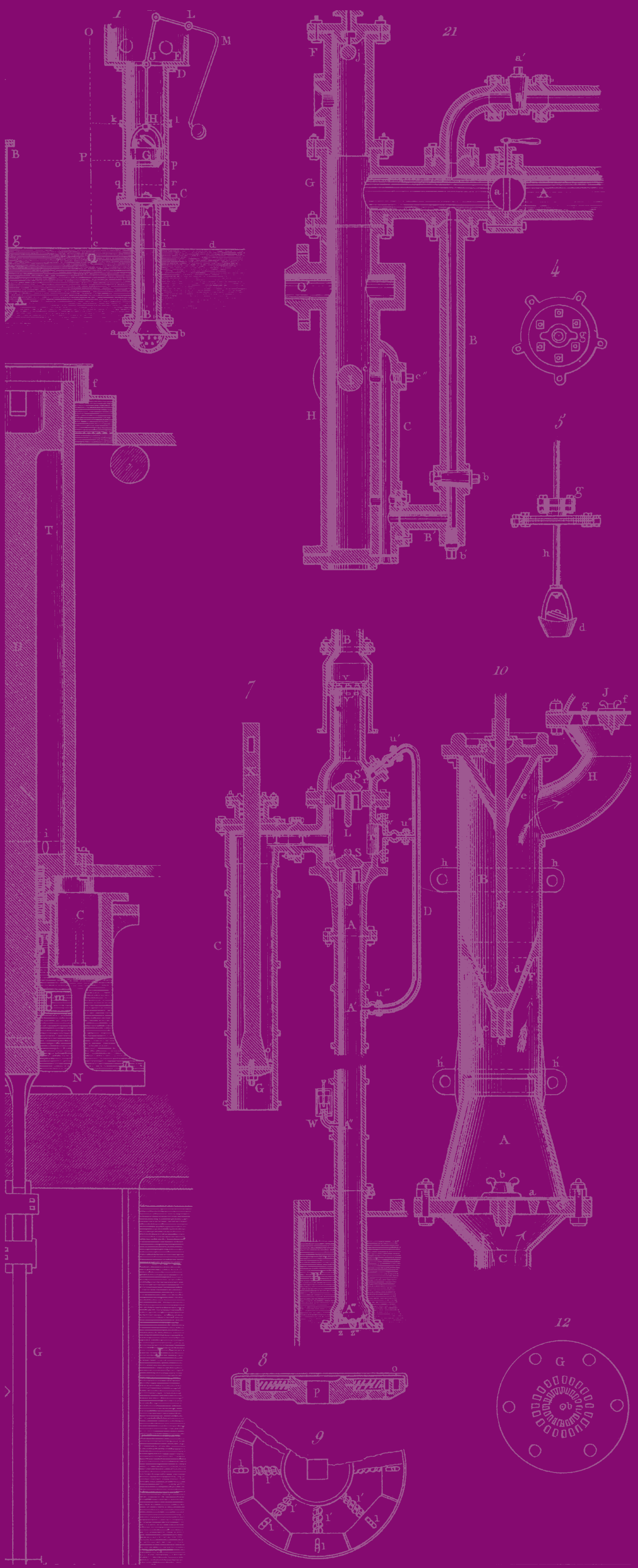
APRIL 2006 VOLUME 31 NUMBER 2

;login:

THE USENIX MAGAZINE



USENIX
The Advanced Computing Systems Association

USENIX Upcoming Events

3rd Symposium on Networked Systems Design and Implementation (NSDI '06)

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

MAY 8–10, 2006, San Jose, CA, USA
http://www.usenix.org/nsdi06

5th System Administration and Network Engineering Conference (SANE 2006)

Organized by Stichting SANE and co-sponsored by Stichting NLnet, USENIX, and SURFnet

MAY 15–19, 2006, Delft, The Netherlands
http://www.sane.nl/sane2006

2006 USENIX Annual Technical Conference (USENIX '06)

MAY 30–JUNE 3, 2006, Boston, MA, USA
http://www.usenix.org/usenix06

First Workshop on Hot Topics in Autonomic Computing (HotAC '06)

Sponsored by IEEE Computer Society and USENIX

JUNE 13, 2006, Dublin, Ireland
http://www.aqualab.cs.northwestern.edu/HotACI/

Second International Conference on Virtual Execution Environments (VEE '06)

Sponsored by ACM SIGPLAN in cooperation with USENIX

JUNE 14–16, 2006, Ottawa, Ontario, Canada
http://www.veeconference.org/vee06

4th International Conference on Mobile Systems, Applications, and Services (MobiSys 2006)

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

JUNE 19–22, 2006, Uppsala, Sweden
http://www.sigmobile.org/mobisys/2006

2nd Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI '06)

JULY 6–7, 2006, San Jose, CA, USA
http://www.usenix.org/sruti06
Paper submissions due: April 20, 2006

2006 Linux Kernel Developers Summit

JULY 16–18, 2006, Ottawa, Ontario, Canada
http://www.usenix.org/kernel06

15th USENIX Security Symposium (Security '06)

JULY 31–AUGUST 4, 2006, Vancouver, B.C., Canada
http://www.usenix.org/sec06

2006 USENIX/ACCURATE Electronic Voting Technology Workshop (EVT '06)

AUGUST 1, 2006, Vancouver, B.C., Canada
http://www.usenix.org/evt06
Paper submissions due: April 3, 2006

7th Symposium on Operating Systems Design and Implementation

Sponsored by USENIX, in cooperation with ACM SIGOPS

NOVEMBER 6–8, 2006, Seattle, WA, USA
http://www.usenix.org/osdi06
Paper submissions due: April 24, 2006

Second Workshop on Hot Topics in System Dependability (HotDep '06)

NOVEMBER 8, 2006, Seattle, WA, USA
http://www.usenix.org/hotdep06
Paper submissions due: July 15, 2006

20th Large Installation System Administration Conference (LISA '06)

DECEMBER 3–8, 2006, Washington, D.C., USA
http://www.usenix.org/lisa06
Paper submissions due: May 23, 2006

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

contents

;login:

VOL. 31, #2, APRIL 2006

EDITOR
Rik Farrow
rik@usenix.org

MANAGING EDITOR
Jane-Ellen Long
jel@usenix.org

COPY EDITOR
Steve Gilmartin
proofshop@usenix.org

PRODUCTION
Lisa Camp de Avalos
Casey Henderson

TYPESETTER
Star Type
startype@comcast.net

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

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

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

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

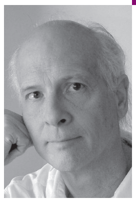
$85 of each member's annual dues is for an annual subscription to *;login:*. Subscriptions for nonmembers are $115 per year.

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

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

RIK FARROW

musings



rik@spirit.com

PROGRAMMING IS AN ART. WHEN different people attempt to accomplish the same tasks, their programs will generally be quite different. And when examined carefully, some programs will stand out as beautiful code, while others could quite easily be described as ugly.

I learned programming in college, working in the painful paradigm of punchcards and mainframes. A single typo meant waiting hours, or even until the next day, to see the results of the correction. While I believe that using punchcards encourgaged a certain discipline, in coding and in exactness, it did nothing to instill in me a real understanding of how to write beautiful programs. I could, and did, write robust, functioning code, but it certainly lacked the elegance that I could sometimes discern in other people's code.

Inelegant code, however, has effects that go beyond aesthetics. I once was tasked with writing a text formatter, with requirements similar to the ones found in Kernighan and Plauger's *Software Tools*. I didn't know of that book at the time (1979), but plowed in with vigor. When finished, I had written a program that worked correctly, taking marked-up text, formatting it, and producing a table of contents, all on a computer that used two 8-inch floppy disks for file storage. Compiling the 16-page program took about 15 minutes, and using the finished program, written in PL/Z (Zilog's version of PL/I) took a long time too.

After I left that company, I found out that my replacement had been given the same task, but used BASIC instead. His version of the code ran *three times faster* because BASIC had much better string handling routines than PL/Z. I knew my code would be inefficient in places, and had I rewritten those places in assembler, my code would (likely) have been as fast. But I sure was embarrassed.

Looking Deeper

Today's computers make the computers I learned on look like electro-mechanical calculators. The mainframe I used in college filled a room, required massive cooling, and actually used other computers for its input and output (reading punchcards, writing them to tape, and printing). The noise of the cooling fans was incredible, but so were the blinking lights, and the computers ran slowly enough that you could actually watch patterns emerge in the display of memory address accesses. With systems like these, every instruction counted.

When we write programs today, we can easily be misled into believing that elegance and efficiency don't matter. Instead, our fast computers can fool us into thinking that everything is running fine. Problems often don't emerge until a program goes into production and fails under real loads, or turns out to include a security flaw that converts code into a back door.

For this issue, I sought out programmers who were willing to write about their art. I was fortunate that Brian Kernighan was willing to share his experience in teaching advanced programming. Brian's article explains how he uses testing to maintain AWK and uses that same testing in his classes. I found myself wondering if I would have been a better programmer had I learned the testing discipline that Brian instills in his students today.

David Blank-Edelman's Perl column also begins by discussing testing in Perl. Various Perl modules provide a framework that can be properly (or poorly) used to aid in building packages that can be tested before installation.

Diomidis Spinellis has written about the effects of the many levels of performance found in modern computer memory. The amount of memory available to run programs at full speed on modern processors is tiny, and each additional level offers lower performance. Diomidis explains how the different levels function, provides hints for improving performance in critical areas, and concludes with an analysis of price/performance of memory that is sure to arouse some discussion.

You will also discover other programming-focused articles. Chaos Golubitsky writes about cflow, a tool she used when analyzing the security of IMAP servers in her LISA '05 paper. Luke Kanies explains why he chose Ruby for his implementation of Puppet. If you have heard about Ruby and are wondering if you should learn it, you should read Luke's article.

Nick Stoughton reports on his work on several standards committees, work that has real impact upon both programming and open source. If you care about these issues, you need to read Nick's report.

Fond Dreams

While I have been busy ranting about the need for new operating system design, Andrew Tanenbaum and his students have been busy writing MINIX 3. I don't know how many times I have written about the need for a small kernel that can be trusted and running services without privileges, in their own protected memory domains, but MINIX 3 actually does this.

Andy wrote MINIX as a tool for teaching operating systems back when the next best thing was UNIX, an operating system that was growing far beyond an easy-to-understand size and was encumbered by copyrights and AT&T lawyers. While we now have open source operating systems, such as Linux and the BSDs, they too have grown in size and complexity over the years. MINIX 3 manages the feat of being a next-generation operating system with an actually comprehensible size. The kernel is only 4000 LoC (almost equally split between C and assembly), and the process management server is 3600 lines of C. The file containing the implementation of `execve()` is 594 LoC in MINIX 3 (servers/pm/exec.c) and 1496 LoC in Linux (2.6.10/fs/exec.c).

By “next-generation,” I mean that MINIX is a microkernel in design and philosophy. Only the kernel runs as privileged, and all other services, including process management, file systems, networking, and all device drivers, run in their own private address spaces. Just moving device drivers out of the kernel and into their own address spaces means that they can crash without crashing the kernel. It also means that system code, including device drivers, can be tested without rebooting, and failed drivers (or servers) can be restarted.

While MINIX 3 is not going to replace your desktop today, it is already a good candidate for embedded systems where robustness, reliability, and a small memory footprint are crucial. Perhaps your cell phone will be running MINIX 3 some day.

What, No Security?

For a change, there is no Security section in this issue of *;login:*. There are two Sysadmin articles, with David Malone writing a detective story about a mysterious flood of HTTP requests and Randolph Langley telling us about software he has created to provide better logging for sudo.

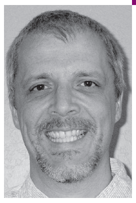
We have two new columns this month. Heison Chak will be writing about VoIP, providing background in this column for later articles that will help system administrators charged with supporting (and implementing) VoIP in their networks. Robert Ferrell has taken charge of the humor department, entertaining us with /dev/random.

The summaries of LISA ’05, WORLDS ’05, and FAST ’05 appear in the back. You might wonder why summaries from December don’t appear until April, but if you look at the publishing schedule of *;login:*, you can see that none of these conferences finished before the articles for the February issue were due. I have, of course, read all of these summaries more than once, and I encourage you to see what is being presented in conferences you don’t attend.

Finally, we have an Opinion piece from Tom Haynes. Tom writes that he got so excited about OpenSolaris that he just had to do something about it. And he did.

TOM HAYNES

OpenSolaris: the model



Tom Haynes is an NFS developer for Sun Microsystems, Inc., and is interested in the cost differences between open source and commercial offerings. He is exploring those costs by using OpenSolaris to design a NAS appliance.

tdh@sun.com

I FEEL LIKE THE NEW CHAIRMAN OF the hair loss club—I liked the product so much I went out and bought the company. Only I didn't buy the company, I simply joined it. The company I am talking about is Sun Microsystems, Inc., and the product is OpenSolaris. The premise is simple: Sun opens up its source vault and sucks more users into its web. Sun has a long development cycle between releases, and interested parties could always download the Solaris Express bits to play with new features. When I was at Network Appliance, Inc., we would do interoperability testing of NFSv4 based on the beta program. We saw the exact same binaries that any other customer of Sun could download. This binary availability was very crucial to the successful cross-deployment of a new protocol. After the release of Solaris 10, Sun decided to release the source code to the majority of the code base at the same time that the binaries, the release called Nevada, were made available. The parts not made available under the CDDL, or Common Development and Distribution License, were those sections that were already entangled under prior copyrights.

You can go to http://www.opensolaris.org to see what all the excitement is all about. There are already multiple distributors: for example, SchilliX (http://schillix.berlios.de) or Nexenta (http://www.gnusolaris.org)—think of these as early-day Debian or Slackware. There are development efforts underway to extend technology already found in Solaris 10, e.g., the BrandZ effort to extend zones (see "Solaris 10 Containers," by Peter Baer Galvin, in the October 2005 issue of *;login:*) to non-native operating systems—first up is Linux. And of course there is the recent release of ZFS in the Nevada Build 27 (or b27, as it is commonly called). ZFS is a radical new file system which has been under development at Sun for the last five years.

The two major draws of OpenSolaris are the commitment to quality and the early access to cutting-edge technology. It is easy to argue that GNU, Linux, and the *BSD efforts all provide the bleeding edge of technology. But the real cost can be in the quality of the built-in supportability of software installed at client sites. Note that I do not mean in the quality of open source products but, rather, in the quality of the support infrastructure in a data center.

A common scenario I have seen is a large data center with a heavy commitment to Linux-based compute servers running a very old kernel, say, a base RedHat 7.3 system with a 2.4.9 kernel. Either the company decided to roll that version out because it was the newest when they upgraded from a 2.2-based kernel, or they bought support from a third party. Regardless of how the decision was made, a further complication is that either a modification was made to the kernel source (the best case is that it was patched up), or the customer's application is dependent on that particular Linux kernel. And, finally, the company no longer has any support for the kernel—perhaps the contract ran out, or someone in management thought that free software was, well, free, and no budget was allocated for maintenance.

I know that if you contact Trond Myklebust, the Linux NFS client maintainer, for support, he will try to help you—no matter if you are a first-year student pounding away on an old hand-me-down laptop or the CIO of a company with a 5,000-node application farm. Depending on the problem, that student might get more help; the maintainer is a volunteer and prioritizes his time accordingly. If you stumble on a major bug he believes will impact the majority of Linux installations, he is going to give you attention. But if you have an interoperability problem with another vendor's product, one he may not have access to, then he is going to give you a fishing rod and teach you how to fish.

This approach is the QA model employed by most open source developers. They simply do not have the time, funds, equipment, or desire to test everything under the sun. So, instead, they provide new features in branches for the brave. These adventurers get bleeding-edge technology and the satisfaction of being able to contribute by finding bugs.

Sun plans to make money from OpenSolaris by being a service provider, and the biggest differentiations are quality and support. Sun already has infrastructure (personnel, equipment, tests) to do interoperability testing. The internal developers still have the same commitment to delivering bug-free software. They also have an organization dedicated to analyzing customer-found issues and providing fixes to customers.

Clearly, the interesting questions about Sun and OpenSolaris are concerns over how Sun and the open software model will interact. For example, if Sun is selling service and that same first-year student finds an issue in SchilliX, what level of support will he get? Or say he not only finds the bug, he fixes it and now wants the fix put back into the Solaris code. How can Sun juggle that need versus the need of the CIO paying an annual support contract for her enterprise data center?

The trick for Sun is that the CIO is going through professional services and the student is going through volunteer services. At the end of the day, they might get help from the same individual, but that depends on the commitment of the developer to the open source movement. Sun has asked its employees to help out with OpenSolaris, but it has not mandated that they do so—it is freedom of choice. And there are not only Sun employees helping out on the project.

Any individual asking for help on the OpenSolaris discussion forums, including that CIO, can expect the same level of support. It might just be more like that fishing rod analogy than some people are wanting and the response time might be in days instead of minutes. And that CIO might even find the student responding.

The other question I posed was how an individual outside of Sun gets fixes put back into the tree. In one approach, the individual or distributor maintains their own source base and does not even try to give back to the community. This model is akin to the way many startups in Silicon Valley operate—they take a FreeBSD release and tack on their IP. Perhaps they feed back general bugs (or even contribute scaled-down versions of their product), but they normally integrate from changes made at the source.

The second approach is for the outside individual to find a sponsor within Sun to champion their change. The sponsor arranges for a bug to be filed, code to be reviewed, and the fix put back into Solaris. Interestingly, the "outside individual" might be a Sun employee. For example, although I work in the NFS team, at night I might want to work on porting OpenSolaris to the UltraSPARC 1 platform. I might get it working and then look for a sponsor—perhaps in the kernel team.

The example also shows that by opening up its source, Sun has to make commitments which seem to run counter to its planning process. The UltraSPARC 1 was supported well into the late releases of Solaris Express for Solaris 10. But as a business, Sun decided to retire support for the system—the EOL was actually for the 32-bit SPARC kernel, but as there were outstanding issues with the UltraSPARC I chips in 64-bit mode, it was retired as well. As an individual, I could decide to backport OpenSolaris to this platform.

Sun has also pledged that it will provide ethical support to their employees who want to contribute to OpenSolaris. While Sun does employ full-time workers to develop OpenSolaris, for the most part such development is completely voluntary. At other companies, I've had to sign an NDA which precluded me from contributing to open source projects that could provide an advantage to competitors. At times, I was asked by the Linux NFS client maintainer to please not even look at that source code. He didn't want to risk contaminating it, under a new licensing model being considered by Linus Torvalds.

But I feel free to contribute to OpenSolaris, not only in NFS but in other modules. I know that if I want a break from my day job, I can still contribute, even if indirectly, to my company. I even know that if I do resurrect the UltraSPARC 1, I am likely to make someone smile in appreciation of the effort.

If I pull off those rose-colored glasses you might think I am wearing, I can see that Sun has taken a large risk. This plan could easily backfire on the company. Consider, for example, ZFS, the new file system. Sun has made it available in source form before it shipped in a commercial product. Instead of joining Sun, I could have gathered some venture capital and started shipping low-end NAS boxes in direct competition. My contribution would have been the business plan, not the cool technology. Also, if Sun is filing any patents on ZFS, it has to do so much earlier than normal (i.e., the technology has been publicly introduced).

Sun is betting the farm on differentiating its product offerings, not through the technology but, rather, through the support and service it can provide once that product is installed at a customer site. One nightmare they will have to contend with is that a customer may no longer be running a Sun kit and may not have bought an AMD-based Ultra 20. Instead, they may have taken their Linux farm with a hodgepodge of x86-based systems and loaded up either a stock Nevada b30 or SchilliX 0.4.

When Sun controlled the allowable hardware, it effectively was managing the service it needed to provide, though admittedly it has always been possible to add third-party hardware even to their proprietary systems. They have also been shipping the x86 version of Solaris for quite some time. But for a long time, the x86 product looked unsupported. I saw a couple of trade articles announcing the end of the product.

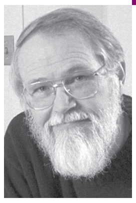
By opening up the vault and committing heavily to the x86 line, they have exposed themselves to the same nightmare of device driver management that other vendors and open source distributions have had to handle.

Sun already has a support model for someone running a Nevada b30 system—they accept bug reports and you can find employees interacting on the OpenSolaris discussions. With an OpenSolaris distribution, support will probably be the same except for a support contract that entails the migration of the boxes to the latest and greatest Solaris. If the concern is the availability of a certain new feature, Sun does backport some technology from Nevada into Solaris 10.

The neat thing about OpenSolaris is that anyone can contribute. Besides testing new technologies, you can see how a commercial product is built. The scripts used to build OpenSolaris are the same ones used to build Nevada, the next commercial release of Solaris. You can also dig into different releases of the source code to try to get an idea of how the underlying technology is changing. A key point to remember is that you are in essence viewing a beta release candidate—from build to build you might find a unique bug. And if you do, make sure to file it!

BRIAN KERNIGHAN

code testing and its role in teaching



Brian Kernighan was in the Computing Science Research Center at Bell Labs and now teaches in the CS department at Princeton, where he also writes programs and occasional books. The latter are better than the former, and certainly need less maintenance.

bwk@cs.princeton.edu

FOR THE PAST SIX OR SEVEN YEARS I have been teaching a course called "Advanced Programming Techniques" [7] to (mostly) sophomore and junior computer science majors. The course covers an eclectic mix of languages, tools, and techniques, with regular coding assignments and final group projects.

Dijkstra famously remarked that "Program testing can be quite effective for showing the presence of bugs, but is hopelessly inadequate for showing their absence." Nevertheless, programmers who think about testing are more likely to write correct code in the first place. Thus I try to encourage the students to do intensive testing of their code, both in one-week assignments in the first half of the semester and as part of their eight-week projects.

My personal interest in testing comes from maintaining a widely used version of AWK for the past 25 years. In an attempt to keep the program working, and to maintain my sanity as bugs are fixed and the language slowly evolves, I have created somewhat over 1000 tests, which can be run automatically by a single command. Whenever I make a change or fix a bug, the tests are run. Over the years this has caught most careless mistakes—once fixed, things tend to stay fixed and new code rarely breaks old code.

The approach is a good way to think about testing small programs, and many of the techniques scale up to larger programs. Furthermore, the topic is good for teaching about other aspects of programming, such as tools, specialized languages, scripting, performance, portability, standards, documentation—you name it, it's there somewhere. So I find myself using this material in class in a variety of ways, I think to the benefit of the students. And when I describe it to friends who work in the real world, they nod approvingly, since they want to hire people who can write programs that work and who understand something of how to produce software that others will use.

This report from the trenches focuses mainly on testing, with some digressions on topics that have led to useful class lessons. I am ignoring other important issues, such as formal methods both for writing correct programs and for validating them after the fact. These are often of great value, but I am reminded of Don Knuth's apposite comment, "Beware of bugs in the above code; I have only proved it correct, not tried it."

A Bit of History

Al Aho, Peter Weinberger, and I created AWK in 1977 [1]; around 1981 I became the de facto owner and maintainer of the source code, a position I still hold. The language is so small and simple that it remains a widely used tool for data manipulation and analysis and for basic scripting, though there are now many other scripting languages to choose from. There are multiple implementations, of which GNU's GAWK is the most widely used, and there is a POSIX standard.

The language itself is small, and our implementation [6] reflects that. The first version was about 3000 lines of C, Yacc, and Lex; today, it is about 6200 lines of C and Yacc, Lex having been dropped for reasons to be discussed later. The code is highly portable; it compiles without #ifdefs and without change on most UNIX and Linux systems and on Windows and Mac OS X. The language itself is stable; although there is always pressure to add features, the purveyors of various implementations have taken a hard line against most expansion. This limits the scope of AWK's application but simplifies life for everyone.

Test Cases

Because both language and implementation are small, the program is self-contained, and there are multiple implementations, AWK is a good target for thorough testing.

This section describes general classes of test cases. In the early days we collected and invented test cases in an ad hoc way, but gradually we became more systematic. Nevertheless, there is definitely a random flavor to many of the tests. In total, there are nearly 7000 lines of tests, in more than 350 files—there are more lines of tests than of source code. This emphasis on testing is typical of software with stringent reliability requirements, which might well have 10 times as much test as code, but it is way beyond what one encounters in a class. Merely citing the scale of testing wakes up a class; it may help convince them that I am serious when I ask them to include tests with their assignments.

One major test category probes language features in isolation: numeric and string expressions, field splitting, input and output, built-in variables and functions, control flow constructs, and so on. There are also a lot of representative small programs, such as the very short programs in the first two chapters of the AWK book. For example, the first test,

```
{ print }
```

prints each input line and thus copies input to output.

AWK was originally meant for programs like this, only a line or two long, often composed at the command-line prompt. We were surprised when people began creating larger programs, since some aspects of the implementation didn't scale, and large AWK programs are prone to bugs. But bigger complete programs like the chem preprocessor [5] make it possible to test features working together rather than in isolation, so there are a number of tests of this kind.

Some aspects of AWK are themselves almost complete languages—for instance, regular expressions, substitution with `sub` and `gsub`, and expression evaluation. These can be tested by language-based approaches, as we will see below.

There are about 20 tests that exercise the most fundamental AWK actions: input, field splitting, loops, regular expressions, etc., on large inputs. The runtimes for old and new versions of the program are compared; although real performance tuning is notoriously difficult, this provides a rough check that no performance bug is inadvertently introduced.

Each time a bug is found, a new set of tests is created. These are tests that should have been present; if they had been, they would have exposed the bug. In general such tests are small, since they are derived from the smallest programs that triggered the bug.

New tests are also added for new features or behaviors. AWK does not change much, but features are added or revised occasionally. Each of these is accompanied by a set of tests that attempt to verify that the feature works properly. For example, the ability to set variables on the command line was added and then refined as part of the POSIX standardization process; there are now about 20 tests that exercise this single feature.

One of the most fruitful places to look for errors is at "boundary conditions." Instances include creating fields past the last one on an input line, trying to set nominally read-only variables like NR or NF, and so on. There is also a group of stress tests: very large strings, very long lines, huge numbers of fields, and the like are all places where implementations might break. In theory, there are no fixed size limits on anything of significance, so such tests attempt to verify proper behavior when operating outside normal ranges.

One useful technique is to move the boundaries closer, by setting internal program limits to small values. For example, the initial size of the hash table for associative arrays is one element; in this way all arrays are forced to grow multiple times, thus exercising that part of the code. This same approach is used for all growable structures, and it has been helpful in finding problems; indeed, it just recently exposed a memory allocation failure on Linux that does not appear on Solaris.

One productive boundary condition test involved trying all AWK "programs" consisting of a single ASCII character, such as

```
awk @          (illegal) single-character program
```

Some of these are legal (letter, digit, comment, semicolon) and possibly even meaningful (non-zero digit), but most are not. This exercise uncovered two bugs in the lexical analyzer, a story to which we will return later.

Every command-line option is tested, and there are also tests that provoke each error message except for those that "can't happen."

I have tried to create enough coverage tests that every statement of the program will be executed at least once. (Coverage is measured with `gcov`, which works with `gcc`.) This ideal is hard to achieve; the current set of tests leaves about 240 lines uncovered, although about half of those are impossible conditions or fatal error messages that report on running out of memory.

I found one bug with coverage measurements while preparing this paper—the nonstandard and undocumented option `-safe` that prevents AWK from writing files and running processes was only partly implemented.

For all tests, the basic organization is to generate the correct answer somehow—from some other version of AWK, by some other program, by copying it from some data source—then run the new version of AWK to produce its version of the answer, and then compare them. If the answers differ, an error is reported. So each of the examples, such as { print } above, is in a separate file and is tested by a shell loop like this:

```
for i
        do
                echo "$i:"
                awk -f $i test.data >foo1 # old awk
                a.out -f $i test.data >foo2 # new awk
                if cmp -s foo1 foo2
                then true
                else echo "BAD: test $i failed"
        fi
done
```

If all goes well, this prints just the file names. If something goes wrong, however, there will be lines with the name of the offending file and the string BAD that can be grepped for. There is even a bell character in the actual implementation so errors also make a noise. If some careless change breaks everything (not unheard of), running the tests causes continuous beeping.

Test Data

The other side of the coin is the data used as input to test cases. Most test data is straightforward: orderly realistic data such as real users use. Examples include the "countries" file from Chapter 2 of [1]; the password file from a UNIX system; the output of commands such as who or ls -l; or big text files such as the Bible, dictionaries, stock price listings, and Web logs.

Boundary-condition data is another category; this includes null inputs, empty files, empty fields, files without newlines at the end or anywhere, files with CRLF or CR only, etc.

High-volume input—big files, big strings, huge fields, huge numbers of fields—all stress a program. Generating such inputs by a program is easiest, but sometimes they are better generated internally, as in this example that creates million-character strings in an attempt to break sprintf:

```
echo 4000004 >foo1
awk '
BEGIN {
        x1 = sprintf("%1000000s\n", "hello")
        x2 = sprintf("%-1000000s\n", "world")
        x3 = sprintf("%1000000.1000000s\n", "goodbye")
        x4 = sprintf("%-1000000.1000000s\n", "everyone")
        print length(x1 x2 x3 x4)
}' >foo2
cmp -s foo1 foo2 || echo 'BAD: T.overflow huge sprintf'
```

(The very first bug in my record of bug fixes, in 1987, says that a long string in printf causes a core dump.) Again, the error message identifies the test file and the specific test within it.

Random input, usually generated by program, provides yet another kind of stress data. A small AWK program generates files with lots of lines containing random numbers of fields of random contents; these can be used for a variety of tests. Illegal input is also worth investigating. A standard example is binary data, since AWK expects everything to be text; for example, AWK survives these two tests:

```
awk -f awk             "program" is raw binary
awk '{print}' awk      input data is raw binary
```

by producing a syntax error as expected for the first and by quietly stopping after some early null byte in the input for the second. The program generally seems robust against this kind of assault, though it is rash to claim anything specific.

Test Mechanization

We want to automate testing as much as possible: let the machine do the work. There are separate shell scripts for different types of tests, all run from a single master script. In class, I describe the idea of running a lot of tests, then type the command and talk through what is happening as the test output scrolls by, a process that today takes two or three minutes depending on the system. If nothing else, the students come away with a sense of the number of tests and their nature.

Regression tests compare the output of the new version of the program to the output of the old version on the same data. Comparing independent implementations is similar to regression testing, except that we are comparing the output of two independent versions of the program. For AWK, this is easy, since there are several others, notably GAWK.

Independent computation of the right answer is another valuable approach. A shell script writes the correct answer to a file, runs the test, compares the results, and prints a message in case of error, as in the big-string example above. As another illustration, this is one of the tests for I/O redirection:

```
awk 'NR%2 == 1 { print >>"foo" }
     NR%2 == 0 { print >"foo" }' /etc/passwd
cmp -s foo /etc/passwd || echo 'BAD: T.redir (print > and >>"foo")'
```

This prints alternate input lines with the ">" and ">>" output operators; the result at the end should be that the input file has been copied to the output.

Although this kind of test is the most useful, since it is the most portable and least dependent on other things, it is among the hardest to create.

Notice that these examples use shell scripts or a scripting language like AWK itself to control tests, and they rely on I/O redirection and UNIX tools such as echo, grep, diff, cmp, sort, wc. This teaches something about UNIX itself, as well as reminding students of the value of small tools for mechanizing tasks that might otherwise be done by hand.

Specialized Languages

The most interesting kind of test is the use of specialized languages to generate test cases and assess their results. A program can convert a compact specification into a set of tests, each with its own data and correct answer, and run them. Regular expressions and substitution commands are tested this way. For regular expressions, an AWK program (naturally) converts a sequence of lines like this:

```
^a.$  ~        ax
               aa
      !~       xa
               aaa
               axy
               ""
```

into a sequence of test cases, each invoking AWK to run the test and evaluate the answer. In effect, this is a simple language for regular expression tests:

```
^a.$  ~        ax        the pattern ^a.$ matches ax
               aa        and matches aa
      !~       xa        but does not match xa
               aaa       and does not match aaa
               axy       and does not match axy
               ""        and does not match the empty string
```

A similar language describes tests for the sub and gsub commands. A third language describes input and output relations for expressions. The test expression is the rest of the line after the word "try," followed by inputs and correct outputs one per line; again, an AWK program generates and runs the tests.

```
try { print ($1 == 1) ?   "yes" :  "no" }
1                          yes
1.0                        yes
1E0                        yes
0.1E1                      yes
10E-1                      yes
01                         yes
10                                 no
10E-2                              no
```

There are about 300 regular expression tests, 130 substitution tests, and 100 expression tests in these three little languages; more are easily added. These languages demonstrate the value of specialized notations, and show how one can profitably separate data from control flow. In effect, we are doing table-driven testing.

Of course, this assumes that there is a version of AWK sufficiently trusted to create these tests; fortunately, that is so basic that problems would be caught before it got this far. Alternatively, they could be written in another language.

Another group of tests performs consistency checks. For example, to test that NR properly gives the number of input records after all input has been read:

```
    { i++ } # add 1 for each input line
END { if (i != NR) print "BAD: inconsistent NR" }
```

Splitting an input line into fields should produce NF fields:

```
{ if (split($0, x) != NF)
        print "BAD: wrong field count, line ", NR
}
```

Deleting all elements of an array should leave no elements in the array:

```
BEGIN {
        for (i = 0; i < 100000; i++) x[i] = i
        for (i in x) delete x[i]
        n = 0
        for (i in x) n++
        if (n != 0)
                print “BAD: delete count ” n “ should be 0”
    }
```

Checking consistency is analogous to the use of assertions or pre- and post-conditions in programming.

Advice

This section summarizes some of the lessons learned. Most of these are obvious and every working programmer knows them, but students may not have been exposed to them yet. Further advice may be found in Chapter 6 of *The Practice of Programming* [3].

Mechanize. This is the most important lesson. The more automated your testing process, the more likely that you will run it routinely and often. And the more that tests and test data are generated automatically from compact specifications, the easier it will be to extend them. For AWK, the single command REGRESS runs all the tests. It produces several hundred lines of output, but most consist just of file names that are printed as tests progress. Having this large and easy-to-run set of tests has saved me from much embarrassment. It’s all too easy to think that a change is benign when, in fact, something has been broken. The test suite catches such problems with high probability.

Watching test results scroll by obviously doesn’t work for large suites or ones that run for a long time, so one would definitely modify this to automate reporting of errors if scaling up.

Make test output self-identifying. You have to know what tests ran and especially which ones caused error messages, core dumps, etc.

Make sure you can reproduce a test that fails. Reset random number generators and files and anything else that might preserve state from one test to the next. Each test should start with a clean slate.

Add a test for each bug. Better tests originally should have caught the bug. At least this should prevent you from having to find this bug again.

Add tests for each new feature or change. A good time to figure out whether a new feature or change works correctly is while it’s fresh; presumably there was some testing anyway, so make sure it’s preserved.

Never throw away a test. A corollary to the previous point.

Make sure that your tester reports progress. Too much output is bad, but there has to be some. The AWK tests report the name of each file that is being tested; if something seems to be taking too long, this gives a clue about where the problem is.

Watch out for things that break. Make the test framework robust against the many things that can go wrong: infinite loops, tests that prompt for user input then wait forever for a response, tests that print spurious output, and tests that don’t really distinguish success from failure.

Make your tests portable. Tests should run on more than one system; otherwise, it’s too easy to miss errors in both your tests and your programs. Shell commands, built-ins (or not) like echo, search paths for commands, and the like are all potentially different on different machines; just because something works one place is no guarantee that it will work elsewhere. I eventually wrote my own echo command, since the shell built-ins and local versions were so variable.

A few years ago I moved the tests to Solaris from the SGI Irix system, where they had lived happily for more than a decade. This was an embarrassing debacle, since lots of things failed. For instance, the tests used grep -s to look for a pattern without producing any output; the -s option means "silent," i.e., status only. But that was true in 7th Edition UNIX, not on other systems, where it often means "don't complain about missing files." The -q of Linux means "quiet," but it's illegal on Solaris. printf on some systems prints -0 for some values of zero. And so on. It was a mess, and although the situation is now better, it's still not perfect.

A current instance of this problem arises from the utter incompatibility of the time command on different UNIX systems. It might be in /bin or in /usr/bin or be a shell built-in (in some shells), and its output format will vary accordingly. And if it's a built-in its output can't be redirected! It's tough to find a path through this thicket; I eventually wrote my own version of time.

It has also been harder than anticipated to use GAWK as a reference implementation; although the AWK language is ostensibly standardized, there are enough dark corners—for instance, when does a change in a field-splitting string take effect?—that at least some tests just produce different answers. The current test suite marks those as acceptable differences, but this is not a good long-term solution.

Check your tests and scaffolding often. It's easy to get into a rut and assume that your tests are working because they produce the expected (i.e., mostly empty) output. Go back from time to time and take a fresh look—paths to programs and data may have changed underfoot and you could be testing the wrong things. For instance, a few years ago, my "big" data set somehow mutated into a tiny one. Machines have sped up to the extent that I recently increased the "big" data by another order of magnitude.

Keep records. I maintain a FIXES file that describes every change to the code since the AWK book was published in 1988; this is analogous to Knuth's "The Errors of TEX" [4], though far less complete. For example, this excerpt reveals a classic error in the C lexer:

> Jul 31, 2003: fixed, thanks to andrey chernov and ruslan ermilov, a bug in lex.c that mis-handled the character 255 in input. (it was being compared to EOF with a signed comparison.)

As hinted at above, the C lexer has been a source of more than one problem:

> Feb 10, 2001: fixed an appalling bug in gettok: any sequence of digits, +, -, E, e, and period were accepted as a valid number if it started with a period. this would never have happened with the lex version.

And one more, just to show how bugs can hide for very long periods indeed:

> Nov 22, 2003: fixed a bug in regular expressions that dates (so help me) from 1977; it's been there from the beginning. an anchored longest match that was longer than the number of states triggered a failure to initialize the machine properly. many thanks to moinak ghosh for not only finding this one but for providing a fix, in some of the most mysterious code known to man.

Teaching

I've mentioned several places where a discussion of testing is a natural part of some other class topic; here are a handful of others.

One early assignment asks the students to program some variant of the compact regular expression code in Chapter 9 of *The Practice of Programming* [3]. As part of the assignment, they are required to create a number of tests in a format similar to the specialized language shown above and to write a program to exercise their code using their tests. Naturally, I combine all their tests with my own. It's sobering to see how often programs work well with their author's tests but not with tests written by others; I continue to experiment with assignments that explore this idea. (It's also sobering to see how often the purported tests are in fact not correct, which is another important lesson.)

I've also tried this assignment with unit tests—self-contained function calls in a special driver routine—instead of a little language. The results have been much less successful for checking individual programs, and it's harder to combine tests from a group of sources. For this application, the language approach seems better.

Another assignment asks the students to write a Base64 encoder and decoder from the one-page description in RFC 2045. This is a good example of writing code to a standard, and since there are reference (binary) implementations like OpenSSH, it's possible to mix and match implementations, all controlled by a shell script, to verify interoperability. I also ask students to write a program to generate a collection of nasty tests, which forces them to think about boundary conditions. (It's a good idea to write a program anyway, since it's easier to create arbitrary binary inputs by program than with a text editor. A surprising number of student programs don't handle non-ASCII inputs properly, and this potential error has to be tested for.)

Yet another assignment gives students a taste of a frequent real-world experience: having to make a small change in a big unfamiliar program without breaking anything. The task is to download AWK from the Web site, then add a couple of small features, like a repeat-until loop or a new built-in function. This is easily done by grepping through the source looking for affected places, then adding new code by pattern-matching old code. Naturally, they also have to provide some self-contained tests that check their implementations, and I can run my own tests to ensure that nothing else was affected.

Two years ago, an especially diligent student ran some GAWK tests against the AWK he had built, and encountered an infinite loop in parsing a program, caused by a bug in my lexer. In 1997 I had replaced the ancient Lex lexical analyzer with handcrafted C code in an effort to increase portability. As might have been predicted, this instead decreased reliability; most of the bugs of the past few years have been in this C code.

In any case, I eventually found the bug but by then it was time for the next class. So I assigned the new class the task of finding and fixing the bug (with some generous hints), and also asked them to find the shortest test case that would display it. Most students fixed the bug, and several came up with tests only two characters long (shorter than I had found) that triggered the infinite loop. Unfortunately, since that bug fix is now published, I can no longer use the assignment. Fortunately, the `-safe` bug described above should work well in its place.

Conclusions

For working programmers, there's no need to belabor the importance of testing. But I have been pleased to see how much testing can be included in a programming course—not as an add-on lecture but as an integral part of a wide variety of other topics—and how many useful lessons can be drawn from it.

It's hard work to test a program, and there are often so many other pressures on one's time and energy that thorough testing can slide to the back burner. But in my experience, once some initial effort has gone into creating tests and, more important, a way to run them automatically, the incremental effort is small and the payoff very large. This has been especially true for AWK, a language that has lived on far beyond anything the authors would have believed when they wrote it nearly 30 years ago.

Acknowledgments

I am deeply indebted to Arnold Robbins and Nelson Beebe for nearly two decades of invaluable help. Arnold, the maintainer of GAWK, has provided code, bug fixes, test cases, advice, cautionary warnings, encouragement, and inspiration. Nelson has provided thoughtful comments and a significant number of test cases; his meticulous attention to portability issues is without peer. My version of AWK is much the better for their contributions. I am also grateful to many others who have contributed bug reports and code. They are too numerous to list here but are cited in the FIXES file distributed with the source. Jon Bentley's essays on scaffolding and little languages [2] have influenced my thinking on testing and many other topics. My thanks also to Jon, Gerard Holzmann, and David Weiss for most helpful comments on drafts of this paper.

REFERENCES

[1] Al Aho, Brian Kernighan, and Peter Weinberger, *The AWK Programming Language,* Addison-Wesley, 1988.

[2] Jon Bentley, *Programming Pearls,* Addison-Wesley, 2000.

[3] Brian Kernighan and Rob Pike, *The Practice of Programming,* Addison-Wesley, 1998.

[4] Donald E. Knuth, "The Errors of TEX," *Software—Practice and Experience,* vol. 19, no. 7, July 1989, pp. 607–685.

[5] Jon Bentley, Lynn Jelinski, and Brian Kernighan, "CHEM—A Program for Phototypesetting Chemical Structure Diagrams," *Computers and Chemistry,* vol. 11, no. 4, 1987, pp. 281–297.

[6] Source code for AWK is available at http://cm.bell-labs.com/cm/cs/awkbook.

[7] The Web site for COS 333 is http://www.cs.princeton.edu/courses/archive/spring06/cos333.

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

modular system programming in MINIX 3

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

jnherder@cs.vu.nl

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

herbertb@cs.vu.nl

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

bjgras@cs.vu.nl

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

philip@cs.vu.nl

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

ast@cs.vu.nl

WHEN THE FIRST MODERN OPERATing systems were being developed in the early 1960s, the designers were so worried about performance that these systems were written in assembly language, even though high-level languages such as FORTRAN, MAD, and Algol were well established. Reliability and security were not even on the radar. Times have changed and we now need to reexamine the need for reliability in operating systems.

