. Chroot is rarely used as a technique by itself but, instead, is combined with other best practices to create a safer environment.

Virtual Operating Systems

Virtual operating systems present a virtualized view of the entire system hardware to allow multiple instances of operating systems to run simultaneously on the same hardware. When that emulation is done well, the operating systems cannot distinguish simulated hardware from actual hardware. A single server can function as many distinct servers, and each instance of an operating system can be fully partitioned from all other instances. All the running operating systems must be managed like separate servers, though, because they essentially are.

Virtual operating systems are attractive in some contexts because they tend to offer very flexible configurations. Each and every instance can be completely controlled separately and independently. In fact, different operating systems can peacefully run simultaneously on the same hardware. There are management advantages to having a single physical system running different operating systems. There can be cost savings to operating a single physical system instead of multiple systems.

The primary disadvantage is that simulating the hardware can be expensive in terms of system resources. Each instance of the OS must have its own disk resources for its file system. The hardware virtualization and mediation is not free; it takes CPU cycles away from the applications themselves. In some contexts a completely separate instance of the operating system is overkill for the level of isolation that is needed.

There are good reasons to use virtual operating systems for certain classes of applications, just like there are good reasons to use chroot environments. Jails, by comparison, fit neatly in between. Jails offer a very compelling cost-benefits ratio for certain classes of problems where a fully virtualized system is too much cost or trouble but chroot is not robust enough.

What Jails Do

Jails combine the virtual file system approach with a limited amount of resource mediation to achieve a middle ground. For instance, creating and managing a jail feels very much like creating and managing a chroot environment. However, the operating system restrictions on jails far exceed a chroot environment and feel more like a virtual machine. The kernel mediates access to global system information and network resources, controls the creation and use of special devices, and logically isolates jailed processes from the main system and from each other. This makes jails safer from a security point of view, and gives jails a more complete feeling of isolation.

What follows is a general explanation of what jails do, why they are interesting, and what they can do for a system administrator.[2] For purposes of this discussion, we will call the regular, unrestricted operating system the "host environment" and the restricted jail environment the "jail environment."

Virtual operating systems are attractive in some contexts because they tend to offer very flexible configurations.

. . . packet sniffing is not possible within a jail.

Jails Limit TCP/IP Access

Processes in a jail are limited to a specific set of TCP/IP socket operations and a single IP address. Typically the host environment is "multi-homed," meaning that the single physical system uses multiple IP addresses.[3] One of these IP addresses is assigned to the jail when the jail is first started. The kernel ensures that every packet leaving the jail environment has this assigned address as its source address. Raw sockets are disabled inside a jail environment. Even ping does not function from within a jail. This means that no spoofed packets originate from a jail; all packets will have the correct source IP address.

Jailed processes are also restricted in the kinds of packets they can receive. In a normal UNIX environment, software that binds a socket on INADDR_ANY (i.e., 0.0.0.0) will receive packets for all legitimate IP addresses on the system. A jailed process, however, only receives packets that are destined for its IP address, no matter how the socket is bound. Furthermore, promiscuous mode for network devices is prohibited to jailed processes, meaning that packet sniffing is not possible within a jail. All of these restrictions add up to a significant improvement in network security. If a hacker should infiltrate a jail, they will find many of their tools inoperative or severely limited.

Similar to TCP/IP are the inter-process communication (IPC) functions, which are also limited by jails. If jailed processes are allowed IPC functions like msgsnd and semctl, they can conceivably affect other processes running on the system. In FreeBSD 4.5-RELEASE, there is a system-wide control for allowing all jails to either do all IPC functions or none. Robert Watson's jail NG work breaks these controls into a per-jail setting,[4] but IPC access is still a binary, all-or-nothing proposition. By default IPC is not permitted to jailed processes, which makes them safer from each other.

Most of these limitations do not interfere with normal operations of normal processes. Mail servers, Web servers, DNS servers, and almost everything else that uses TCP/IP can operate normally in a jail with little or no reconfiguration. They do, however, provide a barricade that is a significant and useful part of an overall security regime.

Jails Limit File System and Device Access

The kernel also prevents the creation of special-device files by jailed processes, and it moderates the use of the special-device files that are available (e.g., /dev/kmem). The administrator should be careful what /dev entries exist in a jail, since those that do exist can be used. Devices that do not exist in /dev, however, cannot be created by jailed processes. The kernel prevents jailed processes from executing the mknod() system call. To simplify things, however, the MAKEDEV script has a "jail"[5] option that makes only those devices which are appropriate and/or necessary in a jail. Because a process in a jail has no access to the direct disk devices, it cannot grope around on the raw disk for data outside its prescribed perimeter. This, too, improves the security of a jail and helps complete the barricade between it and its host environment.

Jails Mask Processes and ID Spaces

Jails are isolated from each other and from the host environment itself. The user IDs (UIDs) and group IDs (GIDs) used inside a jail are the same as those used in the host environment. However, the operating system considers the jail identity (JID) as well, for purposes of determining access and privileges. The superuser in a jail can start and stop processes, send signals, and do many things, but the superuser's effects are limited

to processes with the same JID. Likewise, any processes running as root within a jail can affect other processes, but only those with the same JID.

The quasi-isolation property of jails is the key distinguishing feature between jails and the two other approaches. A process running as root in a chroot environment is only limited in its view of the file system. It can still send signals to processes, reboot the system, open network sockets, etc. In a virtual operating system where an entire instance of the OS is running on simulated or arbitrated hardware, no instance of the OS is limited by the virtual environment. A compromised virtual OS is the same as any normal compromised system. Within a jail, however, a process running with "root" privileges actually has very limited abilities with respect to the rest of the host system. Jails offer just enough functionality to do a lot of legitimate work while isolating and limiting processes' access to unrelated resources.

Using Jails to Your Advantage

Jails can make sense on several different levels. They can save money, isolate risk, and offer an attractive virtualization technique. This section presents some of the benefits and some examples of how jails can be applied to specific problems.

JAILS CAN SAVE MONEY

Every IT department wants to save money, and jails might actually help. Jails can allow a few physical machines to serve as many virtual servers. Since most collocation facilities factor the physical dimensions of servers into their overall charges, fewer physical boxes will lower collocation fees. Even an organization that has no collocation charges to consider will appreciate the simplicity and cost-effectiveness of creating a new jail on an existing server, rather than purchasing new hardware when a new service must be provided.

Since jails are really a subset of the operating system, they can be upgraded en masse differently than real hosts. With some prior planning and automation, jail creation can be automated easily. This also means that jail upgrades can be rolled out easily, since taking down and restarting a jail is a very limited operation. This can translate into time and money savings by reducing management labor.

JAILS CAN BE A SECURITY TOOL

Implementing jails can offer another barricade in the network security battle. Every company must have some number of systems connected to the Internet. By isolating individual services in jails, the impact of a compromise or denial of service can be carefully managed. For instance, a machine with five jails might run a database, mail server, Web server, FTP server, and DNS server each in its own jail. A compromise in one jail would not necessarily lead to a compromise of the host environment or any other jails. The TCP/IP and socket restrictions limit what tools an attacker could use even if they got the foothold of "root" inside of a jail.

JAILS HELP LOGICALLY COMPARTMENTALIZE SYSTEMS

ISPs or IT departments that have very independent user bases may find the virtualization of jails to be an attractive way to partition administrative access. If the Web staff demands full and unfettered access to the Web server, they can have it – in a jail dedicated to the purpose. Now they do not need privileged access to a key server in order to operate just one of its many services. If there are junior administrators who need to manage a few services (for example, DNS and DHCP), those services can be put inside

Jails can allow a few physical machines to serve as many virtual servers.

a jail where the junior administrators can have what access they need. Interestingly, all the existing techniques for compartmentalization and security still function within jails. So commands like sudo(8) or techniques like chroot can still be used inside a jail to further restrict access, or to impose finer-grained security. An ISP that wants to offer virtual hosting to advanced users can use jails to great effect. The user can appear to have "root" access inside their jail despite the fact that they actually have only limited access to the machine itself.

AN EXAMPLE USE OF JAILS: VIRTUAL HOSTING

Consider a system where an ISP wants to offer virtual hosting to its customers, but without allocating a full system and rack space to each customer. After establishing a baseline standard jail to serve as the per-customer virtual host, the ISP can add such customers quickly and easily.

The ISP allocates an IP address to one of its customer servers, and establishes its standard jail on one of its servers using that IP address. The jail runs sshd for secure login access, some kind of Web server, mail server, FTP server, and perhaps even a DNS server. The customer can login and have access to the real, live configuration files for all the important servers. If they pay the appropriate premiums and are sufficiently motivated (and competent), they can install new Web server modules, mail server configurations, FTP login IDs, or whatever especially suits their needs.

This approach to virtual hosting is interesting because giving a customer free rein in their jail does not interfere with any other customers at all. If the customer misconfigures their Web server such that it won't even start, all the other Web servers (in other jails) run unimpeded. If they have specific needs or specialized requirements, it is easy to provide for them in the context of their own jail.

As a variation on this theme, the ISP could install Webmin[6] in the jail and give selective access to selective subsystems in a controlled way. The user has the illusion of full control of the system, so they are happy. The IT staff have absolute control of the actual system, so they are happy.

Tips and Tricks with Jails

JAILS AND FLAGS

Jails can combine with another BSD file system feature, flags, to make them administratively safer, cleaner, and more easily managed.[7] The files which make up the operating system (e.g., /bin, /usr/lib, /sbin, etc.) can be made "immutable" using the chflags(1) command. Even an inept administrator or a joyriding hacker vandal will have a hard time completely ruining the jail. Immutable files can be made such that root (even root in the host environment, if desired!) cannot modify them. To make the operating system honor immutable flags, the kernel's security level must be set higher than the default.[8]

HARD LINKS ARE YOUR FRIENDS

Every jail requires a copy of the operating system – or at least enough to run the necessary software properly. This requirement can make jails disk intensive. A minimal FreeBSD installation with a few interesting services is likely to use between 50 and 100MB. The naïve approach to building jails would create one copy of this hierarchy per jail. However, the vast majority of these files need not be unique to the jail. In an application where many jails are likely, hard links can save a lot of disk space.

To make a hard-linked copy, one must first copy the directory hierarchy, and then recurse the source hierarchy and use the link(2) system call (or the ln command, which is equivalent) to create hard links between all the files in the source and the corresponding destinations.

In a system that has many jails, the hard-linked hierarchy allows the inode and data cache in the kernel to work at optimum efficiency. Consider a system with 10 httpd processes in 10 jails. The naïve approach would cause 10 copies of the httpd binary and copies of the corresponding shared libraries to be loaded into separate process address spaces in RAM. However, if 10 httpd processes and all their shared libraries are all hard links to the same inodes, the operating system can be much more efficient. Only the data blocks for the one httpd binary are loaded in the kernel's disk cache. A single text segment can be shared in RAM by all the running httpds, since they are all loaded from the same inode. There are several other subtle ways in which this scheme allows for shortcuts and efficiency in the kernel. Most of these efficiencies, however, are only realized on an active system with many, many jails. A two-jail system, for instance, would benefit less noticeably.

Hard links create a security concern because they create resources that are shared between jails. Shared files cross the otherwise rigid boundaries and potentially allow one jailed process to write to a file that many other jailed processes might read. Using hard links for disk space efficiency almost demands using the BSD flags above for additional protection.