If you ask ordinary computer users what they like least about their current operating system, few people will mention speed. Instead, it will probably be a neck-and-neck race among mind-numbing complexity, lack of reliability, and security in a broad sense (viruses, worms, etc.). We believe that many of these problems can be traced back to design decisions made 40 or 50 years ago. In particular, the early designers' goal of putting speed above all else led to monolithic designs with the entire operating system running as a single binary program in kernel mode. When the maximum memory available to the operating system was only 32K words, as was the case with MIT's first timesharing system, CTSS, multi-million-line operating systems were not possible and the complexity was manageable.

As memories got larger, so did the operating systems, until we got to the current situation of operating systems with hundreds of functions interacting in such complex patterns that nobody really understands how they work anymore. While Windows XP, with 5 million LoC (Lines of Code) in the kernel, is the worst offender in this regard, Linux, with 3 million LoC, is rapidly heading down the same path. We think this path leads to a dead end.

Various studies have shown the number of bugs in programs to be in the range 1–20 bugs per 1000 LoC [1]. Furthermore, operating systems tend to be trickier than application programs, and device drivers have an order of magnitude more bugs per thousand LoC than the rest of the operating system [2, 3]. Given millions of lines of poorly understood code interacting in unconstrained ways within a single address space, it is not surprising that we have reliability and security problems.

Operating System Reliability

In our view, the only way to improve operating system reliability is to get rid of the model of the operating system as one gigantic program running in kernel mode, with every line of code capable of compromising or bringing down the system. Nearly all the operating system functionality, and especially all the device drivers, have to be moved to user-mode processes, leaving only a tiny microkernel running in kernel mode. Moving the entire operating system to a single user-mode process as in L^{4}Linux [4] makes rebooting the operating system after a crash faster, but does not address the fundamental problem of every line of code being critical. What is required is splitting the core of the operating system functionality—including the file system, process management, and graphics—into multiple processes, putting each device driver in a separate process, and very tightly controlling what each component can do. Only with such an architecture do we have a chance to improve system reliability.

The reasons that such a modular, multiserver design is better than a monolithic one are threefold. First, by moving most of the code from kernel mode to user mode, we are not reducing the number of bugs but we are reducing the power of each bug to cause damage. Bugs in user-mode processes have much less opportunity to trash critical kernel data structures and cannot touch hardware devices they have no business touching. The crash of a user-mode process is rarely fatal, whereas a crash of the kernel always is. By moving most of the code out of the kernel, we are moving most of the bugs out as well.

Second, by breaking the operating system into many processes, each in its own address space, we greatly restrict the propagation of faults. A bug in the audio driver may turn the sound off, but it cannot wipe out the file system by accident. In a monolithic system, in contrast, bugs in any function can destroy code and data structures in unrelated and much more critical functions.

Third, by constructing the system as a collection of user-mode processes, the functionality of each module can be clearly determined, making the entire system much easier to understand and simpler to implement. In addition, the operating system's maintainability will improve, because the modules can be maintained independently from each other, as long as interfaces and shared data structures are respected.

While this article does not focus on security directly, it is important to mention that operating system reliability and security are closely related. Security has usually been designed with the model of the multi-user system in mind, not a single-user system where that user will run hostile code. However, many security problems are caused by malicious code injected by viruses and worms exploiting bugs such as buffer overruns. By moving most of the code out of the kernel, exploits of operating system components are rendered far less powerful. Overwriting the audio driver's stack may allow the intruder to cause the computer to make weird noises, but it does not compromise system security, since the audio driver does not have superuser privileges. Thus, while we will not discuss security much hereafter, our design has great potential to improve security as well.

The observation that microkernels are good for reliability is not new. In the 1980s and 1990s numerous microkernels were constructed, including L4 [5], Mach [6], V [7], Chorus [8], and Amoeba [9]. None of these succeeded in displacing monolithic operating systems with microkernel-based ones, but we have learned a lot since then and the time is right to try again. Even Microsoft understands this. The next version of Windows (Vista) will feature many user-mode drivers, and Microsoft's Singularity research project is also based on a microkernel.

The MINIX 3 Architecture

To test out our ideas, we have constructed a POSIX-conformant prototype system. As a base for the prototype, we used the MINIX operating system due to its very small size and long history. MINIX is a free microkernel-based operating system that comes with complete source code, mostly written in C. The initial version was written by one of the authors (AST) in 1987, and has been studied by many tens of thousands of students at hundreds of universities for a period of 19 years; over the past 10 years there have been almost no bug reports concerning the kernel, presumably due to its small size.

We started with MINIX 2 and then modified it very heavily, moving the drivers out of the kernel and much more, but we decided to keep the name and call the new system MINIX 3. It is based on a microkernel now containing under 4000 LoC, with numerous user-mode servers and drivers that together constitute the operating system, as illustrated in Figure 1. Despite this unconventional structure, to the user the system appears to be just another UNIX variant. It runs two C compilers (ACK and gcc), as well as many popular utilities—Emacs, vi, Perl, Python, Telnet, FTP, and 300 others. Recently, X Windows has also been ported to it. MINIX 3 is available at http://www.minix3.org with all the source code under the BSD license.

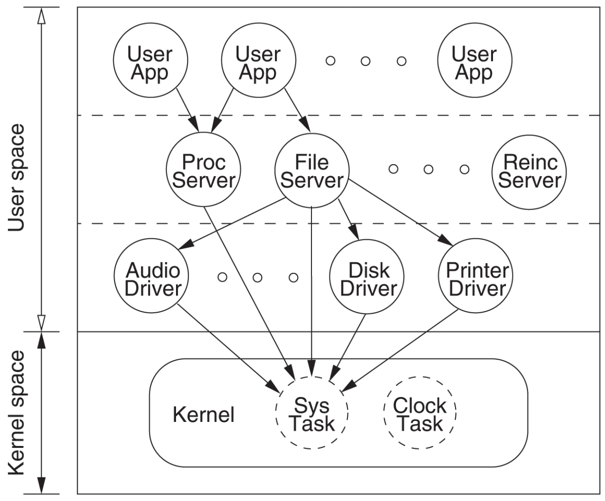


FIG. 1. SKETCH OF THE LAYERED ARCHITECTURE OF MINIX 3

All applications, servers, and drivers run as isolated, user-mode processes. A tiny, trusted kernel is the only part that runs in kernel mode. The layering is a logical one, as all user processes are treated equally by the kernel.

Briefly, the microkernel handles hardware interrupts, low-level memory management, process scheduling, and interprocess communication. The latter is accomplished by primitives that allow processes to send fixed-length messages to other processes they are authorized to send to. Most communication is synchronous, with a sender or receiver blocking if the other party is not ready. Sending a message takes about 500 nsec on a 2.2GHz Athlon. Although a system call usually takes two messages (a request and a reply), even 10,000 system calls/sec would use only 1% of the CPU, so message-passing overhead hardly affects performance at all. In addition, there is a nonblocking event notification mechanism. Pending notifications are stored in a compact bitmap that is statically declared as part of the process table. This message-passing scheme eliminates all kernel buffer management and kernel buffer overruns, as well as many deadlocks.

The next level up contains the device drivers, one per major device. Each driver is a user process protected by the MMU the same way ordinary user processes are protected. They are special only in the sense that they are allowed to make a small number of kernel calls to obtain kernel services. Typical kernel calls are writing a set of values to hardware I/O ports or requesting that data be copied to or from a user process. A bitmap in the kernel's process table controls which calls each driver (and server) can make. Also, the kernel knows which I/O ports the driver is allowed to use, and copying is possible only with explicit permission.

The operating system interface is formed by a set of servers. The main ones are the *file server*, the *process manager*, and the *reincarnation server*. User processes make POSIX system calls by sending a message to one of these servers, which then carries out the call. The reincarnation server is especially interesting, since it is the parent process of all the servers and drivers. It is different from init, which is the root of ordinary user processes, as it manages and guards the operating system. If a server or driver crashes or otherwise exits, it becomes a zombie until the reincarnation server collects it, at which time the reincarnation server looks in its tables to determine what to do. The usual action is to create a new driver or server and to inform the other processes that it is doing so.

Finally, we have the ordinary user processes, which have the ability to send fixed-length messages to some of the servers requesting service, but basically have no other power. While message passing is used under the hood, the system libraries offer the normal POSIX API to the programmer.

Living with Programming Restrictions

Having explained why microkernels are needed and how MINIX 3 is structured, it is now time to get to the heart of this article: the MINIX 3 programming model and its implications. We will point out some of the properties, strengths, and weaknesses of the programming model in the text below, but before we start, it is useful to recall that, historically, restricting what programmers can do has often led to more reliable code. Let us consider several examples.

First, when the first MMUs appeared, user programs were forced to make system calls to perform I/O, rather than just start I/O devices themselves. Of course, some of them complained that making kernel calls was slower than talking to the I/O devices directly (and they were right), but a consensus eventually emerged saying that this restriction on what a programmer could do was worth the small performance penalty, since bugs in user code could no longer crash the computer.

Second, when E.W. Dijkstra wrote his now-famous letter "Goto Statement Considered Harmful" [10], a massive hue and cry was raised by many programmers who felt their style of writing spaghetti-like code was being threatened. Despite these initial objections, the idea caught on, and programmers learned to write well-structured programs.

Third, when object-oriented programming was introduced, many programmers balked at the idea, since they could no longer count on reading or tweaking data structures internal to other objects, a previously common practice in the name of efficiency. For example, when Java was introduced to C programmers, many of them saw it as a straitjacket, since they could no longer freely manipulate pointers. Nevertheless, object-oriented programming is now common and has led to better-quality code.

The MINIX 3 Restrictions

In a similar vein, the MINIX 3 programming model is also more restrictive for operating system developers than what came before it, but we believe these restrictions will ultimately lead to a more reliable system. For the time being, MINIX 3 is written in C, but gradually rewriting some of the modules in a type-safe language, such as Cyclone, might be possible someday. Let us start our overview of the model by looking at some of these restrictions.

Restricted kernel access. The MINIX 3 kernel exports various kernel calls to support the user-mode servers and drivers of the operating system. Each driver and server has a bitmap in the process table that restricts which of the kernel calls it may use. This protection is quite fine-grained, so, for example, a device driver may have permission to perform I/O or make copies to and from user processes, but not to shut down the system, create new processes, or (re)set restriction policies.

Memory protection. In the multiserver design of MINIX 3, all servers and drivers of the operating system run as isolated user-mode processes. Each is encapsulated in a private address space that is protected by the MMU hardware. An illegal access attempt to another process's memory raises an MMU exception and causes the offender to be killed by the process manager. Of course, the file system and device drivers need to interact with user processes to perform I/O, but this is done using safe virtual copies mediated by the kernel. A copy to another process is possible only when permission is explicitly given by that process or a trusted process such as the file system. This design takes away the trust from drivers and prevents memory corruption.

Restricted I/O port access. Each driver has a limited range of I/O ports that it may access. Since user processes do not have I/O privileges, the kernel has to mediate and can check whether the I/O request is permitted. The allowed port ranges are set when a driver is started. For ISA devices this is done with the help of configuration files; for PCI devices the port ranges are automatically determined by the PCI bus server. The valid port ranges for each driver are stored in the driver's process table entry in the kernel. This protection ensures that a printer driver cannot accidentally write garbage to the disk, because any attempt to write to the disk's I/O ports will result in a failed kernel call. Servers and ordinary user processes have no access to any I/O ports.

Restricted interprocess communication. Processes may not send messages to arbitrary processes. Again, the kernel keeps track of who may send to whom, and violations are prevented. The allowed IPC primitives and destinations are set by the reincarnation server when a new system process is started. For example, a driver may be allowed to communicate with just the file server and no other process. This feature eliminates some bugs where a process tries to send a message to another process that is not expecting it.

Operating System Development in User Space

Now let us look at some other aspects of the MINIX 3 programming model. While there are some restrictions, as pointed out above, we believe that programming in a multiserver operating system environment has many benefits and may lead to higher productivity and better code quality.

Short development cycle. The huge difference between a monolithic and a multiserver operating system immediately becomes clear when looking at the development cycle of operating system components. System programming on a monolithic system generally involves editing, compiling, rebuilding the kernel, and rebooting to test the new component. A subsequent crash will require another reboot, and tedious, low-level debugging usually follows, frequently without even a core dump. In contrast, the development cycle on a multiserver system like MINIX 3 is much shorter. Typically, the steps are limited to editing, compiling, testing, and debugging. We will elaborate on these steps below.

Normal programming model. Because drivers and servers are just ordinary user processes, they can use any libraries that are needed. In some cases, even POSIX system calls can be used. The ability to do these things can be contrasted with the more rigid environment available to programmers writing code for monolithic kernels. In essence, working in user mode makes programming easier.

No system downtime. The required reboots for a monolithic operating system effectively kick off all users, meaning that a separate development system is to be preferred. In MINIX 3, no reboots are required to test new components, so other users are not affected. Furthermore, bugs or other problems are isolated in the new components and cannot affect the entire system, because the new component is run as an independent process in a restricted execution environment. Problems thus cannot propagate as in a monolithic system.

Easy debugging. Debugging a device driver in a monolithic kernel is a real challenge. Often the system just halts and the programmer does not have a clue what went wrong. Using a simulator or emulator usually is of no use because typically the device being driven is something new and not supported by the simulator or emulator. In contrast, in the MINIX 3 model, a device driver is just a user process, so if it crashes, it leaves behind a core dump that can be inspected using all the normal debugging tools. In addition, the output of all printf() statements in drivers and servers automatically goes to a log server, which writes it to a file. After a failed run with the new driver, the programmer can examine the log to see what the driver was doing just before it died.

Low barrier to entry. Because writing drivers and servers is much easier than in conventional systems, researchers and others can try out new ones easily. Ease of experimentation can advance the field by allowing people with good ideas but little experience in kernel programming to try out their ideas and build prototypes they would not be able to construct with monolithic kernels. Although the hardest part of writing a new device driver may be understanding the actual hardware, other operating system components can be easy to realize. For example, the case study at the end of this article illustrates how semaphore functionality can be added to MINIX 3.

High productivity. Because operating system development in user space is easier, the programmer can get the job done faster. Also, since no lengthy system build is needed once the bug has been removed, time is saved. Finally, since the system need not be rebooted after a driver crash, as soon as the programmer has inspected the core dump and the log and has updated the code, it is possible to test the new driver without a reboot. With a monolithic kernel, two reboots are often needed: one to restart the system after the crash and one to boot the newly built kernel.

Good accountability. When a driver or server crashes, it is completely obvious which one it is (because its parent, the reincarnation server, knows which process exited). As a consequence, it is much easier than in monolithic systems to pin down whose fault the crash was and possibly who is legally liable for the damage done. Holding the producers of commercial software liable for their errors, in precisely the same way as the producers of tires, medicines, and other products are held accountable, may improve software quality.

Great flexibility. Our modular model offers great flexibility and makes system administration much easier. Since operating system modules are just processes, it is relatively easy to replace one. It becomes easier to configure the operating system by mixing and matching modules. Furthermore, if a device driver needs to be patched, this can usually be done on the fly, without loss of service or downtime. Module substitution is much harder in monolithic kernels and often requires a reboot. Finally, maintenance also becomes easier, because all modules are small, independent, and well understood.

Case Study: Message-Driven Programming in MINIX 3

We will now evaluate the MINIX 3 programming model aided by a little case study that shows how semaphore functionality can be added to MINIX 3. Although this is easier than implementing a new file server or device driver, it illustrates some important aspects of MINIX 3.

Semaphores are positive integers, equal to or greater than zero, and support two operations, UP and DOWN, to synchronize multiple processes trying to access a shared resource. A DOWN operation on semaphore *S* decrements *S* unless *S* is zero, in which case it blocks the caller until some other process increments *S* through an UP operation. Such functionality is typically part of the kernel in a monolithic system, but can be realized as a separate user-space server in MINIX 3.

The structure of the MINIX 3 semaphore server is shown in Fig. 2. After initialization, the server enters a main loop that continues forever. In each iteration the server blocks and waits until a request message arrives. Once a message has been received, the server inspects the request. If the type is known, the associated handler function is called to process the request, and the result is returned unless the caller must be blocked. Illegal request types directly result in an erroneous reply.

As mentioned above, ordinary user processes in MINIX 3 are restricted to synchronous message passing. A request will block the caller until the response has arrived. We will use this to our advantage when constructing the semaphore server. For UP operations, the server simply increments the semaphore and directly sends a reply to let the caller continue. For DOWN operations, in contrast, the reply is withheld until the semaphore can be decremented, effectively blocking the caller until it is properly synchronized. The semaphore has an associated (FIFO) queue of processes to keep track of processes that are blocked. After an UP operation, the queue is checked to see whether a waiting process can be unblocked.

```
void semaphore_server( ) {
    message m;
    int result;
    /* Initialize the semaphore server. */
    initialize( );
    /* Main loop of server. Get work and process it. */
    while(TRUE) {

        /* Block and wait until a request message arrives. */
        ipc_receive(&m);

        /* Caller is now blocked. Dispatch based on message type. */
        switch(m.m_type) {
           case UP:       result = do_up(&m);       break;
           case DOWN: result = do_down(&m);     break;
           default:       result = EINVAL;
        }
        /* Send the reply, unless the caller must be blocked. */
        if (result != EDONTREPLY) {
            m.m_type = result;
            ipc_reply(m.m_source, &m);
        }
    }
}
```

FIG. 2. THE MAIN LOOP OF A SERVER IMPLEMENTING ONE SEMAPHORE, *S*

All servers and drivers have a similar main loop. The function initialize() *is called once before entering the main loop, but is not shown here. The handler functions* do_up() *and* do_down() *are given in Fig. 3.*

With the structure of the semaphore server in place, we need to arrange that user processes can communicate with it. Once the server has been started it is ready to serve requests. In principle, the programmer can construct request messages and send them to the new server using ipc_request(), but such details usually are conveniently hidden in the system libraries, along with the other POSIX functions. Typically, new library calls sem_up() and sem_down() would be added to libc to handle these calls. Although this case study covers a very simplified semaphore server, it can easily be extended to conform to the POSIX semaphore specification, handle multiple semaphores, etc.

The modular structure of MINIX 3 helps to speed up the development of the semaphore server in several ways. First, it can be implemented independently from the rest of the operating system, just like ordinary user applications. When it is finished, it can be compiled as a stand-alone application and be dynamically started to become part of the operating system. It is not necessary to build a new kernel or to reboot the system, which prevents downtime, other users from being kicked off, disruption of Web, mail, and FTP servers, etc. When the server is started, its privileges are restricted according to the principle of least authority, so that testing and debugging of the new semaphore server can be done without affecting the rest of the system. Once it is ready, the startup scripts can be configured to load the semaphore server automatically during operating system initialization.

```
int do_down(message *m_ptr) {
    /* Resource available. Decrement semaphore and reply. */
    if (s > 0) {
        s = s – 1;                      /* take a resource */
        return(OK);                     /* let the caller continue */
    }
    /* Resource taken. Enqueue and block the caller. */
    enqueue(m_ptr->m_source);  /* add process to queue */
    return(EDONTREPLY);             /* do not reply in order to block the caller */
}


int  do_up(message *m_ptr) {
message m;                          /* place to construct reply message */
    /* Add resource, and return OK to let caller continue. */
    s = s + 1;                          /* add a resource */

    /* Check if there are processes blocked on the semaphore. */
    if (queue_size() > 0) {             /* are any processes blocked? */
        m.m_type = OK;
        m.m_source = dequeue();  /* remove process from queue */
        s = s – 1;                      /* process takes a resource */
        ipc_reply(m.m_source, m); /* reply to unblock the process */
    }
    return(OK);                         /* let the caller continue */
}
```

FIG. 3. up **AND** down **OPERATIONS OF THE SEMAPHORE SERVER**

The functions enqueue(), dequeue(), and queue_size() do list management and are not shown.

Conclusion

MINIX 3 is a new, fully modular operating system designed to be highly reliable. Like other innovations, our quest for reliability imposes certain restrictions upon the execution environment, but the multiserver environment of MINIX 3 makes life much easier for the OS programmer. The development cycle is shorter, system downtime is no longer required, the programming interface is more POSIX-like, and testing and debugging become easier. Programmer productivity is likely to increase, and code quality might improve because of better accountability. The system administrator also benefits, since MINIX 3 improves configurability and maintainability of the operating system. Finally, we have illustrated the message-driven programming model of MINIX 3 with the construction of a simple semaphore server and discussed how its development benefits from the modularity of MINIX 3. Interested readers can download MINIX 3 (including all the source code) from http://www.minix3.org. Over 50,000 people have already downloaded it; try it yourself.

REFERENCES

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

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

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

[4] H. Härtig, M. Hohmuth, J. Liedtke, S. Schönberg, and J. Wolter, "The Performance of μ-Kernel–Based Systems," *Proceedings of the 16th Symposium on Operating System Principles*, 1997, pp. 66–77.

[5] J. Liedtke, "On μ-Kernel Construction," *Proceedings of the 15th ACM Symposium on Operating System Principles*, 1995, pp. 237–250.

[6] M. Accetta, R. Baron, D. Golub, R. Rashid, A. Tevanian, and M. Young, "Mach: A New Kernel Foundation for UNIX Development," *Proceedings of the USENIX 1986 Summer Conference*, 1986, pp. 93–112.

[7] D.R. Cheriton, "The V Kernel: A Software Base for Distributed Systems," *IEEE Software,* vol. 1, no. 2, 1984, pp. 19–42.

[8] A. Bricker, M. Gien, M. Guillemont, J. Lipkis, D. Orr, and M. Rozier, "A New Look at Microkernel–Based UNIX Operating Systems: Lessons in Performance and Compatibility," *Proceedings of the EurOpen Spring 1991 Conference*, 1991, pp. 13–32.

[9] S. Mullender, G. Van Rossum, A.S. Tanenbaum, R. Van Renesse, and H. Van Staveren, "Amoeba: A Distributed Operating System for the 1990s," *IEEE Computer Magazine,* vol. 23, no. 5, 1990, pp. 44–54.

[10] E.W. Dijkstra, "Goto Statement Considered Harmful," *Communications of the ACM*, vol. 11, no. 3, 1968, pp. 147–148.

DIOMIDIS SPINELLIS

some types of memory are more equal than others

Diomidis Spinellis is an associate professor in the Department of Management Science and Technology at the Athens University of Economics and Business, a FreeBSD committer, and a four-times winner of the International Obfuscated C Code Contest.

dds@aueb.gr

Parts of this article are excerpted from Diomidis Spinellis's *Code Quality: The Open Source Perspective*, Addison Wesley, 2006. The last section was inspired by the book's Exercise 5–8.

IF WE WANT TO MAKE INTELLIGENT decisions regarding the performance of our systems, we must understand how the various types of memory we find in them work together to provide us with the illusion of a huge and blindingly fast memory store. For example, a program's space requirements often affect its execution speed. This happens because a computer system's memory is a complex amalgam of various memory technologies, each with different cost, size, and performance characteristics. Making our program's working set small enough to get a front seat on a processor's level 1 cache may provide us with a very noticeable boost in its execution speed. Along the same lines, in today's networked computing environments and distributed applications, lower size requirements translate into lower bandwidth requirements and, therefore, swifter program loading and operation. Finally, a program's service capacity is often constrained by the space requirements of its data set.

Due to a number of engineering decisions involving complicated tradeoffs, modern computers sport numerous different memory systems layered on top of each other. (As a general rule, whenever you see "complicated tradeoffs," read "cost.") At any time, our data will be stored in one (or more) of these many layers, and the way a program's code is organized may take advantage of the storage system's organization or be penalized by it. Some of the layers we will talk about are related to caching. In this article we describe them from the viewpoint of storage organization.

Let us summarize how data storage is organized on a modern computer. Figure 1, below, illustrates the hierarchy formed by different storage technologies. Elements near the top represent scarce resources: fast but expensive. As we move toward the bottom the elements represent abundant resources: cheap but slow. The fastest way to have a processor process a data element is for the element to be in a register (or an instruction). The register is encoded as part of the CPU instruction and is immediately available to it. However, this advantage means that processors offer only a

↓ Increasing size	CPU registers
	Level 1 cache (on chip)
	Level 2 cache
	Level 3 cache (off chip)
	Main memory
	Disk cache and banked memory
	Paged out memory
	File-based disk storage
↑ Increasing speed and cost	Off line storage

FIGURE 1. A MODERN COMPUTER'S STORAGE HIERARCHY

small fixed number of registers (eight, for example, on the ia-32; 128 on Sun's SPARC architecture.) See how a data processing instruction (such as add) is encoded on the arm architecture:

31 28	27 26	25	24 21	20	19 16	15 12	11 0
Cond	00	I	Opcode	S	Rn	Rd	Operand 2

Rn is the source register and Rd the destination. Each register is encoded using four bits, limiting the number of registers that can be represented on this architecture to 16. Registers are used for storing local variables, temporary values, function arguments, and return values. Nowadays, they are allocated to their various uses by the compiler, which uses extremely sophisticated algorithms for optimizing performance at a local and global level. In older programs you may find this allocation specified by the programmers, based on their intuition of which values should be placed in a register; here is a typical example from the Korn shell source code:

```
struct tbl *
global(n)
      register const char *n;
{
      register struct block *l = e->loc;
       register struct tbl *vp;
      register int c;
      unsigned h;
      bool_t array;
      int val;
```

This strategy might have been beneficial when compilers had to fit in 64KB of memory and could not afford to do anything clever with register allocation; modern compilers simply ignore the register keyword.

Main Memory and Its Caches

The next four layers of our hierarchy (from the level 1 cache up to the main memory) involve the specification of data through a memory address. This (typically 16, 32, or 64-bit) address is often encoded on a word separate from the instruction (it can also be specified through a register) and thus may involve an additional instruction fetch. Worse, it involves interfacing with dynamic RAMs, the storage technology used for a computer's main memory, which is simply not keeping pace with the speed increases of modern processors. Fetching an instruction or data element from main memory can have the processor wait for a time equivalent to that of the execution of hundreds of instructions. To minimize this penalty, modern processors include facilities for storing temporary copies of frequently used data on faster, more versatile, more easily accessible, and, of course, more expensive memory: a *cache*. For a number of reasons a memory cache is typically organized as a set of blocks (typically 8–128 bytes long) containing the contents of consecutive memory addresses. Keep this fact in mind; we'll come back to it later on.

The *level 1 cache* is typically part of the processor's die. It is often split into an area used for storing instructions and one used for storing data, because the two have different access patterns. To minimize the cache's impact on the die size (and therefore on the processor's production yield[1] and its cost), this cache is kept relatively small. For example, the Sun Micro SPARC I featured a 4KB instruction and a 2KB data cache; moving upward, the Intel 3.2GHz Pentium 4 processor features a 1MB cache.

1. A larger processor die means there is a higher chance for an impurity to result in a malfunctioning chip, thus lowering the production's yield.

Because of the inherent size limitations of the on-chip cache, a *level 2 cache* is sometimes implemented through a separate memory chip and control logic, either packaged with the processor or located near the processor. This can be a lot larger: it used to be 64KB on early 486 PC motherboards; an Intel 3.2GHz Xeon processor comes with 2MB. Finally, computer manufacturers are increasingly introducing in their designs a *level 3 cache*, which either involves different speed versus cost tradeoffs or is used for keeping a coherent copy of data in multiprocessor designs.

How do these levels of the memory hierarchy relate to our code and its properties? By reducing a program's memory consumption and increasing its locality of reference, we can often speed up its performance. All of us have witnessed the pathological case where increased memory consumption coupled with a lack of locality of reference leads to a dramatic performance drop due to thrashing. In the following paragraphs we will examine the winning side of the coin, where appropriate design and implementation decisions can lead to time performance increases.

Memory savings can translate into speed increases when the corresponding data set is made to fit into a more efficient part of a memory hierarchy. In an ideal world, all of our computer's memory would consist of the high-speed memory chips used in its cache. (This ideal world actually exists, and it is called a government-funded supercomputer.) We can, however, also pretend to live in the ideal world, by being frugal in the amount of memory our application requires. If that amount is small enough to fit into the level 2 (or, even better, the level 1) cache, then we will notice an (often dramatic) speed increase. Here is an actual code comment detailing this fact:

```
// Be aware that time will be affected by the buffer fitting/not
// fitting in the cache (ie, if default_total*sizeof(T) bytes
// fit in the cache).
```

Cases where the effort of fitting an application into a cache can be a worthwhile exercise typically involve tight, performance-critical, code. For example, a JVM implementation that could fit in its entirety into a processor's level 1 instruction cache would enjoy substantial performance benefits over one that couldn't.

There are, however, many cases where our program's data or instructions could never fit the processor's cache. In such cases, improving a program's locality of reference can result in speed increases, as data elements are more likely to be found in a cache. Improved locality of reference can occur both at the microscopic level (e.g., two structure elements being only 8 bytes apart) and at the macroscopic level (e.g., the entire working set for a calculation fitting in a 256KB level 1 cache). Both can increase a program's speed, but for different reasons.

Related data elements that are very close together in memory have an increased chance of appearing together in a cache block, one of them causing the other to be *prefetched*. Earlier on, we mentioned that caches organize their elements in blocks associated with consecutive memory addresses. This organization can result in increased memory access efficiency, as the second related element is fetched from the slow main memory as a side effect of filling the corresponding cache block. For this reason some style guides (such as the following excerpt from the FreeBSD documentation) recommend placing structure members together ordered by use.

```
 * When declaring variables in structures, declare them sorted
 * by use, then by size, and then by alphabetical order. The
 * first category normally doesn't apply, but there are
 * exceptions. Each one gets its own line.
```

(The exceptions referred to above are probably performance-critical sections of code, sensitive to the phenomenon we described.)

In other cases, a calculation may use a small percentage of a program's data. When that working set is concentrated in a way that allows it all to fit into a cache at the same time, the calculations will all run at the speed of the cache and not at that of the much slower main memory. Here is a comment from the NetBSD TCP processing code describing the rationale behind a design to improve the data's locality of reference:

```
 * (2) Allocate syn_cache structures in pages (or some other
 * large chunk).  This would probably be desirable for
 * maintaining locality of reference anyway.
```

Locality of reference can also be important for code; here is another related comment from the X Window System VGA server code:

```
 * Reordered code for register starved CPU's (Intel x86) plus
 * it achieves better locality of code for other processors.
```

Disk Cache and Banked Memory

Moving down our memory hierarchy, before reaching the disk-based file storage we encounter two strange beasts: the disk cache and banked memory. The disk cache is a classic case of space over time optimization, and the banked memory is . . . embarrassing. Accessing data stored in either of the two involves approximately the same processing overhead, and for this reason they appear together in our table. Nevertheless, their purpose and operation are completely different, so we'll examine each one in turn.

The *disk cache* is an area of the main memory reserved for storing temporary copies of disk contents. Accessing data on disk-based storage is at least an order of magnitude slower than accessing main memory. Note that this figure represents a best (and relatively rare) case: sustained serial I/O to or from a disk device. Any random-access operation involving a head seek and a disk rotation is a lot slower; a six-orders-of-magnitude difference between disk and memory access time (12ms over 2ns) should not surprise you. To overcome this burden, an operating system aggressively keeps copies of the disk contents in an area of the main memory it reserves for this purpose. Any subsequent read or write operations involving the same contents (remember the locality-of-reference principle) can then be satisfied by reading or writing the corresponding memory blocks. Of course, the main memory differs from the disk in that its contents get lost when power is lost; therefore, periodically (e.g., every 30 seconds on some UNIX systems) the cache contents are written to disk.

Furthermore, for some types of data (such as elements of a database transaction log, or a file system's directory contents—the so-called directory metadata) the 30-second flush interval can be unacceptably high; such data is often scheduled to be written to disk in a *synchronous* manner or through a time-ordered *journal*. Keep in mind here that some file systems, either by default (the Linux *ext2fs*) or through an option (the FreeBSD FFS with soft updates enabled), will write directory metadata to disk in an asynchronous manner. This affects what will happen when the system powers down in an anomalous fashion, due to a power failure or a crash. In some implementations, after a reboot the file system's state may not be consistent with the order of the operations that were performed on it before the crash.

Nevertheless, the performance impact of the disk cache is big enough to make a difference between a usable system and one that almost grinds to a halt. For this reason, many modern operating systems will use all their free memory as a disk cache.

As we mentioned, banked memory is an embarrassment; we would not be discussing it at all but for the fact that the same embarrassment keeps recurring (in different forms) every couple of years. Recall that with a variable N bits wide we can address 2^N different elements. Consider the task of estimating the number of elements we might need to address (the size of our address space) over the lifetime of our processor's architecture. If we allocate more bits to a variable (say, a machine's address register) than those we would need to address our data, we end up wasting valuable resources. On the other hand, if we underestimate the number of elements we might need to address, we will find ourselves in a tight corner.

Intel architecture	*Address bits*	*Addressing limit*	*Stopgap measure*
8080	16	64KB	IA-16 segment registers
IA-16	20	1MB	XMS (Extended Memory Specification); LIM EMS (Lotus/Intel/Microsoft Expanded Memory Specification)
IA-32	32	4GB	PAE (Physical Address Extensions); AWE (Address Windowing Extensions)

TABLE 1. SUCCESSIVE ADDRESS SPACE LIMITATIONS AND THEIR INTERIM SOLUTIONS

In Table 1 you can see three generations of address space limitations encountered within the domain of Intel architectures, and a description of the corresponding solutions. Note that the table refers only to an architecture's address space; we could draw similar tables for other variables, such as those used for addressing physical bytes, bytes in a file, bytes on a disk, and machines on the Internet. The technologies associated with the table's first two rows are fortunately no longer relevant. One would think that we would have known by now to avoid repeating those mistakes, but this is, sadly, untrue.

As of this writing, some programs and applications are facing the 4GB limit of the 32-bit address space. There are systems, such as database servers and busy Web application servers, that can benefit from having at their disposal more than 4GB of physical memory. New members of the IA-32 architecture have hardware that can address more than 4GB of physical memory. This feature comes under the name Physical Address Extensions (PAE). Nowadays we don't need segment registers or BIOS calls to extend the accessible memory range, because the processor's paging hardware already contains a physical-to-virtual address translation feature. All that is needed is for the address translation tables to be extended to address more than 4GB. Nevertheless, this processor feature still does not mean that an application can transparently access more than 4GB of memory. At best, the operating system can allocate *different* applications in a *physical* memory area larger than 4GB by appropriately manipulating their corresponding virtual memory translation tables. Also, the operating system can provide an API so that an application can request different parts of the physical memory to be mapped into its virtual memory space—again, a stopgap measure, which involves the overhead of operating system calls. An example of such an API is the Address Windowing Extensions (AWE) available on the Microsoft Windows system.

Swap Area and File-Based Disk Storage

The next level down in our memory storage hierarchy moves us away from the relatively fast main memory into the domain governed by the (in comparison) abysmally slow and clunky mechanical elements of electromagnetic storage devices (hard disks). The first element we encounter here is the operating system's *swap area* containing the memory pages it has temporarily stored on the disk, in order to free the main memory for more pressing needs. Also here might be pages of code that have not yet been executed and will be paged in on demand. At the same level in terms of performance, but more complicated to access in terms of the API, is the file-based disk storage. Both areas have typically orders-of-magnitude larger capacity than the system's main memory. Keep in mind, however, that on many operating systems the amount of available swap space or the amount of heap space a process can allocate is fixed by the system administrator and cannot grow above the specified limit without manual administrative intervention. On many UNIX systems the available swap space is determined by the size of the device or file specified in the swapon call and the corresponding command; on Windows systems, the administrator can place a hard limit on the maximum size of the paging file. It is therefore unwise not to check the return value of a malloc memory allocation call against the possibility of memory exhaustion. The code in the following code excerpt could well crash when run on a system low on memory:

```
TMPOUTNAME = (char *) malloc (tmpname_len);
strcpy (TMPOUTNAME, tmpdir);
```

The importance of the file-based disk storage in relationship to a program's space performance is that disk space tends to be a lot larger than a system's main memory. Therefore, *uncaching* (Bentley's term) is a strategy that can save main memory by storing data into secondary storage. If the data is persistent and rarely used, or does not exhibit a significant locality of reference in the program's operation, then the program's speed may not be affected; in some cases by removing the caching overhead it may even be improved. In other cases, when main memory gets tight, this approach may be the only affordable one. As an example, the UNIX sort implementations will only sort a certain amount of data in-core. When the file to be sorted exceeds that amount, the program will work by splitting its work into parts sized according to the maximum amount it can sort. It will sort each part in memory and write the result to a temporary disk file. Finally, it will *merge sort* the temporary files, producing the end result. As another example, the *nvi* editor will use a backing file to store the data corresponding to the edited file. This makes it possible to edit arbitrarily large files, limited only by the size of the available temporary disk space.

The Lineup

Component	*Nominal size*	*Worst case latency*	*Sustained throughput (MB/s)*	*$1 buys*	*Productivity (Bytes read / s / $)*	
					Worst case	*Best case*
L1 D cache	64KB	1.4ns	19022	10.7KB	$7.91 \cdot 10^{12}$	$2.19 \cdot 10^{14}$
L2 cache	512KB	9.7ns	5519	12.8KB	$1.35 \cdot 10^{12}$	$7.61 \cdot 10^{13}$
DDR RAM	256MB	28.5ns	2541	9.48MB	$3.48 \cdot 10^{14}$	$2.65 \cdot 10^{16}$
Hard drive	250GB	25.6ms	67	2.91GB	$1.22 \cdot 10^{11}$	$2.17 \cdot 10^{17}$

TABLE 2. PERFORMANCE AND COST OF VARIOUS MEMORY TYPES

(Author pauses to don his flame retardant suit.) To give you a feeling of how different memory types compare in practice, I've calculated some numbers for a fairly typical configuration, based on some currently best-selling middle-range components: an AMD Athlon XP 3000+ processor, a 256MB PC2700 DDR memory module, and a 250GB 7200 RPM Maxtor hard drive. The results appear in Table 2. I obtained the component prices from TigerDirect.com on January 19, 2006. I calculated the cost of the cache memory by multiplying the processor's price by the die area occupied by the corresponding cache divided by the total size of the processor die (I measured the sizes on a die photograph). The worst-case latency column lists the time it would take to fetch a byte under the worst possible scenario: for example, a single byte from the same bank and following a write for the DDR RAM, with a maximum seek, rotational latency, and controller overhead for the hard drive. On the other hand, the sustained throughout column lists numbers where the devices operate close to ideal conditions for pumping out bytes as fast as possible: eight bytes delivered at double the bus speed for the DDR RAM; the maximum sustained outer diameter data rate for the hard drive. In all cases, the ratio between bandwidth implied by the worst-case latency and the sustained bandwidth is at least one order of magnitude, and it is this difference that allows our machines to deliver the performance we expect. In particular, the ratio is 27 for the level 1 cache, 56 for the level 2 cache, 76 for the DDR RAM, and 1.8 million for the hard drive. Note that as we move away from the processor there are more tricks we can play to increase the bandwidth, and we can get away with more factors that increase the latency.

The byte cost for each different kind of memory varies by three orders of magnitude: with one dollar we can buy KBs of cache memory, MBs of DDR RAM, and GBs of disk space. However, as one would expect, cheaper memory has a higher latency and a lower throughput. Things get more interesting when we examine the productivity of various memory types. Productivity is typically measured as output per unit of input; in our case, I calculated it as read operations per second and $ cost for one byte. As you can see, if we look at the best-case scenarios (the device operating at its maximum bandwidth), the hard drive's bytes are the most productive. In the worst case (latency-based) scenarios the productivity performance of the disk is abysmal, and this is why disks are nowadays furnished with abundant amounts of cache memory (8MB in our case). The most productive device in the worst-case latency-based measurements is the DDR RAM. These results are what we would expect from an engineering point of view: the hard disk, which is a workhorse used for storing large amounts of data with the minimum cost, should offer the best overall productivity under ideal (best-case) conditions, while the DDR RAM, which is used for satisfying a system's general-purpose storage requirements, should offer the best overall productivity even under worst-case conditions. Also note the low productivity of the level 1 and level 2 caches. This factor easily explains why processor caches are relatively small: they work admirably, but they are expensive for the work they do.

What can we, as programmers and system administrators, learn from these numbers? Modeling the memory performance of modern systems is anything but trivial. As a programmer, try to keep the amount of memory you use low and increase the locality of reference so as to take advantage of the available caches and bandwidth-enhancing mechanisms. As a system administrator, try to understand your users' memory requirements in terms of the hierarchy we saw before making purchasing decisions; depending on workload, you may want to trade processor speed for memory capacity or bandwidth, or the opposite. Finally, always measure carefully before you think about optimizing. And next time you send a program whizzing through your computer's memory devices, spare a second to marvel at the sophisticated technical and economic ecosystem these devices form.

5th System Administration and Network Engineering Conference

SANE 2006

15–19 May 2006

Aula Congresscentre, Delft, The Netherlands

The 5th System Administration and Network Engineering Conference will offer three days of training followed by a two-day conference program, filled with the latest developments in system administration, network engineering, security, open source software, and practical approaches to your problems and puzzles. You will also have the opportunity to meet other system administrators and network professionals and chat with peers who share your concerns and interests.

www.sane.nl/sane2006

A conference organized by Stichting SANE,
co-sponsored by Stichting NLnet, USENIX, SURFnet, and NLUUG

CHAOS GOLUBITSKY

simple software flow analysis using GNU cflow

Chaos Golubitsky is a software security analyst. She has a BA from Swarthmore College, a background in UNIX system administration, and an MS in information security.

chaos@glassonion.org

A CALL GRAPH IS A TEXT-BASED OR graphical diagram showing which functions inside a code base invoke which other functions. Accurate call graphs aid many debugging and software analysis tasks. For example, when viewing a code base for the first time, an examiner can tell from a call graph whether the code structure is flat or modular, and which functions are the busiest. Later in analysis, a call graph can be used to answer specific questions, such as which other functions within the code invoke a specific function of interest.

GNU cflow is a new tool which can be used to quickly and easily generate flexible and accurate text-based call graphs of C programs. In this article I will introduce cflow, with an eye towards describing how it can be used to easily create accurate call graphs.

History and Motivation

The cflow tool was initially developed in the 1990s, and the older version is referred to as POSIX cflow. I first encountered POSIX cflow while performing a vulnerability analysis of open source software [1], for which I needed a simple source of data about reachable functions within a code base. The POSIX specification for the cflow tool [2] requires that the tool be capable of generating forward and reverse flow graphs up to a specified depth, and that the user be able to specify classes of symbols, such as static functions or typedefs, which should be printed or omitted. The POSIX tool provides this relatively limited functionality, and is no longer being actively maintained.

The cflow project was restarted last year due to interest in a simple tool which could generate call graphs, and the first alpha release of GNU cflow [3] occurred in April 2005. The GNU version of the tool is significantly more flexible than the POSIX specification requires, and is being actively maintained and improved.

Basic Functionality

In its simplest use, cflow is called with the name of one or more C source files as arguments. Cflow uses a custom C lexical analyzer to interpret the source code, and prints a call graph of the code, starting with the main() function.

Cflow's basic functionality can be demonstrated using a classic example:

```
#include <stdio.h>

void howdy();

int main() {
  howdy();
  exit(0);
}
void howdy() {
  printf("hi, world!\n");
}
```

If this example is stored as hello.c, then running cflow hello.c will produce:

```
main() <int main () at hello.c:5>:
    howdy() <void howdy () at hello.c:9>:
        printf()
    exit()
```

GNU cflow's main strength is that it can easily be configured to present call data in useful ways. Cflow's behavior can be modified using two major approaches. First, cflow can be invoked with options which present the call graph data in various ways. These options can be used to quickly find whatever data is needed to answer a particular question, or to format the call data for processing via script or some other external program. Second, cflow can be called with options which modify how it processes the source code and, therefore, what information will be contained in its results. I will discuss each of these in turn.

Customizing Cflow's Output Format

Cflow has two major output modes. In tree mode, which is the default, cflow prints functions one per line, using indentation to indicate call relationships. In cross-reference mode, cflow prints a two-column list containing one line for each caller/callee pair within the code base.

CROSS-REFERENCE MODE

Cross-reference mode, which is invoked using the -x flag, is the simpler of the modes, and is not very customizable. In addition to cross-references, it includes a special line for the beginning of each function definition in a file. Therefore, this mode can also be used to quickly obtain a list of the locations of all function definitions within a given file:

```
cflow -x filename.c | awk '$2=="*" {print $1 "\t\t" $3}'
```

TREE MODE

Tree mode is the default and is much more flexible. When invoked without arguments, cflow looks for a function called main(), and produces an indented call graph of that function and all functions it calls. The -m flag tells cflow to begin the tree using a different function. If the specified function is not found, cflow will print a tree for every function in the examined file or files. Reverse mode (-r) prints a reverse call tree, and always prints information about every function in the file.

Several flags are available which affect how much information, beyond function names, is included in cflow's output. These include -n (number each line of output), -l (label each line of output with the call depth of the function listed on that line), --omit-arguments and --omit-symbol-names (shorten the information printed about each function declaration). The --level-indent flag can be used to gain fine-grained control over the spacing and layout of the functions, but -T provides a good set of defaults which give reasonable visual call tree output. Further, the argument --format=posix can be used to obtain output similar (though not identical) to that produced by the older POSIX cflow program.

In tree mode (either standard or reverse), the -d *N* argument tells cflow to report only *N* levels of output. This option can be used to quickly print a list of all functions which are called by any function within a file of interest. (Note that this is most easily done in reverse tree mode, since forward tree mode examines only the main() function by default):

```
cflow -r -d 1 filename.c
```

I typically format cflow output for automated processing by custom scripts. However, cflow output can also be used as input for other graphing or processing software. A couple of examples are worth mentioning here. Cflow can be used in combination with the tool cflow2vcg to produce visual call graphs under the VCG graphing package [4]. Additionally, Emacs users may be interested in the emacs cflow-mode module which is packaged with cflow [5].

Customizing Cflow's Source Code Analysis

Cflow implements its own lexical analyzer for the C language, and there are several ways to control its behavior. In this section I will discuss some options which affect how cflow finds functions and definitions within C source code.

At the simplest level, the -i flag can be used to define subsets of symbols which should or should not be reported, including static symbols, typedefs, symbols whose names begin with underscores, and external symbols.

PREPROCESSOR OPTIONS

GNU cflow does not use a preprocessor by default. When invoked with the argument --cpp, cflow preprocesses the code using the cpp executable or a user-specified preprocessor. Using --cpp increases the accuracy of cflow's output, but has some visible effects. Most notably, functions which are implemented as #define statements are silently unrolled. This can occasionally cause confusing output: for instance, getc() is often implemented by operating systems as a wrapper for another function. It may be confusing to find __srget() in cflow's output with no indication of what invoked it. The older POSIX cflow always used a preprocessor, and preprocessor mode is likely to be desirable for most analysis, but it can sometimes be helpful to produce GNU cflow output without a preprocessor.

When invoked with --cpp, GNU cflow searches for function definitions in system header files. It is possible to tweak the set of directories which cflow should search for function definitions using the -I (include dir) and -U (undefine) flags. (These flags imply --cpp.) These flags are needed if we wish to use cflow to parse complex source code accurately.

EMBEDDING CFLOW IN MAKEFILES

For very small code bases, or to answer simple or file-specific questions, it can be sufficient to manually run cflow on a small number of C source files. However, in order for cflow to provide accurate results for complex code bases, it must process the code the same way the makefile processes it, to ensure that the function relations cflow finds are the same as those compiled into the software. Some more complex source analysis tools (e.g., the OCaml-based C representation language Cil [6]) compile the code as a side effect of analyzing it, and can therefore be trivially embedded in makefiles as compiler replacements. Since cflow does not do this, it is necessary to manually insert cflow-specific rules into the makefile. Makefile editing requires some effort, but it is often worthwhile due to the increased accuracy.

The general idea is to create a separate make target named, for instance, program.cflow, and configure this target to run cflow using:

- The compiler definitions used for this code base
- The include directives used for this code base
- The preprocessor flags used for this code base
- The file names compiled by this code base

It should be possible to use makefile variables to obtain the correct values for each of these items. In addition, the cflow -o flag is used to save the output to a file, and any desired cflow-specific flags are also set. Here is an example of this configuration which is appropriate for inclusion in a GNU-style Makefile.in file [7]:

```
program_CFLOW_INPUT = $(program_OBJECTS:.@OBJEXT@=.c)
CFLOW_FLAGS = -i^s --brief

program.cflow: $(program_CFLOW_INPUT) Makefile
  cflow -o$@ $(CFLOW_FLAGS) $(DEFS) $(DEFAULT_INCLUDES) \
           $(INCLUDES) $(AM_CPPFLAGS) $(CPPFLAGS) \
           $(program_CFLOW_INPUT)
```

With this configuration, the invocation make program.cflow should suffice to run cflow on the code base as it will be compiled. The CFLOW_FLAGS variable can be changed in order to run cflow with a different set of options.

Summary

In software analysis, it is often useful to be able to identify caller/callee relationships within a code base, and to display such relationships in usable formats. GNU cflow is a simple tool which performs this function accurately. Cflow builds on the decade-old tool of the same name by providing flexible options which significantly increase cflow's utility and ease of use. GNU cflow is recommended as a first-line tool for answering questions about software call flow.

REFERENCES

[1] http://www.usenix.org/events/lisa05/tech/golubitsky.html

[2] http://www.opengroup.org/onlinepubs/009695399/utilities/cflow.html

[3] http://www.gnu.org/software/cflow/

[4] http://www.gnu.org/software/cflow/manual/html_node /Output-Formats.html

[5] http://www.gnu.org/software/cflow/manual/html_node/Emacs.html

[6] http://manju.cs.berkeley.edu/cil/

[7] http://www.gnu.org/software/cflow/manual/html_node/Makefiles.html

LUKE KANIES

why you should use Ruby

Luke Kanies runs Reductive Labs (http://reductivelabs.com), a startup producing OSS software for centralized, automated server administration. He has been a UNIX sysadmin for nine years and has published multiple articles on UNIX tools and best practices.

luke@madstop.com

ABOUT TWO YEARS AGO, I SWITCHED from doing all of my development in Perl [1] to using Ruby [2] for everything. I was an independent consultant at the time, writing a prototype for a new tool (now available and called Puppet [3]); I had been writing tons of Perl since about 1998, and until I tried to write this prototype I really thought Perl was the perfect solution to my programming needs—I could do OO, I could do quick one-offs, and it was available everywhere. It seemed I could also think pretty well in Perl, so it didn't take long to translate what I wanted into functional code.

That all changed when I tried to write this prototype. I tried to do it in Perl, but I just couldn't turn my idea into code; I don't know why, but I couldn't make the translation. I tried it in Python, because I'd heard lots about how great Python was, but for some reason Python always makes my eyes bleed. So, in desperation, I took a stab at Ruby. Prior to this work, I had never seen a line of Ruby and had never heard anything concrete about it or why one would use it, but in four short hours I had a functional prototype; I felt as though a veil had been drawn from my eyes, that I had previously been working much harder than necessary.

I couldn't have told you then what it was about Ruby exactly, but there was something that clearly just seemed to make it easier to write code, even code that did complicated things. Since then, I've gotten much better at Ruby and a good bit more cognizant of what sets it apart.

You could argue that mine was a personal experience and that most people would not benefit as much from a switch to Ruby, and in some ways you would be right—I wrote a lot of OO code in Perl, which isn't exactly pleasant, and I generally hate it when I have to do work that the computer can do (you almost never have to actually use a semicolon to end a line in Ruby). But the goal of this article is to convince you that just about anyone would find benefit from a switch to Ruby, especially if you use or write many libraries or if you use your scripting language as your primary interface to your network.

This article is definitely not meant to teach you how to write Ruby; the Pragmatic Programmers [4] have written a great book [5] on Ruby, and I highly recommend it. I also recommend reading Paul Graham's essays on programming language power [6] and beating the averages [7]; they do a good job of discussing what to think about in language choice.

All of the following examples were written as simple Ruby scripts in a separate file (I keep a "test.rb" file or equivalent for every interpreted language I write in, and that's what I used for all of these examples), and the output of each is presented prefixed with =>. To run the examples yourself, just put the code into something like example.rb and run ruby example.rb. You could also use the separate irb executable to run these examples, but the output will be slightly different.

What's So Special About Ruby?

I can't point to one hard thing that makes Ruby great—if I tried, I would just end up saying something silly like, "It just works the way I expect," and that wouldn't be very useful. Instead, I'll run through some of the things that I love and that really change how I use it.

EVERYTHING IS AN OBJECT

Yes, Marjorie, everything. No, there are no exceptions (heh, rather, even the Exceptions are objects); there aren't special cases. Classes are objects, strings are objects, numbers are objects:

```
[Class, "a string", 15, File.open("/etc/passwd")].each { |obj|
        puts "'%s' is of type %s" % [obj, obj.class]
}
=> 'Class' is of type Class
=> 'a string' is of type String
=> '15' is of type Fixnum
=> '#<File:0x210310>' is of type File
```

Here we have a list of objects and we print a string describing each object in turn (puts just prints the string with a carriage return at the end). When you create a new class, it's an instance of the Class object. This each syntax is how pretty much all iteration is done in Ruby—you can use for loops, but $10 says you won't once you get used to each.

THERE ARE NO OPERATORS

Did you notice that % method in the example above? Yeah, that's a method, not an operator. What's the difference? Well, the parser defines an operator, but the object's class defines a method. In Perl, you have one operator for adding strings, . (a period), and one operator for adding numbers, +, because those operators are part of the language and the parser can't easily type-check the arguments to verify that you passed the right arguments all around. But in Ruby, each class just implements a + method that behaves correctly, including any type-checking.

It gets better. The indexing syntax for hashes and arrays is also a method, and you can define your own versions:

```
class Yayness
        def initialize(hash)
                @params = hash

        end
        def [](name)
                @params[name]
        end

        def []=(name, value)
                @params[name] = value
        end

end

y = Yayness.new(:param => "value", :foo => "bar")
puts y[:foo]
y[:funtest] = "a string"
puts y[:funtest]

=> bar

=> a string
```

That funny term with the colons is called a "Symbol," and it's basically a simple constant, like an immutable string. It's very useful for those cases where you would normally use an unchanging string, such as for hash keys, but you're too lazy to actually type two quotation marks. Actually, one of the reasons I like them so much is that Vim colorizes them quite differently from strings, making them easier to read.

Why would you use these indexing methods? A large number of my classes have a collection of parameters, and this makes it trivial to provide direct access to those parameters. Sometimes it makes sense to subclass the Hash class, but there are plenty of other times where you want to wrap one or more hashes, and these methods make that very easy. I also often define these methods on the classes themselves, so that I can retrieve instances by name:

```
class Yayness
        attr_accessor :name
        @instances = {}
        def Yayness.[](instname)
                @instances[instname]

        end

        def Yayness.[]=(instname, object)
                @instances[instname] = object

        end

        def initialize(myname)
                self.class[myname] = self
                @name = myname

        end

end

first = Yayness.new(:first)
second = Yayness.new(:second)

puts Yayness[:first].name
=> first
```

Here I've used attr_accessor (which is a method on the Module class, and thus available in all class definitions) to define getter/setter methods for name, and then some methods for storing and retrieving instances by name. I also use this frequently, possibly even too frequently. In Ruby, instance variables are denoted by prefixing them with an @ sigil; so, in this case, calling the name method on a Yayness object will return the value of @name.

The initialize method, by the way, is kind of like a constructor in Ruby—I say "kind of" because it's actually called after the object exists and is only expected to (of course) initialize the object, not create it.

INTROSPECTION

Ruby is insanely introspective. We've already seen how you can ask any object what type of object it is (using the class method), and there are a bunch of complementary methods for asking things like whether an object is an instance of a given class, but you can also ask what methods are available for an object, or even how many arguments a given method expects:

```
class Funtest
        def foo(one, two)
                puts "Got %s and %s" % [one, two]
        end
end

class Yaytest
        def foo(one)
                puts "Only got %s" % one
        end
end

[Funtest.new, Yaytest.new, "a string"].each { |obj|
        if obj.respond_to? :foo
                if obj.method(:foo).arity == 1
                        obj.foo("one argument")
                else
                        obj.foo("first argument", "second argument")
                end
        else
                puts "'%s' does not respond to :foo" % obj
        end
}

=> Got first argument and second argument
=> Only got one argument
=> 'a string' does not respond to :foo
```

Here we create two classes, each with a foo method but each accepting a different number of arguments. We then iterate over a list containing an instance of each of those classes, plus a string (which does not respond to the foo method); if the object responds to the method we're looking for, we retrieve the method (yes, we get an actual Method object) and ask that method how many arguments it expects (called its arity).

You can see here that I'm using a Symbol for the method name during the introspection; this is common practice in the Ruby world, even though I could have used a string.

ITERATION

You've already seen the each method on arrays (it also works on hashes), and you probably thought, "Oh, well, my language has that and it's called 'map,' " or something similar. Well, Ruby goes a bit further. Ruby attempts to duck the multiple inheritance problem by supporting only single inheritance but allowing you to mix in Modules. I'll leave it to the documentation to cover all of the details, but Ruby ships with a few modules that are especially useful, and the Enumerable module is at the top of the list:

```
class Funtest
        include Enumerable
        def each
                @params.each { |key, value|
                        yield key, value
                }
        end
        def initialize(hash)
                @params = hash
        end
end
f = Funtest.new(:first => "foo", :second => 59, :third => :symbol)
f.each { |key, value|
        puts "Value %s is %s of type %s" % [key, value, value.class]
}
puts f.find { |key, value| value.is_a?(Symbol) }.join(" => ")
puts f.collect { |key, value| value.to_s }.join("---")
```

```
=> Value second is 59 of type Fixnum
=> Value first is foo of type String
=> Value third is symbol of type Symbol
=> third => symbol
=> 59---foo---symbol
```

Here we define a simple class that accepts a hash as an argument and then an each method that yields each key/value pair in turn (you'll have to hit the docs for more info on how yield works—it took me a while to understand it, but it was worth it). By itself our class isn't so useful, but when we include the Enumerable module, we get a bunch of other methods for free. I've shown two useful methods: find (which finds the first key/value pair for which the test is true, and returns the pair as an array) and collect (which collects the output of the iterative code and returns it as a new array).

You can see that our each code did exactly as we expected, but we also easily found the first Symbol in the list (find_all will return an array containing all matching elements). In addition, we used collect to create an array of strings (just about every object in Ruby accepts the to_s method to convert it to a string, although you generally have to define the method yourself on your own classes for it to be meaningful).

The great thing here is that we just defined one simple method and got a bunch of other powerful iterative methods. Another useful module is Comparable; if you define the comparison method <=> and include this module, then you get a bunch of other comparison methods for free (e.g., >, <=, and ==).

BLOCKS

On the one hand, I feel as though I should talk about blocks, because they really are one of the most powerful parts of Ruby; on the other hand, I know that I am not up to adequately explaining in such a short article how they work and why you'd use them. I'm going to take a shot at such an explanation, but I fear that I'll only confuse you; please blame any confusion on me, and not on Ruby. As you use Ruby, you'll naturally invest more in using and understanding blocks, but you can survive in Ruby just fine without worrying about them. It's worth noting, though, that the iteration examples above all use blocks—each, find, etc., are all called with blocks.

Blocks just keep looking more powerful the more I use them. They're relatively simple in concept, and many languages have something somewhat similar—they're just anonymous subroutines, really—but the way Ruby uses them goes far beyond what I've seen in most places. As a simple example, many objects accept blocks as an argument and will behave differently—files will automatically close at the end of a block, for instance—if a block is provided:

```
File.open("/etc/passwd") { |f|
        puts f.read
}
```

Once you get used to blocks automatically cleaning up after you, it becomes quite addictive. I often find myself creating simple methods that do some setup, execute the block, and then clean up. For instance, here's a simple method for executing code as a different user (the method is significantly simplified from what I actually use):

```
def asuser(name)
        require 'etc'
        uid = Etc.getpwnam(nam).uid
        Process.euid = uid
        yield
        Process.euid = Process.uid
end

asuser("luke") {
        File.unlink("/home/luke/.rhosts")
}
```

We convert the name to a number using the Etc module (which is basically just an interface to POSIX methods), and then we set the effective user ID to the specified user's. We use yield to give control back to the calling code (which just executes the associated block), and then reset the EUID to the normal UID.

This is a very simple example; there are a huge number of methods in Ruby that accept blocks, and there are many ways of using them to make your life easier. I recently refactored the whole structure of Puppet around using blocks where I hadn't previously, and the result was a huge increase in clarity and, thus, productivity. Here's a hint: if you find yourself dynamically creating modules or classes, note that you can use a block when doing so:

```
myclass = Class.new {
        def foo
                puts "called foo"
        end
}
a = myclass.new
a.foo
=> called foo
```

Here I'm defining a class at runtime, rather than at compile-time, and using a block to define a method on that class.

This specific example is no different from just using the class keyword to define the class, but at least for me it provided much more power and flexibility in how I created new classes. This ends up being critical in Puppet, which is composed almost entirely of classes containing classes; the relationships between those classes is one of the most complicated parts of the code—making that easier had a huge payoff.

Conclusion

I hope I've at least interested you in learning more about Ruby. I know that I was short on my descriptions, but I've tried to focus more on why you'd use it than how. I highly recommend looking into some of the discussion around Ruby on Rails [8]; a lot of Java refugees have taken up Ruby as a means of getting more done with less effort, and their discussions on why are very informative.

Even if you don't use Ruby, though, learn more languages, assess them critically, and demand more from them. Your computer should be working for you, and the language you choose for interacting with your computer determines a lot about how you work.

REFERENCES

[1] http://Perl.org

[2] http://ruby-lang.org

[3] http://reductivelabs.com/projects/puppet

[4] http://www.pragmaticprogrammer.com/index.html

[5] http://www.pragmaticprogrammer.com/titles/ruby/index.html

[6] http://paulgraham.com/power.html

[7] http://paulgraham.com/avg.html

[8] http://rubyonrails.org

DAVID MALONE

unwanted HTTP: who has the time?

David is a system administrator at Trinity College, Dublin, a researcher in NUI Maynooth, and a committer on the FreeBSD project. He likes to express himself on technical matters, and so has a Ph.D. in mathematics and is the co-author of *IPv6 Network Administration* (O'Reilly, 2005).

dwmalone@maths.tcd.ie

THANKS
Ian Dowse and I conducted the original investigation. Colm MacCárthaigh provided us with the required Apache changes and Niall Murphy helped improve several drafts of this article.

IN OCTOBER 2002, ONE OF OUR users raised a req ticket with this message:

> When book number p859 is entered in the Web-based Library Catalogue search this error message comes up: Internal Server Error

At first glance, most experienced administrators would give a diagnosis of "broken CGI script"; however, as it would turn out, this was far from the case. Examination of the Web server's logs showed an unusually large number of HTTP requests for our home page, which was causing Apache to hit its per-user process limit. There was no obvious reason for a surge in interest in our Web pages, so we responded:

> It looks like our Web server may be under an attack of some sort! There are lots of people requesting our home page and the server is running out of processes for Webnobody! This isn't leaving enough space to complete the library lookup. I've no idea what is going on at the moment—between 18:00 and 18:59 we saw four times as many requests as we did this time yesterday.

The source of this traffic was far removed from the original error message and was eventually unearthed using a mix of investigation and guesswork. It's the investigation, ultimate causes, and our response to this traffic that I'm going to talk about in this article.

Analyzing the Problem

Our first thought was that our home page had been linked to from some popular Web site or referred to in some piece of spam. Our Apache server had originally been configured to log using the common log file format, so we naturally switched to the combined format, which also records the referring URL and the User-Agent as reported by the browser making the request [1].

To our amazement, the majority of the requests included neither of these pieces of information. However, this seemed to provide a way of easily identifying these unusual requests. Further study showed that the machines making these requests were connecting regularly, some as often as once a minute (to the nearest second), and they rarely requested any page bar our home page. Figure 1 shows the autocorrelation of the requests load, with strong peaks at multiples of 60s. For comparison, the autocorrelation without these unusual requests is also shown [2].

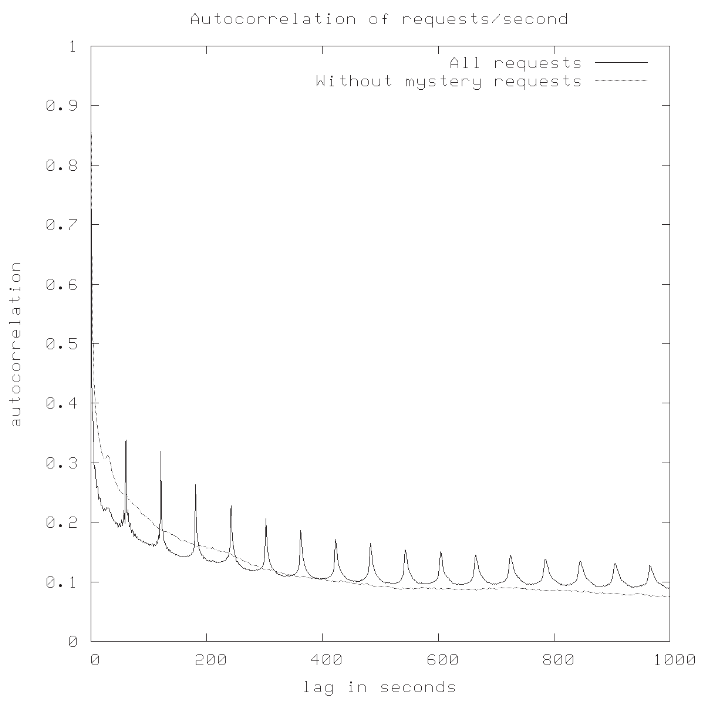


FIGURE 1. AUTOCORRELATION OF THE SERVER'S LOAD

Using *periodic requests for /* as a signature, we found that this had actually been going on for months. The number of hosts making these requests had been gradually increasing. This made it seem less likely that these requests were an orchestrated attack and more likely that it was some sort of quirk or misconfiguration.

Using tcpdump [3] we recorded a number of these requests. A normal TCP three-way handshake was followed by the short HTTP request shown below, contained in a single packet. Our server would reply with a normal HTTP response, headers, and then the content of the home page.

```
GET / HTTP/1.0
Pragma: no-cache
```

The request above is unusual. It does not include the Referer and User-Agent headers mentioned above. Also, it does not include the Host header, which is used to ensure the correct operation of HTTP virtual hosting. Even though the Host header is not part of HTTP 1.0, most HTTP 1.0 clients send a Host header. Virtual hosting is such a common technique that it seemed impossible that this could be any sort of normal Web client. In fact, the lack of a Host header indicated that the software making the request was probably not interested in the content returned.

Other than the content, the only other information returned by Apache was the HTTP headers of the response. These headers were the Date, the Server version, the Content-Type, and a header indicating that the connection would be closed after the page had been sent. After staring at all this for a bit, it occurred to us that something might be using our home page for setting the clocks on machines, as the Date header was the only part of the response that was changing.

We made connections back to several of the IP addresses in question and found that the connecting machines seemed to be Web proxies or gave signs of running Windows. We picked a random sample of 10 to 20 machines and made an effort to identify a contact email address. We sent short queries to these email addresses, but no responses were received.

A post [4] to the comp.protocols.time.ntp Usenet group was more productive. We asked for a list of software that might use the HTTP date header to set the time. This produced a list of possibilities, which we investigated.

Tardis [5] was identified as a likely source of the queries: it had an HTTP time synchronization mode and listed our Web server as a server for this mode. We contacted the support address for Tardis and asked why our server was listed, and why someone would implement an HTTP time synchronization mode when there were other, better protocols available.

Tardis support explained that they had a lot of requests to add a method of setting the time through a firewall, and thus Tardis added a feature using HTTP to do this. At the time they implemented this feature they scanned a list of public NTP servers [6] to find ones also running HTTP servers. The host running our Web server had been a public NTP server around 10 years previously, and due to a misunderstanding had not been removed from the list until mid-2000 [7].

The software only listed four servers for this mode of operation: a host in Texas A&M University, a host in Purdue University, Altavista's address within Digital, and our server. Tardis would initially choose a server at random and then stick with it until no response was forthcoming, when it would select a different server. We suspect that the spike in load that we saw corresponded to the HTTP server on one of these machines being unavailable, resulting in a redistribution of the clients of this machine between the remaining hosts.

Note that the default polling interval in Tardis was once every few hours. However, the software's graphical interface included a slider which allowed the poll interval to be reduced to once per minute.

Tackling the Problem

In the discussions that followed with Tardis support we agreed that future versions of Tardis would only list our official NTP server, and only for Tardis's NTP mode. We also suggested that allowing users to automatically set their clock once per minute was probably a bad idea and suggested modifications to the method used. In particular, using a HEAD rather than a GET request could significantly reduce the amount of data transferred, and setting the User-Agent field would make it easier for server administrators to identify such requests.

We did also suggest that our college be given a complimentary site license for Tardis in exchange for the not inconsiderable traffic shipped to Tardis users. For example, in the first 16 hours after enabling combined logging, we saw 400,000 connections from about 1800 different IP addresses. We estimated the resulting traffic at around 30GB/month.

However, it was some time before a new release of Tardis was planned, so we had to take some action to prevent future incidents of overload on our server. A first simple step was to increase process limits for the Web server, which had plenty of capacity to spare.

A second step was to use FreeBSD's accept filters [8]. Accept filters are a socket option that can be applied to a listening socket that delays the return of an accept system call until some condition is met. Usually the accept system call returns when the TCP three-way handshake is complete. We chose to apply a filter that delays the return of the accept system call until a full HTTP request is available. We knew that the request that Tardis was making arrived in a single packet one round-trip-time later. Thus the filter saves dedicating an Apache process to each request for the duration of the round trip (and avoids a small number of context switches).

While these measures helped prevent our server being overloaded, they did little to reduce the actual number of requests and volume of traffic being served to Tardis users. Using Apache's conditional rewriting rules, as shown below, we were able to match Tardis requests and, rather than returning the full home page (about 3KB), were able to return a much smaller page (about 300 bytes).

```
RewriteCond %{THE_REQUEST} ^GET\ /\ HTTP/1.[01]$
RewriteCond %{HTTP_USER_AGENT} ^$
RewriteCond %{HTTP_REFERER} ^$
RewriteRule ^/$ /Welcome.tardis.asis [L]
```

Using Apache's asis module [9], we were able to return custom headers, including a redirect to our real home page, in case some requests from genuine browsers were accidentally matched by the rewrite rules.

This significantly reduced the amount of data that we returned, but we also wanted to reduce the total number of clients that we were serving. We considered blacklisting the IP addresses of clients making these requests. However, we decided that this was not appropriate, for two reasons. First, a number of the client IPs were the addresses of large HTTP proxy servers and we did not want to exclude users of these proxies from accessing our Web pages. Second, the large number of IPs would make this a high-maintenance endeavor.

Instead, we decided to return a bogus HTTP date header in response to requests matching our Tardis rewrite rule, in the hope that this would encourage Tardis users to reconfigure their clients. By default Apache does not allow the overriding of the date header, but Colm MacCárthaigh of the Apache developer team provided us with a patch to do this. The page was altered to return a date of Fri, 31 Dec 1999 23:59:59 GMT. A link to another page explaining why we were returning an incorrect time was included in the body of this page.

We expected this to cause significant numbers of queries, and so prepared an FAQ entry for our request system to allow our administrators to respond quickly. However, we have only had to reply to a handful of email queries about this anomaly.

This countermeasure had a noticeable impact on the number of clients connecting to our server. Figure 2 shows the number of requests from Tardis users per hour, where we began returning a bogus time at hour number 1609. Alhough the number of requests is quite variable, our countermeasure quickly reduced the number by a factor of roughly five. Tardis support suggests that this is actually users reconfiguring Tardis, rather than some sanity check within Tardis itself. Note that the reduction achieved by this technique is actually more prominent than the impact of the new release of Tardis a year later.

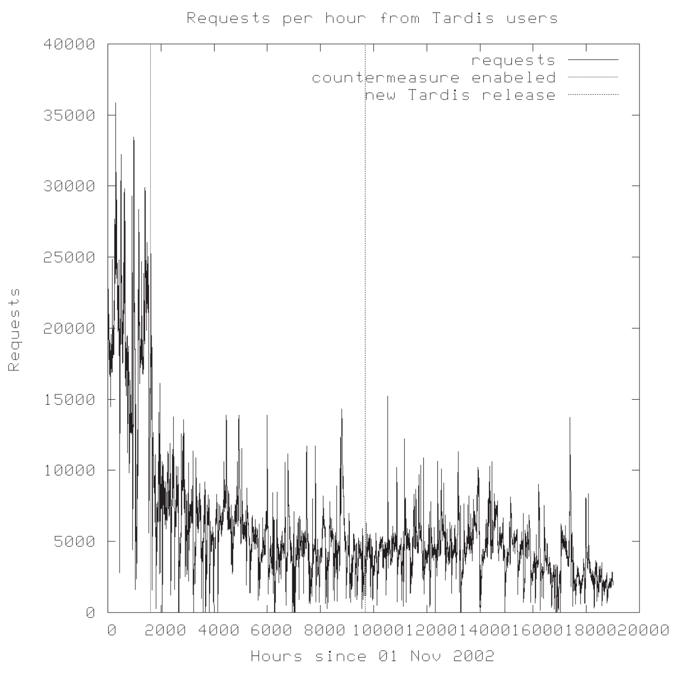


FIGURE 4. NUMBER OF REQUESTS FROM TARDIS USERS PER HOUR

Contemplations . . .

In dealing with this unwanted traffic, we were fortunate in several respects. Though there were only small hints as to the source of the traffic, they were sufficient to find the origin. The traffic also had a clearly identifiable signature, in that common headers were missing, and was not deliberately designed to be hard to identify. This is in stark contrast to spam, where successful attempts to identify features of spam quickly lead to a new generation of further obfuscated spam.

Though unwittingly inflicted upon us, this attack was quite like a DDoS. The number of hosts involved was in the thousands, making it infeasible to deal with each host by hand. Thankfully we were able to reduce the amount of traffic (both bytes and requests), since the hosts were requesting a valid service via a well-formed protocol dialog, allowing us to tailor the response appropriately.

. . . ON DIAGNOSIS

At the time, our efforts to diagnose the problem seemed haphazard. On reflection, the steps followed do seem to have been sensible and moderately generic:

- Identify some rogue requests.
- Try to spot a signature that matches these requests.
- Look at all requests matching the signature (and refine the signature if necessary).
- Examine the corpus of requests, looking for indications of their likely origin.

Of course, to identify a signature requires some knowledge of what constitutes a normal request. In our case, had we been using the combined log file format all along, we might have realized sooner that something unusual was going on. In this case, simply monitoring the number of requests to the server would probably not have identified the problem, as the load on a server can plausibly increase gradually without arousing attention. However, as we saw from Figure 1, higher-order statistics make the problem much more obvious.

. . . ON COUNTERMEASURES

Our initial countermeasure was to send a smaller response to requests matching the signature. This technique has been adopted as a response to being Slashdotted [10] by a number of organizations. For example, Mozilla's Bugzilla database now returns a short static page in response to links from Slashdot. Similarly, it is not uncommon to follow a link from Slashdot to find a page that says, "If you really want to download the 1.5MB PDF report, please go to this page." In the case of one of the other Web servers listed by Tardis, they had no content at /, and were able to create a short page to satisfy the requests.

In our case, we identified that the client making the requests was not a full Web browser. This is why we could use an automatic redirect to accommodate legitimate requests accidentally matched by the rewrite rules. Unfortunately, this option is not available to those who have been Slashdotted.

As a general technique to stop remote sites from linking into specific parts of a Web site, it is possible to generate URLs that have a limited lifetime. However, such systems typically frustrate bookmark systems and search engines alike. Similarly, some people use limited lifetime email addresses to avoid spam. A more extreme version of these techniques could use limited lifetime DNS entries. Options like this were not available to us, as the URL and DNS name in question were too well known.

A consideration that we considered to be important in designing any response to an HTTP problem was that legitimate users and problem users may both be behind the same Web proxy (or NAT device). A student at a local university working on a spidering project repeatedly crawled the Mathworld [11] site. As a response, the Mathworld operators blocked access from the IP they saw the requests coming from. This resulted in blocking the student's entire department!

The final part of our countermeasure was designed to attract the attention of users involved in the problem. Importantly, changing the date returned by our Web server is only likely to attract the attention of users who are using that date for something unusual. It might possibly have confused the caching scheme of some browsers, but we have heard no reports to this effect. Notifying users who are not involved in the problem, as often occurs when virus notifications are returned in response to forged email addresses, can be counterproductive.

. . . ON SIMILAR NTP-RELATED INCIDENTS

There are eerie similarities between this event and a number of other incidents. At about the same time as our incident, CSIRO had to take action because of hundreds of thousands of clients accessing their server from outside Australia. The subsequent incident at Wisconsin [12], where the address of an NTP server was hardwired into a mass-produced DSL router, is probably best known.

Fortunately, our problem was on a smaller scale. Unlike the Wisconsin incident, the extent of the problem had actually been augmented by users configuring the system to poll frequently, rather than simple bad system design (though providing a slider that can be set to poll once per minute probably counts as bad design). It is amusing to note that we actually had to patch the source of Apache to produce our deliberate misconfiguration. This must be a rare example of a Windows GUI providing you with easy-to-use rope to hang yourself, while the config file-based system at the other end requires more work to induce "errors"!

One of the solutions considered at Wisconsin was to abandon the IP address of the host in question; however, this was not the final solution used. There have been incidents of the abandonment of domains because of poor design choices in time synchronization software [13]. We did consider moving our Web server before we had put our countermeasures in place, but this would have placed a much larger burden on our system administration staff.

An interesting question is, why has an apparently innocent service, time synchronization, caused so many problems? A significant part of the problem seems to be misuse of lists of well-known servers [6]. Though the list includes a clear statement that it is "updated frequently and should not be cached," many people serve local copies. A quick search with Google identifies many pages listing our retired server as a current NTP server, even though it has not been on the official list since 2000. Some of these pages include the retired server in example source code.

This suggests that providing standardized dynamic methods for determining an appropriate NTP server might be worth the development effort. Attempts to provide pools of active NTP servers behind one DNS name have proved quite successful in recent years [14]. Multicast NTP would provide a more topologically aware technique for discovering NTP servers; however, the still-limited availability of multicast makes this less practical. Making a number of anycast servers (or, more exactly, shared unicast servers) might also be beneficial. This technique has already been used successfully for the DNS roots and 6to4 relay routers [15]. Anycast NTP has been successfully deployed in the Irish Research and Education Network, HEAnet.

What other protocols/servers may be subject to similar problems? There are obvious parallels with DNS root servers, which are also enumerated with a highly cached list. The high level of bogus queries and attacks arriving at the root servers has been well documented [16].

Conclusions

The investigation of this problem was an interesting exercise in the sorts of problems that sysadmins end up tackling. We had to use many of the standard tools: log files, diagnostic utilities, Usenet, reconfiguration, and a little software hacking.

Our countermeasures remain in place today and seem relatively successful, as the unwanted traffic remains significantly reduced. The incident itself seems to fit into a larger pattern of problems with time synchronization software and statically configured services.

REFERENCES

[1] The common log file format was designed to be a standard format for Web servers and was used by the CERN httpd: see

http://www.w3.org/Daemon/User/Config/Logging .html. More recently the combined format has become more common: http://httpd.apache.org /docs/logs.html.

[2] Autocorrelation is a measure of how much correlation you see when you compare a value now with a value in the future: see http://en.wikipedia .org/wiki/Autocorrelation.

[3] Tcpdump's -X option is good for this kind of thing: http://www.tcpdump.org/.

[4] The thread on comp.protocols.time.ntp can be found at http://groups.google.com/group/comp .protocols.time.ntp/browse_thread/thread /710cc3fb87bd08cc/026820ef0e6b4165.

[5] The home page for Tardis Time Synchronization Software is at http://www.kaska.demon.co.uk/.

[6] David Mills's list of Public NTP Secondary (stratum 2) Time Servers was traditionally at http://www.eecis.udel.edu/~mills/ntp/clock2.htm but now lives in the NTP Wiki at http://ntp.isc.org/ bin/view/Servers/WebHome.

[7] Archive.org is very useful for checking the history of Web pages: http://www.archive.org/.

[8] FreeBSD's accept filters were developed by David Filo and Alfred Perlstein. There are filters that wait until there is data or a complete HTTP request queued on a socket: http://www.freebsd .org/cgi/man.cgi?query=accept_filter.

[9] The Apache Module for sending a file as is, mod_asis, is documented at http://httpd.apache .org/docs-2.0/mod/mod_asis.html.

[10] Being Slashdotted is, of course, being linked to from Slashdot and suffering an unexpected increase in requests as a consequence. Wikipedia has a nice entry at http://en.wikipedia.org/wiki /Slashdotted.

[11] Eric Weisstein's Mathworld is an online encyclopedia of mathematics predating the flurry of Wiki activity: http://mathworld.wolfram.com/.

[12] Dave Plonka's well-known report on routers flooding the University of Wisconsin time server can be found at http://www.cs.wisc.edu/~plonka /netgear-sntp/.

[13] A description of why the UltiMeth.net domain was abandoned can be found at http://www.ultimeth.com/Abandon.html.

[14] The NTP Server Pool project lives at http://www.pool.ntp.org/.

[15] RFC 3258 describes "Distributing Authoritative Name Servers via Shared Unicast Addresses," which is basically making DNS queries to an anycast address. A similar trick for finding a 6to4 relay is described in RFC 3068. Both these techniques seem to work pretty well in practice.

[16] There are a number of studies of requests arriving at the DNS root servers. Check out "DNS Measurements at a Root Server," available on the CAIDA Web site at http://www.caida.org /outreach/papers/bydate/.

Thanks to USENIX & SAGE Supporting Members

Addison-Wesley Professional/
Prentice Hall Professional

Ajava Systems, Inc.

AMD

Asian Development Bank

Cambridge Computer Services, Inc.

EAGLE Software, Inc.

Electronic Frontier Foundation

Eli Research

FOTO SEARCH Stock Footage and
Stock Photography

GroundWork Open Source Solutions

Hewlett-Packard

IBM

Intel

Interhack

The Measurement Factory

Microsoft Research

MSB Associates

NetApp

Oracle

OSDL

Raytheon

Ripe NCC

Sendmail, Inc.

Splunk

Sun Microsystems, Inc.

Taos

Tellme Networks

UUNET Technologies, Inc.

It is with the generous financial support of our supporting members that USENIX is able to fulfill its mission to:

- Foster technical excellence and innovation
- Support and disseminate research with a practical bias
- Provide a neutral forum for discussion of technical issues
- Encourage computing outreach into the community at large

We encourage your organization to become a supporting member. Send email to Catherine Allman, Sales Director, sales@usenix.org, or phone her at 510-528-8649 extension 32. For more information about memberships, see http://www.usenix.org/membership/classes.html.

RANDOLPH LANGLEY

auditing superuser usage

Randolph Langley is a member of the Computer Science Department at Florida State University. Prior to this, he worked both in the financial industry and for the Supercomputer Computations Research Institute.

langley@cs.fsu.edu

IMAGINE BEING THE MANAGER OF A UNIX group who, after receiving a telephone call that a user cannot access his NFS home directory, happens to find the following lines in the shell history file for the root account:

```
ps -elf | grep -i portmap
kill -TERM 2193
portmap -dlv
```

It appears that somebody was trying to debug the portmapper, but when was this done? Who did it? Is it the cause of the current problem, or was it someone working on this problem?

While in this case it might be merely desirable to know more about these lines—after all, you can just do a ps to find out if the portmap program is running and start it if it is not—it is sometimes necessary to maintain records of who does what on some production systems. Programs such as sudo [1] and op [2] provide a means of controlling the who and, to a lesser degree, the what, but determining more exactly what was actually done can still be a challenge.

One method of meeting this challenge, sudoscript [3], was presented in Howard Owen's August 2002 *;login:* article "The Problem of PORCMOLSULB." It wraps the execution of a shell by sudo [1] with a script session.[1] While this is certainly a viable approach, modifying script seemed to me the more natural approach.[2] I wanted to add a remote logging capability since this allows one both to centralize logging and to provide some fraction more capability in the event of a break-in via sudo (although certainly a knowledgeable cracker should be able to stop this logging quite quickly). Modifying script seemed the most direct way to provide such logging.

1. I would like to properly credit the program script to someone, but my detective skills have not sufficed to find the original author.
2. Just to make the sequence of events clear, I had done the main modifications to script before I was aware of sudoscript.

Rationale

In large organizations, the responsibility for system security and its monitoring has natural divisions: system administrators, their managers, the computer security group, and technology auditing all have different roles in preserving and monitoring system security. Division of responsibility also helps maintain accountability in the overall system. To divide responsibility, information technology controls should exist at every level, eliminating any single point of trust.

Traditionally, however, with UNIX system administration there has been an imbalance in accountability for superuser activities by system administrators. While important advances such as SELinux [4] introduce new and useful capability in the form of mandatory access controls in imposing limits, in an audit or forensic situation, tracking superuser actions typically has meant following whatever logs were available from shells and from what can be inferred from reading various system logs. Shell logs typically are not configured to keep timestamps (though many shells, such as bash [5], do have that option). Shell logs keep a record, not of the actual keystrokes, but, rather,of the command line that was eventually entered; shell logs do not keep track of the output from commands; they don't have the ability to automatically forward information to other machines designed to maintain security information.

While a machine such as a honeypot may have a designed-in system for fine-grained tracking of user interaction at a very low level, such as honeynet's use of sebek [6]—typically as a hidden kernel module, since such logging should not be obvious to the intruder—such modifications are not desirable in a typical production system.

Although the program sudo is commonly used in order to improve accountability, it also provides other benefits, such as limiting the number of people who need direct access to a superuser password. In addition, it provides some measure of limiting use of privilege by providing a means of allowing certain programs to be executed by a given user.