Some Limitations of Jails

METERING, MONITORING, MANAGING

Most operations in jails are not especially mediated by the operating system. It is not possible, for instance, to dedicate a CPU to a jail, or regulate CPU usage by particular jails. That lack of control cuts both ways. It means low management overhead so that more CPU cycles and RAM are available to the regular processes. It does make it hard, however, to account for resource usage on a per-jail basis on a server that has hundreds or thousands of jailed processes.

Starting and stopping individual jails can be scripted, but there are no extant management tools yet that make it easy. Starting a jail by running /etc/rc in it works fine, but it is not possible to stop a jail in the same way a system is normally shutdown (i.e., /etc/rc.shutdown). The ability to inject a new process into an already running jail is critical to most management functions, but is lacking in the current jail implementation. JailNG fixes many of these limitations and makes jails easier to manage.

JAILS SOAK UP IP ADDRESSES

Each jail needs its own unique IP address. So the example above of a server with five jails would use six IP addresses: one for each jail and one for the host environment. Inside a firewall where private IP addresses are plentiful, this does not pose a significant problem. It may be significant to some organizations, depending on how they arrange their IP address space. One way around this limitation is to use a NAT mapping at a firewall entry point. Jails can then be assigned private IP addresses. The NAT mapping can direct port 25 connections (email) to one jail's private IP, while directing port 80 connections (HTTP) to a different jail's private IP, and so on.

NO DISK QUOTAS

Limiting the disk resources used by a given jail is difficult, because the operating system's quota facilities cannot be brought to bear on the problem. It is sometimes desirable to impose disk quotas on given jails, or to be able to impose quotas on users inside jails. There are no easy cookie-cutter solutions to this problem. A search on the FreeBSD mailing lists will reveal some attempts at launching jails in virtual (vnode) file systems. A discussion of such file systems is beyond the scope of this article, but they involve creating a large file (e.g., 100MB) and treating that file as if it were a disk device. By formatting the contents of the file as if it were a disk, and using a special mount command, the file's contents can be mounted as a file system. Since the file's size is fixed (100MB in our example), that imposes a hard quota on the entire jail. It does not, however, impose quotas on individual users inside the jail. It's a very coarse measure. This method is also complex and awkward. Increasing the quota is possible, but tricky. Backing up and restoring such file systems is very difficult as well.

Making Jails: Two Techniques

TECHNIQUE ONE: BUILD WORLD

This is what the man page recommends. It works, albeit a bit slowly. To summarize from the man page:

```
D=/here/is/the/jail
cd /usr/src
make world DESTDIR=$D
cd etc
make distribution DESTDIR=$D NO_MAKEDEV=yes
```

There are a few other options that will make the build proceed faster and will limit how many programs get built and installed in the jail. Several options can be enabled in /etc/defaults/make.conf such as NO_SENDMAIL, NO_LPR, and NO_BIND. They prevent large subsystems from being built, which will speed the build time and produce a smaller jail, assuming none of those subsystems is desired in the jail.

The disadvantage to this approach is the build time and disk space required to make "world" from sources. The advantage is the fine grain of control that's possible. To avoid installing compiler tools in the jail, for instance, (a prudent security measure), the directories that correspond to them can be removed from the /usr/src/gnu/usr.bin directory before building "world."

TECHNIQUE TWO: UNPACK DISTRIBUTIONS OFF THE CD

This method is simpler and faster by far, but has less fine-grained control. With the distribution CD mounted or copied into some file system location, each component of the operating system can be installed using its install.sh script. By setting the DESTDIR environment variable, each install.sh script will install its software in DESTDIR instead of overwriting the existing OS installation. A minimum jail should probably consist of the bin distribution and crypto distribution:

```
export DESTDIR=/here/is/the/jail
cd /cdrom    # or wherever your FreeBSD distribution lives
cd bin
sh install.sh
cd ../crypto
sh install.sh
```

The reason this technique has less control is that the entire bin distribution is installed, with compiler tools, sendmail, BIND, and many other programs. If for no other reason than space efficiency, a typical jail probably does not need most of what is installed in the bin distribution. From a security standpoint it is always best to only install exactly what is needed and nothing more. Installing from distributions will require some manual cleaning afterwards to remove unwanted software.

SETTING UP AFTERWARDS

After building a directory hierarchy using either technique above, there are routine chores to do in order to make the jail usable. The devices in /dev must be created, the root password should be set and a few other things must be set specially for a jail. Each command can be launched in the jail until the jail is able to run by itself:

```
cd $DESTDIR/dev
sh MAKEDEV jail
cd $DESTDIR
ln -sf dev/null kernel
touch etc/fstab

/usr/sbin/jail $DESTDIR jail-hostname 10.2.3.4 /usr/bin/passwd root
/usr/sbin/jail $DESTDIR jail-hostname 10.2.3.4 /usr/sbin/adduser
/usr/sbin/jail $DESTDIR jail-hostname 10.2.3.4 /bin/sh /etc/rc
```

The last command boots the jail. Assuming the jail's IP address is 10.2.3.4 and the host environment's networking is correct, it should now be possible to connect to it via SSH. Once logged in, it feels very much like a normal system. The man page for jail(8) discusses various modifications to /etc/rc.conf in the jail environment. It is also important to remove some programs from the jail that can leak information. There is no use for mount(8) or any of its related modules in a jail, and it's arguable whether any compiler tools belong in a jail. Once a good jail baseline is established, though, it's very easy to copy it to make more safe jails.

Conclusion

Jails enable system administrators to build relatively safe sandboxes that feel like virtual environments but have very low computational overhead. They are handy tools for system administrators to have in their toolboxes for certain classes of problems. While not a panacea, jails allow a number of configurations that have not been previously possible in the free UNIX operating systems and commercial desktop operating systems.

Adrian Filipi-Martin (*adrian@ubergeeks.com*) contributed to this article.

REFERENCES

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

2. For a detailed discussion of how the operating system actually implements jails, see Kamp and Watson, "Jails"; E. Sarmiento, "Inside Jail," *Daemon News*, September 2001. *http://www.daemonnews.org/200109/jailint.html*

3. Multi-homing is accomplished through a technique commonly called IP aliasing. See the ifconfig command for more information on IP aliases.

4. R. Watson. "JailNG: From-Scratch Reimplementation of the Jail(2) Code on FreeBSD." *http://www.watson.org/~robert/freebsd/jailng/*

5. See Kamp and Watson, "Jails"; Sarmiento, "Inside Jail."

6. Webmin is a Web-based UNIX administration tool. *http://www.Webmin.com/*

7. See the man page for security(7) for more information on kernel security levels.

8. See *http://www.Webmin.com/*

the bookworm

BOOKS REVIEWED IN THIS COLUMN

IP ROUTING
RAVI MALHOTRA
Sebastopol, CA: O'Reilly &Associates, 2002. Pp. 219. ISBN 0-596-00275-0.

THE PROCMAIL COMPANION
MARTIN MCCARTHY
Edinburgh, Scotland: Addison-Wesley, 2002. Pp. 235. ISBN 0-201-73790-6.

J2ME IN A NUTSHELL
KIM TOPLEY
Sebastopol, CA:O'Reilly & Associates, 2002. Pp. 450. ISBN 0-596-00253-X.

THE J2EE TUTORIAL
STEPHANIE BODOFF, ET AL.
Boston, MA: Addison-Wesley, 2002. Pp. 491 + CD-ROM. ISBN 0-201-79168-4.

JAVA IN A NUTSHELL, 4TH ED.
DAVID FLANAGAN
Sebastopol, CA: O'Reilly & Associates, 2002. Pp. 969. ISBN 0-596-00283-1.

EMBEDDED LINUX
CRAIG HOLLABAUGH
Boston, MA: Addison-Wesley, 2002. Pp. 419. ISBN 0-672-32226-9.

LINUX ADMINISTRATION HANDBOOK
EVI NEMETH, ET AL.
Upper Saddle River, NJ: Prentice Hall, 2002. Pp. 889. ISBN 0-13-008466-2.

VMWARE
BRIAN WARD
San Francisco, CA: No Starch Press, 2002. Pp. 249. ISBN 1-886411-72-7.

INTRODUCTION TO PROGRAMMING IN EMACS LISP, 2ND ED
ROBERT J. CHASSELL
Boston, MA: FSF, 2001. Pp. 292. ISBN 1882114-43-4

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 is Chief Knowledge Officer at Matrix.net. He owns neither a dog nor a cat.

peter@matrix.net

I've got a heap of books I want to talk about this month. And a poster, too, for those of you with space on a wall between "User Friendly" and "Dilbert" clippings.

Getting Around

There are currently over 175 million host machines on the Internet (up from 213 in August 1981). Getting mail, VoIP, streaming video, http[s] packets, ftp and scp packets, AUDIO, and other stuff from box to box is not trivial. I have relied on folks like Huitema and Perlman to elucidate the complexities of routing in the past. Malhotra's small (barely 200 pages) book will now live next to my desk. Though ostensibly about Cisco routing, Malhotra's expositions of RIP, IGRP, EIGRP, RIP-2, OSPF, and BGP-4 are good enough for those working with Juniper, Nortel, etc., hardware. Definitely a must have!

Incidentally, if you're interested at all in Internet history, Peacock (*www.peacockmaps.com*) has a really fine "First Maps of the Internet" poster for $29.95. Just right for covering the hole in the plaster you tossed that CPU through . . .

/dev/null

I get a lot of email. Much of it offers me $$$ or XXX or and opportunity to buy Viagra or enlarge my breast size. For several years, procmail has been my friend: I'm now down to about 100 messages a day; some of them actually meaningful. Marty McCarthy has written a fine book on setting up and using procmail. (I have to confess that I read and commented on an early manuscript and wrote the Foreword to McCarthy's book. I have no financial interest in it.)

Cups of Java

J2ME is the Java 2 Micro-Edition: it is designed for resource-limited devices like cell phones or pagers. It is really cleverly designed. Topley has done the excellent job that I've come to expect from the Nutshell handbooks and references.

J2EE is the Java 2 Enterprise Edition. This tutorial is full of real examples and thoroughly useful pointers on just how J2EE can be used in your enterprise. The CD contains three (!) J2EE tutorials, J2SE and J2EE software development kits, a sample Java BluePrints, and the Forte for Java plug-in. Whew!

For over five years, Flanagan has occupied a prominent place on my bookcase. The 4th edition of his "Desktop Quick Reference" has gained a lot of weight: just over doubling the 438 pages of the 1996 version. Hardly any flab, though.

A Pair of Penguins

It's no secret that I'm a Linux user and a Linux enthusiast. Over the past two years or so, more developers have turned to Linux to provide solutions for embedded systems. Hollabaugh's presentation is more than merely adequate, but I have to admit that I still prefer John Lombardo's presentation (published by New Riders last year).

And here's the book for every penguin-user! Nemeth's UNIX handbook has spun through several editions since 1989. Each one more impressive than the previous. Now here's the Linux version, written by Nemeth and her colleagues, Garth Snyder and Trent Hein. It covers Red Hat, SuSE and Debian. It is well-written. If you use Linux or if someone on your site uses Linux, this book is indispensible. Thanks, Evi. Thanks, Garth. Thanks, Trent.

book reviews

Virtual Machines

A virtual machine enables the operator to pretend to use one OS while running on top of another. VMWare does this for a variety of Windows platforms, Linux and FreeBSD. Ward's presentation is good, though I have a minor problem: my notion of how to "get the most out of Windows" is to just run something else.

Lisping

It has been over a decade between editions of Bob Chassell's Emacs Lisp book. The new edition contains a really fine tutorial as well as the new features included in GNU Emacs v21. It's definitely worthwhile. Bob, another fine piece of work.

IP SANS: A GUIDE TO ISCSI, IFCP, AND FCIP PROTOCOLS FOR STORAGE AREA NETWORKS

TOM CLARK

Boston: Addison-Wesley, 2002
ISBN: 0-201-75277-8

Reviewed by Steve Reames
reames@diskdrive.com

Storage Area Networks (SANs), and storage networking in general, are becoming increasingly important components in corporate data centers. Traditionally dominated by Fibre Channel, a recent plethora of IP-based standards are competing to either augment or replace the entrenched standard. Clark has done an excellent job of putting together the important aspects of these new protocols, and comparing and contrasting them in a fair and even-handed way. If you have anything to do with storage networks, you need to read this book.

IP SANs is just over 280 pages long and packed with illustrations. It would be easy to jump into the technical details of the standards, but instead Clark takes his time and covers the background material that will be needed later. The first few chapters cover shared storage, Fibre Channel, SCSI, and TCP/IP. The level of coverage is introductory, and the knowledgeable reader can skim through these chapters. But network experts learning about storage or storage gurus who need to learn about networking will find one or more of these chapters worth careful reading.

Chapter 8 is the heart of the book, where iSCSI (Internet SCSI), iFCP (Internet Fibre Channel Protocol), and FCIP (Fibre Channel over Internet Protocol) are discussed and contrasted. Clark's extensive Fibre Channel background really shines here as he is able to examine how these new protocols interact with existing Fibre Channel systems. You can spend weeks combing the standards (I've done it!) trying to learn what Clark clearly explains in a few simple pages.

It turns out that these protocols do not really stand on their own, and the next three chapters are devoted to iSNS (Internet Storage Name Server), security, and QoS (Quality of Service). The iSNS protocol provides naming and discovery services that are inherent to Fibre Channel, but require an additional server for IP-based storage protocols. The chapter on security discusses the issue from two perspectives. First, security in Fibre Channel SANs is discussed. This gives the reader the needed perspective for the second section, which covers the same issues for IP-based SANs.

A short chapter on Infiniband gives some insights about how this interface may or may not become part of SANs in the future. The chapter on SAN applications is thorough and detailed, and probably provides the most extensive collection of real-world scenarios yet compiled. Even if you have a Fibre Channel SAN and intend to stay with it, you need to read this chapter to understand how the industry is developing. A final chapter of conclusions examines the potential of IP SANs, and outlines the things that need to happen for IP SANs to succeed.

No book is perfect. In the explanation of RAID (Figure 3-1 in the book), a "0 + 1" RAID is shown as a mirror of two striped arrays (RAID 0, then 1). Almost no one implements it this way; instead, they stripe together a set of mirrored drives (RAID 1, then 0). On page 174 the lower drawing should read "Tunnel Mode Security Association" as opposed to "Transport Mode." To spend more than a couple of sentences on the book's flaws would be an injustice to the tremendous collection of knowledge contained within. This book is destined to stand for many years as the reference on IP SANs.

SOFTWARE FOR YOUR HEAD: CORE PROTOCOLS FOR CREATING AND MAINTAINING SHARED VISION

JIM AND MICHELE MCCARTHY

Boston: Addison-Wesley, 2001. Pp. 464.
ISBN: 0-201-60456-6

Reviewed by Steve Johnson
yacc@yaccman.com

This is one of the more unusual books on software engineering available today. Some of the ideas it proposes are brilliant, some are weird, and some are both.

In an earlier book (*Dynamics of Software Development*), Jim McCarthy states that "Software is the process of turning ideas into bits." He goes on to make a convincing case that the big problem in software engineering is not the individual programmers' abilities to turn their individual ideas into bits but, rather, "aligning all the ideas in the various programmers' heads." If the ideas are aligned, any problems one person may have realizing these ideas are quickly caught and remedied. If the ideas are not aligned, problems will be frequent, contentious, and difficult to find and eliminate.

Software for Your Head is an attempt to provide some algorithms or patterns a

group can follow that will lead to this alignment of ideas. The underlying premise is that most groups are working at a fraction of their possible potential, and by improving the group functioning you can get large productivity gains and increase everyone's job satisfaction. People who have worked in both high- and low-functioning groups will have no trouble with this concept. The key to the "software problem" is not reading another book on multiple inheritance, or even using the latest requirements language, but getting the group aligned and functioning at a high level.

By this time, many readers are probably stifling yawns. There are lots of "teamwork" workshops that teach people their Meyers-Briggs scores and purport to teach "trust" by hanging on ropes. The authors are scornful of such remedies, which typically do not hold individuals accountable for their actions in the group and do not articulate the desired group behavior clearly.

One concept in particular struck me. The authors believe that every group functions at the level of the least engaged person. In many meetings, some people want to be there and so have a vested interest in seeing the issues resolved; others are just warming chairs. These disengaged people may see things that they could contribute but have the attitude, "Why should I speak up? It will just prolong the meeting . . ."

The authors see this attitude as a loss of integrity. If people do not function at the highest level, they are weakening themselves and the group. And, more to the point, they hold the group (and themselves) back. The authors believe in making high-quality group participation a conscious goal. It is not necessary to be fully engaged all the time. But it is required to be honest with yourself and your coworkers about whether you are "in" or "out" at any time.

The authors also believe that when someone is "out," it is often because they are feeling some emotion (work-related or not) that is interfering with their full engagement. So one way to allow more people to be "in" more of the time is to allow emotions, even negative ones, to be acknowledged within the group. They are not suggesting that software teams become encounter groups or therapy groups – far from it. Just that it should be OK for someone whose daughter is in the hospital to acknowledge it in the group, and through this honesty help the group to become stronger.

Another idea put forth in the book is that the "product" of management is, in fact, the group behavior. So if management is unhappy with the group, it needs to look at its own actions and attitudes for the root cause of the problem. This is the kind of obvious statement that never seems to be obvious to the managers themselves.

The description of group maladaptive patterns, and how to break out of them, accounts for some of the most brilliant and insightful writing. The authors are frequently quite vivid in their language. For example, they discuss the phrase "He gets an A for effort." In other words, they point out, he failed, and he wasted a great deal of time and energy failing. They prefer the motto "Fail Cheap."

I found the authors to be very effective in describing the problems that hold groups back and articulating how an effective group would function. I have more difficulty with the remedies they suggest. These are overly structured, even draconian, for my taste. They may well be effective, but I believe other techniques can also be effective and ultimately less invasive.

There are many examples in our society of cooperation at a much higher level than most software teams achieve. Sports teams are an obvious example. A less obvious example can be found by going to an opera (turning ideas into notes), where hundreds of people execute virtuoso feats with split-second accuracy over a span of several hours, with rarely any major errors. Most software teams would literally think this is impossible.

If, as I believe, cooperation is built into our nervous systems (like language and the ability to reason), creating an effective group should be as easy as creating a safe place for our "cooperation genes" to express themselves. And, sometimes, it is this easy.

Of course, musicians are trained from early on to communicate with other musicians, the conductor, and the audience. Athletes in team sports get similar training. Programmers, on the other hand, are still taught as if success depends solely on one's ability to understand multiple inheritance, data structures, and complexity theory. Not only do students not learn to cooperate, but their professors – role models – function in universities that often prize individual achievement over cooperation as well, reinforcing this isolationism.

Is it surprising, then, that many software people find working in a group boring, difficult, and frustrating? And their managers, promoted because they were the best coders, are not typically trained to break this cycle. In fact, the good programmers with integrity often turn down or resign from management jobs, recognizing that they are contributing to the problem rather than the solution. With this background, perhaps some draconian techniques *are* necessary, and may well be effective.

I invite you to read the book and make up your own mind. It will make you think and may challenge your assumptions; even the ideas that you might not accept are interesting and strangely compelling. And they grow on you.

SAGE news

Two Big Steps Forward for SAGE

by David Parter

President, SAGE STG Executive Committee

parter@sage.org

This month I am pleased to announce two big steps forward for SAGE: the hiring of our first full-time Executive Director and the launching of SAGE Certification.

Late last year, the USENIX Board of Directors allocated two additional staff positions for SAGE, as part of our transition from a mostly-volunteer organization to one with more staff support so that we can offer stronger programs and reach a wider community with more consistancy. The two positions are Executive Director and Web Editor.

Rob Kolstad has been selected as the SAGE Executive Director. Rob is well-known in SAGE, having served in many roles over the years. Rob brings to SAGE his experience and enthusiasm, and we are looking forward to working with him. A short biographical statement follows. You can reach Rob at *kolstad@sage.org*.

The other big step forward has already been reported, but deserves mention again. This spring we launched the first SAGE Certification –"cSAGE" – aimed at junior-level system administrators. Information on SAGE Certification – including study guides and information for sysadmins, employers, teachers and others – is online at *www.sagecert.org*. Certification is one of the key building blocks for system administration to progress to the level of professional recognition we think it deserves, and serves the needs of employers in helping to assure that they have a well-trained and knowledgable professional system administration staff. If you are a junior system administrator check out the study guide and sample questions, and become cSAGE certified. If you aren't sure you are ready – or are a more senior system administrator – check it out anyway. It will give you a roadmap for new things to learn, or you can help some junior sysadmins to get their certification (and the Certification Board is currently laying the groundwork for the next level of certification – for more senior sysadmins).

Two big steps in only a few months? These steps are the result of several years work by many dedicated volunteers. I encourage you to get involved in our next big steps. Check out the web site, and you will find many projects in need of volunteers.

All About Rob

Rob's history with USENIX and system administration is a long one. Besides editing the *;login:* magazine for ten years, he has chaired three USENIX technical conferences, served six years on the USENIX Board of Directors, chaired workshops, and was co-founder of the LISA conference. SAGE awarded him its inaugural Outstanding Achievement award in 1993.

Previously, Rob was program director for the SANS Institute, an educational foundation that specialized in system administration, networking, and security, though it concentrates mostly on security these days.

Rob was with Berkeley Software Design, Inc. from 1992 to 1998, serving as its president for several years. Before that he was lead engineer on Sun's Backup CoPilot project and a VP at Prisma

SAGE, the System Administrators Guild, is a Special Technical Group within USENIX. It is organized to advance the status of computer system administration as a profession, establish standards of professional excellence and recognize those who attain them, develop guidelines for improving the technical and managerial capabilities of members of the profession, and promote activities that advance the state of the art or the community.

All system administrators benefit from the advancement and growing credibility of the profession. Joining SAGE allows individuals and organizations to contribute to the community of system administrators and the profession as a whole.

SAGE membership includes USENIX membership. SAGE members receive all USENIX member benefits plus others exclusive to SAGE.

SAGE members save when registering for USENIX conferences and conferences co-sponsored by SAGE.

SAGE publishes a series of practical booklets. SAGE members receive a free copy of each booklet published during their membership term.

SAGE sponsors an annual survey of sysadmin salaries collated with job responsibilities. Results are available to members online.

The SAGE Web site offers a members-only Jobs-Offered and Positions-Sought Job Center.