From a management perspective, simply knowing *who* did *what* can be invaluable, such as when tracking down ad hoc changes that were made in the heat of problem resolution but were not put into the boot-time configuration. For a technology auditor, superior tracking of superuser privilege allows the auditors to have a more informed opinion of operations. For a security officer who may be looking through the logs for security lapses, having better and more accurate logs of actions by superusers may be desirable.

Changes to script.c

3. script.c can be found in the RedHat source RPM util-linux-2.12a-16.EL4.6 .src.rpm.

While quite a bit of this can be done by simply configuring a C or Perl wrapper around script[3] for a standard sudo setup, (such as Owen's Perl script sudoscript [3]), I think that setup is less than optimal. I thought it would be nice if session information could be stored on a common, hardened server; additionally, I thought it would be nice not to have a C or Perl wrapper around script; finally, it would be nice to be able to customize other aspects of the process, such as the exact environmental variables passed, just as the wrapper script sudoscript does. It doesn't need to have the setuid bit set, since it is going to be invoked by sudo, so on its own it shouldn't be a security hazard; the recommended permission is to have it only executable (not readable or writable) by owner, and having root own it.

To effect this, I customized script to

- Write session transcripts to /var/log/super-trans, with each session in a separate file identified by the start time and the PID of the process.
- Write a keystroke log to syslogd (with the idea that syslog is configured to send these securely to another machine). The default setting currently is to use the facility LOCAL2, although there is a runtime option -F to let you (numerically) specify another facility.

- Keep it fairly small and redistributable (it can be linked with dietlibc [7] to create a statically linked binary that is under 50k on a CentOS 4.2 distribution using gcc 3.4.4.)

To install suroot, all you need to do is compile suroot.c (available at http://www.cs.fsu.edu/~langley/suroot), place it in (for instance) /usr/local/bin owned by root and with permissions 0100 (execute bit only for root; it doesn't need to be suid), install one hard link per sudo user (for tracking purposes), and add a line to /etc/sudoers.

For instance, if after you install the binary in /usr/local/bin you want to let user1 use it, you would add this hard link:

```
ln /usr/local/bin/suroot /usr/local/bin/suroot-user1
```

and add the following line to /etc/sudoers:

```
user1 server1=/usr/local/bin/suroot-user1
```

The program suroot is simply a modification of script and keeps script's model. Here's how both script and suroot work, using three processes: (1) the original process, which is used for keyboard input (**parent_p**); (2) a child process, which is used for handling the output to the transcript (**child_p**); and (3) a grandchild (child of the child) process which is our shell (**gchild_p**).

Prior to creating **child_p** or **gchild_p**, we have **parent_p** clear all environmental variables except for TERM and HOME, and obtain a pseudo-terminal, either by the BSD standard openpty(3) or, if it isn't available, by searching for a free /dev/pty[p-s][0-9a-f] device.

Just before the **gchild_p** has a successful exec() to a shell process, its stdin, stdout, and stderr file descriptors are dup2()'ed over to the slave side of the pseudo-terminal. The current version of suroot uses a hard-coded /bin/bash as its shell; the shell is invoked with both the options -i (interactive) and -l (treat this as a login shell).

The **child_p** process has been modified slightly so that the transcript file is now always located in /var/log/super-trans/, and is named first by when the **child_p** process started and then by its PID. For example, the file name /var/log/super-trans/2006-01-20-18:06:38-021592 indicates that it was created on January 20, 2006, by process 21592.

To effect the system logging of keystrokes, the doinput() routine has been augmented with two new buffers, svbuf1 and svbuf2. The buffer svbuf1 records the raw input; if the process is in a default printable mode (non-printable characters are mapped into some visually attractive version, such as ASCII 010 being rendered as C-h), the printable contents of svbuf1 are copied into svbuf2. If raw characters are keystroke-logged, then, one would need to make sure that the receiving syslogd will be happy to receive them.

I like to statically link binaries such as this for three reasons:

- With a security application, I like to be certain that I am not using the wrong shared library; despite the care that sudo takes to make sure that all shared library paths are cleared from the environment (most importantly, of course, LD_LIBRARY_PATH), I am still leery of them.
- The functions that it is calling are simply not likely to be updated by any libc, so why bother to keep looking dynamically for updated versions of those functions every time that it runs?
- If you statically link with a small libc such as dietlibc, the resulting static binary is not much larger than the dynamic version.

However, static linking is not as easy these days as it might be in light of the nss_* situation. I wanted to use glibc's getpwuid() to get home directory information; however, glibc's getpwuid() is now entangled with nss_*, which cannot be statically linked. I didn't want to write my own parser for the password file since, historically, this simple activity has been implicated in various security lapses, and I was already adding two new buffers that could potentially allow buffer overflows.

So I decided to go with a smaller, more compact libc that doesn't share this problem. For Linux I chose dietlibc, since I knew that it was complete enough to use for a full distribution (the Linux distribution DietLinux [8] is wholly built with dietlibc). I haven't managed yet to get suroot to statically link on Solaris. The implication here would be that somehow this would be started with UID 0 and a path such as LD_LIBRARY_PATH would somehow not be wiped out when the code removes all environmental variables except for TERM.

What are the limitations of this approach? The first is that this is only meant to directly run administrative code. It's not a general setup since it doesn't try to solve the problems of handling general users, such as those not in /etc/passwd or creating transcripts for non-root users (presently, transcripts are only created in /var/log/super-trans, which is only root writable). While both of these are addressable, there is a third problem (and one applicable also to the root account): the keystroke logging is not intelligent enough to detect the entry of a password, and will happily log any such that are typed.

REFERENCES

[1] Todd Miller, Chris Jepeway, Aaron Spangler, Jeff Nieusma, and Dave Hieb, Sudo Main Page, http://www.courtesan.com/sudo.

[2] Tom Christiansen and Dave Koblas, The op Wiki, https://svn.swapoff.org/op.

[3] Howard Owen, "The Problem of PORCMOLSULB," *;login:*, vol. 27, no. 4, August 2002.

[4] NSA, "Security Enhanced Linux," http://www.nsa.gov/selinux.

[5] Chet Ramey and Brian Fox, *GNU Bash Reference Manual* (Network Theory Ltd, 2003).

[6] The Honeynet Project, About the Project, http://www.honeynet.org.

[7] Felix von Leitner, "diet lib c—a libc optimized for small size," http://www.fefe.de/dietlibc/.

[8] Bernd Wachter, "Aardvarks DietLinux," http://www.dietlinux.org.

DAVID BLANK-EDELMAN

practical Perl tools: programming, ho hum

David N. Blank-Edelman is the director of technology at the Northeastern University College of Computer and Information Science and the author of *Perl for System Administration* (O'Reilly). He has spent the past 20 years as a system/network administrator in large multi-platform environments, including Brandeis University, Cambridge Technology Group, and the MIT Media Laboratory. He was the chair of the LISA 2005 conference and is an Invited Talks chair for the upcoming LISA '06.

dnb@pobox.com

WELCOME BACK TO THIS LITTLE PERL column. This time the official *;login:* theme is "Programming." Given the subject of the column, that theme is just a walk in the park for this humble columnist. I could simply lean back in my chair, put my boots up on the desk, tilt my hat at a rakish angle, stick a stalk of wheat between my teeth, and say, "Ah yup. Perl's a programming language all right,"[1] and I would have done my part to keep this issue on topic.

But for you, I'll work a little harder. Let's take a look at three programming practices that have cropped up in the Perl world.

Practice #1: Test-First Programming

This first practice isn't actually Perl-specific at all, but we'll look at how it easily can be implemented using Perl. It's not necessarily new either, but the notion has caught on with some of the most respected Perl luminaries and so is receiving more lip service these days than ever before. Plus, this was not something I was taught back when I was a CS major in college (back when you had to learn to whittle your own Turing machine), so it may be new to you.

Simply put, when creating anything greater than a trivial program you need to first write a set of test cases the code is expected to pass. These are written before you write a lick of the actual code. This is the reverse of the standard practice of writing the code and later figuring out how to test it.

This ordering may seem strange, because at first the test cases should completely and unequivocally fail. Since the real code doesn't exist yet (you are just calling stub code at this point), this is to be expected, because there isn't really anything to test. As more and more of the real code is written, ideally more and more of your test cases should begin to pass.

So why write a bunch of test cases that start out failing like this? Perhaps the largest win is that it forces you to think. You are forced to form a clear idea of expected input, output, and (ideally) the possible error-handling the program will exhibit once fully written. This pre-programming pondering can often be pretty difficult, especially for those programmers who like to wander towards their goal, making stuff up as they go along. The

1. Reprising my pivotal, but deleted scene from *Brokeback Mountain*. Look for it in the programming part of the special features when the collector set comes out on DVD.

added discipline may sting a little, but you will find the results are better in the end.

There are other side benefits to this approach as well. It is not uncommon during development to fix one bug and unwittingly introduce one or more new bugs in the process. With test-first development, if your test code is good you should notice those new bugs the very next time you run the tests. This makes debugging at any point easier because it becomes possible to discount many other possible contextual problems when other sections of the code are tested to be known-good.

So now that you are beginning to get religion, let's see how this works in Perl. To Perl's credit, the idea of test cases/code has been present in the community for a very long time. The classic module installation recipe of:

```
perl Makefile.pl
make
make test
make install
```

or the increasingly common:

```
perl Build.pl
./Build          # or just Build (for win32)
./Build test     # or just Build test (for win32)
./Build install  # or just Build install (for win32)
```

both imply that there are tests written that should pass before an installation.

2. Bias alert: one of the co-authors of this book is a student of mine here at Northeastern University. Bias aside, it is a really good book.

There's a whole good book[2] on writing test code with and for Perl, by Ian Langworth and chromatic, called *Perl Testing: A Developer's Notebook* (O'Reilly), so I won't go into any sort of depth on the subject. We'll just get a quick taste of the process and if it interests you, you can pursue more resources online or buy this book.

There are two core concepts:

1. Find ways to encapsulate the question, "If I give this piece of code a specific input (or force a specific error), does it produce the specific result I expect?" If it does, test succeeds; if it doesn't, test fails.

2. Report that success or failure in a consistent manner so testing code can consume the answers and produce an aggregate report. This reporting format is called the TAP (Test Anything Protocol) and is documented in Perl's Test::Harness::TAP documentation.

See Perldoc's Test::Tutorial in the Test::Simple package as a first step toward writing tests. Here's an utterly trivial example, just so you can see the bare-bones ideas made real:

```
use Scalar::Util ('looks_like_number');
use Test::Simple tests => 4;

# adds 2 to argument and return result (or undef if arg not numeric)
sub AddTwo {
    my $arg = shift;
    if (! looks_like_number $arg) { return undef; }
    return ($arg + 2);
}

ok ( AddTwo(2) == 4,              'testing simple addition');
ok ( AddTwo(AddTwo(2)) == 6,      'testing recursive call');
ok ( AddTwo('zoinks') eq undef,   'testing non-numeric call');
ok ( AddTwo(AddTwo('zoinks')) eq 'bogus test',
                                  'testing recursive non-numeric call');
```

Running the code, we get very pretty output that describes the number of tests run and their result (including the last, broken test):

```
1..3
ok 1 - testing simple addition
ok 2 - testing recursive call
ok 3 - testing non-numeric call
not ok 4 - testing recursive non-numeric call
# Failed test 'testing recursive non-numeric call'
# in untitled 1.pl at line 16.
# Looks like you failed 1 test of 4.
```

Test::Simple makes it easy to write quick tests like this. It provides an ok() routine which essentially performs an if-then-else comparison along the lines of "if (your test here) { print "ok" } else { print "not ok" }"; that simple construct is at the heart of most of the more complex testing that can take place. Don't be fooled by how trivial the ok() construct looks. The complexity of the code being called in the ok() is in your hands. If you want to write something that takes eons to compute like:

```
ok(compute_meaning($life) == 42, 'life, the universe, and everything');
```

you can do that.

There are a whole slew of other modules that allow for more advanced tests with more sophisticated comparisons (e.g., Test::Deep will compare two entire data structures), data sources (e.g., Test::DatabaseRow can access a SQL database), control flow items (e.g., Test::Exception for testing exception-based code), and other program components (e.g., Test::Pod to test the code's documentation).

Once you've written a gaggle of individual tests you'll probably want to bring something like Test::Harness into the picture to allow you to run all of the tests and report back the aggregate results. You've probably used Test::Harness before without even knowing it. It is the module called by most modules during the "make test" or "build test" install phase.

If your test scripts output the right TAP protocol, using Test::Harness is super-simple:

```
use Test::Harness;

my @test_scripts = qw( test1.pl test2.pl test3.pl );

runtests(@test_scripts);
```

The three scripts will be run and the results reported at the end. Test::Harness also provides a prove command which can be used to run a set of tests from the command line. See *Perl Testing* for more details on all of these test-related ideas.

Practice #2: Write the Code in Another Language

Oh, the heresy, the sacrilege, the gumption! I hate to be the one to introduce a little existential truth into this issue of *;login:* (usually I'd save that for the philosophy-themed issue), but sometimes you need to program in another language besides Perl to get the job done. Perhaps the vendor of a product you are using only provides C libraries and header files or you've found a really cool Python library that doesn't have a Perl equivalent. The bad news is that sometimes these situations occur; the good news is that you don't necessarily have to write your entire program in that foreign language. You may be able to create a tiny island of strange code surrounded by a sea of Perl.

One easy way to include foreign languages within a Perl program is through the Inline family of modules. Here's a quick example of embedding Python in Perl code (oh, the impiety!) taken from the man page for Inline::Python:

```
print "9 + 16 = ", add(9, 16), "\n";
print "9 - 16 = ", subtract(9, 16), "\n";

use Inline Python => <<'END_OF_PYTHON_CODE';
def add(x,y):
        return x + y

def subtract(x,y):
        return x - y

END_OF_PYTHON_CODE
```

Inline modules exist for a whole bunch of the popular and more obscure programming languages. There's a good chance you'll be able to find what you need to embed that language into your Perl code.

Another potentially useful method of programming in another language involves a language that doesn't really exist yet (certainly not in a finished form): Perl 6. There are two ways to begin enjoying some of the nifty and mind-blowing features of Perl 6:

1. PUGS (http://www.pugscode.org)—I don't think I'd use this for any serious tasks yet, but if you want to play around with Perl 6 well ahead of the actual language being ready, you can use a project started by the worship-worthy Autrijus Tang. Tang and some other programmers have basically been working to implement the Perl 6 language as specified to date using the functional programming language Haskell. This lets people kick the tires on the language design by actually using it. See the URL above for more details.

2. Damian Conway had a similar notion about using implementation to test the design, so he led the charge to create Perl 5 modules that offer test implementations for various pieces of the Perl 6 language design. He and a group of other authors have been releasing modules into the Perl6:: namespace on CPAN for quite a while.

For example, if you'd like to use the new Perl 6 slurp command to read the contents of a file into a variable, you could

```
use Perl6:: Slurp;

$data = slurp 'file';
```

Probably the most useful of these modules is the Perl6::Form module, which allows you to use the Perl 6 replacement for Perl 4/5's sub-optimal format built-ins. See the Perl6:: modules on CPAN for the sorts of Perl 6 features available for use in your Perl 5 programs today.

Practice #3: Add a Little Magic to Your Programs

For our final topic we're going to look at a couple of ways to get work done via "magic." Since we just mentioned Damian Conway in the last section, let's show another one of his creations: Smart::Comments. With this module the normally passive comments in a program's listing can spring to life and do interesting things. For instance, if you wrote code that looked like this:

```
use Smart::Comments;

for $i (0 .. 100) { ### Cogitating |===[%]    |
        think_about($i);

}

sub think_about {
        sleep 1; # deep ponder

}
```

the program would print a cool animated progress bar that would look like this at various stages in the program run:

```
Cogitating |[2%]                    |
Cogitating |====[37%]               |   (about 1 minute remaining)
Cogitating |=============[71%]      |   (about 30 seconds remaining)
Cogitating |==========================|
```

We didn't have to write all of the progress bar code (or even the part that attempts to provide an estimate for how long the program will continue to run), all we had to do was add the comment ### Cogitating |===[%] | next to the for() loop. This module can do other spiffy things that help with debugging your code; be sure to consult its documentation for details.

The last piece of magic I want to bring to your attention is the IO::All module by Brian Ingerson. This module is so magical that it is hard to describe. Here's what the docs have to say:

> IO::All combines all of the best Perl IO modules into a single Spiffy object-oriented interface to greatly simplify your everyday Perl IO idioms. It exports a single function called io, which returns a new IO::All object. And that object can do it all!

And when it says "can do it all!" it isn't kidding. Here are some examples to give you a flavor of its capabilities:

```
io('filename') > $data;            # slurps contents of filename into $data
$data = io('filename')->slurp;     # does the same thing

$data >> io('filename');           # appends contents of $data to filename
io('filename')->append($data);     # does the same thing

io('file1') > io('file2');         # copies file1 to file2

$line = io('filename')->getline;   # read a line from filename
io('filename')->println($line);    # write a line to filename

$io = io 'filename';
$line = $io->[@$io /2 ];           # read a line from the middle of filename

@dir = io('dirname/')->all;        # list items found in dirname
@dir = io('dirname/')->all(0);     # recurse all the way down into dirname
```

From these examples you can see that IO::All makes it easy to read and write to files and operate on directories with a minimum of code. It has both a OO-like interface (e.g. ->slurp) and a set of overloaded operators (e.g., >) for these tasks. Many of these methods can be chained together for even quicker results.

But that's only a small part of the magic. Let's see more of the IO::All pixie dust:

```
io('filename')->lock;                   # lock filename
io('filename')->unlock;                 # unlock filename (could also ->close())

io('filename')->{lulu} = 42;            # write to DBM database called filename
print io('filename')->{tubby};          # read from that database

$data < io->http('usenix.org');         # read a web page into $data
io('filename') > io->('ftp://hostname') # write filename to ftp server

$socket = io(':80')->fork->accept;      # listen on a socket
$socket->print("hi there\n");           # print to the socket
$socket->close;                         # close the connection
```

Easy file locking, database access, and a dash of network operations. Pretty spiffy indeed.

And with that, I'm afraid we have to bring this issue's column to a close. Take care, and I'll see you next time.

RENEW ONLINE TODAY!

Renewing or updating your USENIX membership has never been easier!

You will receive your renewal notice via email and one click will take you to an auto-filled renewal form.

Or visit

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

and click on the appropriate links.
Your renewal will be processed instantly.

Your active membership allows the Association to fulfill its mission.
Thank you for your continued support!

HEISON CHAK

VoIP watch

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

heison@chak.ca

VOIP (VOICE OVER INTERNET Protocol) is becoming an increasingly popular tool for business. For the system administrator, VoIP means new protocols to learn, new security issues, and new servers to be configured and supported. You will also want to become familiar with a long list of acronyms, as VoIP, like the telecoms that precede it, is speckled with TLAs and FLAs (three- and four-letter acronyms). Over the next year, I will expound upon these topics and more, so you can understand how VoIP works and how to support it in the networks you manage, as well as how to take advantage of it personally.

VoIP is now known as the emerging technology that allows home users and businesses to save money by placing calls over the Internet. It brings innovative applications into telecommunication, mainly through the ability to remove the constraints of circuit switching and replace it with packet switching. As a result, larger call volume is achievable on the same raw bandwidth as the PSTN (Public Switched Telephone Network). While developers are striving to improve reliability and availability of VoIP, security experts and government agencies are trying to put in place regulations and processes to protect the interests of end users as well as operators.

Many are using the terms IP Telephony and VoIP interchangeably. VoIP samples analog voice signals, digitizing them into 1's and 0's, then packetizing them before placing them on an IP network for transmission; IP Telephony takes it one step further and supports other POTS (Plain Old Telephone Service) services (e.g., facsimile, modem communications) that PSTN subscribers have been using for decades. In essence, VoIP can be thought of as a subset of IP Telephony.

Benefits of VoIP

Although VoIP has only gained popularity and momentum in the past two years, the technology has been around for much longer. In the mid-1990s, while most homes were using long distance service provided by their local telephone carrier, a.k.a. ILEC (Incumbent Local Exchange Carrier), a wave of CLECs (Competitive Local Exchange Carriers) offering competitive long distance and international rates were born. Many were early adopters of VoIP technologies.

Corporations budget hundreds of thousands of dollars every month on communications to maintain operation of their businesses. With the growth of the Internet, the improved reliability and availability of this global network allow IT managers and decision makers to offload more and more voice calls onto this public network. Interoffice communications and calls destined for PSTNs (Public Switched Telephone Networks) are equally suitable for VoIP switchover. Besides reducing the per-minute cost (or bypassing toll charges altogether), many corporations also find themselves receiving large tax benefits. What was previously budgeted for talk time may now be allocated to expanding the IT infrastructure to cope with the higher demand of bandwidth and stability on IP networks.

Home users may find themselves overwhelmed with billboard and TV commercials, and there are mixed reactions to the introduction of VoIP service. While many are enjoying the convenience and portability of VoIP, others worry about the reliability of the service. Typical features of VoIP in SOHO (Small Office, Home Office) deployment may involve:

- Simplified subscription process and easy setup
- Ability to receive calls on a hometown number in another city or country
- Free or bundled pricing on calls made nationwide
- Voice-mail delivery via email
- Failover from VoIP to landline
- Ability to modify call features (e.g., call forwarding) online

In 2005, 16% (42 billion minutes) of all voice calls were made using VoIP,[1] and that number is growing. Despite the popularity and acceptance of VoIP, there are a number of ongoing concerns. Many early adopters have reported problems with VoIP, such as:

1. Statistics from TeleGeography: http://www.telegeography.com/press/releases/2005-12-15.php.

- Dropped calls
- Line echo
- Clipping sounds
- Touch tone recognition
- E911

While some of the problems require changes in legislation or government intervention, most are the result of placing real-time media in networks originally provisioned for different purposes. With VoIP-aware networks and improvement in protocols and codecs, many issues that adversely impact one's experience with VoIP can be eliminated.

Opportunity to Gain User Trust

One of the benefits of VoIP is the consolidation of voice and data networks running over the same physical wiring. Although this brings great savings to the IT infrastructure, it may also be one of the contributing factors to deployment failure. In traditional PBX systems, dedicated telephone (e.g., Category 3 Twisted Pair) wirings provide the physical connectivity between handsets and the telephone switch. Not only is quality of service maintained, but PBX systems often provide security by transmitting proprietary digital signals across these cablings. With traditional PBX systems replaced by VoIP, security and availability become significant considerations. With voice and data packets mixed together, anyone with access to the routers and switches the packets are transmitted on can potentially compromise the integrity of the media stream. VoIP-ready networking equipment is essential to a successful deployment, as it makes it easier to differentiate between voice and data packets, allowing it to prioritize time-sensitive media according to the urgency of those packets. Improving QoS can better ensure timely delivery of packets and, therefore, higher availability.

On traditional PBX systems, it is not unusual to see telephone switches and small UPS units all crammed into tiny riser rooms. When these legacy systems are replaced by VoIP, UPS power may need to be increased to support network switches, routers, and VoIP handsets. Cooling capacity and air circulation will very likely require adjustments. Due to the nature of IP networks, VoIP outages are more likely to occur as compared to the PSTN. It is considered good practice to keep a couple of POTS lines for emergencies.

Communication between IT professionals and users plays a significant role in the success of any major project. Since VoIP allows for deployment in phases, it may be worthwhile to spend time and share with users the potential enhancement that VoIP can bring to their communication needs.

Gathering Requirements

VoIP provides many features, but not all may be beneficial or suitable for an individual or organization's needs. It is important to understand which VoIP application can most significantly enhance business development or daily communications. For example, if a corporation has high-volume teleconferencing requirements, it may be more cost-effective to acquire VoIP conferencing-capable servers to work with existing legacy systems rather than spend resources replacing the entire system with technologically advanced handsets.

CDR (Call Detail Records) logs and phone bills are generally good places to start investigating where telecom resources are spent. If long-distance or international calls show up frequently in these log histories, it may be time to investigate more competitive pricing. Some VoIP providers offer 3–5 cents per minute charge to major cities when most PSTN long distance providers are still selling 20–50 cents per-minute rates.

In terms of providing VoIP services, some VSPs (VoIP Service Providers) only provide call termination (e.g., allow incoming calls only for toll-free service), while others provides call termination as well as call origination (i.e., incoming and outgoing calls). Most residential VoIP service on ATA (Analog Telephone/Terminal Adapter) provides incoming and outgoing capabilities, but may not support simultaneous calls. There exists VSP wholesalers that support multiple incoming and outgoing calls, with toll-free numbers termination, and the best part—no monthly charges.

Moving Forward

There are pluses and minuses to any technology, and VoIP is no exception. On one hand, it allows for feature-rich deployments with relatively low ongoing costs. Communication becomes much more efficient and affordable, especially for those who travel a lot (e.g., salespeople, field engineers). In larger corporations, savings as a result of toll bypass may be significant. End users of commercial deployment are likely to enjoy the flexibility of taking the VoIP ATA away from their hometown while they travel, or the ease of giving overseas relatives a hometown number to save on long distance charges.

On the other hand, VoIP is one of those technologies that is evolving quickly. Because of that, there aren't very many standards and guidelines to allow long-term survivability. Carriers and IT managers are having to face the pros and cons of different technologies and interoperability issues. A classic example would be the two competing protocols H.323 and SIP. Although favored by the academic world, SIP still has some of the same problems that have haunted H.323 (e.g., NAT/firewall issues). Until one of these protocols becomes dominant, deploying VoIP-aware routers and switches that support both protocols will guarantee a safer investment.

USENIX Membership Updates

Membership renewal information, notices, and receipts are now being sent to you electronically. Remember to print your electronic receipt, if you need one, when you receive the confirmation email.

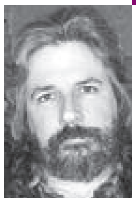
You can update your record and change your mailing preferences online at any time.

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

You are welcome to print your membership card online as well.
The online cards have a new design with updated logos—all you have to do is print!

ROBERT G. FERRELL

/dev/random



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

rgferrell@direcway.com

WELCOME TO /DEV/RANDOM, A KAYAK tour along the sporadically navigable wide spots in my stream of consciousness. Various incarnations of UNIX have played a major role in my life, both professionally and personally, for a quarter-century now. Before I get too old and doddering to set fingers to keyboard, I thought I'd share some of my own UNIX experiences, those of close acquaintances, and possibly even a few from total strangers if the mood strikes me. It would have been better for public relations if I could have found some computing luminary to introduce me in glowing terms as a shining UNIX guru and heavyweight player in the industry, but the price for that sort of bald-faced hype has gone through the roof since Oracle declared themselves "hackproof," and frankly we just don't have the budget for it. You're free to pretend, if you like.

I first tumbled down the UNIX rabbit hole in 1981, as a graduate student at Texas A&M University. Prior to then I'd encountered only O/S 370, and that via a blistering eight-baud teletype terminal, although I actually started on the path to hackerdom in the early 1970s with the occasional bout of phreaking. One might reasonably wonder how a person who styles himself a classic "geek" avoided contact with perhaps the ultimate technological expression of that proclivity, the personal computer. Easy—there weren't any. OK, that's not literally true: there were the Apples, the Altair, the TRS-80, and a few others. But as one of the original scions of West Texas poverty, none of those were available to me. No, I earned my geek appellation in part by building several radio telescopes (including a three-meter parabolic dish and a multi-element Yagi interferometer array) in my backyard and on the roof of the science building at my high school. My neighbors must have thought I was some sort of spy, since home satellite dishes were largely unknown in 1975.

Like any good geek, I loved the SR-10 calculator I had to rake a lot of rocks to afford, but it was at the beginning of my sophomore year of college that I landed the part-time job that would largely determine my future career path: remote terminal operator. I learned JCL (Job Control Language) and keypunch and suddenly found myself alone in a data center at nights and on weekends—just me, the blinking lights, and all that free time. IBM 370 JCL was a wonderfully cryptic jumble of punctuation marks worthy of an NSA analyst, and I ate it up like candy. I felt all intellectual and tingly inside, punching in those slashes and asterisks and assorted operands, then hitting “autoverify.” My very first hacker handle (I used it with my CB radio, too) was “ddname,” in fact, although I eventually just shortened it to “deedee.” Another warning sign of incurable geekhood and future government service.

In the summer of 1981 a friend of mine had an account on a VAX in the Physics Department at Texas A&M running, I seem to remember, AT&T version 7. It was my first UNIX, my shining virgin leap into the interactive console command line. I was hooked in 10 minutes, maybe less. No cards to punch, for one thing, and a seemingly bottomless pool of commands to explore, most of which resembled actual words. The idea that you could write and run a shell script in “real time” was pretty seductive, too. My prior experience with academic computing was that you punched in a program, put the box of rubber-banded cards with your name on it in a basket, and came back the next day for your error messages. With UNIX, however, you got instant gratification and/or smackdown. Intoxicating it was, young Skywalker.

Printed manuals were hard to come by in those days, which meant that any problem solving had to be done by trial and error. At my age now that sort of mental exercise just makes me tired, but back then it was an irresistible challenge to my nerditude. Hacking out a shell script in the wee hours was all part of the game. I still see some of the Perl scripts I wrote (albeit much later) for performance monitoring and so on floating around in old archives on the Web, like resin-cast trilobites in a museum gift shop.

Another fateful defining moment occurred when my friend and I discovered, via a physics grad student, that a text-based game we had heard about called “Adventure” resided on this very box. Unfortunately, playing it required an account with much better privileges than those of a mere student assistant. We paged through the user roster until we found a professor who was on sabbatical, a professor with faculty-level access to the crucial VAX. All we needed was the elusive account password. In those prehistoric days you couldn’t just go online and download Jack the Ripper or L0phtCrack. If you wanted to launch a dictionary attack, you had to supply your own injection code and your own wordlist. We did our research and loaded it with words and numbers that seemed as though they might be significant to the professor in question, including family member/pet names, phone numbers, faculty ID number, office number, and so on. Finally nailed it after about an hour of runtime with his street address and dog’s name. Piece o’ cake. We did the happy hacker hop of victory, cracked our knuckles, and got down to serious entertainment.

We played Adventure all that summer, breathlessly mapping out the twisty little passages on the back of used green and white-striped tractor-fed printer paper and typing “plugh” every few minutes to see what might happen, always careful to scrub logs and reset quotas after each session. It was my introduction to computer hacking, computer gaming, and UNIX all rolled into one, and it was a heck of a lot of fun. Call me an old fuddy-duddy (you won’t be the first), but the “point, click, and r00t” mantra of the Metasploit generation just doesn’t have the same allure.

NICHOLAS M. STOUGHTON

USENIX Standards Activities

Nick is the USENIX Standards Liaison and represents the Association in the POSIX, ISO C, and LSB working groups. He is the ISO organizational representative to the Austin Group, a member of INCITS committees J11 and CT22, and the Specification Authority subgroup leader for the LSB.

nick@usenix.org

2005 was a busy year for me as the USENIX standards representative. There are three major standards that I watch carefully:

- POSIX, which also incorporates the Single UNIX Specification
- ISO-C
- The Linux Standard Base (LSB)

In order to do that, USENIX funds my participation in the committees that develop and maintain these standards. Throughout 2005, the Free Standards Group (FSG) also helped fund these activities. For each of these, let's look at the history of the standards, then at what has happened over the past 12 months or so, and, finally, what is on the agenda for this year. Each of these standards is critical to a large proportion of our members. Without these standards, open source software as we know it today would be very, very different!

POSIX

The POSIX family of standards was first developed by the IEEE, arising from earlier work from /usr/group and the System V Interface Definition (SVID), and was published as a "trial use" standard in 1986. In 1988, the first full-use standard was published. The difference between "trial" and "full" use is principally in the use of the term "should" rather than "shall" in the requirements for any interface.

In 1990, the 1988 API standard was revised, clarifying a number of areas and expanding them. At the same time, the API standard became an ISO standard. At this point in history, there were about 10 separate POSIX projects under development, ranging from the basic OS system calls and libraries, through commands and utilities, to security, remote file access, super-computing, and more. In 1992, the second part of POSIX was published (the Shell and Utilities volume), and it became a second ISO standard. Amendments to these standards were also under development, and led to the addition of real-time interfaces, including pthreads, to the core system call set. Many of the other projects died away as the people involved lost interest or hit political roadblocks (most of which were reported in *;login:* at the time).

Until the end of the twentieth century, POSIX was developed and maintained by IEEE exclusively. At the same time, the Open Group (also known as X/Open) had an entirely separate but 100% overlapping standard, known as the Single UNIX Specification. This specification started from the same place in history, and many of the participants around the table at an X/Open meeting were the exact same people who had met a few weeks before at an IEEE POSIX meeting to discuss the same set of issues!

This duplication of effort became so annoying that a new, collaborative, group was formed to produce a single document that would have equal standing for each of ISO, IEEE, and the Open Group. That group held its first meeting in Austin, Texas, in 1998, and was therefore named the "Austin Group." The Austin Group published a full revision of the POSIX and Single UNIX specifications as a single document in 2001. It was adopted by all three organizations and is maintained by the same team, which represents the interests of all three member organizations.

Since the 2001 revision, work has been steadily progressing maintaining this 3762-page masterpiece. Every week, there is a steady stream of "defect reports," which range from typos in the HTML version (the document is freely available in HTML on the

Web; see http://www.unix.org/single_unix_specification), through major issues with ambiguous definitions, and so on. Some of these defects can be quickly and cleanly fixed, and two "Technical Corrigenda" documents have been approved, which alter the wording for some of the interfaces to clarify their meanings.

Every ISO standard (and every IEEE standard, too) has a five-year "reaffirm/revise/withdraw" process, where the document is examined to see if it is still relevant, whether it needs revision to meet current needs, or whether it is now outdated and should be withdrawn. For POSIX, the Austin Group has elected to revise the specification during 2006.

Under the Austin Group rules, the group as a whole cannot invent new material. One of its sponsor groups (IEEE, ISO, and the Open Group) must have prepared a document and had it adopted under its own organization rules before it can be presented to the group as a whole. Therefore, the Open Group has been developing, and is now in the final stages of approving, a number of documents which include new APIs to become a possible future part of a UNIX branding program. Once approved, these documents can then be examined by the Austin Group (OK, so it's still the same group of people who developed the set in the first place) for inclusion into the POSIX revision.

The new interfaces under consideration are ones that have been popular in the GNU-C library (glibc) and Solaris for some time, but have not been formally standardized before. They include support for standard I/O functions to operate on memory buffers as well as external files, getline and getdelim, some multibyte string-handling functions, robust mutexes, and versions of functions that take pathnames relative to a directory file descriptor rather than plain pathnames (this helps avoid certain race conditions and helps with really long pathnames).

I would expect to see official drafts of this new revision this summer, and the final version in 2008.

POSIX has long had support beyond the C language world. There are Ada and Fortran official "bindings" to POSIX. However, there has never been a real connection between the C++ world and the POSIX world; C++ programs can use C to call POSIX functions. But this leads to all sorts of complications for C++ programmers and, more seriously, to much reinvention of the wheel in providing mappings between C++ constructs and those of POSIX. The Austin Group has received several defects from C++ programmers who want to know why they can't do x, to which the traditional answer has been "don't use C++, use C"! And to make matters worse, the C++ language committee is also going through a revision at present, and they want to add all sorts of features to the language that might make it harder to access some of the fine-grained features of POSIX (since they want the language to work on other platforms, they deliberately try to be OS-neutral).

All that may change soon. A study group has recently been formed to look into the need for, and desire to build, a C++ binding to POSIX. USENIX is hosting the wiki for this group, and you are welcome to join: http://standards.usenix.org/posix++wiki.

ISO C

The first version of the ISO C standard, then known as ANSI-C, was published in 1989. It took the original language from Kernighan and Ritchie's book and tightened it up in a number of places. It added function prototypes and considerably improved on the standard C library. The first versions of POSIX used this language as the underlying way to describe interfaces, and included a c89 command to invoke the compiler.

Between 1989 and 1999, the C committee added wide character support and addressed several language "defects"—internal discrepancies in the way various features were described. The committee included a number of compiler vendors, who were also keen to have the language permit ways to guide an optimizer: features such as constants, volatile variables and restrict pointers were added to the language for this purpose.

In 1999, a new revision came out which included several new features such as these, along with major rework for floating point support (including things such as complex numbers).

At this point, the committee is fairly happy with the state of the core language and is fighting back against proposals to change it. However, they have not stopped working! They are currently preparing several technical reports that optionally extend the C language in a number of directions. Of these, by far the most significant to most USENIX members is the report formerly known as the "Security TR." I say formerly because the term "Security" (and it turns out, many other related words) are so overloaded and charged with meaning that by far the most objections to the document were to its title.

The report formerly called the "Security TR" actually attempts to deal with the fairly common problem of buffer overflow. It does so in a very simple fashion: every interface in the ISO-C standard library that takes a buffer has a secure variant which includes the size of the buffer. Now, while that is the meat of the original concept, it isn't all that the report currently proposes. The report introduces the concept of runtime constraints, that is, various things that must hold true when an interface is invoked. The original standard library simply had undefined behavior when you passed a null pointer to an interface that expected a pointer to a buffer. So

```
char *p = malloc(10);
gets(p);
```

could fail in a variety of ways, despite being well-formed, legal C.

The new "secure" library version of this,

```
char *p = malloc(10);
gets_s(p, 10);
```

will invoke a runtime exception handler (analogous to a signal handler) if p is null (because the malloc failed) or if there are more than 10 characters on the next line of standard input.

According to its current stats, this document proposes a library that might be of benefit to someone going over thousands or millions of lines of existing code and trying to find and plug all of the possible buffer overflow spots. It is likely to end up obfuscating some of the code. It is also possible that if the buffer size is not well known, it could end up hiding bugs where the programmer simply guesses at a buffer size but is wrong; now the code looks as if it has been retrofitted to prevent buffer overflows, but it hasn't!

It will also likely change the ABI of third-party libraries that want to use this; they must now have a way of receiving the size to check against. This suggests to me that this library will have little uptake as it stands, though Microsoft has implemented it and has updated all of its core programs to use it (is this a good thing?).

The core of the problem is that memory handling in C is complicated and error-prone. Nobody will doubt that improvements in the supporting APIs are useful, but the existing APIs already provide the means to write correct programs. It is just cumbersome to do so. The proposed interfaces won't change that; on the contrary, they could make programs even more complex. An alternative approach is to take as much of the memory handling away from the programmer as possible.

To that end, I am preparing a second part to this technical report that uses dynamic memory allocation instead of static buffers. For new programs (rather than retrofits of old code), this approach leads to a cleaner, more robust application, with fewer possibilities for problems. For example, instead of reading data from an input with gets into a static buffer (that might be too small), the getline function allocates a buffer big enough to hold the entire input line, however long it was (or returns NULL if there was insufficient memory). The only problem with such an interface is that the programmer must remember to release the memory when he or she is done with it, by means of a call to free. Some have argued that this, too, can lead to unexpected bugs, as programmers forget to free these buffers, and the application slowly leaks memory. However, I believe this is a smaller problem than the use of static buffers with guessed sizes.

Back to the name of this report: as I said , "Secure Library" got a ringing "no" vote. This report does not address any of what many people regard as security issues. The name "Safer Library" was suggested, but the owners of a product called "Safer-C" objected. In the end it has come down to "Extensions to the C Library—Part 1—Bounds Checking Functions."

THE LINUX STANDARD BASE

The LSB is an Application Binary Interface (ABI), rather than an Application Programming Interface (API). As such, it covers details of the binary interfaces found on a given platform, providing a contract between a compiled binary application and the runtime environment that it will execute on. The first version was published in 2000 and has developed rapidly since then. It now consists of a Core specification (including ELF, Libraries, Commands & Utilities, and Packaging), a Graphics module (including several core X11 libraries), and a C++ module.

Each specification has a generic portion that describes interfaces that are common across all architectures and seven architecture-specific add-ons that spell out the differences between the architectures.

For the past year or more, I have been acting on behalf of the Free Standards Group as the technical editor for the ISO version of this standard. ISO 23360 was unanimously approved last September by the national bodies that contribute to the subcommittee responsible for programming languages and their runtime environments.

We have had to jump through a few hoops in the final publication phase, but now it looks as though the document is ready. You will soon be able to buy a CD from ISO with the LSB on it (see http://www.iso.org), or you can just download the PDF for free from the Free Standards Group (though the copyright notice is subtly different, as are the running headers and footers—see http://refspecs.freestandards.org.

What now for the LSB? Are we done? Of course not! The LSB workgroup has a new chair, Ian Murdock (the Ian of Deb*ian*). A new subgroup is developing a desktop specification, with an increased focus on libraries needed by desktop applications such as GTK, Qt, PNG, XML, more X, imaging, etc.

And with the pace of development in the open source community, it is necessary to continually revise the specification to match current practice. For example, until recently the pluggable authentication modules (PAM library) had no symbol versioning, but the upstream maintainers have now decided to add that (which makes maintaining an ABI possible). The LSB now has to be updated to discuss which version of which symbol you should be using to get the promised behavior.

Additionally, the LSB Core Specification is mostly a superset of the POSIX APIs. However, there is a small handful of places where the two specifications are at odds. For the most part, these differences won't bother most programmers most of the time, but there are corner cases you can creep into where you'll find your application isn't portable between an LSB-conforming platform and a POSIX-conforming platform. For example, POSIX requires the error EPERM if you attempt to unlink a directory, while the LSB requires this error to be EISDIR.

A document describing these differences is now available from ISO as Technical Report 24715.

During the revision of POSIX this year, and as a part of any future LSB development work, we will review these changes to see if there is any way that either specification can accommodate the behavior of the other in some deterministic fashion.

A well-supported standard for Linux is a necessary component of Linux's continued success. Without a commonly adopted standard, Linux will fragment, thus proving costly for ISVs to port their applications to the operating system and making it difficult for end users and Linux vendors alike. With the LSB, all parties—distribution vendors, ISVs, and end users—benefit as it becomes easier and less costly for software vendors to target Linux, resulting in more applications being made available for the Linux platform.

It is important for the LSB workgroup not to slip into the comfortable feeling that the job is now done. If the workgroup does not remain focused on the core document, that core document will quickly become irrelevant, overtaken by the pressures of distribution vendors to have their product be the de facto standard in the absence of a good de jure base.

My work with the LSB over the past year has not just been as the technical editor of the ISO standard, although this has been a major part of my work. I have also been one of the principal technical editors of the specification as a whole. With the completion of the submission of the initial core specification to ISO, the sources of funding for this critical project have largely dried up. I end with a plea: if your organization believes that standards for UNIX, Linux, and C are important, consider donating money to USENIX to help fund the development and maintenance of these standards.

ANNUAL MEETING OF THE USENIX BOARD OF DIRECTORS

The Annual Meeting of the USENIX Board of Directors will take place at the Boston Marriott Copley Place during the week of the 2006 USENIX Annual Technical Conference, May 30–June 3, 2006. The exact location and time will be announced on the USENIX Web site.

book reviews

ELIZABETH ZWICKY

zwicky@greatcircle.com

with Sam Stover and Rik Farrow

THE TCP/IP GUIDE: A COMPREHENSIVE, ILLUSTRATED INTERNET PROTOCOLS REFERENCE

Charles M. Kozierok

No Starch Press, 2005. 1,539 pages. ISBN 1-59327-047-X

When I started to review books, my husband found that several of his cherished illusions about book reviewing were shattered. First, having publishers send us free books was not as exciting as he had hoped. Second, he had believed that reviewers always lovingly read every page of every book. As an author, I've never believed that, and have in fact cherished the theory that any reviewer who dislikes my book just didn't pay sufficient attention. Different reviewers have different standards; I feel that it's necessary to read every page, except in truly extreme circumstances, but I'm willing to gloss rapidly over some of them.