SAGE EXECUTIVE DIRECTOR
Rob Kolstad: *kolstad@sage.org*

SAGE MEMBERSHIP
office@sage.org

SAGE ONLINE SERVICES
list server: *majordomo@sage.org*
Web: *http://www.sage.org/*

Technica, a gallium-arsenide SPARC-compatible supercomputer startup. Rob was a member of the original engineering team of Convex Computers and is co-holder of the patent on the Convex C-1 minisupercomputer.

Rob is the head coach of the USA Computing Olympiad, the organization that chooses the set of four pre-college computer programmers to represent the USA in international competitions. He is also the head judge at the Pikes Peak Regional Science Fair.

Rob earned his Ph.D. in high level language constructs for distributed computing from the University of Illinois at Urbana-Champaign after completing an M.S.E.E. at Notre Dame. His undergraduate B.A.Sc. degree from Southern Methodist University was among the first computer science bachelor's degrees offered in the United States.

SAGE Mentoring Program Update

by Strata R. Chalup
Program Chair
strata@virtual.net

The SAGE Mentoring program is ramping up from a long period of inactivity, in which many applications for mentors and apprentices were received. We have had a great deal of feedback regarding folks' interest in senior-to-senior mentoring as well, and are currently considering the best way to implement "mentor matching" so that SAGE members can get in touch with each other directly via the soon-to-be-revamped Mentoring area on the SAGE Web.

If you have submitted an application, and have not heard from us, please accept my personal apologies. I have quite a backlog of applications, including a large tarball of applications from 2000 and 2001, prior to my involvement with the program. I am reviewing those applications, trying to establish good email addresses for people who contacted us in prior years, and refreshing the mailing lists so that they are current. Please feel free to contact me personally if you are in a jam and really, really need to make a mentor connection! And, as always, the *sage-members@usenix.org* list is a terrific resource.

USENIX news

And So It Goes

by Daniel Geer
President, USENIX Board of Directors

geer@usenix.org

A little like writing your own obituary, this is my last column as President of USENIX. We have term limits here and, all in all, that is a good thing. I may still find something to do around here, but I welcome the new Board and President Kirk McKusick and wish them well.

Over the time I've been associated with USENIX an awful lot of changes have transpired. In that regard, USENIX mirrors changes going on in society in general due in large part to technical change – our communications are overwhelmingly electronic when they are not face to face, distance is measured in either network latency or the dollar cost of airline tickets, security is no longer a geek-only issue (which is arguably a fearsome development), it is much cheaper to retain electronic data in full than to selectively cull it, and it is the capital markets rather than tenure committees that ultimately sort technologies into winners and losers.

Running an organization like the USENIX Association is getting harder at least insofar as our core mission is concerned. We exist to accelerate the advance of knowledge in our field, what Mike O'Dell famously called "moving information from where it is to where it is not." The reason running USENIX well is getting harder is that our success to date does nothing so much as raise the standard to which we now have to perform.

In the meantime, the latency between invention and exploitation is shrinking – which fundamentally is a good thing in the bigger scheme of things. The human dynamics of scheduling new workshops around emerging topics and disciplines (which is exactly where we can do the most good) are such that USENIX pretty much cannot get anything effective done in under 9-12 months despite the fact that, just as in business, there is a really substantial and growing first mover advantage to that professional society that best calls trends right, that has the first meeting on a new topic at exactly the right time in exactly the right venue.

Knowing when to strike, what to strike, and who needs to be involved are at the critical core of the risk-reward tradeoff that USENIX as a business lives within. But let me be clear about something: For those of you who have something to say and who want to create vehicles for knowledge transfer, USENIX is absolutely remarkable. No other professional society relieves you of as much logistical detail. No other professional society comes close to the price-performance value that USENIX delivers. No other professional society has meetings where the signal-to-noise ratio is more favorable.

Sure, I am a true believer, but I invite you to try to get something together under the umbrella of any other group and compare it to what you can get done here. Measure your experiment in how much time you have to put in versus how much intellectual value you get out. Measure it by looking at the quality of our Proceedings and don't take my word for it, use CiteSeer (*http://www.citeseer.org*) to confirm that when measured by citation frequency USENIX meetings are tops. Yeah, USENIX is cliquish but at no other society can you expect to talk to

the actual authors of the tools you can't live without.

Want to call the USENIX regulars elitist? No problem as far as I am concerned since every bit of elitism in these here parts is earned. USENIX is about getting things built and recognizing those who get it done. Just as the IETF is the dominant standards organization because of its simple creed, "Rough consensus and working code," so, too, our simple idea is that USENIX is where you go for that working code; it's where you go if you want to know what actually works rather than what might be theoretically interesting.

Were it not such a low bar, I'd point out that we have an awful lot better taste in what is "novel, non-obvious and reducible to practice" than the US Patent & Trademark Office does, not to mention that we are becoming more selective in our selection process in the face of a rising volume of submissions.

As I have said here several times before, I'd like to urge those of you with ambition to remember that it is never too early to choose whether you are going to lead, follow or get out of the way. I'd ask all USENIX members to recognize that if you want something to happen the surest way to get it to happen is to realize that you have more leverage here than you are going to get anywhere else. Samuel Johnson observed that knowledge is of two sorts, where you know a thing yourself and where you know where to find out about it.

Let me tell you a secret of career success: That secret is simple – You can either (on your own) go out and scour the countryside for knowledge, or you can (by serving on USENIX program committees) get the countryside to bring its best work to you. I recommend the program committee approach; you will find that it concentrates the interesting traffic like nothing else, and all you have to pay is your time while all you have to risk is your reputation. Expensive and scary? Sure, but consider the alternative.

I've asked each of the current USENIX officers to do something that has not been part of the USENIX managerial tradition heretofore and that is to write a report on their term of office, what got done and what didn't. This is not an easy thing to do – USENIX is a lot more complex than it seems or, to put it differently, that it looks simple from the outside is a triumph rather than just something that you can buy at the store like milk.

As such, it is actually hard to write in one document something that is readable by the mildly curious, meaningful for the serious student, reassuring for the well-wisher, and a counterweight to the heckler. That it is hard to be at once universal and concrete is precisely why it is a good thing to try, even when the real role of an individual Board member is much more like an adverb than a noun, i.e., we modify more than we pre-empt. I'll finish my report when my term is actually over, which will be about when you read this. Read it if you care but don't if you don't.

In the meantime, I want to thank you all for the opportunity to lead over the past decade and a half, and just as it is the duty of a teacher to be surpassed by his students, it is now your job to prove that you're better than I was.

Godspeed.

2002 USENIX Board of Directors Elections Results

The results of the elections for Board of Directors of the USENIX Association for the 2002-2004 term are as follows. Names in bold are the elected officials.

Newly elected directors will take office at the conclusion of the next regularly scheduled board meeting which will be held June 14, 2002 in Monterey, California.

Ballots Received:	1295
Invalid Ballots:	17
Participating Ballots:	1278

President:

Marshall Kirk McKusick	1100
Abstain	159
No Selection	19
TOTAL	1278

Vice President:

Michael B. Jones	679
Trey Harris	543
No Selection	56
TOTAL	1278

Secretary:

Peter Honeyman	758
Steve Simmons	479
No Selection	41
TOTAL	1278

Treasurer:

Lois Bennett	1117
Abstain	126
No Selection	56
TOTAL	1278

Directors:

Jon "maddog" Hall	829
Tina Darmohray	688
John Gilmore	662
Avi Rubin	656
Ted Ts'o	530
Æleen Frisch	401
Peg Schafer	357
Darrell Long	282
Adam Moskowitz	206
Clem Cole	192
James Yaple	117

Summary of the USENIX Board of Directors Actions

by Ellie Young

Executive Director

ellie@usenix.org

The following is a summary of some of the actions taken by the USENIX Board of Directors between mid-November 2001 and April, 2002.

Conferences

BSDCon: It was decided to sponsor another BSDCon in approx. 18 months (Fall of 2003), with a goal to improve the quality and attendance, while keeping costs low. Gregory Neil Shapiro will serve as program chair.

FAST: After a successful inaugural conference in January, it was decided that another conference will be held in 2003 with Jeff Chase as program chair.

Linux Kernel Summit II: It was agreed that USENIX will co-sponsor another summit with OSDN in June 2002.

Mobisys: An agreement between ACM SIGMOBILE and USENIX to co-sponsor a conference on mobile systems, applications and services was signed.

Good Works

Open AFS Project: It was agreed that USENIX will commit $35,000 to this effort contingent upon the receipt of matching funds from at least two other donors. USENIX will oversee the distribution of funds to the Open AFS council of elders (currently CMU, U/Michigan and MIT), and also publicize the donors' support of the OpenAFS development.

It was agreed to fund once again the Status of Women in Computing's mentoring program of the Computing Research Association's Committee on Women (*http://www.cra.org/craw/*) for $10,000. USENIX will also be a sponsor for the CRA's Snowbird Conference in July 2002.

It was agreed to fund $10,000 for student stipends for the Internet Measure Workshop 2002 which is being co-sponsored by ACM SIGCOMM, SIGMETRICS, and USENIX.

SAGE

The following statement was approved and posted on the USENIX and SAGE web sites for a 2 weeks period in December '01: "The USENIX board wishes to apologize to Barb Dijker and Peg Schafer for dismissing them from the SAGE Executive Committee without appropriate due process."

A hiring committee to hire a SAGE Executive Director was formed (Dan Geer, David Parter, and Ellie Young.) A SAGE/USENIX relationship oversight committee composed of respective liaisons (Parter and Hume), Young, and others to be announced was formed.

Finances

Sponsorship and exhibit packages recommended for 2002 were approved.

The recommendation that an outside accounting firm (Burr, Pilger and Mayer) conduct the audit of the Association's 2001 finances was approved.

The 2002 budget was discussed and approved.

News from NUUG

by Jon Petter Bjerke

Jon Petter Bjerke is a NUUG board member and NUUG's primary contact with USENIX.

jonp@nuug.no

NUUG is the Norwegian UNIX User Group, established in 1984 as the Norwegian arm of the EUUG (later to become Europen). In the last several years Europen has existed only as a loosely joined group of European "UNIX" groups.

The NUUG membership of about 300 has been receiving *;login:* on a subscription basis for a number of years, and in the annual meeting of June 2001, NUUG decided to join the USENIX Association as affiliate members. As of January 2002, our members joined USENIX, and a few also signed up for SAGE.

NUUG organizes a monthly technical evening, each on a different subject of interest. We have also held full-day tutorials and national conferences, often with invited speakers from the USENIX side of the Atlantic. We are participating in the yearly NordU conferences as well.

Apart from NUUG, Norway has several special interest computer groups, many of which focus on Linux and open source developments. Some of these groups share space on the NUUG Web server, and events are announced in a common calendar. We are also looking for other ways to cooperate with and support local or special interest groups.

As a result of taking part in the EUnet initiative of EUUG since the mid-1980s and establishing it as a commercial company in the early 1990s, NUUG received considerable money when EUnet was sold in 1998. NUUG decided to put these assets into a separate entity, and the NUUG Foundation was established in 2000. The foundation is actively looking for projects to support, both nationally and internationally. We recently selected the first such project: the "Skolelinux" (Linux for Schools) project, an effort to provide schools with a reliable and easy-to-handle Linux distribution, with all relevant programs supported in the two variants of the Norwegian language (Bokmål and Nynorsk) as well as the northern Sami language. Participation from the other Nordic countries may lead to Danish and Swedish versions of the project.

References and further reading (mostly in Norwegian):

NUUG: *http://www.nuug.no/*
The NUUG Foundation:
http://foundation.nuug.no/
Linux for Schools:
http://www.linuxiskolen.no/

Twenty-Five Years Ago in USENIX

by Peter H. Salus
USENIX Historian
peter@matrix.net

The June 1977 meeting of the UNIX User's Group (not yet USENIX), was held at the University of Illinois at Urbana-Champaign. Knowing that Mike O'Brien (the amanuensis of Mr. Protocol) had been there, I asked him for his recollections. Here they are. (My addenda are in [].)

At the time of the 1977 UNIX User's Group combined East-West meeting in Shampoo-Banana, I was an eager UNIX booster. UNIX Version 5 (and, later, Version 6) was at the same time so interesting and so hard that I couldn't imagine anyone in computing not being as enthralled as I was . . .unless, of course, they worked for IBM, the then-current Evil Empire.

The conference was organized by Steve Holmgren and Greg Chesson, who knew Ken Thompson from Berkeley days or some such. Steve and Greg were responsible for having imported UNIX to the U of I campus downstate, where it replaced the home-grown software on ANTS, the ARPANET Terminal System, which ran on a PDP-11. ANTS' sole winning feature was that it had ants stenciled all along the top panels of the cabinet. It ran like it was full of ants, too. UNIX was a big step up. On the Chicago campus, I had just hired on as a research assistant to a gigantic project ($3 million) to build a medical information system, based on what is probably the worst grant proposal I have ever read. It was funded out of desperation by government officials who had a record amount of money to spend in a hurry due to the Supreme Court decision that told Dick Nixon what part of the purse-strings he did not control.

I was the lone man on the Chicago campus running UNIX, and I dimly remember pestering the daylights out of both Steve and Greg as I came up to speed. I seem to remember that I had founded and was running the UNIX User's Group Software Distribution Center by the time of this conference, so, hopefully, I was less of a pest by the time it rolled around.

Ken and Dennis Ritchie were both scheduled to be in attendance. I think I'd met them before, either at one of Mel [Ferentz]'s get-togethers in New York or one of Lew Law's get-togethers at Harvard. I seem to recall both of those occurring before the big national meeting. I decided to commemorate the occasion. I don't remember much at all about the technical content of the meetings, but I remember this.

I lived in Chicago at the time, and so did a man named Phil Foglio. Phil was a comic artist, later to become rather famous. [Foglio is very active as an artist. Among his credits is *Girl Genius*.] He was in school at the time. I knew him slightly, through Chicago science fiction fandom, in which I was active. One day Phil called me up. His apartment was apparently equipped with a wall safe. His roommate, the only one of the two of them who knew the combination, had locked a prop Star Trek phaser in the safe and blown town. Phil knew that I was, at that time, a bonded locksmith, and wanted me to open the safe.

I knew how to open garden-variety pin-tumbler locks, but I'd never tackled a safe before. I knew that there was an elementary manipulation algorithm that worked on the cheaper sort of combination padlocks, but I figured a safe would be proof against that. Still, I was willing to give it a try. So expectant of failure was I that I brought along another locksmith, a friend of mine, for moral support.

I am amazed to this day that it worked. I opened the safe in under fifteen minutes. I should have written down the name of the manufacturer, to make certain that I never purchased a safe from them.

I had agreed with Phil to take payment in trade. In return for my success in returning his phaser to him, he prepared full-color artwork to my specifications, a now rather famous picture of a PDP-11 cabinet in a maze of pipes, complete with pitchfork-carrying demons running along the pipes. There was a rain barrel with "/dev/null" written on it, but no front panel, due to contradictory specs on my part as to exactly which panel held all the buttons and lights.

At that time, T-shirt "art-to-order" printing houses were few and far between. I found a ma-and-pa operation in suburban Chicago, quite a drive from my place as I recall, who used a 3M color copier to make T-shirts. They were very helpful in turning Phil's artwork into the first UNIX T-shirts ever produced. Because the printing process reversed the artwork, they whited out Phil's trademark signature and carefully forged it in reverse, so that he would get proper credit.

The first four shirts produced were intended for Ken, Dennis, me, and my wife. Only these four shirts were produced with red piping on the sleeves and collars – all shirts made after this had white collars and sleeves. I still have mine and my wife's. As I recall, only Ken made this meeting. I gave him his shirt, and one for him to take back to Dennis. I recently asked Dennis about that, and he could not recall ever having received his shirt. I suspect that the embarrassment at being caught out in this after all these years is responsible for Ken's recent retirement.

The ma-and-pa operation produced several hundred shirts with this artwork in the years that followed. They retained the artwork in their files to fulfill future orders. They told me that their largest single order came from Bell Labs, where about 40 shirts were ordered for a picnic.

Years later, Armando Stettner of DEC asked me about that artwork. He wanted to obtain the rights to it, in order to use it in a marketing campaign for Ultrix. I dug out the phone number of Ma and Pa in suburban Chicago (I lived in Los Angeles by this time) and found out that they had ceased operation years before and were on the point of throwing everything out. They returned the artwork to me, and I sent it on to Armando stating that as far as I knew, I owned the rights, and he (and DEC) could have 'em. The artwork was used in an Ultrix poster showing how much better Ultrix was than that stovepipe clattery amateur UNIX stuff, and the original artwork passed into oblivion, along with the entire Digital Equipment Corporation.

At some point the USENIX board of directors gave Phil Foglio some money to compensate him for the unexpected success of his artwork. [It was in 1986, when I was Executive Director of USENIX. — PHS] He never complained to me personally, but had been heard to gripe that he never saw a dime from the art. I'm glad they did this. I'd never expected the art to go beyond the first four shirts, frankly.

It remains an open question as to the degree of influence that all this had on the BSD Daemon. Kirk McKusick says that John Lassiter created the original BSD Daemon artwork without reference to Phil's work, and I believe him. It's an obvious visual pun. However, I still regard the BSD Daemon as a cultural child of the first four shirts I produced at the Urbana meeting, even if there is no direct connection.

Henry Spencer wore a shirt produced with that artwork for many years, and was by many years the last survivor to do so, until the shirt reached the stage where he was in danger of being picked up for vagrancy. My own are in better shape, partly because I long ago ceased to fit into either mine or my wife's.

The only other clear memory I have of the Urbana meeting is of standing around Steve Holmgren's driveway, at one of the few keggers of any sort I've ever attended. Considering who all was there, it has to rank right up there in terms of "semi-famous people connected with UNIX all in one place doing something non-technical." I recall being as nerdy as possible and also trying desperately to think of something semi-intelligent to talk to Ken about. It was many more years before I became socialized, I'm afraid.

Of course, it was also at that meeting that I was recruited to come work for The Rand Corporation, which turned out to be the making of my career. Yes, in that respect I have mighty fond memories of the Urbana meeting.

[I'm afraid that it would be absurd for me to add to this. Mike, many thanks.]

conference reports

BSDCon 2002

San Francisco, California

February 11–14, 2002

Summarized by George V. Neville-Neil

This issue's reports focus on BSDCon, held in San Francisco, California, February 11-14, 2002.

Our thanks to the Summarizer:

George V. Neville-Neil

KEYNOTE ADDRESS

Software Strategy from the 1980 Time Capsule

John R. Mashey, Sensei Partners

The opening keynote talk by John Mashey treated us to a time capsule with slides used in the original talks from 20 and 25 years ago.

John's first talk, originally titled "Small Is Beautiful and Other Thoughts on Programming Strategies" was a proposal from 1977 to put a UNIX machine on everyone's desk. The argument for this at the time was that terminal room space was expensive, and that smaller projects, done by teams in their offices, would be more effective.

The talk was really about projects and how they get executed. His basic thesis was that there are three ways of doing projects:

1) complete planning (do it right);
2) pessimism (plan to do it over);
3) build small and quick (the UNIX philosophy).

Some of the more interesting facts he presented were that a survey of projects in 1971 showed that out of 18 total, five were infeasible, five were feasible, but could not gain acceptance, two projects had good fall-outs but no real monetary return, three were partial successes, and three were actual successes. Success is very rare.

He then presented a list of how software goes wrong (which, hopefully, didn't surprise anyone in the audience):

1) too many features (creeping featurism); 2) upwards compatibility; 3) intuition about what's important is usually wrong.

What made this interesting is how little things have changed in 30 years.

The "build it quick" philosophy, which he still holds to, says that you need to be ready to throw a project away and that the purpose of each piece you build is insight into the problem space. Another radical idea in 1977 was to use existing tools.

One suggested practice to keep a project small he termed the "Lifeboat Theory," which says that you need to get rid of a feature to add a new one.

His conclusions were again not surprising: 1) failure is the norm; 2) build fast; 3) keep it small; 4) build it to be changed.

His next talk, from 20 years ago, was called "Software Army on the March," a discussion of project organization for projects of about 200 people. The concept is that there are different types of contributors to a project and they can be organized as an army.

There are scouts on motorcycles who are far ahead of the army. They can fail and are expendable. Instead of building the final product, the scouts should learn how to solve the hard problems and find the problems that the rest of the army isn't even seeing as it moves forward more methodically.

Then there are engineers with bulldozers and cranes who build the actual thing you want to ship (the road, bridge, etc.).

Due to the nature of the audience most of the talk focused on the scouts and on decision making. Decision order (what gets done when) was what he felt was most important.

Questions:

1) What do you see as our failures and successes in software?
UNIX is a success even though Microsoft makes more money. But big coalitions have not been a success. There

was a 1967–77 project to automate Bell Telephone. It cost over $1 billion, and people cheered when it died. It was replaced by a small project run by two managers.
2) Why have we learned so little in 20 years?
There are a couple of answers. One of the biggest is that people are not encouraged to talk about failure. We have a poor institutional memory.
There is a certain amount of pessimism in the philosophy, and that goes against the grain.
3) The talks speak to the human condition. The primary problem is that humans are doing this. Most project management is a way of smoothing over or making them work together better. Our ability to understand things does not keep up with Moore's law.

That's actually another talk that I do, "Hardware, Software, Wetware."

Refereed Papers

SESSION: HARDWARE AND DOCUMENTATION

Porting NetBSD to the AMD x86-64: A Case Study in OS Portability

Frank van der Linden, Wasabi Systems, Inc.

The first technical talk covered a port of NetBSD. The biggest general point was that when programming we should never make assumptions about the sizes of types. The assumption that an integer is 32 bits, which was correct for more than a decade, breaks software on 64-bit architectures. This led to the second general point: a programmer should always expect someone else to use their code on a different platform.

One simple recommendation for building more easily ported code was never to use "#ifdef <arch>". The comment was that "if you use this, there is something wrong with you or the code," and you should go back and check your work.

Questions:

1) How is the optimized syscall different from the normal one?
A bunch of unnecessary checks were removed.
2) Is the way that you do the compilations really separate from the host environment?
In NetBSD we are moving toward a different build system which says that "if you have an ANSI compiler you can build our system." This can now be done, which means we support full cross-builds.