At 1,539 pages, meeting that standard for the *TCP/IP Guide* has taken me quite a long time. And it's probably not representative of what other readers will do; the book is not intended to be read end-to-end like a novel. But I found that I actually got fonder of the book as I kept going. My early experiences were marred by an indexing issue (I tried to look up the port number for DHCP, which isn't indexed, although it turns out the information is there) and a fundamental disagreement with the author about what constitutes a protocol (I'm sorry, but I know of no coherent definition of the term which allows network address translation to be considered a network protocol). But as I went along I found that while I have issues with the book, it's actually informative and easy to read, even when discussing rather nasty protocols, and when it covers something, it generally covers it quite completely.

The book is something of a strange beast. I would have made some different choices about what to include and what to leave out; for instance, I've seen some pretty odd things on networks—including noncontiguous netmasks, which Kozierok asserts were never used—and I've never seen the ICMP traceroute message type in use. He does point out that it never made it out of experimental status, but only after two pages of discussion. More important, the book's only gesture towards non-UNIX systems is to discuss implementations of UNIX-based TCP/IP protocols. There's minimal coverage for Microsoft extensions and oddities, and no coverage at all of Microsoft file sharing or naming. But it's not just Microsoft that gets shorted; there's no AFS, Kerberos, or LDAP, and RPC is mentioned very briefly in passing during the discussion of NFS.

In the protocols that it does discuss (and there are lots of them), I would have made some different choices about the information to put in, preferring less history and more security, for instance.

So why is it twice as long as volumes 1 and 3 of *TCP/IP Illustrated*, which cover basically the same protocols? Well, it covers IPv6 quite thoroughly, it assumes less expertise on the reader's part, and it covers some topics (like much of the theory behind routing) that *TCP/IP Illustrated* leaves for more specialized books. If you don't have a TCP/IP background, and you're looking for understandable, implementation-neutral descriptions of protocols, it's a good choice for a reference work. Despite my initial misgivings (and my continued pedantic snarling), I'm going to give this one a place on my bookshelf.

OPEN SOURCE FOR THE ENTERPRISE: MANAGING RISKS, REAPING REWARDS

Dan Woods and Gautam Guliani

O'Reilly, 2005. 217 pages. ISBN 0-596-10119-8

You love open source, but you're not sure how to get it into your IT shop; it's not that everybody is committed to what you've got now, but they're nervous about something that seems to involve too many hippies and fanatics. Or, for that matter, you don't love open source, you're a traditional IT manager trying to figure out what to do about open source for one reason or another (not enough money to buy commercial, you're surrounded by hippies and fanatics, your vendor just snapped your last nerve). This sensible book is a good place to start. It's very much in favor of open source software, while maintaining a good grasp of the pitfalls involved, and it speaks in language that nicely bridges the worlds of open source and IT manager.

It is a small book; it left me feeling hungry for more. But it does a nice job of filling its niche. If you need to figure out how much you can reasonably do with open source, or how to convince people to do that, and you're dealing with people who think in traditional IT terms, this book will point you in the right direction and reassure you that it is possible and reasonable.

DESIGNING INTERFACES

Jennifer Tidwell

O'Reilly, 2005. 331 pages.
ISBN 0-596-00803-1

I am by no means an interface designer. On the other hand, I've ended up designing my fair share of interfaces, either because I was the only option or because everybody else involved in the project was even less able. This has left me with the unsurprising insights that interface design matters, it's a lot of work, and that people who do it seriously are better at it than I am. So I was enthusiastic about the idea of a book that would either improve my ability or at least allow me to take a reasonably interested programmer on a team and get them to my level of semi-competence.

Happily, I believe this book meets both goals. It's a book of user interface patterns meant for people who are just starting to think about the design of user interfaces. If you're a serious human-computer interaction person, it's going to be way too basic for you. If you were hoping somebody would just tell you what to do and get it over with, it's going to be too fuzzy for you. But if you're willing to do your own thinking and need somewhere to start, this book should give you the tools to work with.

INTERNET FORENSICS: USING DIGITAL EVIDENCE TO SOLVE COMPUTER CRIME

Robert Jones

O'Reilly, 2006. 216 pages.
ISBN 0-596-10006-X

Internet Forensics is somewhat misleadingly titled. If you're hoping to find out what professionals do when they track down serious crimes, or you're already familiar with computer security, you're likely to find it disappointing. It's a sensible, interesting book on amateur Internet forensics, the sort of thing you might do at home to track down people who are really annoying you. I enjoyed it, although as somebody who already has a security background, I didn't find anything particularly novel in it.

I recommend this book if you don't know a lot about security and want to do something about nasty mail and Web pages. It's also a great lesson in a bunch of basic parts of the Internet; if you want a really motivating way to learn how IP and DNS and HTTP work, it's a lot more fun than reading abstract descriptions, and it will give you a good reason to play around with things until they make sense to you.

SARBANES-OXLEY IT COMPLIANCE USING COBIT AND OPEN SOURCE TOOLS

Christian B. Lahti and Roderick Peterson

Syngress, 2005. 333 pages.
ISBN 1-59749-036-9

This book takes on challenging territory. "Sarbanes-Oxley" and "COBIT" (Control Objectives for Information and related Technology) are the sort of words that inspire simultaneous terror and boredom. Anybody involved in trying to comply with Sarbanes-Oxley is probably going to turn to COBIT as a way of getting a handle on things, but they don't map perfectly, so it's a confusing mess where the only possible downside for getting it wrong is huge fines and jail time. Just the kind of situation in which you'd like a good book to come along and hold your hand, and all the better if it includes the tools you need.

Unfortunately, this book doesn't do a particularly good job of hand-holding. It gives a nice introduction to the issues involved in Sarbanes-Oxley and COBIT, and how the two relate (although I could have done without the intro that portrayed the reader as too technology-obsessed to even pay minimal attention in important meetings; thanks, but I get enough insulting stereotypes from people who're not trying to sell books to me). After that, things go downhill. There are a lot of statements and not a lot of the sort of scaffolding you'll need to make your own decisions about your own site.

Open Source for the Enterprise (see above) does a much better job of discussing the issues and advantages of open source to meet whatever needs you have. Coverage of open source consists of a brief discussion and a CD containing a selection of open source tools that might or might not be a useful part of your Sarbanes-Oxley compliance plan. These tools are mentioned when they talk about the relevant parts of COBIT, but they aren't discussed in enough detail to help you decide whether they're the right tools for you.

As an example of compliance policy, they offer a password compliance policy that violates almost every rule for a good password policy. It mixes information of interest only to administrators with information for users. It doesn't give the users an understandable reason for the policy. It states rules for passwords almost entirely in the negative ("Don't do . . .") and includes pointlessly specific rules. An editing error has caused the only useful information on picking a password to be attached to the "Enforcement" section. And there's no verification mentioned.

I'd pass this one up. Stick to Web resources and separate books on Sarbanes-Oxley and open source.

SOFTWARE PIRACY EXPOSED

Paul Craig

Syngress, 2005. 310 pages.
ISBN 1-93226-698-4

REVIEWED BY SAM STOVER

Since I'm not in on the piracy scene, I can't vouch for the technical accuracy of this book, nor can I just build a lab and put its assertions to the test. But what a fascinating read. I mean, this book had me hooked from page 1 to page 296 (right before the Index). Literally, I couldn't wait to get back to it after setting it down. I'm no stranger to BitTorrent, and we've all been hearing the media hype on Napster, Gnutella, etc., for years. When I first picked up this book, I expected to read about those very applications and their detriment to the Internet, and society as a whole.

What I found was an extremely detailed and thorough journey into the world of piracy, a world that most people don't know exists, much less interact with. Let's get one thing straight—P2P applications like eDonkey and BitTorrent are NOT piracy, at least not the piracy this book speaks to. Piracy is the high-adrenalin world of stealing or cracking applications and posting them to private sites. This world seems to be fueled by peer acceptance rather than monetary gain. Not that there isn't a an underlying "stick it to the man" attitude in the piracy groups, but as presented in this book, fiscal gain is not the primary motivation. Whether it's two couriers racing to get the same application distributed first, or the cracker pitting his sk1llz against the latest anti-piracy measures, it's all about competition.

Unlike other (dry) technical books, this one was extremely thought provoking. I still find myself discussing or contemplating the points the author brings to light. Throughout the first two-thirds of the book, I kept thinking "I can totally see why people get into this." Then I got to Chapter 9, where the high-profile FBI busts were discussed, complete with actual names and sentencing details. Then I started thinking, "Why would anyone do this?" Risking 10 years in prison for something that doesn't pay the bills seems a little extreme to me. A lot of the pirates seem to have day jobs, and piracy is more of a hobby/passion than a career. And some of the achievements are just astounding. Disk storage is measured in Terabytes, bandwidth is FastEthernet or even GigE, and the number of applications distributed in the thousands. Wow. If only the dot-bomb businesses had been this efficient.

The book has a lot of facts and plenty of interviews with real pirates. The research seems very sound, and the interviews ring true. Each aspect of the piracy scene is discussed in depth, from the Suppliers, to the Crackers, to the Distribution Chain. I found the technology discussed in the Cracking section especially interesting, as the author goes into a fair bit of detail when describing common reverse-engineering methods.

The blurb on the front cover bills this book as a "Must Read for Programmers, Law Enforcement, and Security Professionals." I agree totally. In fact, I think it should be required reading, especially for application developers, because if you code something, someone is going to pirate it. You need to know how and why.

My only complaint was the number of editorial oversights in the book. Misspellings and grammatical errors kept popping up. As with some other Syngress books I've read, I'd say that this was rushed to press because they thought the content was ground-breaking. Well, I agree. Just an amazingly fun read.

OS X FOR HACKERS AT HEART

Ken Caruso, Chris Hurley, Johnny Long, Preston Norvell, Tom Owad, Bruce Potter

Syngress, 2005. 439 pages.
ISBN 1-59749-040-7

REVIEWED BY SAM STOVER

A lot of folks are (or consider themselves) "Apple bigots." I tend to prefer the label "OS X bigot," but after reading this book, I'm starting to convert. Having only used OS X for about three years, and never once with a classic application, I think of OS X as "*NIX that works" or "*NIX for the masses" or "*NIX that's so freaking sexy I can't believe it." Take your pick.

I knew that Snort, Nessus, and KisMAC worked just fine. I knew you could integrate a Mac into a predominantly Windows-biased environment via SMB support, Entourage (the Mac version of Outlook), and Open Directory (the Mac version of Active Directory). I knew I could compile my own source code manually, or use Fink and/or DarwinPorts for a more automated experience. I knew that most/all of these issues were in this book, and figured there wouldn't be much left for me to take home.

I knew nothing.

I didn't know that my Powerbook knows when it's being dropped, and reacts accordingly by parking the hard drive head. In fact, it does more than that—it keeps a running three-dimensional profile of its position in space and monitors G-forces to determine when it should panic. Turn your Powerbook sideways and you can read it like a book. It knows.

I didn't know that I could run CD-based Linux distributions from VirtualPC. Want to give the new Helix or Auditor ISO a spin? Drop it into VirtualPC, and away you go. Obviously, this isn't a long-term solution, but it will do if your Linux box dies (which never happens, right?).

And, most important, I really knew nothing about the great stuff that long-time Mac users take for granted, like Automator and AppleScript. Sure, I've messed around with AppleScript every now and then, but I always end up going back to Python or possibly shell scripting to get things done. The chapter on getting the most out of combining Automator, AppleScript, and any other language (Python, bash, Perl, C, etc.) totally rocked my world. Apple really goes out of their way to make it easy for the user to find the best way to get things done, and this book is truly the hackers' cookbook for putting it all together.

I totally enjoyed this book and would recommend it to anyone who has picked up a Mac and wants to run it through its paces. I would also recommend it to anyone contemplating getting a Mac, because I guarantee you'll end up making the purchase after you start salivating over what it can do.

My only true gripe with this book is that the editing really needed more attention. There weren't many chapters that didn't have at least one error, with the top scorer containing 19. This tells me that Syngress really rushed this book through to get it out to me, and, well, to you too. I suspect the 2nd edition will be a bit cleaner, but don't wait for it. If you want to learn what you can do with a Mac, you need this book—warts and all.

ESSENTIAL PHP SECURITY

Chris Shiflett

O'Reilly Media, 2006. 109 pages.
ISBN 0-596-00656-X

REVIEWED BY RIK FARROW

PHP is a very popular language for creating Web scripts, and one with a bad reputation for security. Shiflett argues that much of this reputation is undeserved, and the issues can be avoided by carefully following a set of principles when writing with PHP. I agree, to some degree.

This little book is an excellent way to learn about the security pitfalls one may encounter, and defend against, when writing Web scripts in any language. By following all of Shiflett's recommendations, you would avoid most, if not all, security vulnerabilities in PHP. If you use PHP, I highly recommend that you get this book, read it, and adhere to the suggestions found within it.

My only reservation is that I prefer languages that make it more difficult, if not impossible, to do the wrong thing. PHP lets you shoot yourself in the foot so many ways, that caution becomes the watchword.

UNDERSTANDING THE LINUX KERNEL, THIRD EDITION

Daniel P. Bovet and Marco Cesati

O'Reilly Media, 2006. 923 pages.
ISBN 0-596-00565-2

REVIEWED BY RIK FARROW

I really didn't want to understand the Linux kernel. Operating system programming is difficult, the Linux kernel is immense, and I have other things I must focus on. But when I found myself having to tinker with the kernel, or interested in learning about how modern memory management with 80x86 CPUs works, I needed a reference that could help me. And *Understanding the Linux Kernel* really worked for me.

Explaining a program that is millions of lines of code long is an enormous challenge. This book focuses on the operating system aspects of the kernel, as opposed to networking or device drivers (which are covered in other books). Given that focus, I feel that the authors have done an excellent job. They take the time to explain the issues clearly, and they provide cross-references to other areas of the book (and the kernel).

This was not the first Linux kernel book that I looked at, but it is the one I can recommend.

USENIX notes

USENIX MEMBER BENEFITS

Members of the USENIX Association receive the following benefits:

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

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

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

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

DISCOUNTS on registration fees for all USENIX conferences.

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

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

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

TO THE EDITOR

I just read in the December issue of *;login:* Peter Salus's "Ave Atque Vale" piece (pp. 65f.).

I shall miss Peter's column. I always enjoyed reading it and, on a few occasions, contributing to it.

Peter, my best wishes, Happy 2006, and thanks for all the memories . . .

Ted Dolotta

FUND TO ESTABLISH THE JOHN LIONS CHAIR IN OPERATING SYSTEMS AT THE UNIVERSITY OF NEW SOUTH WALES

USENIX announces the creation of a matching fund to establish the John Lions Chair in Operating Systems at the University of New South Wales.

The University of New South Wales is establishing an endowed Chair to recognize the enormous contribution made by John Lions to the world of computing. USENIX will match up to $250,000 in donations made through USENIX, now through December 31, 2006.

The Chair, to be called the John Lions Chair in Operating Systems, will enable an eminent academic to continue the John Lions tradition of insightful and inspirational teaching in operating systems. The creation of the Chair will perpetuate the John Lions name, and new generations of students will benefit from his legacy.

Donations can be made by sending a check, drawn on a U.S. bank and made out to the USENIX Association, to John Lions Fund, USENIX Association, 2560 Ninth St., Suite 215, Berkeley, CA 94710, or by making a donation online at https://db.usenix.org/cgi-bin/lionsfund/donation.cgi.

Your contribution may be tax-deductible as allowed by law under IRS Code Section 501(c)(3). Check with your tax advisor to determine whether your contribution is fully or partially tax-deductible.

conference reports

THANKS TO THE SUMMARIZERS

Alex Boster
Ali R. Butt
Stefan Büttcher
Laura Carriere
Marc Chiarini
Timothy Denehy
Matt Disney
Rik Farrow
Chaos Golubitsky
Kevin Greenan
Roman Valls Guimera
Abhishek Gupta
Jin Liang
KyoungSoo Park
Charles Perkins
Kristal Pollack
Florentina Popovici
Vijayan Prabhakaran
Josh Simon
Aameek Singh
Shafeeq Sinnamohideens
Matthew Wachs

CONTENTS OF SUMMARIES

LISA '05: 19th Large Installation System Administration Conference

San Diego, CA
December 4–9, 2005

Keynote Address

SCALING SEARCH BEYOND THE PUBLIC WEB

Qi Lu, Vice President of Engineering, Yahoo! Inc.

Summarized by Roman Valls Guimera

Qi Lu told us the challenges that Yahoo is facing to adapt their infrastructure to a new search level: personal and social search. These new forms of searching, as opposed to the traditional and well-known public search, are really difficult to scale.Take the example of Yahoo Mail: gigabytes of personal mail that cannot be lost under any circumstances. Hence, a personal space should provide high levels of fault tolerance, replication, and data-partitioning schemes. One thing is clear here: it's really complex to achieve all of them when you have a massive number of users. Without going into details, the Yahoo approach to solving those issues is cleverly simple, analogous to a biological cell: when the data cell grows, it divides and replicates itself throughout the system, keeping the properties we've seen before (redundancy and fault tolerance). Of course this process runs unattended, but it can be monitored in real time.

Without leaving the infrastructure point of view, we need to think about new ways to relate data from different users without losing performance or search quality. As del.icio.us does, a community of friends improves user search, and when there's a critical mass of users, we can improve the quality of results for a lot of users.

You can check Qi Lu's personal 360° Yahoo space for more info: http://360.yahoo.com/profile-dHFl7togcqomOrUGtvI-

CONFIGURATION MANAGEMENT WORKSHOP

Moderator: Paul Anderson

Summarized by Matt Disney

Based on a completely unscientific survey, the odds are high that you do not use a configuration management (confmgt) tool for managing systems. And if you do use a confmgt tool, you probably wrote it yourself (despite the availability of a small number of other confmgt tools) and that nobody else uses it. Why? What are you seeking in a confmgt tool? Are you ready for systematic management of your systems? Is it possible to create a confmgt tool that will be accepted by a majority of system administrators?

The confmgt community asked itself these questions, and many others, at the LISA '05 Configuration Management workshop. The unscientific survey mentioned above was taken at this year's workshop, a gathering of system administrators, researchers, and tool developers interested in the challenge of confmgt.

By some accounts, confmgt problems are characterized by the lack of popular adoption of confmgt tools. Some attendees, while not entirely unconcerned about adoption, are principally concerned with the underlying theory. They believe a solid foundation will yield tools that are attractive and, more important, correct according to certain metrics. Although the differing priorities of these two groups are not necessarily mutually exclusive, the workshop next year will likely be divided into the two categories of tools and theory.

One popular topic this year was the prospect of an OSI-like layered model for confmgt, which could facilitate the progress of tools as well as represent the boundaries between tools so that developers can focus on specific challenges. Such a model emerged from that discussion:

5. Service level goals. Example: .5 second response time for service X.
4. Invariants. Example: port numbers.
3. Services. Example: IMAP service.
2. Configurable elements. Examples: users, groups, resolvable hosts.
1. OS API. Examples: file contents, process memory state.

That definition led to an exploration of related issues, such as the general notion of feedback among the layers and the prospect of suboptimal restrictions potentially inherent in such a framework.

The challenge of federated confmgt was also covered. Existing tools do not reflect the complex political structure of large organizations. Some suggested methods for addressing this included combining abstraction and delegation, separation by infrastructure ownership, and the separation of functional administrative domains.

Andrea Westerinen of Cisco gave a presentation about the Common Information Model (CIM) and helped the attendees frame ways in which it might be used in the context of confmgt. Increased attention to a well-defined and popular, if not technically standard, model for describing system objects could be important and useful to confmgt in the future. Some tools already use CIM to some extent.

Tom Limoncelli also joined the workshop with a presentation from an outsider's perspective. Entitled "What I've Learned from Avoiding Configuration Management," his talk included some tips on how the core confmgt group could do a better job of connecting with the greater system administration community.

While some themes for the workshop recurred this year and will undoubtedly continue to arise on mailing lists and future workshops, there is traction on some new ideas and a continued interest in both confmgt tool development and theory. For detailed workshop notes and general information, see http://homepages.inf.ed.ac.uk/group/lssconf/.

ADVANCED TOPICS WORKSHOP

Moderator: Adam Moskowitz
Summarized by Josh Simon

In answer to the question of how much system administration has changed in the past year, attendees at the Advanced Topics Workshop (businesses, including consultants, outnumbering universities by about 4 to 1), the general consensus was "not much" on a professional level, although various compliance issues (local and federal regulations on IT, including SOX) have affected many. There's an expectation that compliance will take up more of our time and budget. Furthermore, automation is becoming a more obvious necessity to more people; folks are learning that scale, especially with clustering, simply requires it. We also agreed that the so-called soft problems, such as user interaction and customer service, will increase. We noted that many of us seem to be leaving system administration–type roles for networking, security, and, in at least one case, company executive (CIO), and others are losing interest in pure SA-type work.

Next was a quick around-the-room for tools we've seen. Many people said "wiki"; other tools included cfengine and other configuration management tools, Google Earth, IM clients within and across workgroups, Nagios and monitoring tools in general, Ruby, System Installer Suite (SIS), VMware and other virtual machine tools, VNC, ILO, other Lights-Out Management (LOM) software, and Xen. Others mentioned methodologies for development and testing, and code reviews, or hardware tools such as label makers for cables and power-consumption monitoring.

After the morning break, we discussed security and some of the hardware VPN solutions—using security incidents as catalysts for change on both an organizational and a technical level—and when to allow exceptions to your mandated security policy. This segued into a discussion on compliance; two of the points someone stressed were that (1) there's no established case law for SOX, so the auditors get to define what compliance is, and (2) making the collection of reports (or at least data) for the auditors should be both automated and reproducible. This is much like ISO 9000 all over again in some places.

Our next discussion was on scaling and automation. You should never say, "We can do this stuff with less staff," but, rather, "We can do more stuff with the same staff," lest you lose budget. It's essential to plan for growth at the beginning, because you'll rarely get the opportunity to go back and fix it. Many places run homegrown systems (especially configuration management and automation), because there's no off-the-shelf software that does everything we want, and such products as there are tend to have a steep learning curve or cost. Furthermore, getting different single-OS groups to agree on a multi-platform product is hard, and some people fear losing their jobs to automation, as opposed to getting rid of the mundane tasks to focus on the more challenging.

We next discussed personal productivity tools, ranging from changing OS ("Mac OS X"), to documentation (more wikis), to simple command-line tools (vi, grep, glimpse), books, PDA-specific applications, Web calendaring and sharing tools, sleeping pills (for ourselves, not our customers), unsubscribing from magazines and mailing lists, delegating to others, and even going to the gym.

After lunch, we briefly discussed autonomic computing, and how we as system administrators will interact with these self-modifying systems. In summary, it won't change what we do overnight, there'll be a cost/value tradeoff in outsourcing, and it'll probably be inappropriate for organizations with open-ended problem sets (such as research organizations and other places where the problems the systems are there to solve are not well contained or easily programmed).

Our next discussion was on professionalism and seniority. Some expressed concern that fewer institutions of higher education were offering courses specifically aimed towards system administrators, though others argued that as long as the candidates have thinking and problem-solving skills, that (plus experience as needed) was sufficient. Some concerns were raised about forthcoming regulation of IT personnel as an industry; between compliance issues such as HIPAA and SOX and the issues caused by not patching systems regularly, several people predict that regulation is coming sooner rather than later. The usual analogies were mentioned: are we doctors, are we janitors, or are we on a spectrum like the electricians/electrical engineers?

We next named tools we thought we'd need to learn in the next year or so. Answers included concepts from AJAX to ZFS, along with new operating systems for people (Mac OS X, Solaris 10, and Windows), and the usual suspects (documentation and knowledge management, process management, project planning, and virtualization).

We next discussed storage and efficiency, followed by network versus system administration. Some argue that netadmin is five to ten years behind sysadmin; others argue the reverse. The consensus seems to be somewhere in the middle and depends a lot on how you define your terms. For example, it's harder to have the Internet in your test lab, and routers and switches tend to make changes immediately rather than "when you reboot" or "when you send a signal to a process," as with systems. There was also a discussion about the relative security models for systems (data) and networks (keys to the kingdom).

Finally, we discussed physical plant issues (power, cooling, weight, and remote access) and social technologies. Most places are using some form of wiki or other documentation and collaboration software; many are using some form of instant-messaging client. One novel approach taken by some places is to use podcasting for information broadcasts.

Technical Sessions

VULNERABILITIES

Summarized by Roman Valls Guimera

■ ***GULP: A Unified Logging Architecture for Authentication Data***

Matt Selsky and Daniel Medina, Columbia University

GULP (Grand Unified Logging Project), a distributed approach to logging centralization, was born from the difficulties managing logging info at Columbia University: lots of servers saved different logs on local disk with unrelated informationmaking searches and correlation painful.

GULP aims to solve that problem by applying custom XML templates to the log files and extracting the interesting information from them. When validated, this data is stored on a MySQL database. Now the security team can construct queries to solve their problems: find stolen laptops, missing people, owners of infected machines; confirm stolen accounts; etc.

http://www.columbia.edu/acis/networks/advanced/gulp

■ ***Toward an Automated Vulnerability Comparison of Open Source IMAP Servers***

Chaos Golubitsky, Carnegie Mellon University

Awarded Best Student Paper!

Chaos Golubitsky presented a way to measure the attackability of code. That is, the relation between not commonly accessed code (which the standard user is not supposed to reach) and the code that is accessed under normal circumstances can be expressed in the following weighted formula:

attackability(codebase) =
$\Sigma f\ functionsweight(priv(f))\ weight(access(f))$

She applied this to UW IMAP, Cyrus-IMAP, and Courier IMAP. Using a code analysis tool called cflow (http://www.gnu.org/software/cflow/), she managed to split privileged code functions from the user-accessible ones and applied the above weighted formula.

The winner was Courier-IMAP, because it's designed to have a good privilege separation, while UW and Cyrus were tied.

If you want more information on this presentation, please see http://www.glassonion.org/projects/imap-attack/slides.pdf.

■ ***Fast User-Mode Rootkit Scanner for the Enterprise***

Ti-Min Wang and Doug Weck, Microsoft Research

Almost any enterprise or user who uses Microsoft Windows will eventually be infected by malware. Tools such as Ad-Aware perform quite well to wipe out the adware, trojans, and viruses that infect Windows machines.

Unfortunately, a new form of malware has appeared on the scene: ghostware. Ghostware evades any attempt to clean the system if you use current utilities. It does so by intercepting the API calls, which is just a step away from owning the whole OS. In other words, ghostware cannot be detected from inside the infected machine because it has kidnapped the OS itself, and that "ghost program" responds to the other programs by lying when asked for its presence.

The main concept behind Strider GhostBuster is the cross-view diff approach. Forget about the normal time diff (standard diff) we all know. Cross-view is a diff between what we see *inside* the infected machine, and what we see *outside* of it, so we can see the lie and the truth at the same time. We can then erase the ghost(s): it takes just seconds to see the liar.

http://research.microsoft.com/csm/strider

INVITED TALK

Summarized by Charles Perkins

■ ***Network Black Ops: Extracting Unexpected Functionality from Existing Networks***

Dan Kaminsky, DoxPara Research

Introduced as a "white hat" hacker, Dan Kaminsky presented practical and, in many cases, real-time exploits of network and cryptographic protocol weaknesses or unintended behaviors.

The MD5 hash function is broken both in theory and in practice. Dan demonstrated how an unsafe hash (which can be found in about 45 minutes) can be used to create two pages that hash to the same value. Key to the demonstration are that Web pages accept garbage and that you can present Web content programmatically.

Dan then described how for the receiver, keeping track of IP fragments turns a stateless protocol into a stateful one, and that IP fragmentation makes IDS harder. While attention to this has resolved many of the issues, timing attacks remain a problem. When an intrusion protection system operates upstream of a protected host, differences in fragment expiration timing between the host and the IDS can be exploited. A stream of fragments can be created by an attacker such that the IDS will construct a different packet from the fragments than the supposedly protected host will. Dan then described the temporal attack in detail.

Some firewalls, intrusion protection systems, and intrusion detection systems attempt to mask their existence. Dan listed a number of existing packet behaviors, responses, and contents that will reveal the existence of even "transparent" defenses; IPv6 will be even easier to fingerprint, due to encapsulation and reassembly issues.

Dan asserted that IPSes should not insert rules to ban traffic from hosts or networks after receiving invalid, excessive, or anomalous traffic. Simplistic rules will result in banning important services (such as root DNS servers), but, more important, through DNS poisoning an attacker could subvert your infrastructure and use your own rules against you.

Dan next described his project of probing the Internet DNS infrastructure, which he performed using copious bandwidth and novel techniques, including requesting the addresses of dynamically generated names satisfiable only by his DNS servers. Of 9 million nameservers scanned, 2.5 million do recursion; 230,000 forward to Bind8, which is a security problem; and 13,000 have the precise configuration that caused trouble for Google. Dan's resulting data set is quite large, and most interrelationships among nameservers are one hop deep (40,000 are connected graphs that are two hops deep—e.g., ask alice, get a request from bob).

As a result of his study, when the Sony Rootkit was exposed Dan already had a list of all the nameservers in the world and was able to use his tools to get an understanding of the breadth of the rootkit's distribution. It connects to connected.sonymusic.com, and that requires a DNS lookup which goes into the nameserver's cache. Dan performed a scan requesting connected.sonymusic.com of each of the nameservers without recursion. Nameservers that were able to respond with the IP address therefore had already been queried for it. Dan found 556,000 hosts with Sony-linked names. Dan acknowledged the margin of error in the survey due to time-to-live filters, some nodes recursing anyway, etc. Dan was interested to find indications that more nodes were trying to uninstall the rootkit (based on a different Sony domain name) than had gotten the rootkit in the first place.

Dan then showed graphs of the DNS server relationships, animations of router source-destination pairs, and a 65KB/sec video stream encapsulated in and delivered over DNS replies from an outside host.

CONFIGURATION MANAGEMENT THEORY

Summarized by Marc Chiarini

■ ***Configuration Tools: Working Together***

Paul Anderson and Edmund Smith, University of Edinburgh

Anderson took a look at the current state of system configuration tools, outlined why there are no clear successes, and made some simple suggestions for improving the technology. Configuration management needs to be viewed as a continuum, and we are just beginning to understand how to translate from high-level goals to the best low-level network and machine configurations to achieve those goals. This understanding will be facilitated by moving toward a common, generalized framework that represents distinct layers in the continuum and standard means for transforming data between layers.

Anderso focused on generic, semantically unaware operations for the deployment and management of configuration data. First of these are classing operations, which, as implemented in many current tools, cannot easily handle conflicts, such as those that may occur when multiple inheritance is in effect, and do not effectively address cross-cutting concerns. One way in which to shore up the first of these drawbacks is to implement powerful mechanisms for constraining subclasses and prioritizing inherited values. The second type of operation, aggregation, involves the (semi-)automatic creation of server configurations based on the needs of the client. The advantages of aggregation include a reduction both in the time required for manual specification and in the number of configuration errors. Sequencing and planning operations that enforce user-defined invariants in a declarative environment will be integral to any effective configuration tool. Finally, Anderson delivered a convincing argument that delegation and authorization should become multi-valued in order to make meaningful distinctions among required services.

We do not need a common system configuration lexicon or a strictly enforced operational architecture. Rather, we require a data structure for information exchange in the continuum and between independent tools, a "library" of generic operations for configuration data manipulation, and a simple interface for performing these operations. During the Q&A, someone asked about the lack of clear guiding theories, standards, and leaders in the configuration management space, and whether clarity is required to move forward. Anderson replied that arriving at high-level de facto standards will very likely happen naturally.

■ ***A Case Study in Configuration Management Deployment***

Narayan Desai, Rick Bradshaw, Scott Matott, Sandra Bittner, Susan Coghlan, Rémy Evard, Cory Lueninghoener, Ti Leggett, John-Paul Navarro, Gene Rackow, Craig Stacey and Tisha Stacey, Argonne National Library

Narayan presented a case study based on the rollout of the BCFG2 configuration management tool developed at ANL. The talk focused on the human aspects of CM tool adoption, which have not been extensively researched. Narayan began by stating that CM tools are not widely used and posited a reason: the upside is not well understood. The reason his division wanted to deploy a tool was because they were experiencing serious configuration problems (change propagation issues, patching, etc.) due to many years of ad hoc management. He described a two-year timeline of in-house events that began with the development of BCFG1 (and the eventual realization that it was a miserable failure) and culminated in the successful deployment of BCFG2. Narayan went on to present a retrospective analysis of key discussions within his group and how they arrived at their success.

Among the many issues addressed by the team, four stood out: tool fitness, group consensus, initial buy-in, and group dynamics. An effective approach was to give admins whitebox access, address their technical questions as quickly as possible, and take their input seriously. Not surprisingly, this also helped in reaching group consensus. Narayan pointed out that this consensus was built by increasing each person's familiarity with BCFG2 and implementing critical features. Communication was hard, since individual assessments of the tool embedded strong personal beliefs, and confidence in the tool varied over time.

The authors make four recommendations for helping to get a high-impact tool adopted at one's site. First, the tool needs an evangelist. This person consistently touts the prospective benefits of the tool and remains optimistic, but does not ignore complaints. Second, the audience must be shown a short-term payoff. Third, every effort must be made to address the concerns of the users (system administrators), whose instincts are usually correct. Try to incorporate in minor revisions those suggestions that make sense for the tool in the big picture. Lastly, try to keep everyone on the same page whenever possible. This may require sorcerer-like social skills.

Narayan freely admits that they had several factors working in their favor. Their group already believed that new configuration management techniques were needed; their strongest advocate was also their primary toolsmith; and they had an amicable and highly interactive group from the start. Your mileage may vary.

■ ***Reducing Downtime Due to System Maintenance and Upgrades***

Shaya Potter and Jason Nieh, Columbia University

Awarded Best Student Paper!

Shaya Potter mentioned a few well-known reasons why managing computer systems is hard work: software is buggy, hardware suffers from various faults, security can be compromised, and forcing downtime to upgrade or patch for any reason will usually annoy users. Common approaches to mitigating the impact of such events include the replication of services, OS-based isolation (such as FreeBSD Jail and Solaris Zones), and hardware virtualization using true VMMs like Xen and VMware. The first of these is only useful for shorter-term transactions such as Web requests and is very difficult to implement for longer-term stateful services such as user desktops. OS-based isolation suffers from serious limitations on the types of applications that can be run and may also require extensive non-modular kernel modifications. Lastly, the biggest drawback to heavyweight VMMs is that they still require tight coupling with the underlying OS, making migration costly and inflexible.

The AutoPod system provides secure, virtual private environments (PODs) in which a multitude of processes can execute normally with minor restrictions: a lightweight virtualization layer is installed on a host OS (currently only Linux) via kernel module. The virtualization layer intercepts and potentially rewrites all system-call communication between processes and the real kernel. AutoPod also features a facility to migrate whole PODs across machines or even virtual machines running different OS kernels.

When considering the initial design of AutoPod, the authors identified several hurdles. Most existing applications are not designed to migrate between computers, primarily because their running images are coupled to a specific instance of an OS. Clearly, it is not feasible to rewrite all applications of interest. In conjunction with virtual namespaces, this hurdle is overcome by briefly stopping all POD processes, recording important high-level state information for each process, translating into an efficient intermediate representation, transferring the process state and POD-specific info to a POD on an alternate machine, and restarting the processes where they left off. Another hurdle that needed to be cleared was the isolation of processes for security purposes. In particular, processes running with super privileges are rarely restricted by an underlying OS.

Questions were asked about transferring network state, especially long-lived connections. Potter responded that AutoPod can handle most situations. In some cases, however, such as when a Web server is migrated to another system with a running Web server, an external proxy must be in place to redirect requests to the correct virtual port. Another questioner asked how AutoPod compared to VMware's Vmotion, a migration facility for entire virtual machines. The difference is primarily in the speed with which a migration can be performed (especially for fully loaded VMs) and the limitations on kernel variations.

INVITED TALKS

■ *What Big Sites Can Learn from Little Sites*

Tom Limoncelli, Cibernet Corp.

Summarized by Alex Boster

Tom Limoncelli gave a relatively high-level talk about lessons he has learned turning about the IT department of a small site. He began with "why things aren't getting better." Using a pyramid diagram, Tom illustrated the earlier state of IT with a small number of "Good IT" sites at the top and a large number of "Bad IT" sites at the bottom. The state of IT today was illustrated with the same pyramid with a much larger base labeled "Really Bad IT." This was, he asserted, the result of the proliferation of small sites with "small sysadmin" attitude and abilities. However, he asserted that small sites are important because (1) they become big and (2) most big sites are really federations of small sites. These "broken" sites, he said, slack on the fundamentals.

Tom then asked, "Are best practices the solution?" He made an analogy between electricians versus electrical engineers: a construction project stops rather than do something "not up to code." He claimed that what's missing from this analogy in IT is an inspector who signs off on a project. The overall state of best practices is very fragmented: vendor's recommendations, SAGE and LISA publications and tutorials, CMM for sysadmins. Tom made special note of applying Maslow's "hierarchy of need" from the field of psychology to IT users as a good practice.

Finally, he presented his lessons from rebuilding a small site. The first lesson was that, at first, he only had time to deal with the basics, and, furthermore, "being there" crystallized what those basics were. Tom presented his experience in phases. Phase 0, acclimation, was where he learned who the players were and dealt with emergencies. In Phase 1, basic stability, the goals were to make the most important services reliable, establish backup procedures, learn the corporate purchasing process, and replace "accidents of history" design decisions. He emphasized the importance of the email service, meeting with users, a rudimentary documentation repository, and physically labeling everything he touched. Then in Phase 2, he could move on to establish basic IT applications: ticket tracking, network monitoring, documentation wiki, remote access, and automating backups.

Questioners asked about the size of the small company (100 users). In response to a query about backups, Tom stated that he started with rsync and Retrospect and has since moved to Bru apps. This was followed by a back-and-forth about fixing sites that, once stable, can be outsourced.

■ ***Building MIT's Data Center: An IT Perspective***

Garrett Wollman, Infrastructure Group, MIT Computer Science & AI Laboratory

Summarized by Charles Perkins

IT infrastructure was not considered early in the design process for the $300 million CSAIL building, which, at the time of the initial planning for the new building, contained four IT labs with independent IT staff.

Garrett outlined the differences among residential, commercial, and institutional architecture. Institutional architecture usually ends up being one-off construction. This new building had to shelter 1,000 people and meet the needs of 150 faculty, 50 frozen monkeys, four IT organizations, three lecture halls, and three wealthy donors, while reflecting the artistic vision of a well-known architect. Garrett and his team, the Net32 committee representing the computing labs, were brought into the project six years in, well after most of the physical parameters had been set and budget and space had been allocated.

The Net32 committee quickly determined that several misconceptions by management had resulted in a woefully inadequate allocation of space and access for IT infrastructure, including: (1) Computers are smaller and need less space than they used to, never mind that the computing clusters are growing by leaps and bounds. (2) Switches are $50 . . . manageability? What? Why? (3) You can just move the racks, switches, UPSes, power supplies, and all of the rest of the infrastructure over from the old building . . . except that the old system has to stay up and be usable while the move is taking place. (4) The building AC in the ceiling is good enough, and the IT staff doesn't need to monitor the HVAC independently of the facilities people . . . although in the past it has always been the IT staff telling the facilities people that the AC is broken and the computers are overheating. (5) Conventional quad power outlets in the floor will be fine.

The Net32 committee wanted all new racks with room-wide UPS power, under-floor AC with humidity control, power and network pre-wired, SNMP monitoring of the UPS and HVAC, and accessible cable-trays throughout the building for easy network changes.

A compromise was reached: some smaller spaces were coalesced into an irregular larger space and the group got all new racks, roomwide UPS, under-floor AC without humidity control (as the water pipes for chilling had not been designed into the building), power and network partially pre-wired, and separate proprietary UPS and HVAC monitoring.

Lessons learned: You can avoid a great deal of pain by getting involved in the planning early: make sure that management knows what IT costs, get closets, watch your wiring contractors like a hawk, get complete drawings, give complete requirements, think about where office infrastructure goes (printers, etc.), pre-wiring is great, play hardball with vendors, get freebies for naming things after vendors, hold coordination meetings after lunch instead of during lunch, and raised floors outside of machine rooms will make you sad.

CONFIGURATION MANAGEMENT PRACTICE

Summarized by Roman Valls Guimera

■ ***Integration of MacOS X Devices into a Centrally Managed UNIX Environment***

Anton Scultschik, ETH Zürich

Software management has always been complicated, especially on large, shared UNIX environments. Even with the help of package management tools, the admin has to deal with system diversity.

Template tree 2 helps to ease that diversity by providing modularized, self-isolated, meaningful configuration entities. This approach combined with SEPP package manager, which allows on-the-fly software provisioning (using automount), simplifies the daunting task of installing and updating software.

Template tree: http://isg.ee.ethz.ch/tools/tetre2/

SEPP: http://www.sepp.ee.ethz.ch/

■ ***RegColl: Centralized Registry Framework for Infrastructure System Management***

Brent ByungHoon Kang, Vikram Sharma, and Pratik Thanki, University of North Carolina

Managing large networks of Windows clients can be a daunting task: users tend to install their own programs (if they have the privileges to do so), and with those changes eventually comes breakage of their workstation.

Regcoll allows a system administrator to monitor Windows registry changes the same way a revision control system does, but with a real-time feature. If the user complains about a system malfunction, by using regcoll the system administration can revert the offending changes and go back to a state known to be fully operative.

In addition, regcoll can be used as a monitoring tool and a security analysis and auditing framework. To sum up, regcoll helps you keep your computer park free from unexpected failures caused by third-party software and/or user intervention.

■ ***Herding Cats: Managing a Mobile UNIX Platform***

Wout Mertens and Maarten Thibaut, Cisco Systems, Inc.

Users of laptops behave as if the laptops are their property; they will customize them, install programs, change default configurations, etc. As a result, the task of keeping those systems updated and clean becomes really difficult for the administrator or help-desk support staff.

Maarten and Wout solved the problem using Mac OS X as the preferred platform (while also supporting others). They use radmind plus their own additions to distribute software updates efficiently, with a pleasant interface on the user side, and, most important, safely (users need their laptops to always be operative). Additionally, they've made backup scripts to keep clients' data safe on a server and configured FileVault (a ciphered file system) properly to ensure users' privacy. (They've also used their own automated scripts to manage the process of issuing client SSL certificates!)

They deployed all these features quite successfully and, more important, usefully and painlessly.

radmind: http://sourceforge.net/projects/radmind

backup software: http://rsug.itd.umich.edu/software/radmind/contrib/LISA05/TacSync.tar.gz

INVITED TALKS

■ *Under 200: Applying IS Best Practices to Small Companies*

Strata R. Chalup, Virtual.Net, Inc.