Problems Updating FreeBSD's PCCard System from ISA to PCI

M. Warner Losh, Timing Solutions, Inc.

The old method of updating PCCard support in FreeBSD was to have an adapter that made PCI look like ISA, which allowed the old ISA code to work. But this caused two major problems: there was no IRQ sharing, so the system consumed many IRQs, and it was hard to configure.

Experiences on an Open Source Translation Effort in Japan

Hiroki Sato and Keitaro Sekine, Tokyo University of Science

Hiroki Sato presented work done in Japan to keep up with the mainly English source of documentation in the FreeBSD project. The FreeBSD Japanese Documentation Project (doc-jp) was started in 1996 by FreeBSD developers. Its main goal was translation of FreeBSD documents to Japanese. There are two ongoing efforts: the Japanese Manual Project and doc-jp. Currently, 70% of the documents in the source tree are available in Japanese.

One of the major problems the project has to face is that evaluation of translated documents is harder than evaluating code because you can't run the documents. There isn't a good parser for the output of the translation.

Another issue is release engineering because of the lag time between software/doc release and translation. The team is playing a constant game of catch up.

To help alleviate these problems they have developed a toolchain to process the documents. Unfortunately, the JadeTeX system cannot handle Japanese characters at the moment, so there are no PDF or PostScript versions of the documents available.

There were no questions for this speaker.

SESSION: KERNEL STUFF

Locking in the Multi-Threaded FreeBSD Kernel

John H. Baldwin, The Weather Channel

This talk was intended to show kernel programmers how to use the new locking system and tools. The major tool that the FreeBSD team is using is called Witness and was contributed to the project by the BSD/OS team.

Witness is really a set of macros for locking the store state to ensure that locks are always taken in the same order. An out-of-order set of locks can lead to a deadlock situation, which would be a major failure for the kernel.

One of the beauties of Witness is that it learns on the fly what locking order the software is using. Other systems for detecting deadlocks depend on the programmer to declare the locking order for the whole system.

Questions:
1) Is the granularity of locking on entire processes or are there locks on structures?
This was an example of locking for processes but other locks are possible.
2) Do you have any results on how much time you're spending in the locking code?
Not as yet because the Witness system is very CPU intensive.

3) You mentioned that you were going to go through one of the tools. Can you mention others?
One tool is to use assertions. Another tool on the horizon is a lock profiler to tell us which locks we depend on heavily and where we depend on those locks in the code.
4) What sort of lock push-out has occurred? Are there any subsystems depending on fine-grained locks?
Ninety-five percent of the kernel is still under the giant lock although there has been some work on locking file descriptors.
5) What advice do you have for people maintaining device drivers with regard to pushing locks into the drivers? Are there any recommendations on what to do about spl()?
Leave spl() in there for now. Device drivers are the last thing you want to lock. Subsystems should be locked first.
6) What is the difference between the SMP efforts in NetBSD and FreeBSD?
I'm not completely familiar with NetBSD but from what code I have seen they're using different APIs.
7) You were saying that VM had already been converted.
The only thing that has been done is the file descriptors. We have no performance numbers now.
8) What are you doing about priority inversion?
The mutex code from BSD/OS does include some code to protect against priority inversion.

Advanced Synchronization in MacOS X: Extending UNIX to SMP and Real-Time

Louis G. Gerbarg, Apple Computer, Inc.

This talk discussed Xnu, which is Darwin's kernel and is based on Mach 3.0 and 4.4BSD-Lite2.

The system provides a novel synchronization primitive called Funnels, which are like mutexes but they allow non-reentrant code to run safely. It is not a locking construct. At the present time there are two funnels, one for the network and one for everything else.

Another difference in working with Xnu/Darwin is that it depends on a completely different driver model called IOKit. IOKit abstracts the drivers from the rest of the kernel and provides a stable interface.

Drivers handle multi-threading by using IOWorkLoops. They are used to serialize event sources such as timers and interrupts.

IOKit can also filter interrupt events. This makes it easier to handle a large number of interrupts because they appear to the system as a single event.

Questions:
1) You said that you have a network funnel and a kernel funnel when you read from a socket. Which do you get?
The system calls have flags saying which funnel to get.
2) Do you intend to adopt a particular locking strategy from a particular kernel?
I do not handle policy. The answer is that it's very difficult.
3) It seems to me that the IOWorkLoop that you mentioned is similar in concept to the networking ISR code on *BSD. Is that the same concept?
Yes, they are similar, but the implementation details are different. IOKit is more generic.
4) As more and more devices get attached via USB, does that change your viewpoint on how device drivers should be written?
The USB stack does, in fact, hook up functions to handle asynchronous events instead of depending on interrupts.
There is only one IOWorkLoop for all of USB.
5) Have you analyzed the performance of this implementation? What are the interesting effects of the model?
People have done extensive measurements. Just how fast, I cannot say.
6) How many context switches are involved from a NIC card to user space?
I couldn't tell you. Could be less than five, but it is definitely not less than three.
7) Have you benchmarked performance with Darwin vs. something else on the same hardware?
There are answers on the Net. It is not better or worse in totality than anything else.
8) What does the CPU context switch cost?
I don't know.

An Implementation of the Yarrow PRNG on FreeBSD

Mark R.V. Murray, FreeBSD Services, Ltd.

Mark's talk discussed the problem that random numbers aren't random enough.

One of the claimed advantages of this system is that it is number-theoretically correct. The software that was implemented does not block when giving out a random number so it is vulnerable to DoS attacks. Asking for a lot of random numbers does not slow the system down.

Other advantages are that it's resistant to entropy starvation, prediction attacks, as well as being fast and simple to augment.

The target market for this work is the kernel (process IDs, TCP serial numbers) and user-land (SSL/SSH, Kerberos, simulation).

The non-target market is "real" entropy users for things like one-time pads.

The system was designed in an open way from theory to practice.

Questions:
1) Has this been merged from FreeBSD's -CURRENT branch to -STABLE?
No.

2) What is the source of the word "yarrow"?
On *http://www.counterpane.com* take the labs link. A plant with a straw like stalk that is used in China to tell fortunes.

BSD STATUS REPORTS

This session consisted of status reports from representatives of each BSD project. Each section lists the name of the project (OpenBSD, NetBSD, FreeBSD, etc.) the person who spoke, and a laundry list of their latest achievements.

OpenBSD

Todd Miller

- Version 3.0 released in December of 2001 is the 11th release.
- Concentration is on security. Robust code equals safe code. This means using the safe variants of the libc code.
- Integrated cryptography, IPSec, OpenSSH support for hardware crypto
- Much improved UltraSPARC with support
- Alpha resurrection
- Better support for crypto boards Hifn 7951 Broadcom 5820
- Improved 802.11 support
- Support for I2O adapters
- Gigabit Ethernet
- New "pf" packet filter has IPv6 support – easy conversion of IPF-based filters; packet normalization.
- BSD authentication from BSD/OS used throughout the system, including OpenSSH.
- Integrated ALTQ
- Heimdal (Kerberos V)
- Better behavior in low kmem situations
- Updated RAIDframe
- Updated Adaptec AIC7XXX driver
- Better sizing of kernel buffers

Long-term plans:

- IPSec-aware TCP hardware crypto in OpenSSL rate limiting within crypto framework striping network interface

NetBSD

Jason Thorpe

- New toolchain and build framework for fast cross-building
- Highly optimized buffer cache and NFS implementation
- Vastly improved paging behavior
- Performance enhancements for FFS
- Ports to many new platforms (PlayStation 2), including several systems on Chips
- Improved locale support
- Improved standards compliance
- Too many new drivers to count
- Getting ready to branch the 1.6 release

Waiting in the wings:

- devvp: deprecate dev_t in favor of vnodes
- devfs: the end of dev_t as we know it
- wedges: a new approach to disk partitioning
- Sun-style LWPs: finer-grained scheduling
- Scheduler activations: scalable kernel support for threads
- New pthreads library based on scheduler activations
- POSIX real-time extensions
- Support for MIPS64 and PA-RISC
- Performance improvement data structures and algorithms; data movement primitives; concurrency for multi-processors; page replacement algorithms
- Better quality control
- Automated regression testing
- Automated weekly snapshots
- Better bug tracking
- Instant releases
- Embedded
- No persistent storage
- Improved flash support
- Run-in-place support

FreeBSD

Jordan Hubbard

- There are now almost 6500 ports.
- Biggest changes in the last year were social or structural. These included the purchase of Walnut Creek, which was then bought by Wind River and the later spin-off of the FreeBSD team from Wind River.
- The system is still used by lots of large customers (Hotmail, Yahoo, etc.).
- Still doing three releases per year
- More government funding (security)
- Project management is now more work because of the increased number of contributors.
- FreeBSD Foundation has gotten full version of Java for FreeBSD.
- Doc project is doing really well.
- SMP progress continues; a lot of work to be done there.
- KSE (thread scheduler) has reached a milestone.
- SPARC is multi-user but does not self-host yet.
- PowerPC is single user.
- CardBus and PCCard support has taken a leap forward.
- C99 and POSIX are moving forward.
- devfs are used by default.
- Polling network driver support is in -CURRENT and is backported to -STABLE.
- PAM has been cleaned up.
- POSIX ACL support was added to FS.
- Hardware drivers for NVIDIA and other 3D stuff
- Audio drivers improved
- Improved resilience to DoS attacks

Future:

- Wait for NetBSD to get bug tracking right and take it into FreeBSD
- Figure out hosting third-party projects (e.g., SourceForge)
- Continue to work on security infrastructure

BSD/OS

Don Seeley

- Focus on networking and storage
- Snags: tech shakeout, RIFs, FreeBSD debacle, licensing carryover, BSD/OS 4.3 delays
- Wind River is backing BSD in a big way.

In the Pipeline:

- Itasca Release
- Preemptive kernel with locking
- Real embedded release
- Native BSD tools with visionWARE
- Intel IA32, PowerPC targets
- Margaux release
- Tornado IDE
- Cross-development from a Windows host

Future History:

- Building alliances and partnerships
- Spreading BSD through the industry

Darwin

- MacOS X released on March 24
- Targeted the consumer market
- Deployed in a hardened mode
- Released Darwin 1.3 in April
- MacOS 10.1 released in September – a lot of software updates.
- Darwin release with 10.1 was supposed to be 5.0.
- WebDAV is now open source.
- A lot of software is coming out.
- MacOS X will now be the default with every system Apple ships.
- BSD on the desktop now outnumbers Linux by 2:1.
- All hardware comes with developer tools.
- Next event is the World Wide Developer conference, which will be UNIX oriented.
- New release focuses on gcc3.0, FreeBSD synchronization, better pthreads, Kerberos, QuickTime 6, package management.

Questions:

The questions were put to all the speakers, whose answers are noted by project name.

1) I'd like to hear the plan that will stabilize the API for device driver authors.
Darwin: Has its own and it's open source. No plans to change.
BSD/OS: Is going toward the CardBus-based system.
FreeBSD: Most problems are legal and not technical.
NetBSD: Closest to BSD/OS.

2) (Comment) I created the API mailing list after a recent debacle. No one else seems to want to care. The lists are moderated. There is an announcement list and a discussion list:
bsd-api-announce@wasabi-systems.com
bsd-api-discuss@wasabi-systems.com

3) Where lies the official support for Darwin Intel? Is there a plan to ship anything?
Apple will only ship a CD-ROM, no hardware.
4) What's up with the OpenPackages project?
Project on hold waiting for developer time but can be found at *http://www.openpackages.org*.
5) Darwin/MacOS X plans for IPv6?
No formal plans as there is no real demand yet, but the code is in the repository.
6) Four different platforms are doing locks differently. Would it make sense to reduce the divergence?
BSD/OS is synced up with FreeBSD APIs.
Perhaps there should be a forum for SMP.
7) LFS is cool — why not use it?
Although we've made the cleaner a little more robust it's still not ready for prime time.
8) What are the current plans for transferring the FreeBSD trademark to FreeBSD from Wind River?
No comment.
9) Very little seems to happen with merging the user-land. Speaker proposes that we get someone to drive this.
Most people are waiting for automated tools to help with this.
The core problem is a difference in goals.
Best off collaborating on APIs and not on implementations.
It's not always true that multiple implementations are bad.
10) (Comment) At the recent FAST conference there was a paper presented on the write buffering LFS that significantly improved the garbage collection. Overhead was very small.

KEYNOTE ADDRESS

UNIX: Not Just for Geeks Anymore

Brett R. Halle, Director, Core OS Engineering, Apple Computer, Inc.

The second-day keynote was presented by Brett R. Halle from Apple Computer. He made the usual assertion that UNIX has for decades been designed for engineers but is not usable by mere humans.

Most modern UNIX's primary customers remain servers, scientific users, colleges and universities, specialized workstations. These are fundamentally highly educated technical users in centrally managed environments.

What's missing?

General purpose desktop/mobile computing. This means that the current UI is un-acceptable and that applications are important.

He then presented a MacOS X architectural slide that shows how applications are supported independent of the kernel.

Apple found that UNIX on the desktop uncovered many challenges and presented some interesting opportunities. A central question is, "Who uses the desktop?" Apple found that desktop users include PC users (kids and parents), cre-

ative professionals (in advertising, film etc.), and people working in higher education and scientific/technical fields.

In general desktop users have no access to central administration, aren't computer scientists, and want to use the computer as a tool. These users run every type of software imaginable and are in highly dynamic environments.

One of the major challenges to UNIX in this environment is the file system layout. Names like "/etc /usr /bin" do not mean anything to a typical desktop user. Files are often removed accidentally in some environments because it's common for users to clean their disks and remove things they don't think they're using. For this reason MacOS X provides a filtered view of the file system.

Metaphorically this is a "trees vs. forest" argument. The UNIX file system tree is an extremely powerful concept, but desktop users expect to see a forest and not the trees. Another important feature is that devices should be visible as objects in the system as should remote file systems.

Desktop users often identify files with a descriptive phrase used for identification, including spaces and punctuation. To most users case doesn't matter. Another complication is that paths can change because users reorganize their data frequently.

Another area that UNIX does not address on the desktop is security. On a personal computer, users are the administrators. Unfortunately, root is too granular and highly dangerous. In MacOS X, support for administrative roles is much more flexible, and roles may change during a session. All systems are shipped with root turned off.

What makes this all more difficult is that security issues scare most computer users. A personal computer should have most network services turned off by default. Services should come online in the most secure manner. Updating software for security fixes is difficult and so support for this is built in.

File system permissions are another area of security that has to be dealt with. The UNIX permission model is too restrictive for desktop users, and it falls apart with removable media because of the problems with administrating UIDs and GIDs on someone else's desktop computer. File system permissions only work in a managed environment.

Next Brett turned to the role cron and other background services play in maintaining the system. These are typically used for housekeeping, but they don't work well for computer users with portables or energyStar support, which routinely put systems in sleep mode. These services can also have unpredictable side effects.

The software development cycle is also different for desktops. Commands and libraries are part of the public API, not just what's inside libc. Commercial applications depend on these and do not update on the same schedule as OS releases. Unlike open source projects, source compatibility isn't good enough. Customers cannot be expected to update third-party applications for each OS release.

UNIX could improve by avoiding fixed length data such as usernames, passwords, etc. Brett challenged the audience by asking, "What if the API you were working on had to survive 10–20 years? In a binary compatible way?"

He also pointed out that people make products that extend the kernel to such as file systems, VPN, and device drivers. We've got a lot of work to do here!!!

Apple has put a lot of work into user interface issues. A command is not a good interface for a desktop user and neither is X Windows. Consistency is a very important goal for the UI. There are also accessibility requirements for all users.

Interoperability between applications – data interchange, cut/paste, drag and drop – also has to be addressed.

One of the major goals for OS X was to be able to have multiple localizations from a single binary. This has now been achieved.

Other challenges to UNIX on the desktop are:

- Power management
- Systems can be expected to change speeds
- Subsystems will power down disks, displays
- Systems will sleep when idle
- Mobility
- Wireless networking
- Moving from managed to unmanaged networks

In summary Apple believes we in the UNIX community still have a long way to go but that we can surmount these challenges, in part because we now know what some of them are.

Questions:

1) (Comment) There was one thing I think you might not have gotten right. The problem of backwards compatibility has been solved. You can run old NetBSD or Sys V.3 on NetBSD. I think we've been concerned about that as naïve users. All the old syscalls are still there.

I accept that there are areas where this is true, but I'd like to propose a stronger challenge including evolving kernel plug-ins. We still have the command-line problem as it is an API.

2) I do object very strenuously to the Apple thread of bashing UNIX by referring to us as geeks.

I'm sorry you interpreted it that way.

3) With respect to roles, are these distinctions obvious in Darwin?
It's visible in Darwin; there are authorization APIs. There are no setuid apps.
4) Why did Apple choose UNIX with all of its current problems on the desktop?
I think the choice of UNIX for the base system was the right decision. It's the right technology, but we need to solve these broader issues. I think this is the right way to do an OS.
5) There is a distinction between UNIX programs and Mac applications. It's a different way of writing things. You have to be aware of that.
UNIX has this notion that files should be in plaintext ASCII; the Mac does not have this.
With MacOS X we're trying to balance the goals of original UNIX with building a real Mac.
6) The UNIX security model is pretty well understood. How do you go about ensuring that the new models don't have holes?
Good question. As a commercial entity we have more resources to throw at the problem. The model of moving from a most secure system to a least secure one helps.
7) One thing that concerns me is Netinfo.
This will be addressed in a near-future release.
8) When will I get multiple mouse buttons?
I'll give that feedback to our people.
9) What about central administration?
UNIX already does that well.
10) Is there some thought within Apple to get some ideas in this talk out to open source?
Yes, that's why I'm here.

SESSION: FILE SYSTEMS

Running "fsck" in the Background

Marshall Kirk McKusick, Author and Consultant

This paper was one of two given the "Best Paper Award."

The goal of this work is to be able to quickly and reliably restore file systems to a consistent state after a crash. The work is intimately tied to soft updates. In a soft updates enabled file system, the only possible inconsistencies are that there might be blocks or inodes marked in use that are free.

With this in mind it is safe to run immediately after a crash, though, eventually, the lost space must be reclaimed. The problem is the loss of resources during this period.

To address these issues there is background resource recovery. The steps to achieve this are to snapshot the file system and then run standard fsck on the snapshot. Making this happen requires adding a system call to allow fsck to put lost blocks and inodes back into the file system map.

Information on the creation and management of snapshots and background file system checking can be found in the paper.

The status of the work is that snapshots and background fsck have been running on FreeBSD 5.0 (-CURRENT) systems since April 2001. All relevant code is under BSD license and can be found on *http://www.freebsd.org*.

Questions:
1) Can this work be re-used to revive the union mount system?
That's an independent system, so it's unrelated.
2) What are the potential failure modes during snapshot generation?
There is no failure mode since a snapshot is not a snapshot until it's marked as finished.
3) When you're performing copy-on-write is there a failure mode?
There are problems there because you have to write synchronously to make the snapshot consistent.
4) Modern ATA disks have problems with soft updates due to their lying about operations.
If you have disks that lie to you then you're hosed no matter what. The current answer is tagged queuing, which is only in SCSI right now.
5) Since you can adjust the link count on any file, how does that interact with security?
These operations can only be done at security level 1 (root).
6) Your modifications to the scheduler — how do these affect multi-user performance?
By default everyone runs at nice 0, so if you aren't running niced you won't notice it. If you do nice things it will slow you down. It makes nice a CPU and I/O thing now.
7) Have you given any thought to making the whole process FS independent?
This is too tightly integrated and is not generalizable.

Design and Implementation of a Direct Access File System (DAFS) Kernel Server for FreeBSD

Kostas Magoutis, Harvard University

This paper was one of two given the "Best Paper Award."

DAFS is an NFS derivative that provides user-level file access for data-center applications. The protocol itself is specified by the DAFS Collaborative led by Network Appliance and Intel and is targeted for virtual memory mapped networks.

A VM mapped network has user application memory mapped directly into the network interface controller. This gives user-level access to network data with low overhead. It is most commonly used in Server Area Networks (SAN).

The protocol supports both remote DMA and reliable, in order, message passing. Like NFS the protocol is based on Remote Procedure Call (RPC).

Within the kernel the DAFS server talks to the vnode layer for file access. All file I/O goes through the buffer cache. Connection setup is through its own driver. Network data transfer is direct via the NIC. The system uses the PMAP layer of the VM system to register the protocol with the NIC.

As part of ongoing work, they are trying to provide asynchronous vnode file I/O and to integrate NIC MMU directly into the VM system.

For more information: *http://www.eecs.harvard.edu/vino/fs-perf/dafs*
http://www.dafscollaborative.org

Questions:
1) What's the security model for RDMA. Security is kind of weak right now. At the device level it is possible to corrupt exported pages.
2) This work shows that UNIX I/O is problematic for user space. Perhaps we need a new model.
Conversation taken offline.

SESSION: NETWORKING

Flexible Packet Filtering: Providing a Rich Toolbox

Kurt J. Lidl, Zero Millimeter LLC; Deborah G. Lidl and Paul R. Borman, Wind River Systems

Paul Borman discussed packet filtering in the BSD/OS system. Although their system is named IPFW it has no relation to FreeBSD's ipfw subsystem. The system was developed in the early 1990s when the only technology available was based on rules and not programmatic (screend, IPFirewall, IPFilter, etc.)

BSD/OS's IPFW is a framework for IP filtering that supports multiple types of filters and is agnostic about how filtering is done. It provides a programmatic interface to the system. The primary method of programming is based on the Berkeley Packet Filter. Filters can have names.

IPFW filters packets at several points in the network stack. Each filter point is a chain of filters and each chain can have zero or more filters. Filters can be of multiple types and are normally pushed onto the list like a stack. They can be prioritized and named as well.

Types of filters and some examples are given in the paper. Paul presented these to the audience. The examples showed how to deal with things such as the NIMDA and Code Red worms.

Questions:
1) Is the language static? Is there a compiler?
You need the source.
2) Is this going to be open source?
Currently undecided by their employer (Wind River Systems).
3) Can the system handle IPv6?
Yes.

Resisting SYN Flood DoS Attacks with a SYN Cache

Jonathan Lemon, FreeBSD Project

Jonathan presented work with TCP SYN caches and SYN cookies for resisting denial of service attacks. A SYN-based DoS attack floods the machine with SYN request packets. Each SYN tries to set up a new connection to the machine, and this ties up machine resources, including CPU and memory. Eventually, the machine under attack becomes unresponsive or crashes.