Summarized by Alex Boster

Chalup's talk examined the question, "What of the big company practices can be applied to small companies?" As smaller companies grow to 50–70 people, staff moves on or the junior IS staff does not know how to handle the larger site.

She implored listeners to eschew the term "IT" in favor of "IS," since the ultimate goal of the job is to provide a service, not just the technology itself. This is part of an overall attitude adjustment required of most IT shop patterns.

Chalup's specific recommendations included: control access (widespread root access causes chaos); standardize and modularize everything you touch; have a standard plan for debugging issues; build a knowledge base; make full use of email lists; and use change control everywhere. She also discussed the importance of having written policies published on the intranet. She placed great emphasis on using a ticketing system with built-in metrics for all IS tasks. Proper ticketing system priorities were mentioned.

There was a question about what to do to keep users from walking up to your desk if you don't have a door to close. She stated that she's seen yellow police tape used in place of a door to good effect. A discussion then took place about ticketing systems. Chalup also noted the importance of learning how to get the information you need out of a user.

■ *What's a PKI, Why Would I Want One, and How Should It Be Designed?*

Radia Perlman, Sun Microsystems Laboratories

Summarized by Charles Perkins

Radia showed the usefulness of public key–based systems for authentication and authorization, as compared to symmetric key encryption. She described problems with current models (the monopoly of Verisign or oligarchy of self-signed certificates in browsers vs. the anarchy of PGP) and then outlined a model that avoids the concentration of trust inherent in the first two while addressing the scalability issues of the third.

Participants in encryption systems need to get their keys from somewhere. If each participant (n) required a shared secret for each other participant it might need to talk to, n^2 keys would need to be configured. In shared secret systems, such as Kerberos and Windows NT domains, the n^2 requirement is relaxed by using central servers to hold secret keys for participants (e.g., users' workstations and the services that they connect to). The only initial shared secret required is that which allows the participant to talk to the KDC or domain controller.

Public key encryption also requires key distribution, because participants need to get the public keys of their intended destinations from somewhere. The certificate authority is the equivalent of the KDC or domain controller in a Public Key Infrastructure. A certificate authority has significant advantages over its private key equivalent: a KDC is less secure, contains a highly sensitive database, must be online, and must be replicated. The CA, on the other hand, may be offline. Revocation makes CAs harder to implement, however.

Radia asked, "What can I do with PKI?" and answered: establish secure conversation without online introduction service, send encrypted email, send signed email widely, distribute signed content and single sign-on to mutually distrustful sites. Radia doesn't believe we can avoid names in a PKI.

Radia then explored how PKI with access control lists can create a scalable system for revocable granting of permission to resources. The system allows resources to require membership in groups, with the groups nested in hierarchies. On an access attempt the group server will (1) sign a certificate vouching that an identity is a member of that group or (2) require the client to walk up and/or down the tree acquiring proof of membership in sub- and/or super-groups in order to prove membership in the group the resource requires. Proven membership certificates, which may be timestamped, may be cached by the client, and revocation is provided for by allowing the resource requiring the certificate to accept only recently minted certificates.

There are three models of PKI widely used today:

1. The Monopoly model, whereby Verisign signs all the certificates, which is easy, understandable, vulnerable to monopoly pricing, introduces vulnerabilities getting the certificate from a remote organization, is dependent on Verisign's key never changing, and requires the security of the world to depend on the honesty and competence of one organization forever.

2. The Oligarchy model, used by Web browsers, wherein 80 or so self-signed certificates are implicitly trusted, which allows users to add to or delete from the set of certificates, eliminates monopoly pricing, is less secure (any of the 80 keys may be compromised), and makes it impractical to check the trust anchors.

3. The Anarchy model, used by PGP, wherein anyone may sign a certificate for anyone else; users consciously configure starting keys; proof of identity is inferred from traversing chains of trust, which does not scale as the number of certificates grows and it becomes computationally difficult to find a path; there is no practical way to tell if a path should be trusted; and there is too much work and too many decisions for the user.

Trust in a CA should not be binary; a CA should only be trusted for certain things, and a name-based system seems to make sense.

Radia proposes a bottom-up hierarchical model where each arc in a name tree has a parent cert (up) and child certs (down). The namespace has a CA for each node and lookups don't start at the root—they start at the member's group CA and go up to the least common ancestor. Cross-links are allowed, and this system allows organizations to choose top-level cross-link services. Importantly, the organization can revoke the up-certificate to one cross-linking service and select another if it is unhappy with the service. In intranets, no outside organization is required, inside security is controlled from the inside, and no single compromised key requires massive reconfiguration. A uniform PKI policy across all participants is not required.

Asked why we don't have elliptic curves in all this stuff, Radia replied that the patent situation around elliptic curves is unclear. Also, using the RSI private key is slow, but using the public key is fast. Verifying a certificate using RSI might actually be faster than using elliptic curves.

A concern was raised that fast factoring might make the PKI infrastructure obsolete. Radia conceded that it could happen. However, a fundamental concept of cryptography is to pick a problem mathematicians have been working on for a long time, meaning, hopefully, that it is a hard problem. She predicted that quantum crypto hardware might be able to factor the number "15" in a few years!

She was asked if the bottom-up PKI architecture described in her talk was in the book she co-authored (*Network Security: Private Communication in a Public World*, 2nd ed.). She replied that it was.

■ ***Modern Trends in UNIX and Linux Infrastructure Management***

Andrew Cowie, Operational Dynamics

Summarized by Laura Carriere

Andrew Cowie delivered a thought-provoking session, postulating that the profession of system administration continues to follow numerous divergent paths when solving new problems and does not appear to be converging on a set of standard solutions to these problems. He stated that it was unusual for an industry to fail to converge on standards by this stage in its development.

Cowie observed that system administrators are being asked to solve increasingly complex problems with static or reduced resources and that there are frequently two schools of thought on how to solve these problems. Our profession seems to cycle between the options and often chooses to apply the wrong solution to a given situation.

Cowie gave a number of examples to support his hypothesis. He first addressed the issue of when to scale vertically (using a few powerful systems) and when to scale horizontally (using many small systems), stating there's no consensus within the industry on the criteria to be used when making such decisions. The end result is that many companies choose the wrong solution.

He discussed the related issue of server consolidation versus increasing complexity. A reasonable solution to limited floor space is to consolidate services onto a single UNIX system. However, a conflicting trend is to isolate services on separate servers, which simplifies the administration required to load, deploy, tune, and ghost. The end result is that organizations may be reducing or increasing the number of systems, or, possibly, following both trends at once.

The issue of using multiple blade servers versus moving to virtualization is a similar problem. Multiple small boxes provide plenty of resources but are a management nightmare. Putting multiple virtual systems on one powerful box works well until the virtual systems overuse one resource, thereby creating a bottleneck (which is frequently the I/O system).

Additional conflicting themes discussed by Cowie included Web interfaces without a command line interface, which make it impossible to write management scripts. The irony is that Web interfaces are designed to simplify management but ultimately prevent the best mechanism we have to do that—automation.

Cowie went on to consider desktop deployment. Although vendors have developed tools such as JumpStart and KickStart to automate installation, maintenance is difficult, and vendors are not providing solutions for that, the only exception being RedHat Satellite Servers.

Configuration management (CM) also has two competing approaches—convergence and congruence. Cowie cited cfengine as an example of convergent configuration management, where desired lines are added to the configuration files if they are missing. With a congruent CM system, entire configuration files are regenerated. The industry currently has no guidelines to determine which solution best fits a situation. Cowie briefly discussed the idea of encapsulation, an OO approach to CM that allows the administrator to specify policy (i.e., SwitchToPHP) and let the software do the required configuration.

Cowie concluded with a warning that Grid computing is coming and will radically change the industry. Again there are two competing approaches, a tightly linked cluster with shared memory, such as an SGI predicting the weather, and an aggregate of individually maintained systems, such as the systems that comprise SETI@home. He expressed his concern that Grid computing will drive the development of effective management tools and that this will threaten the livelihood of the junior sysadmin who enjoys repetitive tasks. During the Q&A period, Cowie expanded on this, saying that change is good and more evolutionary solutions free us to do more interesting work.

■ ***Incident Command for IT: What We Can Learn from the Fire Department***

Brent Chapman, Great Circle Associates

Summarized by Marc Chiarini

Brent Chapman, a California Civil Air Patrol incident commander and local fire department volunteer, gave a talk about applying the principles of incident command in IT departments. An IC system is used by various public safety organizations (Coast Guard, local fire and police departments, FEMA) to coordinate themselves and communicate with other agencies in an efficient manner during major unplanned incidents. Often, many different individuals and organizations are involved, and there needs to be a structure to determine who is in charge and exactly what needs to be done. Brent gave several real-world examples (car accident, raging wildfires, total data-center power outage) to help the listeners understand the scale of situations that occur. He also stressed that IC can be applied to nonemergency situations, such as facility moves and major system/network upgrades.

A typical ICS follows nine key principles:

1. Maintain a modular and scalable organizational structure. There may be five "sections" or groups responsible for different tasks: a Command Section with a capable IC (incident commander) must always be available; a mandatory Operations Section executes plans to achieve command objectives and worries about the now; a Planning/Status Section collects and evaluates information needed to prepare action plans and tracks progress; a Logistics Section is responsible for obtaining all resources required to deal with an incident; an Admin/Finance Section, necessary for the largest and longest-running incidents, will track costs and administer procurements.

2. Maintain a manageable span of control. Limit section sizes and grow the hierarchy as necessary.

3. Maintain unity of command. A strict tree structure (each person has only one boss) facilitates communication and reduces freelancing.

4. Transfers of responsibility must be explicit.

5. Maintain clear, expedited communication. Use no shorthand or codes and speak directly to resources when possible.

6. Keep action plans consolidated. Command maintains the top-level (preferably written) plan for the current operational period (hour, shift, day, etc.).

7. Manage by objective. Tell subordinates what to do, not how to do it.

8. Maintain comprehensive resource management. Track all assets and personnel. Establish a sign-in process and "report-to" site.

9. Use designated incident facilities. Must always identify a Command Post (CP).

Brent went on to give a compelling example of using ICS in the IT world. He presented the timeline of an IC response to a data center failure, including the creation of subgroups in Operations, an explicit transfer of responsibility, assignment of a liaison, and ongoing organizational restructuring.

The talk ended with some important tips for implementing ICS effectively: initiate incident response as soon as possible, use ICS as a toolbox, keep things simple, and practice all the time with routine and pre-planned events.

More info can be found at http://www.greatcircle.com/blog.

During the Q&A, David Blank-Edelman asked how people stay updated in the field. Brent recommended wikis, bulletin boards, top-down word-of-mouth, and whiteboards and Post-Its for areas without power. John Millard mentioned having standardized ICS kits ready for immediate use. I asked whether there are any standard metrics for judging the efficiency of a response. Brent replied that a good way to do this is follow the paper trail and do not get emotional when reviewing performance.

THEORY

Summarized by Marc Chiarini

■ ***Toward a Cost Model for System Administration***

Alva Couch, Ning Wu and Hengky Susanto, Tufts University

Awarded Best Paper!

Alva Couch presented a novel first step in approximating the costs of system administration. System administration incurs both tangible and intangible costs; the former, as described in Patterson's cost model (LISA '02), tend to result in financial or productivity losses. The latter are much more difficult to measure, but an appropriate model would allow organizations to assess and improve their current processes. To arrive at such a model, Couch's team combined queuing theory, risk analysis, and simulation with an analysis of 400+ days of request ticket data (obtained from Tufts' EECS support group).

At first glance, measuring time spent waiting seems like a daunting task. It is, however, possible to view it as a function of certain parameters (request arrival rate, service rate, number of workers, etc.). This naturally leads one into queuing theory. Couch demonstrated how viewing request arrivals from the appropriate height, removing outliers from the ticket data, and adjusting for daily work cycles can ultimately reveal Poisson processes. To estimate the expected service rate, it is possible to apply risk analysis to the decision trees used by system administrators to resolve requests.

After examining real data, the authors chose to simulate a trouble-ticketing environment with non-product behaviors. As Couch explained, the motivation behind this was to account for phenomena that cannot be analyzed effectively via queuing theory. The team found that running a system near absolute capacity will cause chaotic and utterly unpredictable increases in service wait times. The important point is that in order to be useful, the new cost model cannot be applied to networks on the edge of steady state. When the capacity to resolve standard requests comfortably exceeds load, however, estimating the cost of administrative practice by indirect methods such as risk analysis can be made much more accurate.

Some interesting points were clarified during the Q&A session. Mark Burgess asked whether the data had been overly massaged. Couch responded that it was within reasonable limits for obtaining a decent model of steady-state behavior and extracting inhomogeneous trends. On the service side, non-product (realistic) systems could be approximated by introducing interruptions into an ideal system and analyzed via perturbation theory. When Couch mentioned that the study of realistic systems suffered from lack of data, someone suggested that SAGE or LOPSA could volunteer data sets. Couch was ecstatic about this prospect and stressed the importance of anonymized submissions.

■ ***Voluntary Cooperation in Pervasive Computing Services***

Mark Burgess and Kyrre Begnum, Oslo University College

Mark Burgess spoke of a worldwide move toward pervasive computing, with multiple decentralized services provided by individual actors implementing autonomous policies. The authors believe strongly that the sysadmin tasks of tomorrow must integrate ideas about this explosion of autonomy. Mark's "promise theory" provides a different risk model for service provision. Whereas modern services are driven by demand and the server and client trust each other almost implicitly, this new approach takes an individualistic view of how an actor protects its own resources and acquires those it needs. In a future with very limited resources, client demand will no longer be the governing factor; clients and servers will have to cooperate voluntarily to keep things humming. The focus of every transaction in promise theory is on minimizing the risk of the involved parties.

The authors demonstrate the strengths of their approach by implementing a proof-of-concept voluntary RPC mechanism in cfengine. They observe that cooperative agreements now become the key to eliminating unpredictability. As opposed to traditional services, the protocol does not enforce reliability. Actors learn over time the probability that their peers will deliver on their promises, and then fall into stable patterns. The protocol itself was analyzed and verified for correctness using Maude, a programming language for reasoning about temporal logic and proving certain properties. Combined with the POC, this analysis revealed several limitations: the mechanism for initial agreement is made out-of-band; there is no current means of reprisal for uncooperative actors;

and the protocol does not easily provide a HA environment.

An interesting question was asked by Alva Couch about the quandary of having to put a file system into the pervasive network. Mark answered that there does need to be an addressable superblock out there.

INVITED TALK

■ ***Automatic PC Desktop Management with Virtualization Technology***

Monica Lam, Stanford University/ SkyBlue Technologies

Summarized by Alex Boster

Monica Lam's talk was about a new x86 PC virtualization system in its pre-alpha stage (details are available on itCasting.org). She started by describing their team's motivation: to allow end users to turn over management of their desktops to professionals by breaking old assumptions. Their solution, called itPlayer, solves issues of mobility, management, and security.

The itPlayer software is built on a small, bootable Linux system and VMware Player. itPlayer is placed on any bootable storage device, such as an SD card, micro drive, or iPod. The whole VM resides at a known place on the network but is cached locally—similar to the way virtual memory works. Changes can be written back over the network, giving the user an online backup of the system. The system can also run in disconnected mode, provided the local storage device is large enough to hold the entire image (e.g., a hard drive, but probably not an SD card).

According to Lam, itPlayer is fast if the local cache is good; is as easy to use as a television ("just turn it on"); cannot be lost—just grab a new copy from the network; has disconnected operation; and has low virtualization overhead. It's limited by what Linux device drivers are available, having no virtualization of advanced graphics, and the fact that the desktop must be USB-bootable.

This new system results in new assumptions: that the state of the computer is always backed up, and that hardware is interchangeable. Lam then compared this system to other ways of doing desktop management: stand-alone PCs, mainframes, and thin clients.

Lam addressed the issue of updates by pointing out that the image provider (an IT department, for example) can update an image. Upon reboot, the users of that image will simply swap in the new image blocks from the network and run the new image. She said that currently desktop customization is easy, and standardization is hard. Lam asserted that the itPlayer system reverses that arrangement.

The talk ended with a demo of itPlayer. A Windows XP SP1 image was booted, the backing store image was replaced with an updated image running SP2, and the itPlayer restarted into SP2 upon reboot.

Questions focused on licensing issues, which Lam addressed mostly by pointing out that there is lots of freely available software. This was followed by a discussion of practical difficulties in customizing itPlayer environments per user in a corporate setting.

NETWORK VISUALIZATION

Summarized by Charles Perkins

■ ***Visualizing NetFlows for Security at Line Speed: The SIFT Tool Suite***

William Yurcik, NCSA

William Yurcik demonstrated Security Incident Fusion Tools, which leverages human ability to discern patterns in visual displays.

CANINE provides NetFlows interoperability by converting and anonymizing NetFlow events from many commercial formats. It performs multi-dimensional anonymization of fields to facilitate secure data sharing and it reads both Cisco unidirectional NetFlows and Argus bi-directional NetFlows (see http://security.ncsa.uiuc.edu/distribution/CanineDownLoad.html).

NVisionIP shows the user the state of the IP address space, with default configuration for a class-B range, in a single screen. Activity is displayed by address in a pixilated matrix, with subnets across the top and station addresses down the side. It provides for drilling down to graphical views of activity on subnets, sets of hosts, and a single machine (http://security.ncsa.uiuc.edu/distribution/NVisionIPDownLoad.html).

VisFlowConnect-IP shows who is connected to whom on the network in a parallel axis chart with an inside view and an inside/outside view of network traffic. One-to-many, many-to-one, scanning activity, and unusual connection behavior can be observed in real time on the parallel-axis views, and both drill-down functionality and a filter language are provided: http://security.ncsa.uiuc.edu/distribution/VisFlowConnectDownLoad.html).

Yurick completed his talk by pointing interested parties to the VizSEC community at http://www.ncassr.org/projects/sift/vizsec/ and http://www.ncassr.org/projects/sift/.

Question: How do the tools scale above a class B network? Answer: One would open different windows, one for each class B. Question: How much trouble is it to make the software handle different data sources? Answer: It takes hard work, some "bribing," and a clear understanding of the protocols and formats. Also, the software is going open source.

■ ***Interactive Traffic Analysis and Visualization with Wisconsin Netpy***

Cristian Estan and Garret Magin, University of Wisconsin, Madison

Cristian Estan described adding interactive drill-down and flexible analysis to real-time traffic monitoring of network traffic. The Hierarchical Heavy Hitter approach reports traffic that exceeds a threshold and can use subnets, ports, and routing table prefixes, as well as user-defined groupings as hierarchies with ACL-like rules.

Cristian demonstrated the advantage of real-time interactive drill-down to determine the cause of anomalous network behavior, with "heatmap" charts of sender/receiver pairs making network traffic hotspots visually apparent.

Analysis may be conducted through text, time series plots, bar charts, and bi-dimensional reports across hierarchies. The user can select the time interval, bytes, packets or flows, and filters to be applied. The software handles router sampling and can use a database or files.

The software will be open source and more information can be found at the Netpy home page, http://wail.cs.wisc.edu/netpy/.

■ ***NetViewer: A Network Traffic Visualization and Analysis Tool***

Seong Soo Kim and A.L. Narasimha Reddy, Texas A&M University

Seong Soo Kim presented the paper, demonstrating, producing, and analyzing video from captured packet header information in order to detect DoS, DDoS, and worm behavior in the network. He asserted that DDoS flows look like any other flow and require aggregate analysis.

NetViewer aggregates seconds of traffic header information in a concise data structure in order to compare sequential frames with image-processing algorithms. Variations in pixel intensity and movement indicate DoS, DDoS, and worms. He displayed representative sequences and showed characteristic visual patterns produced by network attacks.

NetViewer has been run on several university and ISP connections, and they found things that snort did not. NetViewer is not looking for known attacks, is generic, is real-time with latencies of a few seconds, is simple enough to be implemented inline, and has a Windows and a UNIX GUI.

Email seongsoo1.kim@samsung.com or reddy@ece.tamu.edu for more information.

INVITED TALKS

■ ***Internet Counter-Intelligence: Offense and Defense***

Lance Cottrell, Founder, President, and Chief Scientist, Anonymizer, Inc.

Summarized by Alex Boster

Lance Cottrell began by describing his company, Anonymizer, Inc., and their history, products, and services. He described some of the basic problems in intelligence analysis, pointing out that simple log file analysis is still the most common method. He also noted that tech companies are far from the only ones doing this.

However, whenever you have exposed IP addresses, Cottrell claims, you are leaking information about your business out to the world. Even if you engage in IP blocking (which people can see you do) or IP spoofing (having different versions of Web sites for different visitors), you are still "hemorrhaging" data out. For example, competitors can read your whitepapers, product listings, press releases, and so forth to discover your business and research profile.

Cottrell then cited a number of examples: that prior art is a huge intellectual property issue, and if you have visited a competitor's Web site, you may be exposed; Cisco employees who surfed to a competitor's Web site were presented with a job offer; European hackers who would launch automatic DDoS attacks against visitors to their Web site who were seen to be running Microsoft IE and coming from a Washington, D.C., IP address.

One solution to conducting this kind of intelligence analysis is to anonymize traffic by routing it through another network and rewriting the headers. However, Cottrell pointed out, it is tricky to do this without introducing inconsistencies (e.g., traffic made to look as though it originated in Hong Kong, but the time zone was PST). Further examples of intelligence analysis were given: airlines scraping all their competitors' fares; retailers profiling users both on their buying habits and on their geographical location.

Next, Cottrell moved on to examples of counter-intelligence. Less aggressive companies can monitor their traffic closely, for example, for a 3 sigma change in interest in whitepapers. Companies in a bidding war might bug the investor section of their Web site.

A questioner asked if companies block Anonymizer. The answer was, yes, they try, but they cannot do so effectively, due to Anonymizer's large, scattered, frequently changing IP address space. Another question was about ethical boundaries of Anonymizer. Cottrell said that they try to detect and reject attacks, spam, IP floods, and the like. Their policy, he said, was that they would block activities that are illegal in the U.S.—however, all other uses by enterprises were permitted after a committee review. He also stressed the importance of Anonymizer ensuring privacy by never, ever keeping logs. Other questions dealt with: issues of ISP trust (Anonymizer must engage in long discussions when buying IP blocks from a new ISP); working with law enforcement (Cottrell said they are usually respectful and that Anonymizer cooperates where appropriate); how many companies engage in dynamic customer profiling, for example, offering different prices to different people (he said it was "very widely used" and most big sites did it).

■ ***Preventing Child Neglect in DNSSECbis Using Lookaside Validation (DLV)***

Paul Vixie, Internet Systems Consortium, Inc.

Summarized by Chaos Golubitsky

In this talk, Paul Vixie proposed DNSSEC Lookaside Validation (DLV) as a means of overcoming the road blocks which currently prevent deployment of Secure DNS. He justified the need for such a solution with some history. First deployed in 1987, DNS was not designed to enable authentication of name data. The IETF has been working on Secure DNS since 1994, but it has still not been deployed at any sites.

The current Secure DNS proposal, DNSSECbis, works by introducing a new set of DNS RR types, most of which are used by a zone to enable authentication of its own DNS data using public key cryptography.

DNS is hierarchical by design: just as DNS validators hard-code the locations of the root nameservers, DNSSECbis validators will hard-code the root nameserver DNSKEY. The effect is that no zone can deploy DNSSECbis until the zone's parent has deployed it. In particular, DNSSECbis cannot be meaningfully deployed until it is present in the root and .com zones. Since parties higher on the DNS tree see more of the costs of DNSSECbis and fewer of the benefits, this may never happen.

DLV is designed to allow zones to deploy Secure DNS even if their parents have not deployed it. It introduces a DLV resource record, which is functionally similar to the DS (Delegation Signer) record. It also introduces DLV namespaces, zones which have offered to serve DLV data for all or part of the DNS space. A validator looking for Secure DNS data for a given zone must first look for a DS record at the zone's parent. If none is found, the validator may then look for entries within any DLV namespaces it knows. For example, if dlv.isc.org is a DLV namespace and there is no DS entry for vix.com, then a DLV entry can be stored at vix.com.dlv.isc.org. Therefore, vix.com can deploy DNSSECbis even if none of its parents have done so.

DLV is intended as a temporary solution, which should be shut down either when deployment of DNSSECbis reaches critical levels or when it becomes clear that DNSSECbis will fail. As a result, the DLV namespace should be introduced by a public benefit corporation which uses a cost-based fee structure. Vixie identified his employer, ISC, as committed to this model. BIND 9.4.0, to be released soon, will contain support for DLV, and ISC will operate a DLV registry using BIND9. For further information, search for "ieice vixie dlv" to find Vixie's 2004 paper introducing DLV.

Attendees asked how individuals can convince their employers to roll out DLV, and how ISC plans to authenticate DLV registrants. First, the announcement of BIND 9.4.0 will announce DLV, since many sites will deploy as soon as possible. Second, Vixie is compiling a set of marketing whitepapers to advertise DLV. Authentication of registrants involves liability risk for ISC; the exact mechanism has not been determined. Possibilities include: initially registering DLV records only for people with whom ISC has an existing business relationship; charging a fee to cover the cost of verifying registrants' identities; obtaining identity information from existing registrars; or using a web-of-trust scheme, starting with existing ISC business partners.

PLENARY SESSION

■ ***Picking Locks with Cryptography***

Matt Blaze, University of Pennsylvania

Summarized by Alex Boster

Matt Blaze did not, in fact, give a talk on lock picking using cryptanalysis. Instead, he talked about his more recent research into wiretap eavesdropping and applying computer and network security techniques to wiretap systems. Blaze pointed out that there are important legal implications to vulnerabilities in wiretap systems that might cast doubt on the reliability of the tap.

Blaze then described the two basic types of wiretaps: pen registers, which record the numbers dialed but not the audio, and full audio taps, which have greater legal restrictions. A description of basic telephone and wiretap terminology and functions followed. Blaze's research focused not on the many ways one could do wiretaps but, rather, on how law enforcement agencies actually do them.

Various types of wiretap equipment were then presented. Blaze pointed out that wiretaps do not perform exactly the same as the phone company's central office (CO) equipment—and that opens up some vulnerabilities. He was able to reverse-engineer the signals used by wiretap systems. Taking advantage of differences in tolerance (the phone tap equipment is more sensitive to the on-hook signal than the actual CO equipment), he was able to play two recordings of the same phone conversation: a short one where the wiretap had been fooled into halt-ing recording, and the full version recorded directly from the line.

Questioners asked if audio and call detail logs are correlated. Blaze replied that they were not standard operating procedure. Blaze was also asked about parallels between the talk he gave about wiretaps and his research on lock picking and cryptography. He said that parallels included understanding the limits to mechanical devices, noting that we tend to upgrade them to electronic devices, and that reducing the problem to software might not be a good idea.

INVITED TALKS

■ ***How Sysadmins Can Protect Free Speech and Privacy on the Electronic Frontier***

Kevin Bankston, Electronic Frontier Foundation Staff Attorney

Summarized by Rik Farrow

Bankston began with a history of U.S. laws relating to wiretapping. Until a Supreme Court decision in 1967, U.S. citizens could expect almost no privacy from surveillance via taps installed on telephone lines. The Wiretap Act of 1968 placed federal law in line with the court decision, but the law and later court decisions still permitted pen-traps, collection of call log information. In 1986, the Electronic Communications Privacy Act attempted to modernize the law. In 1996, CALEA forced telephone providers to include mechanisms for install taps and/or pen-traps via phone switches, in support of law enforcement armed with judicial permissions.

The Patriot Act changed much of the landscape, making it possible for a tap to be installed and the target never informed of it, unlike earlier laws. NSLs (National Security Letters) issued directly by the FBI can also not be challenged or made public, ever, and an article in the *Washington Post* suggests that these letters are being used for surveillance of domestic opposition to the current administration.

What can sysadmins do to protect the privacy of their users? Bankston had a series of suggestions:

- Minimize logfiles; storing logs forever is more likely to cause problems than to help you.
- Have a clear policy about how long you keep log files, and follow it.
- Negotiate to keep the government software and hardware out; you don't have to redesign your networks—yet.
- Lobby for legal challenges (you can call a lawyer).
- Give notice whenever possible.

If you are asked to do surveillance, do check on the law. Contact EFF, even if you get a supersecret order, or you can go to a lawyer (ask your boss). You often do have the power to inform people if their info has been subpoenaed. Yahoo has done this.

You can also join the EFF (eff.org).

■ ***Wireless Security***

Michael H. Warfield, Internet Security Systems, Inc.

Summarized by Chaos Golubitsky

Michael Warfield provided an overview of the current state of wireless security. The focus of the talk was classification of methods of attacking networks, outcomes of successful attacks, and available means of protection.

While war driving for insecure access points is the best-known exploit of wireless networks, others are also in use. Attackers can run their own APs, either to opportunistically snoop on any machine with an open wireless configuration (inverse war driving) or with a specifically chosen SSID to mirror a legitimate network (evil twin attack). In a hotspot battle, an attacker launches a denial of service attack on a specific wireless network by interfering with the channel used by that network.

Once a network has been exploited, the attacker's target may be the network itself (simple bandwidth theft, denial of service), the contents of machines using the network (information theft, extortion), or the use of the network to anonymize illegal activity (spam, visiting illegal Web sites). Warfield noted that arp cache poisoning can be used to redirect interesting traffic from adjacent wired networks, and that owners of wireless networks may face liability or reputation problems due to illegal activity on their networks.

The last portion of the talk focused on the benefits and shortcomings of common wireless network defenses. Warfield stated that MAC address control is not very valuable—the administrative overhead of maintaining tables is high, and guessing a valid address can be trivial. Since tools such as Kismet can easily probe silent access points, turning off SSID broadcasting is not a good security measure either. In general, WPA should be preferred to WEP. However, both protocols have a history of weak implementations, and a modern WEP network may require more traffic in order to break a key than a broken WPA network. Virtual Private Networks should be used, but they provide no protection against poorly configured legitimate machines. To the extent possible, wireless networks should be protected against physical threats—for instance, by placing APs in the interior of a building rather than near the outside.

Warfield repeatedly made the point that it is useful to classify attacks according to whether they are opportunistic or targeted. Evil twin attacks and hotspot battles necessarily explicitly target the network being attacked, while others may be indiscriminate attacks against any nearby network, or may even be accidents. Similarly, weak countermeasures may have value because they prove intent. WEP is easy to crack, but it cannot be cracked accidentally, so an intruder on a WEP-protected network can be assumed to be launching a deliberate attack on that network.

The full slides for the presentation are available at http://www.wittsend.com/mhw/2005/Wireless-Security-LISA.

ACCESS CONTROL

Summarized by Chaos Golubitsky

■ *Towards a Deep-Packet-Filter Toolkit for Securing Legacy Resources*

James Deverick and Phil Kearns, The College of William and Mary

The goal of this project is to provide a toolkit for authenticating access to non-secured legacy resources through a firewall. The toolkit should consist of a central library of solutions which can secure many network services with minimal per-service coding, and should not require that the protected software be altered in any way. Jim Deverick presented a proof-of-concept implementation which used the Linux netfilter packet filter to authenticate NFS mount and umount requests and LPR printing.

Both services are wrapped using a netfilter rule set which captures packets representing new requests and holds these packets while they are examined by user-space code on the firewall. The firewall code performs an external authentication step, generally by contacting a daemon on the client system with a challenge/response request. If authentication is successful, the connection request is forwarded to the server. If not, the toolkit cleans up any loose TCP connections created on the server.

As implemented, the toolkit secures only NFS mount and umount requests and initial LPR connections. No authentication is required in order to submit packets to a connection already in progress, and, in the NFS case, no authentication is required in order to perform NFS operations on a mounted file system. Since netfilter operates on TCP packets, authorization could be provided at the granularity of source and destination IP/port pairs, although the current implementation authorizes the entire source host to send packets to the target port. In the future, the authors hope to improve the implementation so that wrappers can be added and modified more easily.

■ *Administering Access Control in Dynamic Coalitions*

Rakesh Bobba and Himanshu Khurana, NCSA and University of Illinois at Urbana-Champaign; Serban Gavrila, VDG Inc.; Virgil Gligor and Radostina Koleva, University of Maryland

Radostina Koleva introduced a prototype of a set of tools for administering dynamic coalitions. A dynamic coalition is a set of independent organizations (domains) that share resources for use in a joint project. The example given was that of a pharmaceutical company, an FDA review board, and a research hospital working together on a new drug. For a coalition to form, each domain must have an incentive to bring private resources to the table. A flexible framework is needed to control other domains' access to these resources. The coalition may create shared resources, which will be owned and administered by consensus among domains. In addition, new domains may join an existing dynamic coalition for certain projects, and previous member domains may leave.

Negotiating a coherent access policy is a challenge, as is implementing a formal policy specification. The tool set presented here can help negotiate coalition policies in a semi-automated fashion, allow consensus-based administration of joint resources, distribute and revoke privileges efficiently, and provide each member organization with tools to assess current and proposed policies.

The tool set is implemented over a Windows 2000 server and consists of the Common Access State, a formal specification of the access policy implemented using an RBAC tool and Active Directory; policy management tools for domain administrators; three types of certificate authorities, for authenticating users within each domain, for authorizing access to resources belonging to each domain, and for authorizing access to joint resources using a shared-RSA cryptosystem; and a secure communication framework allowing trusted communication between domains.

An attendee asked how the coalition verifies that the domains are not passing shared information to outside parties. Koleva replied that confidentiality would need to be enforced using a non-technological mechanism such as a legal agreement.

■ *Manage People, Not Userids*

Jon Finke, Rensselaer Polytechnic Institute

Jon Finke contends that it is possible to maintain a single source of data about the people at your institution, and that the system administration group is well placed to run such a system. In this talk, he discussed details and strategies for such a database, using the implementation he oversaw at RPI as an example.

The driving principle is that every person in the system should have a status ("student," "faculty," "staff," "guest") and that a reasonable provider should maintain data related to each status. For instance, Human Resources should maintain staff data, while the registrar handles students. This system appeals to prospective data providers because they can be given total authority over their data. Consumers can use the database to group people accurately based on status. For instance, the library can set different book check-out intervals for professors and for students.

Finke then discussed technical details of the implementation, including the types of information stored in the database for each class of users. He discussed the maintenance of guests, which is complicated because universities have a large number of types of guests (e.g., visiting professors, dependents of other people in the system). In order to manage guests more easily, he requires that someone be responsible for each guest's data (the hosting department for visitors, the employee for dependents), and that guests expire from the database unless their data is explicitly renewed.

One attendee asked about problems encountered when correlating multiple sources of data. Finke replied that his group attempts to ensure that each person has only one database entry, but it does not always succeed. Once data providers are using the system, getting them to maintain their data is not hard, since users now know where to complain if their information is inaccurate.

INVITED TALKS

■ *Wikis, Weblogs, and RSS for System Administrators*

Dr. Jonas Luster, Socialtext, Inc.

Summarized by Laura Carriere

Luster began his highly entertaining talk by acknowledging that wiki and blog technologies have been around for a number of years now and are well established. Sociologists believe that the strongest human drives are to communicate and to make sense of communication; Luster stated that everyone is a sender but pointed out that there is no way to filter or roll back the data once it has been sent. It is the receiver's job to filter the data stream. Weblogs are an example of sending without filtering. RSS is an example of the receiver filtering the data. To emphasize his point, Luster observed that as the speaker he could choose to moon the audience and we would be unable to stop him, only to try to filter the image.

Wikis, as opposed to Weblogs, give permission to the receiver to participate. This makes them collaborative and creates fertile ground for communication.

Luster went on to describe the Pastures Theory, which explains that areas with the greenest grass attract the most cows. These cows then fertilize these areas, and this promotes the growth of more green grass, which attracts more cows. He compared this process to a busy wiki such as Wikipedia. Luster proposed that adding syndication to Weblogs, although it adds value by providing filtering, decreases the opportunities for fertilization and leads to empty pastures and deserted Weblogs. Luster then cautioned the audience to resist the temptation to compare our users to cows processing grass, but many of us were stuck with this image.

Luster presented survey results which found that there are 486 Weblog projects and 198 wiki projects currently available, and he suggested that we'd be better off with more wiki software and less Weblog software. He also reported that there were 16 million Weblogs in November 2005 and 13,000 contributors to Wikipedia. The average user is comfortable with this technology and users reported that their coding and HTML skills improved with Weblog development, although he expressed some skepticism about this result, based on his observations of many Weblogs.

Luster offered his view that the future holds tighter integration of video, audio, text, and collaboration and that these technologies may converge. He acknowledged that the required increase in complexity will increase the burden on the software maintainers. He also expressed concern that legal issues related to freedom of speech may soon come into play but suggested that the technical people leave this to the lawyers.

During the Q&A period, Adele Shakal, Caltech, asked for advice on social engineering strategies to deal with outdated content. Luster offered two recommendations: tie it to the user's paycheck by making it standard company practice, and automate a congratulations email after every 1,000 visitors, to encourage voluntary page maintenance.

At the conclusion of the talk the author expressed his pleasure at being able to share both the cows and the mooning images with us and then performed a live blog update rather than a live moon. The audience was profoundly grateful for his discretion.

■ *Using Your Body for Authentication: A Biometrics Guide for System Administrators*

Michael R. Crusoe

Summarized by Josh Simon

Michael Crusoe, a recent escapee from the biometrics industry, spoke about using biometrics from a sysadmin point of view. It was a high-level overview of the major biometric modalities, or methods of using body parts for identification. Techniques included:

- Facial recognition, which are error-prone in two dimensions due to changes in position and lighting.
- Fingerprinting, which can use the actual image, and the minutiae or the changes and breaks in ridges;

real-world testing shows that errors, both false positives and false negatives, decrease as the number of fingers examined increases.

- Hand geometry readers, the largest-deployed technology today.
- Iris recognition, which is the most accurate, due to the large amount of data available in a small space (striations, positioning, etc.), but which is very expensive to calculate; only one vendor is in this space (with soon-to-expire patents, so this may change).
- Speaker recognition, or voice-response.

Other modalities were mentioned, including vein recognition (using the pattern of the veins in the hand) and dynamic signature recognition (specifying the location, pressure, and velocity of the pen). Efforts are made to ensure that the body part is live (either by prompted motion, such as smiling or blinking on cue, or by scanning for temperature or motion).

WORK-IN-PROGRESS REPORTS

Summarized by Charles Perkins

- ***Bedework Open Source Institutional Calendar System***

Jon Finke, Rensselaer Polytechnic Institute

An open source standards-conforming calendar system designed to meet institutional needs, Bedework presents a Web interface, supports subscriptions, and presents a calDAV interface. iCal and skins are supported. Oracle is not used. Bedework is written in Java. For more information, see www.bedework.org.

- ***DeSPAC-SE: Delegated Administration Framework for SELinux***

Ryan Spring, Herbey Zepeda, Eric Freudenthal, and Luc Longpre, UTEP; Nick West, Stanford University

Eric Freudenthal presented a delegated administration framework for SELinux. DeSPAC-SE uses Mandatory Access Control to create security domains, and an active classifier with human intervention creates security tables of program types and allowed behavior. Security classification can be delegated and is amortized over many systems.

- ***Deployment of BladeLogic for Access Control Restriction, Change Tracking, and Packaged Software Distribution Primary to Ensuring Sarbanes-Oxley Compliance***

Michael Mraz

Developed for Solaris on SPARC as well as RedHat and SUSE x86 Linux, the software enables logging and auditing from development, through QA, and into production of complete software systems.

- ***VNC Manger: A Software Thin Client Using Perl, VNC, and SSH***

Wout Mertens

Mertens showed a brief live demo of Perl + TK software for managing multiple sessions of VNC over SSH with load sharing. The software thin client works on any UNIX, and special attention has been paid to server-side Solaris. Wout's presentation tied for best WiP. For more information, see http://sf.net/projects/vncmgr.

- ***An Exoskeleton for Nagios: Scalable Data Collection Architecture***

Carson Gaspar

Gaspar shows how to solve limitations of Nagios by adding a queueing server, a modular client agent, a config-file generator, an rrd-based trending server, and a ping agent. Multiple Nagios servers in passive pipe mode display and act on queued data.

- ***A Brief Look at RSA Moduli***

James Smith, Texas A&M

In his presentation, subtitled "What an English Major Learned in Class," James took the audience on a quick spin through the set of mathematical knowns and unknowns when narrowing the search space for finding factors of an RSA key.

- ***Mail Backup***

Dan McQueen, Cisco

Designed by Dan and coded by Ed Miller, this Sendmail/procmail backup system makes local copies of incoming mail automatically and allows users to initiate restoration of messages that might be lost due to user action before the nightly filesystem backup occurs. Text- and GUI-based restore tools are provided. Retention periods can be set. Restoration is a resend. Docs are forthcoming, and there are plans for open source. For more information, email dmcqueen@cisco.com.

- ***What I Did on My LISA Vacation***

Dave Nolan, CMU Network Services

Dave described the network architecture set up for the LISA conference. He addressed problems with network performance, reliability, and transparency, suggesting that for success one should "clone Tony" and spend money. Good results were had for LISA '05 because of a hotel-link upgrade, donated hardware, and excellent volunteer staff. Monitoring was done with the cricket collector, drraw drawing engine, argus network flow analysis tool, and mon nagios.

- ***Pretty Network Pictures***

Dan Kaminsky, DoxPara Research

In his presentation, subtitled "I Like Big Graphs and I Cannot Lie," Dan explained that while visual displays allow a human to absorb more complexity than text, animation encodes even more complexity. He then demonstrated real-time tcpdump data piped through OpenGL and displayed as video. With this codebase, Dan asserts that "OpenGL does the graphing, Boost does the layout, the programmer gets to be lazy." Tied for best WiP. For more information, email dan@doxpara.com.

■ ***How to Ask Questions the Right Way***

Cat Okita

Cat promoted asking better questions of those seeking technical help, including: What do you want to do? What have you tried to do? What happened? A little more detail please . . Got any ideas?

■ ***Portable Cluster Computers and Infiniband Clusters***

Mitch Williams, Sandia National Labs

Mitch described his work with clustered computers from the extremely small (one foot tall and 6x6 inches wide) to the Thunderbird system, which is #5 in the supercomputer list. For more information, see eri.ca.sandia.gov/clustermatic.org.

WORLDS '05: Second Workshop on Real, Large Distributed Systems

San Francisco, CA
December 13, 2005

INFRASTRUCTURE

Summarized by Rik Farrow

■ ***Experience with Some Principles for Building an Internet-Scale Reliable System***

Mike Afergan, Akamai and MIT; Joel Wein, Akamai and Polytechnic University; Amy LaMeyer, Akamai

Joel Wein described Akamai's Content Distribution Network (CDN) as having 15,000 servers in 1,100 third-party networks, with a NOCC managed by a day crew of eight and a night crew of three. The focus of this paper is not on CDN but on Akamai's experience in its seven-year experiment: in particular, keeping its distributed system running using Recovery Oriented Computing. In a single day, it is not unusual to lose servers, racks of servers, and even several data centers. The base assumption is that there will be a significant and constantly changing number of component or other failures occurring at all times in the network. The development philosophy is that their software must continue to work seamlessly despite numerous failures.