Prior solutions had performance that was a function of the connections in the socket listen to the backlog which is O(N); this consumed 30% of system time. Socket state allocated with incoming SYN was also roughly 512 bytes, and the data accompanying the attacking SYN was preserved.

The SYN cache implementation borrowed from NetBSD, which in turn borrowed from BSD/OS. This implemented a hash table, with limits on the size of the hash chains. The maximum number of entries was also limited so that data within the initial SYN is not preserved.

The hash function used a boot-time secret, so an attacker could not target a specific hash bucket to create a specific DoS attack on the SYN cache itself. The SYN cache only keeps a small amount of state, about 100 bytes.

SYN cookies had advantages over a SYN cache. They allocate no state on the server. The system creates a TCP initial sequence of numbers that is an encrypted cookie returned in ACK, to determine whether a connection is accepted. The ISN is a secret with a finite lifetime. SYN cookies allow an infinitely deep queue of connections.

The SYN cookie system has some problems. It does not allow for the use of any TCP options and cannot handle retransmission of SYNs or ACKs. Other issues are that it is possible for a connection to be accepted with no initial SYN, and it requires a crypto hash on an incoming ACK.

Questions:
1) Do you have any performance comparison to NetBSD?
No comparison is made to other BSDs. The algorithm is roughly identical, but they don't implement SYN cookies.

A FreeBSD-Based Low-Cost Broadband VPN Router for a Telemedicine Application

Gunther Schadow, Regenstrief Institute for Health Care

This talk focused on applying various pieces of *BSD technology to a user installation. The work was funded by the National Institutes of Health to study a next generation Internet as applied to remote medicine. The test system provided for direct physician-patient teleconferencing between 6-9 physicians and a 240-bed nursing facility, serving high-risk, multiply ill patients.

The telemedicine VPN provided H323 video on top of UDP within a wireless network. The physicians' homes were hooked up through cable modems to the rest of the system. Because of the public/private nature of the connections, IPSec technology was used for private data over the public network.

To provide for a decent quality of service the system used ALTQ. This was to prevent starvation of more important signals by outgoing video. Initially camera control was impossible and outgoing audio was useless. ALTQ literally rescued the project.

As routers they used the Soekris net4501. This is a small, cheap ($250), embedded router that can run *BSD and other operating systems (*http://www.soekris.com*).

The router throughput maxes out at 30 Mbps, but this is sufficient for their needs.

Questions:
1) Were there problems with cable modems degrading performance?
Initially we had a 6Mbps downlink. In some neighborhoods we have a problem with going down to 1.5Mbps.
2) On the PicoBSD mailing list someone showed how to use NFS.
I thought about using NFS also but the problem with that is the path length between the systems.
3) You mentioned NAT at some point. How did you get IPSec to get through NAT?
IPFilter will be on the outside and IPSec with be on the inside. You must let IPSec packets through the filter. You trust your tunnel.
I wanted to make it all a package but there is too much fiddling.
4) You mention that you use DNS to get information from the central system?
What they have is a certificate. This is what distinguishes boxes/sites. They query for their internal IP overlay addresses using DNS.
5) Is this HIPA compliant?
Yes.

SESSION: SYSTEM ADMINISTRATION

SystemStarter and the MacOS X Startup Process

Wilfredo Sanchez, MIT; Kevin Van Vechten, UC Berkeley

This talk was a discussion of the MacOS X startup process, which is completely different from any of the other *BSDs. The system grew out of work done in NextStep and Rhapsody. Originally it was a hybrid BSD/System V solution.

Some basic problems with the original system were that it depended on lexographic ordering to indicate who ran first. This is fragile. For MacOS they must go into GUI right away and need to launch the display system early.

The new design defines the startup sequence as a progressive bring-up of services. A single startup item describes a service and contains logic to start or stop that service. The SystemStarter manages startup items.

The startup items themselves are contained in the /sys/library directly.

Each startup item contains an executable (typically a script), a StartupParameters.plist, which contains a description, the list prerequisites, and the list of services.

In the future they hope to provide for the partial startup and shutdown of services. This would allow system administrators to bring up all services that X depends on and service X but nothing more.

The implementation provides for parallel startup. Each script is run as its own process via fork/exec. As soon as a script's prerequisites are met, the script is executed. When a script terminates, the list of waiting scripts is reevaluated to look for scripts newly eligible to run.

The new startup system created new needs for the Inter-Process Communication system. Scripts need to display messages, get configuration information, and report success or failure.

To fill these needs SystemStarter listens for messages on a Mach port, and the scripts themselves use a tool to send messages. Messages can go to the console or query for information.

The system is currently deployed on MacOS X and is used by vendor packages.

Questions:
1) Will this be open sourced?
Yes, it's in Darwin.
2) How portable is this?
Have to port part of CoreServices.
3) Documentation?
Not much documentation but there is stuff in the sysadmin manual.
4) Why is there a duplication on the startup messages?
Because plists are going away.
5) Does this deal with things like walking into range of a wireless network?
Right now this only works for boot. There is another state change system called Configuration or something. That's all a little further down the road.
6) How do you handle deadlocks in the dependency change? What if someone messes it up?
The algorithm is dumb. It goes through each item and says, "Are you ready to run?" In your case they would just never get to run.
7) Why do we run /etc/rc to start SystemStarter?
There is a migration plan.

Log Monitors in BSD UNIX

Brett Glass, Glassware

A log monitor is an intelligent agent that automatically responds to conditions revealed by one or more system-log

messages. A log analyzer does the same thing but offline.

A log monitor can detect abnormal usage patterns, recognize abuse, catch worms, detect vulnerability scans, and detect intruders.

The reason for much of this work is that logging via syslogd is now antique.

One of the major applications for this work was the Apache Web server. Apache does not normally use syslogd for logging. Conditional logging in Apache can be done in the configuration files but this is a dirty hack.

The author used SNOBOL4 to implement a powerful worm blocker. The reasons for choosing SNOBOL4 were that it is a powerful string matcher, can call programs to go upstream to inject a firewall rule and can be implemented in 28 lines of executable code.

Some future enhancements are an option to turn off log compression in FreeBSD, integration of algorithms from MIT AI Lab work on determining "interestingness," and a drop in replacement for syslogd, specifically tuned to allow efficient log monitoring.

Questions:
1) (Suggestion) If you want those flags to appear, use the bsd-api-discuss list.

SUSHI: An Extensible Human Interface for NetBSD

Tim Rightnour, NetBSD Project

SUSHI (Simple-to-Use System-Human Interface) is a menu-based sysadmin tool. Application is based on curses and CDK and is easily extensible. The main thrust of the system is around system administration.

SUSHI provides a hierarchical menu structure that can be used to group similar actions together. The system is based around forms (which are a series of questions and answers). Because the system's menus are stored in plaintext, it can be updated without recompilation.

In terms of implementation of the system, the menus are defined by a hierarchy of directories and command files. Forms and menu indices are stored in text files in each directory. Each form can execute arbitrary programs or specific scripts. Scripts can be written in any language or be an executable.

Questions:
1) How does this relate to SMIT on AIX?
Similar but not the same behind-the-scenes.
2) Doesn't this need to be integrated with the rc/system startup?
It's an adjunct.
3) What about having things changed out from under you?
The system reads the rc.conf file and works with what you did.
4) Is this difficult to port to FreeBSD?
No. The only issue might be curses vs. ncurses.
5) Are you planning to make an install replacement with this library?
No, because SUSHI is more of a question/answer system. An installer would be difficult because the user would already have to know what to do.

NEW MEMBER BENEFIT

eLearning from DigitalThink

NOW AVAILABLE FOR OUR MEMBERSHIP – AT A 25% DISCOUNT

ALL YOU NEED IS A BROWSER!

USENIX and SAGE members can now register for DigitalThink's Web Based Training courses at a 25% discount. Through DigitalThink, you can learn online, 24 hours a day, anytime, anywhere! If you want to improve your skills, get certified, get promoted, learn about management, finance or other key skills, these courses can help get to where you want to go.

The DigitalThink courses were selected by a committee of USENIX and SAGE members after a comprehensive review of all the major eLearning providers. Courses cover a wide variety of topics including: UNIX, Linux, Java, Databases, Networking & Project Management. DigitalThink also offers unique, integrated tutor support to give you personal feedback and assist you with questions.

If you, your colleagues or subordinates need an educationally effective AND cost effective way to learn a specific skill, DigitalThink might be the best solution. Try one today!

The outline below offers a glimpse of the over 200 courses offered.

Certification Database Concepts
IT Management
Java Programming
Microsoft Certification
MCDBA
MCSD
MCSE
Networking
Oracle
Programming
Red Hat Linux
System and Network Administration
UNIX
Web Development
Business Fundamentals
Compliance
Project Management
Desktop Operating Systems

For a complete course catalogue and to register please visit::
http://www.usenix.org/membership/digitalthink/index.html

NordU USENIX 2003

NordU2003 –
The fifth NordU/USENIX Conference
February 10–14, 2003
Aros Congress Center
Västerås, Sweden

Announcement and Call for Papers

Information regarding NordU2003 – the fifth NordU/USENIX Conference, to be held in Västerås, Sweden, February 10–14, 2003.
A Conference organized by EurOpen.SE – The Swedish Association of Unix Users, and affiliate of USENIX, The Advanced Computing Systems Association.
Complete program and registration information will be available November 2002.
To receive information about the fifth NordU/USENIX Conference, please visit http://www.nordu.org/NordU2003/ or send an e-mail to NordU2003@europen.se

Important Dates

Extended abstracts due September 2, 2002
Notification of acceptance October 2, 2002
Final papers due December 2, 2002

Call for Abstracts

Authors are invited to submit a 1/1 page abstract in English on any of the topics below to the Conference Secretariat.
Submission should be original work and will be reviewed by the Technical Review Committee.
All accepted papers will be available on Internet after the Conference.
Authors must register for the Conference and present their papers in person.

Topics

- Security
- Operating Systems
- Open Source/Free Unix
- High Performance Computing
- High Availability
- Mobile Computing
- Software Development
- Interoperability
- Storage Area Network, SAN

Technical Review Committee

Will be announced May 30, 2002.
Please visit http://www.nordu.org/NordU2003/

Please send your abstract to:
Congrex Sweden AB
Attn: NordU2003
P.O. Box 5619
SE-114 86 Stockholm, Sweden
Phone: +46 8 459 66 00
Fax: +46 8 661 91 25. E-mail:congrex@congrex.se

.SE EurOpen DKUUG EurOpen NUUG USENIX FUUG

NordU2003

CONNECT WITH USENIX

MEMBERSHIP AND PUBLICATIONS

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710
Phone: 1+ 510 528 8649
FAX: 1+ 510 548 5738
Email: office@usenix.org
login@usenix.org
conference@usenix.org

WEB SITES

http://www.usenix.org

http://www.sage.org

EMAIL

login@usenix.org

COMMENTS? SUGGESTIONS?

send email to ah@usenix.org

CONTRIBUTIONS SOLICITED

Your are encourgaged to contribute articles, book reviews, photographs, cartoons, and announcements to *;login:*. Send them via email to *login@usenix.org* or through the postal system to the Association office.

The Association reserces the right to edit submitted material. Any reproduction of this magazine in its entirety or in par requeires the permission of the Assocation and the author(s).

USENIX & SAGE

The Advanced Computing Systems Association &
The System Administrators Guild

;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