Wein outlined six design principles, organized in two sets of three. The first three principles are to ensure significant redundancy, use software logic instead of dedicated pipes for message reliability, and use distributed control coordination. Wein then gave examples of how these principles aid in operation during failures. The next three principles have to do with software design: fail cleanly and restart, zoning (their term for their brand of phased rollout), and notice and quarantine faults. No software is perfect, and these principles have helped to catch faults in software or configurations. Sometimes faults do not show up until a change has been rolled out to many systems. While most aborted rollouts occurred during phase one (36), the next most commonly aborted rollout occurred at the world level (23).

During the Q&A, Armando Fox asked why, if Akamai stages rollouts, there were ever any world aborts. Wein answered that sometimes that was when the problem showed up, and it could be caused by hardware, order of events, or corner cases. Fox followed up by asking if this was the only way to tickle the bug? Wein answered that stupid mistakes caused many of the world aborts, followed by needing to run on 50,000 servers before the problem shows up. Paul Lu asked how much of the system is homebrewed? Wein answered that a lot of this is custom code, but they are open to using other people's ideas and try not to be religious about these things. Jeff Mogul commented that most companies try to get down to one person per server, while the Akamai approach is different. Wein answered that their design notices a problem in an automated way, detects it right away, and removes it automatically. They have large brute force redundancy.

■ ***Deploying Virtual Machines as Sandboxes for the Grid***

Sriya Santhanam, Pradheep Elango, Andrea Arpaci-Dusseau, and Miron Livny, University of Wisconsin, Madison

Sriya Santhanam presented this research into the use of VMs in distributed computing. As most research Grid computing projects will run code that cannot be trusted, this code poses a security challenge. VMs provide security and isolation, environment independence, finer resource allocation, support for a wider variety of jobs, and a flexible, generic solution. They used Xen for their project, as Xen adds very little overhead when running applications on Linux. The target environment was Condor, software that watches for idle workstations so they can be used in Grid computing.

Santhanam described four different sandbox configurations, starting with the least restrictive and going to a very restricted environment. Even the least restrictive version has Condor alone installed within the VM, but arbitrary programs can be executed, and Condor itself is still exposed to network attacks. In the next version, VM gets launched on demand, and eager whole file caching is used, so no network access is required. In the next version, system calls get executed on the submitting machine rather than on the local system, and the final sandbox configuration includes lazy whole file caching and remote system calls on the submitting machine. Santhanam then presented graphs comparing the performance of the difference sandboxes.

Sean Rhea asked why sandbox 1 showed such low overhead compared to the other versions. Santhanam answered that only in this version is the VM already running. All other sandboxes include the time to start the VM in their overhead. Armando Fox asked, which sandbox would you choose for your friends? For people you trust to run code, sandbox 1 is easiest, whereas sandbox 4 adds additional components and complexity. Rhea asked if only one job is run at a time, and Santhanam answered yes, because the goal was limited and focused on defense. Rhea asked if the VM gets flushed after running each job. Santhanam answered that these are student workstations, running in labs, so the focus is on protecting these machines.

- ***MON: On-Demand Overlays for Distributed System Management***

  Jin Liang, Steven Y. Ko, Indranil Gupta, and Klara Nahrstedt, University of Illinois at Urbana-Champaign

  Jin Liang described the problems that can occur when running applications on the PlanetLab Grid: monitoring and control require connections from the many remote systems each to a separate process. Monitoring the status of remote applications—noticing if they have crashed, if they need to be restarted, or if all applications need to be stopped and a new version uploaded—has been difficult with the existing tools. The goal also includes software distribution to all nodes.

  MON is a management overlay network that uses an equivalent of a spanning tree to aggregate the results of all commands and to distribute commands to all the nodes. The overlay network is built on demand when needed, and is simple, lightweight, and suited to management, irregular/occasional usage, and short/medium-term command execution. When execution completes, the overlay goes away. Each remote host runs one daemon process that not only executes commands, but also participates in the construction of the tree. The construction of the tree must itself be lightweight and satisfy the requirements of both status query and software distribution. Liang described research into the best method of tree construction, a combination of random tree construction followed by local selection of neighbors. The paper provides more details of tree construction.

  The Q&A focused more on what MON can and can't do than on tree construction. Someone asked, how can you be sure that a response from a node is calculated exactly once? Liang answered that this is not a problem, as each parent aggregates responses from children and sends just one response to its parent. Someone else asked, how can you find nodes that aren't working properly? Jiang said that MON is not designed for this purpose, but is focused on reliable, occasional monitoring.

CHOOSING WISELY

Summarized by Jin Liang

- ***Supporting Network Coordinates on PlanetLab***

  Peter Pietzuch, Jonathan Ledlie, and Margo Seltzer, Harvard University

  Jonathan Ledlie first briefly reviewed what network coordinates are. Network coordinates such as Vivaldi try to approximate delay between two nodes using a geometric space. Thus, they are a powerful abstraction for distributed systems. However, the delay between nodes is not static. There could be gradual changes as well as unpredictable, unusually large deviations. The authors used a moving minimum filter to deal with this problem. Specifically, at any time, the next delay is predicted as the minimum of the previous four measurements. The second problem with network coordinates is that the changes in network coordinates might cause expensive application-level adjustments. For this, some update filter is used. Specifically, the centroid of the starting coordinate window is computed. The application is notified about the change only when the current centroid of the coordinate window is significantly different from the starting centroid.

  Ledlie also showed a movie that illustrates how the coordinates would change, with and without the link (moving minimum) and update filters. Their evaluation results are based on the delay measurement on about 270 machines on PlanetLab.

  One audience member commented that the filters are similar to network time protocol (NTP), including the update filter (whether a node is trustable in NTP). Ledlie said he will look at the differences. Another audience member asked if it is possible to report distribution as well as coordinates to the application, so that the application is aware of how much variance there is. Ledlie said this is currently not in the system but can be added. Someone else asked if delay is correlated with load, and Ledlie answered that there is a correlation.

- ***Fixing the Embarrassing Slowness of OpenDHT on PlanetLab***

  Sean Rhea, Byung-Gon Chun, John Kubiatowicz, and Scott Shenker, University of California, Berkeley

  Awarded Best Paper!

  There is a lot of hype about DHTs (Distributed Hash Tables). However, many previous results were obtained in benign environments (i.e., in lab). The authors of this paper wanted to improve the performance of DHT "in the wild," and by considering 99th-percentile performance numbers. Real-world applications may not have dedicated machines, and the authors want to provide an OpenDHT service. There are two flavors of slowness in nodes. The first is unexpected, which is discovered only when a request is routed to the node. The second is consistent slowness, which can be avoided by maintaining a history. The authors provided two solutions to node slowness: (1) Delay-aware routing, in which the delay to the next hop and the distance in the key space between hops are considered when selecting the route.This is in contrast to traditional DHT, where routing is purely greedy in the key space. (2) Parallelism. Using iterative routing, the requester can keep multiple outgoing RPC requests. Thus, even if some slow nodes are encountered, other requests can quickly get results. The user can also send the initial request to two different gateways.

Their results, obtained from PlanetLab using concurrent execution methods (i.e., a particular approach is randomly selected for lookup each time) show that delay-aware routing is clearly best, reducing the 99th-percentile latency by 30% to 60% without increasing overhead. Other techniques can also reduce the delay, but will increase overhead.

- ***(Re)Design Considerations for Scalable Large-File Content Distribution***

Brian Biskeborn, Michael Golightly, KyoungSoo Park, and Vivek S. Pai, Princeton University

Well-designed systems may not work efficiently in a real environment. In redesigning the Coblitz file transfer service, the authors achieved a 300% faster download and a 5x load reduction on the origin server. Coblitz uses a content distribution network for file transfer. A smart agent will divide the request for one file into multiple requests for file chunks. The requests are sent to different CDN nodes that have the chunks cached. There are several techniques that are used to improve the downloading. For example, some nodes are consistently slow, and these are removed. Also, when a node is slow, instead of waiting for time-out and retry, the new design keeps several connections to compete with each other. Also, before a node requests the file from the origin server, it looks at other nodes to see if they are more suitable for making the request. Using these techniques, the new Coblitz system's performance is much improved.

One audience member asked if the set of slow nodes is stable, because they have found it (in terms of delay instead of bandwidth) unstable. KyoungSoo said they have done a lot of bandwidth measurement and the set is stable. Another questioner asked where the bottleneck is for downloading, and KyoungSoo answered, the bandwidth cap. Another audience member asked whether they have run comparisons with SHARK, and KyoungSoo answered, yes, and with BulletPrime.

FROM THE TRENCHES

Summarized by KyoungSoo Park

- ***Non-Transitive Connectivity and DHTs***

Michael J. Freedman, New York University; Karthik Lakshminarayanan, Sean Rhea, and Ion Stoica, University of California, Berkeley

Michael Freedman started by pointing out the difficulty of running DHT applications due to the non-transitive connectivity problem. Non-transitive connectivity is A not being able to communicate with B, while A and C and also C and B can communicate. Because the conceptual DHT design assumes full connectivity, people must often resort to their own hacks to get around this problem. Non-transitive connectivity occurs for various reasons, such as link failures, BGP routing updates, and ISP peering disputes, and 9% of PlanetLab nodes are reported to show such phenomena, according to Stribling.

For DHTs, Michael classified the problems as "invisible nodes," "inconsistent roots," "broken return paths," and "routing loops." Node B is said to be A's invisible node if A can communicate with B via C, but not directly with B. A simple fix for this would be to let A add (or remove) B only when A directly communicates (cannot communicate) with B. In order to get over the performance impact of invisible nodes, Michael proposes (1) timeout estimates via network coordinates, (2) parallel lookups, and (3) caching of unreachable nodes. Another problem is "inconsistent roots," possibly caused by network partition. Two distictive nodes, say R and R', which cannot communicate with each other, may each act as if it were the root. An expensive consensus algorithm is one way of solving this; another is to use link-state routing among the leaf set with FreePastry 1.4.1. "Broken return paths" means that a direct return path between the destination node and the entry node may not exist, while a forwarding path in the DHT lookup does exist. One solution is to route backward along the lines of the forward path, and the other is one-node source routing via a leaf node randomly chosen by the destination node.

Justin Cappos asked whether such non-connectivity is mostly unidirectional or bidirectional, and Michael responded that he did not measure it, but he thinks it is mostly asymmetrical. Indranil Gupta mentioned that the problem is being solved by RON, and asked if the problem is the fundamental limit of DHT. Michael and Sean Rhea responded that it is a problem of whether to store more states in the routing table. Rick McGeer added that Tapestry has a backup path, and Sean confirmed that.

■ ***Why It Is Hard to Build a Long-Running Service on PlanetLab***

Justin Cappos and John Hartman, University of Arizona

Justin Cappos began by asking why we do not see many long-running services on PlanetLab, even though PlanetLab was mainly created to support them. He divided the types of services that researchers are interested in into three categories. The first category consists of highly novel services that are publishable but unstable and that usually end up perishing right after publication (e.g., Bullet and Shark). Another category includes services such as AppManager and Sirius, which have high stability but are not novel enough to be made into papers.The last category, comprised of services that combine the two features, includes Stork, Bellagio, and CoDeeN. Justin explained that the reason why we do not see many research services on PlanetLab is because there is not much incentive to provide long-running services, which would take non-trivial maintenance time that cannot be rewarded with publication, and he emphasized the need to give more credit to long-running services.

He described the process by which Stork was shaped into a reliable, long-running service and the lessons to be drawn from Stork's example: the need to have a reasonable fall-back scheme for unreliable services, to build on other research services, to be aware of corner cases, and to use other research systems and provide the feedback essential to improving their quality and usability.

Jeff Mogul commented that it is not the novelty of the idea but the novelty of the results that draws the attention of paper reviewers, and conferences like OSDI mainly focus on such results. Sean Rhea commented that in the past, good projects all started with a simple scheme but evolved into a novel system by fixing problems in the middle.

■ ***Using PlanetLab for Network Research: Myths, Realities, and Best Practices***

Neil Spring, University of Maryland; Larry Peterson, Andy Bevier, and Vivek Pai, Princeton University

Years of operation of PlanetLab have created various myths that used to be true. Still, some research folks believe PlanetLab is too flaky or too overloaded for some experiments. Neil Spring talked about what is and what is not true about PlanetLab, based on his careful observation.

He started with what's true. First, the experimental results are not reproducible on PlanetLab, because it is designed to provide a real-world Internet environment rather than a controlled testbed. Even so, short experiments can be measured more carefully by avoiding what CoMon has determined to be heavily loaded nodes. For reproducible results, Emulab and Modelnet can be alternatives to PlanetLab. Also, PlanetLab is not representative of the Internet or peer-to-peer network nodes, because PlanetLab cannot cover the entire Internet and its nodes are not desktop machines as in P2P systems. However, more and more traffic on PlanetLab includes lots of commercial sites and is not PlanetLab-exclusive. Although PlanetLab does not use P2P nodes, its nodes can be used as managed core nodes in P2P systems, as in End System Multicast (ESM).

Neil also enumerated myths that are no longer true. First, PlanetLab is no longer overloaded. Measurement shows that 20 to 30% of available CPU cycles are available at any given time, even right before major conference deadlines. The current per-slice scheduling prevents any single slice from hogging all the CPU cycles. Another myth is about PlanetLab's supposed inability to guarantee resources, but resources are available because they are managed by a brokerage service, especially in running short-term experiments.

Best practices also help in demystifing some myths and in improving the reliability of the experiments on PlanetLab. By using kernel timestamps and instrumenting traceroute one can time the packets on PlanetLab with great accuracy, and measuring bandwidth via precise packet trains is still possible with the use of an appropriate system call such as `nanosleep()`. Random site measurement restriction imposed by PlanetLab AUP can be implicitly lifted by soliciting outside traffic, and observation of such rules is easily achieved by using a service like Scriptroute. Finally, it is beneficial to know that surviving excessive churns is essential in making long-running experiments.

Mic Bowman asked if PlanetLab suffers from memory pressure due to the large memory footprint of some Java programs. In response, Vivek mentioned that a recent memory pressure test shows that 80% of all PlanetLab nodes have at least 100MB available. Besides, pl_mom is effective in maintaining a good level of memory status by killing the highest memory consumer when memory pressure arises.

Sean Rhea commented that one problem is that people tend to try a random tool that works on a stock Linux, but get frustrated to see it not working on PlanetLab. He also mentioned that it would be useful to make the packet trains into a tool like KyoungSoo and Vivek's CoTop. Neil responded that using Scriptroute will provide accurate timing without carefully implementing it individually.

FAST '05: 4th USENIX Conference on File and Storage Technologies

San Francisco, CA
December 13–16, 2005

KEYNOTE ADDRESS

■ *Greetings from a File-System User*

Jim Gray, Distinguished Engineer, Microsoft Bay Area Research Center

Summarized by Stefan Büttcher

Jim Gray's message was that we have arrived at an era of infinite storage. He argued that in today's storage systems, I/O bandwidth and seek latency are limiting factors. In order to keep a single CPU core busy, 100 hard drives are needed; for future 10-terabyte hard drives, it will take 1.3 days to read all data sequentially and five months to read them randomly.

File systems are becoming so large that we need database systems in order to be able to use them effectively: how do we find data in a file system containing 30 million files of 1GB each? Integrating a database into the file system and combining the hierarchical structure with a content-based addressing mechanism would help.

According to Jim, we are heading towards a backup-free world, because the file systems are getting so large that it would take too long to restore the contents. Since we have more storage space than we need, we might as well keep many versions of the data (snapshot file systems) instead of backups.

As a side blow in the direction of Garth Gibson, one of the inventors of RAID, Jim pointed out that RAID-5 is the wrong tradeoff, as it sacrifices bandwidth for more efficient space utilization. In the Q&A, Garth, of course, disagreed, noting that for many people storage space efficiency is still very important.

FILE SYSTEMS SEMANTICS

Summarized by Vijayan Prabhakaran

■ *A Logic of File Systems*

Muthian Sivathanu, Google Inc.; Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, and Somesh Jha, University of Wisconsin, Madison

Muthian began the talk by discussing how important it is to ensure the correctness of file systems. The current approaches, such as stress testing and manual exploration, are inadequate and error-prone. A logic of file systems is a formal framework for reasoning about file systems. It focuses on file system interaction with disk and targets file system design rather than implementation.

Then Muthian gave some background on file systems, describing metadata and data consistencies. The key challenge in reasoning is the asynchrony which arises in file systems due to buffering and delayed writes. There are three basic entities in the model: containers, pointers, and generations. A file system is a collection of containers linked through pointers. A container is a placeholder of data, and generation is defined as an instance of a container between reuse and free. Muthian then explained the concept of containers and generations through an example.

Concepts such as beliefs, actions, and ordering operators (e.g., before, after, and precedes) were explained. The proof system followed by the logic is based on event sequence substitution. Muthian gave examples of basic postulates, for example, "If container A points to B in memory, a write of A will result in the disk inheriting the belief."

Three case studies that are described in detail in the paper were briefly explained by Muthian. The first case study verifies the data integrity under various file system mechanisms, such as soft updates and journaling. The second case study examines a performance bug in ext3. The last case study looks at the non-rollback property under journaling. Other case studies detailed in the paper deal with generation pointers and semantic disks.

■ *Providing Tunable Consistency for a Parallel File Store*

Murali Vilayannur, Partho Nath, and Anand Sivasubramaniam, Pennsylvania State University

Parallel file systems distribute portions of a file across diff servers. With multiple data servers and client-side caches, consistency becomes an important issue. Current configurations provide either much weaker consistency (e.g., PVFS) or much stronger consistency (e.g., Lustre). However, the applications running on a parallel file system know better about their concurrency/consistency needs than does the file system.

The approach taken in CAPFS is to export the mechanisms and leave the policy to the applications, which provides tunable granularity. CAPFS uses content-addressable data stores and optimistic concurrency control mechanisms to provide serialization.

The architecture consists of two server components: hash servers, which are the metadata servers, and content addressable servers (CAS), which are the data servers. Hash servers provide a NFSv4-like interface. The CAS servers are multi-threaded servers. Murali then described how writes are handled. Whenever a write is issued it goes to the hash server first, which computes the hash, and then the write goes to the CAS. During commit, the old hash of the data is compared with the new hash. If they match, the commit succeeds. Write serialization is achieved this way. The system is verified with a 20-node experimental setup.

SENSOR STORAGE

Summarized by Shafeeq Sinnamohideens

■ ***MicroHash: An Efficient Index Structure for Flash-Based Sensor Devices***

Demtrios Zeinalipour-Yazti, University of Cyprus; Song Lin, Vana Kalogeraki, Dimitrios Gunopulos, and Walid A. Najjar, University of California, Riverside

Dimitrios Gunopulos first described a sensor network developed for, among other applications, a U.C. Riverside Conservation Biology project to monitor soil organisms. While the network has a large number of sensors, sensing several parameters over a long period of time, only a few time points or parameters are interesting. The sensor nodes are based on the RISE platform, have local flash RAM as their main storage, and limited network and power resources. Since nodes in their system only transmit data in response to queries, each node must store and index the data it collects in its local flash memory. The problem is that while existing storage and indexing structures are well suited to the properties of RAM and hard disks, flash memory has a few unique properties of its own: it can only be erased a whole block (several pages) at a time; a page can only be written into an already erased block; and a page physically wears out after being written 10,000–100,000 times.

Using structures meant for other media will result in poor performance as a result of having to read and rewrite an entire block whenever any of its contents change, as well as wearing out some pages more rapidly than others. The goal of their proposed structure (MicroHash) is to efficiently support value-based and time-based queries for single data points or ranges while maximizing the lifetime of the flash memory.

MicroHash contains data records that are both hashed into buckets and indexed. It uses four types of pages: data pages that store data records, index pages that store indices to the data, directory pages containing information on hash buckets, and a root page that stores properties for the structure. Pages are always written to flash in a circular order to provide wear-leveling. In normal operation, as the sensor generates data records, it inserts them into a data page. When the data page is full, it is written to flash in the next available position. The corresponding index record is updated and the index page written to the next position, if necessary. As writing proceeds, the oldest page will be overwritten when there are no more free pages. Because the index is always updated after data is written, an index page is never deleted until the data it indexed is also deleted. If a particular hash bucket contains too great a proportion of indexed records, a repartitioning step will split it into two less-full buckets.

Searching by time is simple, since all pages are written in chronological order. Searching by value requires first hashing the value to select a dir page. The dir page will point to the most recent index page for that value. The index page will either point to a data page with the data or to another index page that can be followed.

Jason Flinn asked how the number of directory buckets ever shrinks. The answer is that when splitting produces two new buckets, the old bucket is eventually reclaimed by the normal overwriting process.

■ ***Adaptive Data Placement for Wide-Area Sensing Services***

Suman Nath, Microsoft Research; Phillip B. Gibbons, Intel Research Pittsburgh; Srinivasan Seshan, Carnegie Mellon University

These sensor nodes differ in scale from those in the previous talk; they are assumed to have more computing power and may be distributed anywhere in the Internet. Query-issuing clients may also be anywhere in the Internet. In addition to sensor nodes, the system may include other infrastructure nodes, which can also replicate data, perform data aggregation, and process queries. The system aims to assign functions to nodes automatically, in order to optimize efficiency, robustness, and performance across the entire system. Additionally, the IrisNet infrastructure may be supporting several different sensor networks, with different access patterns which may change over time.

The IrisNet Data Placement (IDP) algorithm attempts to determine, for a given network hierarchy and node capabilities, the data placement that optimizes query latency, query traffic, and update traffic. It is a distributed algorithm that runs on each node, using only local knowledge to approximate the globally optimal solution, while rapidly responding to flash crowds. Each node builds a workload graph representing all data objects necessary for its queries, with edges weighted by the traffic across that edge. The algorithm must select fragments (subgraphs) to partition and allocate to each node. The optimal solution is $O(n^3)$, which is too slow to be used. By only considering subtrees, an approximate solution can be found in $O(n)$. By contrast, all better-performing algorithms require global knowledge, and no distributed algorithms perform as well.

After partitioning, IDP must choose where to locate each fragment of the workload. It does this using two heuristics. One attempts to cluster data objects together. This reduces the number of nodes involved, but requires consideration of whether nearby machines can handle the extra load. The other places fragments as close to the data source or sink as possible. This reduces traffic and latency, but may involve additional nodes. Repartitioning or replication is performed when load on a node exceeds a set threshold. When replicas are available, a query can select either a random replica or the nearest one. If the nearest is selected, it may become persistently overloaded, whereas selecting a random one will cause all replicas to have an equally light load. As a compromise, IDP selects a replica randomly, but with weighted distribution, so nearby replicas are selected more often.

Christopher Hooper asked whether energy consumption was considered and whether IrisNet could take advantage of heterogeneous power availability. The answer was that power had not been considered, but could be considered one element of a node's capacity.

FAULT HANDLING

Summarized by Kevin Greenan

- ***Ursa Minor: Versatile Cluster-based Storage***

Michael Abd-El-Malek, William V. Courtright II, Chuck Cranor, Gregory R. Ganger, James Hendricks, Andrew J. Klosterman, Michael Mesnier, Manish Prasad, Brandon Salmon, Raja R. Sambasivan, Shafeeq Sinnamohideen, John D. Strunk, Eno Thereska, Matthew Wachs, and Jay J. Wylie, Carnegie Mellon University

Awarded Best Paper!

John Strunk presented the work on versatile cluster-based storage at CMU's Parallel Data Lab. To get the audience into the right state of mind, Strunk presented a quick brain-teaser which illustrated the fact that a single storage system generally stores different data sets, with different requirements. Unfortunately, all of the data in the system may share the same fault model and encoding scheme.

Even though cluster-based systems provide more cost-effectiveness and scalability than today's monolithic approaches, these systems do not necessarily provide versatility. Ursa Minor attempts to solve the challenge of versatility in cluster-based storage.

Essentially, Ursa Minor is a cluster-based storage system which supports multiple timing models, fault models, and encoding schemes among multiple data sets in a single system. In addition, changes to the distribution of data can be made online, thus configuration choices are adaptive. The architecture of Ursa Minor is quite simple and provides object access similar to NASD architecture and the emerging OSD standard. Basically, clients are required to consult an object manager for metadata requests and I/O request authorization. Versatility is accomplished using a protocol family that supports consistent access to data in the storage system. Each member in a protocol family is defined by three parameters: timing model, fault model, and encoding scheme. Online data distribution changes are made using back-pointers from the new data locations to the old data locations. The object manager can then revoke access to the old locations, forcing the client to request the new location of the data.

In the end, we find there is a lot to gain from defining specialized configurations for different workloads in a cluster-based storage system, especially when the workloads are running at the same time.

A great many questions came up during the Q&A. One member of the audience asked how an object is re-encoded upon distribution change. A distribution coordinator works its way through the object by actively re-encoding in the background, ensuring that newly written data does not get overwritten. A few of the questions were directly related to the encoding schemes used in Ursa Minor. Currently, the user is responsible for choosing which information dispersal encoding is used for a given data set. Lastly, Strunk was asked whether failures were assumed during migration, and he answered they are not.

- ***Zodiac: Efficient Impact Analysis for Storage Area Networks***

Aameek Singh, Georgia Institute of Technology; Madhukar Korupolu and Kaladhar Voruganti, IBM Almaden Research Center

Aameek Singh presented work on impact analysis, starting with a photograph of a woman pulling her hair out, which was strategically placed to symbolize the frustration involved in storage management. The work focuses on the change-analysis problem. The Zodiac framework is provided to help system administrators determine the impact of changes to a SAN (storage area network) before actually making the change. This framework integrates proactive change analysis with policy-based management; thus, the impact of an administrator's action is assessed with respect to a set of user-defined policies. In the context of impact analysis, policies can be thought of as best practices.

Singh briefly explained the four primary components of Zodiac: SAN-state for incremental operation within a single analysis session; optimization structures for efficient policy evaluation; a process engine for impact evaluation; and a visualization engine, which acts as the output interface to the user. The main SAN data structure is represented by a graph of entities connected by network links; thus graph traversals are required when policies are added or evaluated. In order to make policy evaluation more efficient, the authors exploit policy classification, caching at every node in the SAN graph, and aggregation. All of these optimizations allow for a reduced graph traversal space when evaluating policies.

Singh showed that the three policy evaluation optimizations significantly decrease the latency of policy evaluations and allow for more scalable evaluations as the size of the SAN increases. Overall, this work provides an efficient framework that provides what-if analysis under a policy-based infrastructure.

A member of the audience asked whether this framework could be used for root-cause analysis. Singh replied that this work did not focus on finding a root cause, but such a tool used in conjunction with their framework would be very helpful.

- ***Journal-Guided Resynchronization for Software RAID***

Timothy E. Denehy, Andrea C. Arpaci-Dusseau and Remzi H. Arpaci-Dusseau, University of Wisconsin, Madison

Timothy Denehy presented an approach to software RAID resynchronization after a system crash, which was done with some of the other folks at the University of Wisconsin, Madison. Denehy pointed out that it is hard to maintain consistency at the RAID layer, since a very long window of vulnerability may exist during updates. Failures that occur within this window of vulnerability may leave a stripe within a RAID array in an inconsistent state. The authors show that this window of vulnerability can be removed through the use of a write-ahead log, which may result in poor performance.

Instead of relying on a write-ahead log or offline scanners, the authors propose a solution that leverages the functionality of a client-journaling file system, such as ext3. Denehy gave a quick overview of ext3 with respect to transactions and journaling. A new mode of operation, declared mode, is added to the underlying file system. This new mode of operation, which is similar to ext3's ordered mode, requires the write record for each data block to reside in the journal before issuing the actual write. Unlike ordered mode, this results in a record of outstanding writes, which can be used to restore consistency upon a system crash.

In addition to the new mode of operation, an interface is created, which allows the file system to communicate inconsistencies to the RAID layer. This interface between the file system and software RAID is very straightforward. The file system can tag a block with a synchronize flag, which results in a verify read request at the software RAID layer. At the RAID layer, the corresponding stripe is read and checked. If the parity is inconsistent, a new parity for the stripe is computed and written.

The process of file system recovery and RAID resynchronization is done using the new functionality. After a system crash, the file system can scan the journal and communicate possible inconsistencies to the RAID layer. These possible inconsistencies are handled at the RAID layer using the verify read request.

Denehy further justifies the effectiveness of declared mode by comparing it to ordered and data-journaling mode on a set of benchmarks, which shows that declared mode generally outperforms data-journaling mode and incurs very little overhead with respect to ordered mode. Another important side effect of this new form of RAID resynchronization is the reduction of the window of vulnerability from 254 to 0.21 seconds.

CACHING

Summarized by Ali R. Butt

- ***DULO: An Effective Buffer Cache Management Scheme to Exploit Both Temporal and Spatial Localities***

Song Jiang, Los Alamos National Laboratory; Xiaoning Ding, Feng Chen, Enhua Tan, and Xiaodong Zhang, Ohio State University

Song explained that the motivation for their work is the increasing gap between disk and processor speed. Hard disks remain a performance bottleneck when accessing data at high speeds. He explained that the main reason for this bottleneck is the lack of sequential accesses to the disk, which, among other things, causes expensive disk-head movements. Although the application accesses are more sequential than random, the filtering effect of the buffer cache results in the accesses to disk becoming randomized. To address this issue, Song presented a new buffer cache-replacement algorithm, DULO, which uses both temporal and spatial patterns. The main goal of the paper is to increase the sequential accesses that are issued to disk.

Song explained that there are two schemes that are employed to improve the number of sequential accesses: namely, disk request scheduling and file prefetching. The buffer cache sits on top of the file prefetcher and I/O scheduler. The cache filters the requests from the application to the lower layers, and has the potential problem of filtering the patterns, which in turn makes the pattern more random. Hence, the buffer cache has the ability to shape the disk requests. Since other schemes only consider temporal locality of the blocks being accessed, they may result in a smaller number of blocks being read from the disk, but these blocks may have poor sequential properties, resulting in more disk head movements.

In the Q&A someone asked whether the authors have compared DULO to LIRS (an algorithm also proposed by the same authors). Song responded that he has not yet done that but is considering extending DULO to a more general scheme that can accommodate any cache-replacement algorithm rather than only LRU. Kai Shen from Rochester University inquired how DULO compares to the optimal case in this situation. Song replied that it is hard to define "optimal" in this scenario, due to the complexity of the components involved.

■ ***Second-Tier Cache Management Using Write Hints***

Xuhui Li, Ashraf Aboulnaga, Kenneth Salem, University of Waterloo; Aamer Sachedina, IBM Toronto Lab; Shaobo Gao, University of Waterloo

Examples of second-tier cache management, the topic of the paper presented by Kenneth Salem, are the file server acting as the lower cache, and the client as the top cache, or, more interestingly, the lower cache as the storage server and the top cache as a database system. An important thing to note in this context is that database systems that handle OLTP workloads perform a lot of write requests.

There are two difficulties that are faced in two-tier cache management. One is that of cache inclusion, i.e., a page is stored in both the caches, which essentially wastes space. Therefore the challenge is to maintain exclusivity between the two caches. The second challenge is that the second-tier cache exhibits poor temporal locality. Kenneth pointed out that other people have looked at various schemes to manage two-tier caches by employing hierarchy-aware schemes, interpreting storage data, using explicit notifications between caches, and providing hints to the higher tiers. The approach presented in this paper is based on hint-based schemes that require only simple changes to the first-tier cache management. The main focus of the work is on write requests so as to improve the hit ratio of the second-tier cache.

In the Q&A session, Song Jiang of LANL inquired about the effect of the first-tier cache management algorithm on the second tier, and he pointed out that the interaction may adversely affect the cache performance. Kenneth agreed that this was possible, which is why they are interested in evaluating the scheme for more applications. Prashant Pandey of IBM Research said that in the current scheme the second tier is expected to interpret the hints on its own and asked whether there will be any benefit in telling the storage exactly what to do. Kenneth replied that they have made an effort to keep the hints open for interpretation by the second tier, but it would be interesting to see if the second tier can simply use the hints as classification. However, this aspect remains part of their future work. Another questioner asked about the distinction between write hints and eviction hints. Kenneth replied that they currently only interpret the hints at the second tier, but possibly could introduce two additional bits in the hints to ask the second tier for direct eviction.

■ ***WOW: Wise Ordering for Writes—Combining Spatial and Temporal Locality in Non-Volatile Caches***

Binny S. Gill and Dharmendra S. Modha, IBM Almaden Research Center

Binny presented an innovative idea that aims at improving the performance of writes to hard disks. He pointed out that the writes have often been ignored in caching research, which mainly focuses on improving performance of reads. The presentation started with a brief history of caching's important part in improving the I/O time of disks. But although read caches have significantly improved the performance of disks, there are six times more writes in terms of disk seeks. He also pointed out that write caches are typically 1/16th the size of read caches. Hence, improvement in write time can have a significant impact on the overall I/O performance, but the small size of write caches requires careful planning in order to get any benefit from them.

Binny then presented WOW, which uses reordering of writes in the NVRAM write cache to reduce the disk seeks associated with writes. The order in which writes are destaged to disk is critical. WOW aims to use the smallest amount of disk time for writes and to use most of the time to service read requests. For this purpose, it utilizes both temporal and spatial locality of the writes. To create spatial locality, WOW uses reordering.

The WOW algorithm is produced via an innovative marriage of the CSCAN and CLOCK algorithms, and has the good qualities of both. Basically, WOW uses CLOCK bits for temporal locality information, and weights of CSCAN to give the spatial order information. WOW keeps the sorted order of CSCAN and temporal bits of CLOCK to give both spatial and temporal locality information. The evaluation of the scheme shows that WOW indeed provides improved throughput and response time.

SECURITY

Summarized by Aameek Singh

■ ***Secure Deletion for a Versioning File System***

Zachary Peterson, Randal Burns, Joe Herring, Adam Stubblefield, and Aviel D. Rubin, The Johns Hopkins University

Due to increasing federal regulations and other business requirements, versioning file systems are being deployed rapidly. These file systems maintain multiple versions of the data and can be used to restore to an earlier version. For space efficiency, different versions can share data blocks. This paper makes two contributions: (1) secure deletion of a file, implying that a deleted file cannot be retrieved by any forensic techniques; (2) authenticated encryption, ensuring that data has not been corrupted between a disk write and its corresponding read.

Some of the earlier approaches—repeated overwriting, encrypting, and deleting the key—require more storage or need data blocks to be contiguous. This paper's approach minimizes the amount of secure overwriting and eliminates the need for contiguity. The main idea is to use a keyed transform to create a short stub representing the data blocks with the additional property that deleting the stub by secure overwriting automatically deletes the data.

The authos also presented techniques that are better optimized for deleting an entire version chain. The techniques have been implemented in ext3cow versioning file system.

■ ***TOCTTOU Vulnerabilities in UNIX-Style File Systems: An Anatomical Study***

Jinpeng Wei and Calton Pu, Georgia Institute of Technology

Time-of-check-to-time-of-use (TOCTTOU) vulnerabilities occur in UNIX-style file systems when applications perform two non-atomic steps—first establish an invariant about the state of the file system and then perform an operation assuming the invariant to hold. For example, Sendmail first establishes that the mailbox is not a symbolic link (a malicious user's attempt to corrupt an important system file) and then writes to the mailbox. Between these two steps, a malicious user can modify the file system so that the invariant does not hold, but the application does not check for it in the second step.

The paper attempts to define a formal model, called CUU, that can be used to identify such TOCTTOU vulnerabilities. For this, they identify pairs of operations that establish invariance and then operate on it: for example, <stat, open>. Such pairs, called TOCTTOU pairs, can then lead to potential attacks.

The paper identified 224 such pairs in various utility programs such as Sendmail, vi, and RPM. They also checked the feasibility of attacks on vi, which shows that such attacks can have nearly a 50% success rate for large files.

■ ***A Security Model for Full-Text File System Search in Multi-User Environments***

Stefan Büttcher and Charles L.A. Clarke, University of Waterloo

With increased interest in desktop search, there are many tools available now from companies such as Google, Microsoft, Apple, and Yahoo. However, a multi-user environment presents new and interesting challenges. Keeping a separate index for each user in the system is inefficient, since many files are actually accessed by multiple users and thus a single file system change would need to be pushed into each index.

A second approach, that of keeping a single index and postprocessing search, requiring that files that a user should not see are removed, suffers from a subtle problem: since query ranking uses statistics that in a single index case would be systemwide, carefully formed queries can leak out potentially critical information.

As a solution, the paper proposes GCL, a structured query language developed in the 1990s by one of the authors which evaluates on-the-fly query ranking using security primitives, ensuring that no file or its influence on statistics is revealed through the results. One of the shortcomings is the memory caching of security properties of each file, which is 32 bytes for each inode.

The system shows good performance and is available at http://www.wumpus-search.org.

MULTI-FAULT TOLERANCE

Summarized by Florentina Popovici and Timothy Denehy

■ ***Matrix Methods for Lost Data Reconstruction in Erasure Codes***

James Lee Hafner, Veera Deenadhayalan and K.K. Rao, IBM Almaden Research Center; John A. Tomlin, Yahoo! Research

Jim Hafner addressed two general problems pertaining to erasure codes, with the ultimate goal of recovering lost data whenever it is information-theoretically possible. First, can the system recover from uncorrelated errors and, if so, how? Second, how can the system efficiently recover partial strip data? To solve these problems, the author presented the following theorem: for any linear erasure code and a set of sector failures, there exists a simple mechanism that identifies which sectors cannot be recovered and provides formulas for the reconstruction of those sectors that can be recovered. Jim presented their method, based on matrix theory and pseudo-inverses, which completely solves the first problem and provides the formulas for solving the second problem.

He also presented a hybrid approach which uses the matrix methods along with the code-specific recursive reconstruction methods to improve efficiency. Finally, he demonstrated their methodology for recovering lost array sectors with a TCL/Tk application.

The first questioner asked if the ordering of sector recovery matters? Jim responded that if the sectors are lost simultaneously, the ordering of recovery does not matter. Garth Gibson asked how often an additional sector can be lost and recovered under existing erasure codes. In his experience, Jim estimated that a third lost sector could be recovered about 50% of the time.

■ ***STAR: An Efficient Coding Scheme for Correcting Triple Storage Node Failures***

Cheng Huang, Microsoft Research; Lihao Xu, Wayne State University

Cheng Huang asked how to ensure both reliability and performance for storage systems. Some of the characteristics of such systems are that they are built from less reliable components in order to achieve large capacity, and that they may also be geographically distributed.

Reliability is achieved by redundancy. Usually the codes used are (n,k) threshold codes. n is the number of nodes where the shares of the data are distributed, and k represents the minimum number of shares that need to be gathered to reconstitute the original data. Most systems use MDS schemes, which allow for the recovery of $r = n - k$ nodes, where r is called the reliability degree of an (n,k) scheme.

But all practical schemes use Reed Solomon schemes as MDS, and they are slow, so the question is whether there are other, better-performing schemes. The alternatives are MDS array codes such as XOR ($rr = 1$) and EVENODD ($r = 2$). There is a generalized EVENODD algorithm that recovers from three failures, but the authors wanted to reduce its decoding complexity further and so propose a new algorithm, called STAR.

Cheng exemplified the recovery schemes for the EVENODD and STAR algorithms and showed how the algorithms recover from failures. The extended EVENODD algorithm uses diagonal parities with slopes of one and two. STAR, however, uses diagonals with slopes of one and negative one. Cheng showed how this geometric symmetry used by STAR leads to faster decoding.

■ ***WEAVER Codes: Highly Fault Tolerant Erasure Codes for Storage Systems***

James Lee Hafner, IBM Almaden Research Center

Jim Hafner started by discussing why there is a need for another erasure code. The focus is on distributed storage systems and distributed RAID with more vulnerable components, and there is a need for another performance metric.

The proposal is a vertical code, with properties of symmetry, balance, and localization. Symmetry allows for easy implementation and natural load balancing. Localization means that I/Os do not involve the entire stripe. There is also more sequentiality from longer I/Os. The array size can be varied with fixed parity in-degree (number of inputs). Furthermore, the data-out degree is constant and equal to the fault tolerance.

The focus of this work is on codes with 50% efficiency. One of the features is variability of fault tolerance. The fault tolerance level can be changed by adding or subtracting an element without remapping or readdressing existing blocks. The disadvantage is that there is only 50% efficiency.

Ed Gould asked Jim to estimate how much fault tolerance is needed for a level of reconstructability of 90%. Jim answered that it depends on the components, as different batches of components from manufacturers have different errors. Also, it depends on the configuration and the combination of independent versus dependent domains.

WORK-IN-PROGRESS REPORTS

Summarized by Matthew Wachs

■ ***Controlling File System Write Ordering***

Nathan Burnett, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau, University of Wisconsin, Madison

The order in which writes occur to a file system must be controlled, or it may not be possible to restore the file system to a consistent state after a crash. While operating systems use techniques such as write-ahead logging to manipulate file system structures safely, it is difficult for an application to do the same for data consistency within its own files, because the operating system does not expose primitives for write ordering to the application. Nathan Burnett described two conventional ways that applications can ensure that commits to stable storage occur in the desired order: direct I/O and `fsync()`. Direct I/O allows applications to write directly to the raw storage device, avoiding all caches; however, it is slow (because of synchronous writes) and not portable (it is not universally available and APIs are not consistent). `fsync()` is portable, but slow as well. The fast alternative, ignoring write ordering, cannot ensure recoverability after a crash. Burnett suggested that the OS export an interface allowing the application to describe ordering constraints for writes. Not all writes may need to be ordered; taking advantage of this might yield better performance.

He proposed two methods for expressing ordering constraints: a `barrier()` system call which (globally) prohibits reordering of writes across the call; and asynchronous graphs, which express the constraints using an implicit graph data structure. In conventional applications, calls to `fsync()` could easily be replaced by calls to `barrier()`. The graph approach requires more extensive modifications:

write() returns an identifier for each call, and future invocations of write() can be passed a list of identifiers corresponding to writes (if any) that must occur first.

The authors have simulated both techniques and shown that the graph approach reduces the number of writes and the number of non-sequential writes in a TPC-B-like workload over both synchronous writes and the barrier approach, because it allows the coalescing, in some cases, of hundreds of small writes into one large write. They are now implementing the techniques in FreeBSD 5.4.

■ ***Rethink the Sync!***

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

Another way of thinking about durability is at a much higher level: transactions must not become visible externally until they have been committed to stable storage, but the application issuing them may continue executing while the transactions are still in volatile memory. When writes ultimately occur, they are performed in the order issued, maintaining the proper write ordering. Jason Flinn presented this approach as "external synchrony" or "visible synchrony." The authors are implementing this type of durability in the Linux kernel using mechanisms from their Speculator project (which addresses speculative execution in a distributed file system). Synchronous operations are performed asynchronously, and each operation is a transaction in the ext3 file system with data journaling.

Synchronous I/Os taint the calling process with an annotation prohibiting external output (such as to the screen or network) until all preceding I/Os are complete. If processes engage in IPC, taint annotations are inherited as appropriate. Because progress is being made on I/O in the background, the latency during which external output is withheld while pending commits finish is expected to be short enough not to be noticed by a human. Postmark results using visible synchrony show that performance is within 6% of an asynchronous implementation.

■ ***Amino: Extending ACID Semantics to the File System***

Charles Wright, Richard Spillane, Gopalan Sivathanu, and Erez Zadok, Stony Brook University

Applications such as mail servers and text editors often need to enforce transactional semantics on file manipulations: atomicity, consistency, isolation, and durability (known as ACID). While databases provide ACID semantics, there is no standardized interface to databases, which limits portability for applications that might use them. The availability of ACID at the file system would simplify error handling (transactions could simply be aborted), enhance security (time-of-check-to-time-of-use security vulnerabilities could be avoided by serializing concurrent accesses), and ensure durability.

Gopalan Sivathanu presented the idea of providing support for arbitrary transactions in the file system as a first-class service. To make this possible, the operating system itself must also support transactions at layers such as the cache. The authors have created a prototype file system, Amino, that provides begin, commit, and abort calls alongside the standard POSIX interface. Legacy applications automatically have each system call wrapped in a transaction; enhanced applications can wrap begin and commit calls around arbitrary sequences of POSIX I/O calls and computational activity. Back-end storage and transactional primitives are provided by Berkeley DB. The prototype is implemented in user level through a ptrace monitor, allowing existing applications to run unmodified and avoiding fundamental modifications to the OS.

■ ***PASS: Provenance-Aware Storage System***

Margo Seltzer, David Holland, Kiran-Kumar Muniswamy-Reddy, Uri Braun, Jonathan Ledlie, Harvard University

Provenance is metadata about the history of an object. For instance, if an application reads files A and B, then later writes file C, the provenance of file C includes files A and B, the application itself, and other environmental information that may have been used to derive C. Kiran-Kumar Muniswamy-Reddy explained that provenance is useful to scientists in understanding how results were arrived at, to homeland security applications in determining the information used to suggest a possible threat, and to business compliance systems in tweaking policies for information life-cycle management. He believes that the operating system and file system should be in charge of tracking provenance, because all data flows through them. Provenance should be a first-class entity which is automatically annotatable, indexable, and queryable; the authors are designing a storage system that meets these goals.

Muniswamy-Reddy highlighted several research questions: first, how provenance should be stored so that it is indexable and queryable; second, what the proper security model for provenance should be (does access to a file imply access to its provenance?); and third, how it can be sent over "the wire." A prototypical implementation added only 2% overhead for a Linux kernel build. More information can be found at http://www.eecs.harvard.edu/syrah/pass.

■ ***Logistical Storage***

Surya Pathak, Alan Tackett, and Kevin McCord, Vanderbilt University

Scientific computing, especially for efforts such as high energy physics, often requires sharing large data sets among collaborators around the world (for instance, some projects generate 3TB per day, 1PB per year). Surya Pathak introduced L-Store (Logistical Storage), a framework to address this need using software agent technology and the Internet Backplane Protocol. The software agents provide automated resource discovery and fault tolerance. The scalability of the authors' distributed approach has allowed them to achieve 10Gb/sec sustained reads and writes to distributed storage using a RAID-5 encoding on moderate hardware.

■ ***A Unifying Approach to the Exploitation of File Semantics in Distributed File Systems***

Philipp Hahn and Carl von Ossietzky, University of Oldenburg

Many distributed file systems exist, but few are widely used in practice. One reason for this may be the fact that they are often specialized for particular types of environments or applications. File systems that have seen widespread adoption because of their generality may suffer from "compromise" designs that optimize for average performance and excel at nothing. Philipp Hahn suggested that it would be ideal to have a universal abstraction for a distributed file system that allows for per-file optimizations and special cases and permits requirements to change over time. Various dimensions of configurability include concurrency, latency, availability, and consistency; the anticipated fault mode, access frequency, and access pattern; and the caching, versioning, encryption, and compression strategies employed.

Benefits from his work might include being able to bypass locking for backups, to avoid strong consistency in disconnected operation, to suppress replicas for temporary files, and to use different replica placement strategies for different files. He seeks to achieve this flexibility by creating a framework for a distributed file system with pluggable modules that allows the user to control all of these options up to administrator-configured limits, and falls back to a default configuration when none is specified. Hahn anticipates that self-tuning may relieve some of the burden of configuration.

■ ***A Centralized Failure Handler for File Systems***

Vijayan Prabhakaran, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau, University of Wisconsin, Madison

Commodity file systems have "broken" failure handling, because they assume that disks fail in a fail-stop manner. Moreover, failure handling is complex, because the code that actually performs I/O on behalf of applications is diffused throughout the system (for instance, journaling code or the sync daemon), and thus the code that must handle failures is distributed throughout the system. Vijayan Prabhakaran believes that this results in illogically inconsistent policies, because the reactions to the same error depend upon which component received it: some may propagate while others retry. It is also difficult to separate policies from mechanisms in this regime; and each component is subject to bugs.

The authors' proposed solution is a centralized failure handler, which addresses each of these shortcomings; it also relieves programmers of the need to add error-handling code to each new function, because the global handler already takes care of errors. Prabhakaran pointed out three issues with this approach: semantic information about a particular I/O needs to be available at the handler so it can respond to errors appropriately; the handler has parts that must be specialized to a particular file system while other parts are generic across file systems; and I/O paths are time-critical, requiring the common completion path to be separated from the error case.

■ ***Storage Benchmarking for HPC***

Mike Mesnier, James Hendricks, Raja R. Sambasivan, Matthew Wachs, and Gregory Ganger, Carnegie Mellon University; Garth Gibson, Carnegie Mellon University and Panasas

High-performance computing (HPC) applications are one important class of programs that use storage systems, but it is difficult to simulate their access patterns with existing benchmarks. In particular, the coordination and data dependencies between multiple compute nodes that are accessing storage may need to be modeled in a benchmark to capture the true nature of HPC workloads. Mike Mesnier discussed the idea of explicitly capturing this coordination in an existing workflow-specification language; specifications could then be used by a distributed workload simulator to synthetically generate multi-client accesses similar to those of a given HPC application.

The modeling language might include data sources and sinks with flows between them passing through compute nodes that perform transformations on the data. At the same time, the language must also incorporate I/O characteristics such as read/write ratio, request size, and randomness for the simulator to follow. The authors plan to select and extend an appropriate workflow modeling environment and to begin a repository of specifications expressed in this language by providing reference specifications—for example, HPC codes—and then soliciting the contributions of domain experts from different fields such as computational chemistry, bioinformatics, and so on.

■ ***POSIX I/O Extensions for HPC***

Brent Welch, Panasas

Just as it is important to benchmark HPC applications, so, too, is it fruitful to optimize for them at each level of a storage system. The POSIX semantics for I/O, which are intended for single-node access, are not ideal for environments using collective I/O and clustered compute-node access to shared storage. Brent Welch discussed an initiative, being undertaken by a large working group, to draft proposed API enhancements to POSIX that may boost the performance of this class of applications. Many of the changes are based on the ideas of relaxing expensive semantics and providing hints to the storage system. Among the proposals are support for vector I/O, coherence (propagation or invalidation of data), lazy attributes in metadata, locking schemes, shared file descriptors, and layout hinting. For instance, a `statlite()` call extends `stat()` to poll only those attributes actually needed by the application; and ACLs match the new NFSv4 semantics rather than the old POSIX ACL semantics. More information can be found at http://www.pdl.cmu.edu/posix.

■ ***Storing Trees on Disk Drives***

Medha Bhadkamkar, Fernando Farfan, Vagelis Hristidis, Raju Rangaswami, Florida International University

Many modern applications store tree-structured data, such as those using XML, those storing directory hierarchies, and those implementing suffix-tree alignments for bioinformatics. Because of this, being able to store tree-structured data efficiently is an important factor affecting the performance of these applications. Currently used schemes (such as relational databases or flat files) do not take advantage of the tree structure or the performance characteristics of disk drives.

Raju Rangaswami proposed tree-structured placement, a way of matching the data structure of a tree to the semi-sequential access patterns of a disk drive. Under this technique, the root is placed at the outermost track, with its children residing on the next free track, placed such that accessing the first child results in a semi-sequential access (that is, one which incurs no rotational delay because it falls under the disk head just as the seek to that track completes). Subsequent children are placed just after the first one and incur only a slight rotational delay. The drawbacks of this approach are high space fragmentation and poor random access times. A second strategy, the optimized tree-structured placement strategy, places child nodes in non-free tracks and permits some limited rotational latency to reach the first child on that track, increasing the flexibility of placement; it also stores multiple nodes in a single disk block. In the future, the authors plan to explore how to store arbitrary graphs more efficiently on disks.

■ ***Efficient Disk Space Management for Virtual Machines***

Abhishek Gupta and Norman Hutchinson, University of British Columbia

Virtual machines are being used for various purposes, but the problem of efficiently providing storage for each virtual machine has not been entirely solved. Frequently, multiple VMs share the same disk image and software configuration; existing solutions such as LVM (the Linux Volume Manager) and Parallax share blocks between the images and provide copy-on-write to achieve good space utilization. Abhishek Gupta described weaknesses in these systems: LVM has a high cost when the VM running on the master image overwrites a block (the original copy of the block must then be propagated to all the mirrored images or else they will see the changed block, unless they have performed a copy-on-write to that block). LVM also does not support hierarchical copy-on-write images (recursive snapshots).

Parallax can do recursive snapshots, but it is unclear how efficient it is: the cost of traversing the radix-tree data structure to translate a block address may be high, and there is no space reclamation. Gupta discussed how to explore possible solutions to these limitations: first, the authors have implemented radix trees in LVM so that they can be benchmarked and the existing solutions can be quantified; next, they will either try to fix the problems in current approaches or propose a new data structure that will support faster snapshots.

■ ***Intelligent Data Placement in a Home Environment***

Brandon Salmon, Carnegie Mellon University

Consumer media devices are proliferating in the home, and they are increasingly capable of handling high-quality videos, music, and photos. At the same time, the devices have varying degrees of mobility, storage capacity, and access to power. Because of this, Brandon Salmon highlighted the fact that there is a data synchronization problem in getting desired data to the right device at the right time. Currently, data transfer is typically done manually, which is not a scalable solution. Pushing data to all devices is not feasible, because power or capacity may be at a premium or some mobile devices may be out of range; yet on-demand access is not sufficient, because it is often not reliable.

Salmon's plan is to match data to appropriate destination devices by using metadata (such as ID3 tags in music files), easily observed access patterns, and machine learning to anticipate upcoming requests. Unlike hoarding, he plans to use information about data and device access patterns to match data to a device, rather than using inter-file access patterns alone. He also plans to leverage known cliques (such as a cell phone usually being near a laptop, but rarely near a DVR) to optimize caching.

■ ***Functionality Composition Across Layers in a Storage System***

Florentina Popovici, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau, University of Wisconsin, Madison

Various functions, such as caching, prefetching, layout, and scheduling, are implemented at each component in a layered system. For instance, a Web server may have a cache that duplicates data in the buffer cache, RAID controller cache, and hard disk cache, resulting in inefficient use of available memory. Florentina Popovici argued that exclusive caching would result in superior use of resources, and that analogous coordination of other functions such as prefetching and scheduling would similarly improve efficiency. She enumerated a number of research questions, such as where the best layer is to implement a particular algorithm (such as prefetching); how performance is influenced by a combination of decisions at different layers; and how quality of service is influenced by the hierarchy of layers.

■ ***Transaction Support in the Windows NTFS File System***

Surendra Verma, Microsoft

Windows Vista's NTFS file system implementation is expected to include ACID semantics for transactions consisting of arbitrary file-system operations. Surendra Verma gave a product demo of TxF, the code name for the transactional support in Vista, showing how file-system manipulations wrapped in transactions being performed in two different command prompt windows were not visible to each other. For instance, if a directory is deleted in one window but the transaction has not yet been committed, then the directory is still visible from the other window. Conflicts between concurrent transactions result in errors and aborted transactions to preserve the semantics.

ON THE MEDIA

No summaries available

■ ***On Multidimensional Data and Modern Disks***

Steven W. Schlosser, Intel Research Pittsburgh; Jiri Schindler, EMC Corporation; Stratos Papadomanolakis, Minglong Shao, Anastassia Ailamaki, Christos Faloutsos, and Gregory R. Ganger, Carnegie Mellon University

Awarded Best Paper!

■ ***Database-Aware Semantically-Smart Storage***

Muthian Sivathanu, Google Inc.; Lakshmi N. Bairavasundaram, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, University of Wisconsin, Madison

■ ***Managing Prefetch Memory for Data-Intensive Online Servers***

Chuanpeng Li and Kai Shen, University of Rochester

ON THE WIRE

Summarized by Kristal Pollack

■ ***A Scalable and High Performance Software iSCSI Implementation***

Abhijeet Joglekar, Michael E. Kounavis, and Frank L. Berry, Intel Research and Development

This work focused on iSCSI software solutions rather than common hardware implementations that use TCP/IP offload engines or iSCSI host bus adapters. An iSCSI software implementation offers the advantage that it can scale with CPU clock speed and the number of processors. Furthermore, it was shown in earlier work that hardware offload engines can become a bottleneck for small block sizes.

For the majority of their work, the authors used a user-level sandbox implementation of the iSCSI protocol, coupled with an optimized TCP/IP implementation. They discovered that the two main bottlenecks in iSCSI processing are CRC generation and data copies.They therefore set out to make these two operations more efficient.

They were able to improve CRC generation performance by a factor of 3 by replacing the standard CRC algorithm, developed over a decade ago by Sarwate, with a new algorithm, Slicing-by-8 (SB8), which takes advantage of more modern computer architectures. SB8 requires fewer operations per byte on the input stream and takes advantage of the size of the processor cache by using appropriately sized lookup tables. Data copy performance was improved by interleaving the CRC generation with the data-copy operations.

iSCSI processing performance in the authors' sandbox environment showed a factor-of-two improvement by changing the CRC algorithm to SB8. They gained an additional 32% performance improvement when the interleaving data copy with CRC generation was added. With both optimizations they improved throughput from 175MB/sec to 445MB/sec. SB8 was then implemented in UNHs iSCSI implementation and improved the overall throughput by 15%. The authors attribute this lower gain to the significant overheads in the Linux 2.4 implementation of SCSI and TCP/IP.

In the Q&A session someone asked why they chose 8 for their algorithm. It was from empirical results and may be processor-dependent. Another questioner asked how close they were to running at maximum memory—very close to maximum, almost memory limited, was the answer.

■ ***TAPER: Tiered Approach for Eliminating Redundancy in Replica Synchronization***

Navendu Jain and Mike Dahlin, University of Texas, Austin; Renu Tewari, IBM Almaden Research Center

TAPER is a solution for synchronizing data across distributed replicas. The authors' solution aims to minimize the bandwidth required for this task by using multiple phases of redundancy elimination. The authors introduced a method for quick elimination of identical files by using content-based hierarchical hash trees. They also developed a method for similarity detection using bloom filters.

These new methods were combined with existing techniques to form their overall protocol. In phase 1 they eliminate all identical files using their content-based hierarchical hash tree technique. In phase 2 they eliminate all identical data chunks in the remaining files by using content-defined chunks similar to LBFS. In phase 3 they use their bloom filter technique to find a similar file at the target for each file that has not been completely matched at the source. The unmatched pieces of the files at the source are broken into fixed-sized blocks, and their signatures are sent to the target. At the target these signatures are used in a sliding-block technique over the chosen similar file to find identical blocks. Finally, in phase 4 they use their bloom filter technique again for similarity detection between the remaining unmatched chunks and the already matched data at the source. The unmatched chunks are delta-encoded against the matched data, and the delta encodings are sent to the target to complete the synchronization.

TAPER was compared with rsync for several software sources, object binaries, and Web data sets. Gzip compression was used before sending data over the wire. In terms of bandwidth reduction, TAPER saved 18–25% for software sources, 32–39% for object binaries, and 12–57% for Web data when compared with rsync.

In the Q&A session someone asked if TAPER was compared with any other products, such as Tivoli. The answer was that rsync was the most relevant comparison. Another questioner asked if most of the savings came from phase 2. The answer was that 60% of the total savings came from the first two phases.

■ ***VXA: A Virtual Architecture for Durable Compressed Archives***

Bryan Ford, MIT CSAIL

Both general-purpose compression and multimedia encoding schemes have evolved rapidly over the past few decades. This presents a challenge for digital preservation of compressed data as encodings and the software to read them become obsolete. The author observes that instruction encodings are far more durable than data encodings. He points out that the x86 architecture has experienced few major changes over time, and has made efforts to be backwards-compatible. The author takes advantage of this observation by implementing Virtual eXecutable Archives (VXA), which save executable x86 decoders along with compressed data.

The VXA architecture uses a specialized virtual machine to run the decoders in. Decoders have access to computational primitives, but can only read from a given stream and write the decoding back. The decoders are extremely isolated and cannot use any of the operating system services. An implementation of this architecture was built using the zip/unzip tools. When compressed files are input into the system they are attached with decoders to the encoding they are already in. If the file can be compressed further, a lossless compression technique is used that best matches the file type, and the file is tagged with the appropriate decoder. When files are read, the appropriate decoder is loaded into the virtual machine, then executed on the stream of encoded data to produce the decoded data. The decoders are stored in a compressed format using a standard compression algorithm to reduce their overhead as well.

The performance for the VXA implementation was tested using six common decoders. The storage overhead for these ranged from 26KB to 130KB. The performance overhead on an x86-32 execution was 0–11%, while the performance overhead for the x86-64 execution was 8–31%. This can be attributed to the fact that the VXA decoders are 32-bit.

In the Q&A session someone asked if the system assumptions were violated by gzipping the gzip compiler. The answer was that even though he demonstrated VXA with open source decoders, the goal was really to use this system for proprietary encodings that are more likely to disappear. If one was worried that gzip might go away, the gzip compiler could be left unencoded. The next questioner was concerned that this is only for the x86 instruction set and wondered why it and not a universal one, such as Raymond Lorie's, was chosen. The answer was that we won't forget x86; it's ubiquitous. Someone asked if extracting semantic content was addressed, and the answer was no. The last question was, how do you ensure that decoder code is trusted and how do you verify that the sandbox environment is safe? The answer was that you have to trust the library for the emulator.

TOOLS

Summarized by Abhishek Gupta

■ ***I/O System Performance Debugging Using Model-Driven Anomaly Characterization***

Kai Shen, Ming Zhong, and Chuanpeng Li, University of Rochester

Performance problems in complex systems are hard to identify and debug, due to the presence of manifold system features and configuration settings coupled with dynamic workload behaviors and special cases. In a nutshell, the approach presented by Kai Shen is to construct simple and comprehensive models of system components using their corresponding high-level design algorithms. Later, discrepancies between model prediction and actual system performance are used to discover performance anomalies. In order to quantify these, Kai introduced the notion of a parameter space, a multi-dimensional space in which each workload condition and system configuration parameter is represented by a single dimension. The occurrence of a performance anomaly under one setting is identified as a single point in this space. It was observed that if samples were chosen randomly and independently, the chances of missing a bug decrease exponentially with the increase in the number of samples. Since, anomalous settings could be due to multiple bugs, a hyper-rectangular clustering algorithm was invented to offset the shortcomings of classical algorithms such as k-means.

For evaluation purposes these models were applied to data-intensive online servers hosted on Linux 2.6.10. These servers access large disk-resident data sets while serving multiple clients simultaneously. Using this scheme, four performance bugs in Linux were successfully discovered.

- ***Accurate and Efficient Replaying of File System Traces***

Nikolai Joukov, Timothy Wong, and Erez Zadok, Stony Brook University

Nikolai Joukov presented Replayfs, an accurate and efficient method to replay file system traces. Replayfs can replay traces faster than any known user-level system, and can even handle replaying of traces with spikes of I/O activity or high rates of events. In fact, with their optimizations in place, Replayfs can replay traces captured on the same hardware faster than the original program that produced the trace.

Nikolai opined that in developing a file system trace replayer it is often difficult to identify its suitable position within the operating system stack. To this extent, user-level replayers are easier to implement and thoroughly exercise the file system, but they do not support memory-mapped operations and have high memory/CPU overheads. Network-level replaying avoids the high memory/CPU costs, but it often misses out on client-side cached or aggregated events that do not translate into protocol messages. Replayfs overcomes all of these shortcomings by installing itself, as a kernel module, just beneath the VFS level and above classical file systems. In doing so it enjoys direct access to the buffer cache, exercises control over process scheduling, and benefits from reduced context switching, though at the cost of reduced portability.

During Q&A, Ralph Becker from IBM Almaden Research asked how Replayfs could handle traces from large-scale clusters. Nikolai replied that in such a case they would have to run Replayfs on multiple clients and be more intelligent while capturing traces. Daniel Ellard from Sun Microsystems wanted to know if the zero-copy optimization could be turned off. Nikolai replied, yes, it is configurable.

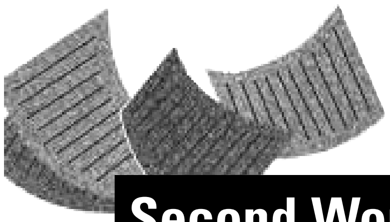
- ***TBBT: Scalable and Accurate Trace Replay for File Server Evaluation***

Ningning Zhu, Jiawu Chen, and Tzi-Cker Chiueh, Stony Brook University

In this talk, Ningning Zhu presented the design, implementation, and evaluation of TBBT, a comprehensive NFS trace replay tool. The author described TBBT as a turn-key solution that can automatically detect and repair missing operations in a trace, derive a file-system image required to successfully replay the trace, initialize and age the file-system image appropriately, and eventually drive the file server according to a user-configurable trace workload.

The author began her talk by highlighting the shortcomings of synthetic benchmarks, which are currently the most common workloads for file-system evaluations. Time-varying and site-specific parameters make it harder for synthetic benchmarks to mimic real-world workloads. Also, the time taken to develop a high-quality benchmark is often outpaced by the time taken for changes to trickle in to the workloads of specific target environments. TBBT is proposed as a complementary approach to synthetic benchmarks and is aimed at evaluating the performance of a file system/server on a site by capitalizing on the file access traces collected from that site.

During Q&A, someone from Seagate wanted to know how good this approach is in replaying the traces on a server that has capacities different from those of the one from which the traces were collected. The author replied that in order to evaluate this they would first have to classify traces according to localities within them.



Announcement and Call for Papers **USENIX**

Second Workshop on Hot Topics in System Dependability (HotDep '06)

Sponsored by USENIX, The Advanced Computing Systems Association

http://www.usenix.org/hotdep06

November 8, 2006 **Seattle, WA, USA**

HotDep '06 will be held immediately following the 7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6–8, 2006.

Important Dates

Paper submissions due: *July 15, 2006 (firm deadline, no extensions)*
Notification of acceptance: *August 31, 2006*
Final papers due: *September 18, 2006*

Workshop Organizers

Program Co-Chairs

George Candea, *EPFL* and *Aster Data Systems*
Ken Birman, *Cornell University*

Program Committee

Lorenzo Alvisi, *University of Texas at Austin*
David Andersen, *Carnegie Mellon University*
Andrea Arpaci-Dusseau, *University of Wisconsin, Madison*
Mary Baker, *Hewlett-Packard Labs*
David Bakken, *Washington State University*
Christof Fetzer, *Technical University of Dresden*
Roy Friedman, *Technion—Israel Institute of Technology*
Indranil Gupta, *University of Illinois at Urbana-Champaign*
Farnam Jahanian, *University of Michigan and Arbor Networks*
Petros Maniatis, *Intel Research Berkeley*
Andrew Myers, *Cornell University*
David Oppenheimer, *University of California, San Diego*
Geoff Voelker, *University of California, San Diego*
John Wilkes, *Hewlett-Packard Labs*

Overview

Authors are invited to submit position papers to the Second Workshop on Hot Topics in System Dependability (HotDep '06). The workshop will be co-located with the 7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6–8, 2006. The HotDep '05 program is available at http://hotdep.org/2005.

The goal of HotDep '06 is to bring forth cutting-edge research ideas spanning the domains of fault tolerance/reliability and systems. HotDep will center on critical components of the infrastructures touching our everyday lives: operating systems, networking, security, wide-area and enterprise-scale distributed systems, mobile computing, compilers, and language design. We seek participation and contributions from both academic researchers and industry practitioners to achieve a mix of long-range research vision and technology ideas anchored in immediate reality.

Position papers of a maximum length of 5 pages should preferably fall into one of the following categories:

- describing new techniques for building dependable systems that represent advances over prior options or might open new directions meriting further study
- revisiting old open problems in the domain using novel approaches that yield demonstrable benefits
- debunking an old, entrenched perspective on dependability
- articulating a brand-new perspective on existing problems in dependability
- describing an emerging problem (and, possibly, a solution) that must be addressed by the dependable-systems research community

The program committee will favor papers that are likely to generate healthy debate at the workshop, and work that is supported by implementations and experiments or that includes other forms of validation. We recognize that many ideas won't be 100% fleshed out and/or entirely backed up by quantitative measurements, but papers that lack credible motivation and at least some hard evidence of feasibility will be rejected.

Topics

Possible topics include but are not limited to:

- automated failure management, which enables systems to adapt on the fly to normal load changes or exceptional conditions
- techniques for better detection, diagnosis, or recovery from failures
- forensic tools for use by administrators and programmers after a failure or attack
- techniques and metrics for quantifying aspects of dependability in specific domains (e.g., measuring the security, scalability, responsiveness, or other properties of a Web service)
- tools/concepts/techniques for optimizing tradeoffs among availability, performance, correctness, and security
- novel uses of technologies not originally intended for dependability (e.g., using virtual machines to enhance dependability)
- advances in the automation of management technologies, such as better ways to specify management policy, advances on mechanisms for carrying out policies, or insights into how policies can be combined or validated

Deadline and Submission Instructions

Authors are invited to submit position papers by 11:59 p.m. PDT on July 15, 2006. **This is a hard deadline—no extensions will be given.**

Submitted position papers must be no longer than 5 single-spaced 8.5" x 11" pages, including figures, tables, and references; two-column format, using 10-point type on 12-point (single-spaced) leading; and a text block 6.5" wide x 9" deep. Author names and affiliations should appear on the title page.

Papers must be in PDF format and must be submitted via the Web submission form, which will be available on the Call for Papers Web site, http://www.usenix.org/hotdep06/cfp.

Authors will be notified of acceptance by August 31, 2006. Authors of accepted papers will produce a final PDF and the equivalent HTML by September 18, 2006. All papers will be available online prior to the workshop and will be published in the Proceedings of HotDep '06.

Simultaneous submission of the same work to multiple venues, submission of previously published work, and plagiarism constitute dishonesty or fraud. USENIX, like other scientific and technical conferences and journals, prohibits these practices and may, on the recommendation of a program chair, take action against authors who have committed them. In some cases, program committees may share information about submitted papers with other conference chairs and journal editors to ensure the integrity of papers under consideration. If a violation of these principles is found, sanctions may include, but are not limited to, barring the authors from submitting to or participating in USENIX conferences for a set period, contacting the authors' institutions, and publicizing the details of the case.

Authors uncertain whether their submission meets USENIX's guidelines should contact the program co-chairs, hotdep06chairs@usenix.org, or the USENIX office, submissionspolicy@usenix.org.

Registration Materials

Complete program and registration information will be available in September 2006 on the conference Web site. The information will be in both HTML and a printable PDF file. If you would like to receive the latest USENIX conference information, please join our mailing list: http://www.usenix.org/about/mailing.html.

Announcement and Call for Participation

20th Large Installation System Administration Conference (LISA '06)

Sponsored by USENIX and SAGE

http://www.usenix.org/lisa06

December 3–8, 2006 **Washington, D.C., USA**

Important Dates

Extended abstract and paper submissions due: *May 23, 2006*
Invited talk proposals due: *June 1, 2006*
Notification to authors: *July 12, 2006*
Final papers due: *September 12, 2006*

Conference Organizers

Program Chair
William LeFebvre, *Independent Consultant*

Program Committee
Narayan Desai, *Argonne National Laboratory*
Peter Galvin, *Corporate Technologies, Inc.*
Trey Harris, *Amazon.com*
John "Rowan" Littell, *Earlham College*
Adam Moskowitz, *Menlo Computing*
Mario Obejas, *Raytheon*
Tom Perrine, *Sony Computer Entertainment America*
W. Curtis Preston, *GlassHouse Technologies*
Amy Rich, *Tufts University*
Marc Staveley, *SOMA Networks, Inc.*
Rudi Van Drunen, *Leiden Cytology and Pathology Labs*
Alexios Zavras, *IT Consultant*

Invited Talk Coordinators
David N. Blank-Edelman, *Northeastern University CCIS*
Doug Hughes, *Global Crossing*

Guru Is In Coordinator
Philip Kizer, *Estacado Systems*

Workshops Coordinator
Luke Kanies, *Reductive Labs*

Get Involved!

Experts and old-timers don't have all the good ideas. This is your conference, and you can participate in many ways:

- Submit a draft paper or extended abstract for a refereed paper.
- Propose a tutorial topic.
- Suggest an invited talk speaker or topic.
- Share your experience by leading a Guru Is In session.
- Submit a proposal for a workshop.
- Present a Work-in-Progress Report (WiP).
- Organize or suggest a Birds-of-a-Feather (BoF) session.
- Email an idea to the Program Chair: lisa06ideas@usenix.org.

Overview

The annual LISA conference is the meeting place of choice for system, network, database, storage, security, and all other computer-related administrators. Administrators of all specialties and levels of expertise meet at LISA to exchange ideas, sharpen old skills, learn new techniques, debate current issues, and meet colleagues and friends.

People come from over 30 different countries to attend LISA. They include a wide range of administration specialties. They hail from environments of all sorts, including large corporations, small businesses, academic institutions, and government agencies. Attendees are full-time, part-time, student, and volunteer admins, as well as those who find themselves performing "admin duties" in addition to their day jobs. They support combinations of operating systems ranging from open source, such as Linux and the BSD releases, to vendor-specific, including Solaris, Windows, Mac OS, HP-UX, and AIX.

Refereed Papers

Refereed papers explore techniques, tools, theory, and case histories that extend our understanding of system and network administration. They present results in the context of previous related work. The crucial component is that your paper present something new or timely; for instance, something that was not previously available, or something that had not previously been discussed in a paper. If you are looking for ideas for topics that fit this description, the Program Committee has compiled a list of some good open questions and research areas, which appear in a separate section below. This list is not meant to be exhaustive; we welcome proposals about all new and interesting work.

It is important to fit your work into the context of past work and practice. LISA papers must provide references to prior relevant work and describe the differences between that work and their own. The online Call for Papers, http://www.usenix.org/lisa06/cfp, has references to several resources and collections of past papers.

Proposal and Submission Details

Anyone who wants help writing a proposal should contact the Program Chair at lisa06chair@usenix.org. The conference organizers want to make sure good work gets published, so we are happy to help you at any stage in the process.

Proposals can be submitted as draft papers or extended abstracts. Draft papers are preferred. Like most conferences and journals, LISA requires that papers not be submitted simultaneously to more than one conference or publication and that submitted papers not be previously or subsequently published elsewhere for a certain period of time.

Draft papers: A draft paper proposal is limited to 16 pages, including diagrams, figures, references, and appendices. It should be a complete or near-complete paper, so that the Program Committee has the best possible understanding of your ideas and presentation.

Extended abstracts: An extended abstract proposal should be about 5 pages long (at least 500 words, not counting figures and references) and should include a brief outline of the final paper. The form of the full paper must be clear from your abstract. The Program Committee will be attempting to judge the quality of the final paper from your abstract. This is harder to do with extended abstracts than with the preferred form of draft papers, so your abstract must be as helpful as possible in this process to be considered for acceptance.

General submission rules:

- All submissions must be electronic, in ASCII or PDF format only. ASCII format is greatly preferred. Proposals must be submitted using a Web form located on the LISA '06 Call for Papers Web site, http://www.usenix.org/lisa06/cfp.
- Submissions containing trade secrets or accompanied by nondisclosure agreement forms are not acceptable and will be returned unread. As a matter of policy, all submissions are held in the highest confidence prior to publication in the conference proceedings. They will be read by Program Committee members and a select set of designated outside reviewers.
- Submissions whose main purpose is to promote a commercial product or service will not be accepted.
- Submissions can be submitted only by the author of the paper. No third-party submissions will be accepted.
- All accepted papers must be presented at the LISA conference by at least one author. One author per paper will receive a registration discount of $200. USENIX will offer a complimentary registration for the technical program upon request.
- Authors of an accepted paper must provide a final paper for publication in the conference proceedings. Final papers are limited to 16 pages, including diagrams, figures, references, and appendices. Complete instructions will be sent to the authors of accepted papers. To aid authors in creating a paper suitable for LISA's audience, authors of accepted proposals will be assigned one or more shepherds to help with the process of completing the paper. The shepherds will read one or more intermediate drafts and provide comments before the authors complete the final draft.
- Simultaneous submission of the same work to multiple venues, submission of previously published work, and plagiarism constitute dishonesty or fraud. USENIX, like other scientific and technical conferences and journals, prohibits these practices and may, on the recommendation of a program chair, take action against authors who have committed them. In some cases, program committees may share information about submitted papers with other conference chairs and journal editors to ensure the integrity of papers under consideration. If a violation of these principles is found, sanctions may include, but are not limited to, barring the authors from submitting to or participating in USENIX conferences for a set period, contacting the authors' institutions, and publicizing the details of the case. Authors uncertain whether their submission meets USENIX's guidelines should contact the program chair, lisa06chair@usenix.org, or the USENIX office, submissionspolicy@usenix.org.

For administrative reasons, every submission must list:

1. Paper title, and names, affiliations, and email addresses of all authors. Indicate each author who is a full-time student.
2. The author who will be the contact for the Program Committee. Include his/her name, affiliation, paper mail address, daytime and evening phone numbers, email address, and fax number (as applicable).

For more information, please consult the detailed author guidelines at http://www.usenix.org/events/lisa06/cfp/guidelines.html. **Proposals are due May 23, 2006.**

Training Program

LISA offers state-of-the-art tutorials from top experts in their fields. Topics cover every level from introductory skills to highly advanced. You can choose from over 50 full- and half-day tutorials covering everything from performance tuning, through Linux, Solaris, Windows, Perl, Samba, TCP/IP troubleshooting, security, networking, network services, backups, Sendmail, spam, and legal issues, to professional development.

To provide the best possible tutorial offerings, USENIX continually solicits proposals and ideas for new tutorials. If you are interested in presenting a tutorial or have an idea for a tutorial you would like to see offered, please contact the Training Program Coordinator, Daniel V. Klein, at tutorials@usenix.org.

Invited Talks

An invited talk discusses a topic of general interest to attendees. Unlike a refereed paper, this topic need not be new or unique but should be timely and relevant or perhaps entertaining. An ideal invited talk is approachable and possibly controversial. The material should be understandable by beginners, but the conclusions may be disagreed with by experts. Invited talks should be 60–70 minutes long, and speakers should plan to take 20–30 minutes of questions from the audience.

Invited talk proposals should be accompanied by an abstract describing the content of the talk. You can also propose a panel discussion topic. It is most helpful to us if you suggest potential panelists. Proposals of a business development or marketing nature are not appropriate. Speakers must submit their own proposals; third-party submissions, even if authorized, will be rejected.

Please email your proposal to lisa06it@usenix.org. **Invited talk proposals are due June 1, 2006.**

The Guru Is In Sessions

Everyone is invited to bring perplexing technical questions to the experts at LISA's unique The Guru Is In sessions. These informal gatherings are organized around a single technical area or topic. Email suggestions for Guru Is In sessions or your offer to be a Guru to lisa06guru@usenix.org.

Workshops

One-day workshops are hands-on, participatory, interactive sessions where small groups of system administrators have discussions ranging from highly detailed to high-level.

A workshop proposal should include the following information:

- Title
- Objective
- Organizer name(s) and contact information
- Potential attendee profile
- An outline of potential topics

Please email your proposal to lisa06workshops@usenix.org.

Work-in-Progress Reports (WiPs)

A Work-in-Progress Report (WiP) is a very short presentation about work you are currently undertaking. It is a great way to poll the LISA audience for feedback and interest. We are particularly interested in presentations of student work. To schedule your short report, send email to lisa06wips@usenix.org or sign up on the first day of the technical sessions.

Birds-of-a-Feather Sessions (BoFs)

Birds-of-a-Feather sessions (BoFs) are informal gatherings organized by attendees interested in a particular topic. BoFs will be held in the evening. BoFs may be scheduled in advance by emailing bofs@usenix.org. BoFs may also be scheduled at the conference.

Suggested Topics for Authors and Speakers

Challenges

- Architecture: Keeping an infrastructure up to date
- Autonomic computing: Self-repairing systems, zero administration systems, fail-safe design
- Configuration management
- Content and collaborative systems: Building them, managing them, growing them
- Data center design: Modern methods, upgrading old centers
- Scaling: Dealing with a doubling in storage, backup, networking, address space, database, etc.
- Spam: Will we ever be able to control it?
- Virtualization: Benefit or bane?
- Viruses: Preparing for the onslaught of UNIX viruses

Profession

- Information flows: Creating, understanding, and applying them
- Management: Transitioning from technical to managerial, "managing" your manager, measuring return on investment
- Metrics: Inventing and applying meaningful metrics
- Outsourcing/offshoring system administration: Is it possible?
- Proactive administration: Transitioning from a reactive culture
- Problem-solving: Training sysadmins to solve problems
- Quality: Control, measurement, and quality assurance
- Standardizing methodologies: IT Infrastructure Library (ITIL), best practices, practical use, applicability

Technologies

- IPv6: Deployment, large-scale implementation
- Peer-to-peer networking
- Scripting languages
- VoIP
- XML: Usage for configuration management, other potential applications

Tools

- Content and collaborative systems
- Diagnostics: Tools that explain what's wrong
- Google tools: Making use of them in system administration
- Tools for system administration: Implementation, use, and applicability

Case Studies

- Practical implementations and deployments of ideas or tools
- Scaling and expanding an infrastructure
- Theory meets practice

Contact the Chair

The Program Chair, William LeFebvre, is always open to new ideas that might improve the conference. Please email any and all ideas to lisa06ideas@usenix.org.

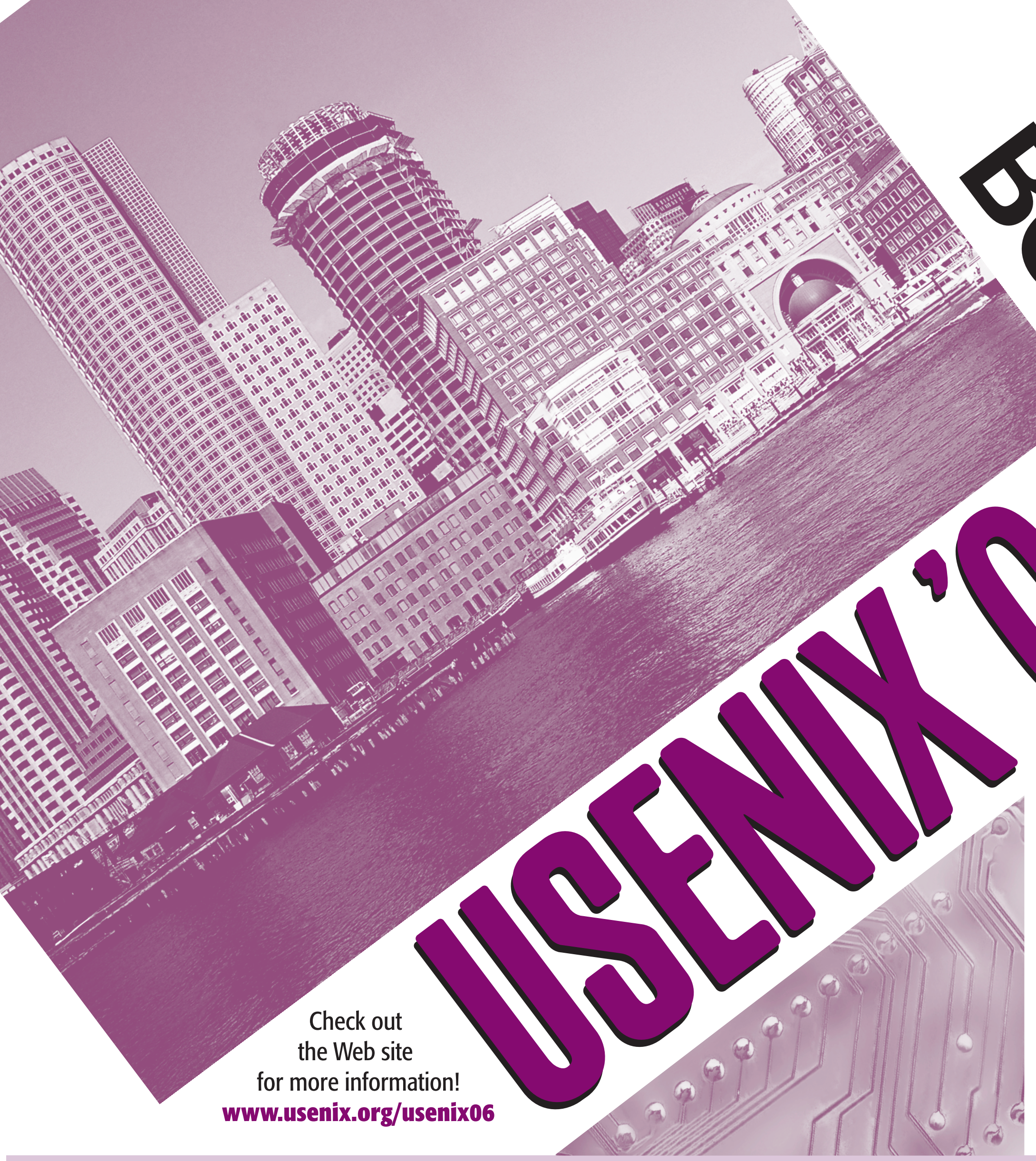
Final Program and Registration Information

Complete program and registration information will be available in September 2006 at the conference Web site, http://www.usenix.org/lisa06. If you would like to receive the latest USENIX conference information, please join our mailing list at http://www.usenix.org/about/mailing.html.

Sponsorship and Exhibit Opportunities

The oldest and largest conference exclusively for system administrators presents an unparalleled marketing and sales opportunity for sponsoring and exhibiting organizations. Your company will gain both mind share and market share as you present your products and services to a prequalified audience that heavily influences the purchasing decisions of your targeted prospects. For more details please contact sponsorship@usenix.org.

Rev. 1/17/06



May 30 – June 3, 2006

Boston Marriott Copley Place

BOSTON

USENIX '06

Annual Technical Conference

Join us in Boston for 5 days of groundbreaking research and cutting-edge practices in a wide variety of technologies and environments.

Don't miss out on:

- **Extensive Training Program featuring expert-led tutorials**
- **New! Systems Practice & Experience Track (formerly the General Session Refereed Papers Track)**
- **Invited Talks by industry leaders**
- **And more**

Please note: USENIX '06 runs Tuesday–Saturday.

Check out the Web site for more information!
www.usenix.org/usenix06

www.usenix.org/usenix06

;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