motd

by Rob Kolstad

Rob Kolstad is currently Executive Director of SAGE, the System Administrators Guild. Rob has edited USENIX's *;login:* magazine for over ten years.

kolstad@sage.org

Ch-ch-ch-changes

"Change is the only constant."

– Denise McCluggage, U.S. race car driver.

I have a new job. I know what you're thinking; it surprised me, too.

I am the new SAGE Executive Director. I am charged with carrying out the wishes of the Executive Board of the System Administrators Guild. You'd just be amazed at how many wishes one can conjure up when there's someone to carry them out :) .

One of SAGE's primary stated goals is to 'raise the perceived level of professionalism of System Administrators'. I think this is a dandy goal and I've taken it upon myself to create plans to accomplish this goal.

One of the interesting parts about such goals is the prerequisites required to accomplish the goal. For instance, in order for SAGE to "represent the interests of System Administrators," SAGE must acquire some sort of standing in the community. This might mean having enough members to claim to represent some significant fraction of the entire community (20%?). I reckon that one could claim 750,000 System Administrators in the USA alone (that includes network and security admins, but not help desk personnel). That means expanding SAGE's membership to six figures from its current four.

Why would someone affiliate with SAGE? I'm guessing that people do like to belong to clubs and the like, but rational people do enjoy having a reasonable set of membership benefits that are supplied in exchange for any money required as organizational dues.

To that end, I'm working on several projects to expand SAGE's visibility and bring more people into the fold. Hardly any of these is a revenue generator (i.e., they are free for all comers), but they will get SAGE and its good work in front of an ever growing number of people.

David Parter discusses the developing Web site, particularly SAGEwire, in his article later in this issue. This news and discussion forum is an ideal place for users who like to "pull" their news. The website will also feature white papers (1-20 page missives that dissect or digest a relevant topic). Please let me know if you'd like to write one.

I am also working to create an e-mail newsletter (biweekly?) that will be sent to those who prefer to have their system administration information "pushed" to them. If you'd like to be one of the newsletter writers, please let me know. It's a great way to contribute and be recognized.

I am also in the process of contacting other members of the media in order to get SAGE's word out through other publications. As we ramp the number of projects (certification, newsletters, white papers, SAGEwire, etc.), this is the sort of news that other trade publications enjoy publishing.

On another front, "professionalism" also connotes mastery of a body of knowledge. You've probably already heard about cSAGE, the SAGE certification effort (check out the SAGE website to learn more). But a profession not only has certification, it needs two more very important things (among others):

- A body of knowledge that encompasses what its members know
- A way to obtain a university degree in that field

Work continues on the Sysadmin Book of Knowledge. John Sechrest has been leading the charge to design a viable university curriculum. Both Mark Burgess and Alva Couch are assisting in the best possible way by initiating work in the theoretical bases of system administration. When coupled with David Paterson's work, it appears that sysadmin research is becoming a warm topic (if not a hot one).

John and I will be leading a day-long workshop at LISA that will unite those working on the Body of Knowledge and those working on education for system administrators. If you have interests in either of these areas, I hope you will come.

Like any other successful technical/trade organization, volunteers are the lifeblood that makes everything work. If you have ideas or would like to contribute in any way, please contact me and tell me what you'd like to do! Together, we can build SAGE into a world-wide success story.

apropos

What's New?

by Tina Darmohray

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

<tmd@usenix.org>

"What's new?" is a common greeting between friends, usually exchanged in addition to an opening "How ya doin'?" if you haven't seen the person for awhile. At the USENIX Annual Technical Conference in Monterey in June, there was plenty of opportunity to find out "What's new?" There were new attendees to meet, old friends to catch up with, and a wealth of cutting edge technical topics to learn about. For me, personally, I started my two-year stint as a new board member at the board meeting held during the conference, which turned out to be a new feeling, as do most new things.

I've attended USENIX board meetings before. I've always found them interesting and felt that I learned from them. I'd go away from those meetings feeling informed, educated, entertained, and often energized by what I'd heard. This one was different. I still feel informed and educated, but the entertainment and energy levels are way down. This may be partly due to the marathon meeting, from 9 a.m. to well after 7 p.m., but I think, more likely, it was because of an unspoken change in status. I think the reality of the new responsibility took a bite out of the previous entertainment and energy value I'd experienced.

USENIX is also dealing with new issues. To be expected, the attendance at our conferences is down. Since conference venues of this size are booked years in advance, we have financial obligations to the conference site, regardless of our attendance figures, or even whether we hold the conference at all. The bottom line is that if we don't meet an expected level of attendance, we lose money. That's been the case since September 11 and is a significant concern for the organization until things turn around.

Probably the other top topic is on the SAGE front. The new position of SAGE Executive Director, filled by Rob Kolstad, and the ongoing certification backing represent an unprecedented level of financial commitment to SAGE on behalf of USENIX. I'm optimistic that this stepped-up support will enable SAGE to develop some programs that will both enhance and diversify our services and offerings such that the community will benefit and our organizations will be less dependent on a single type of income. On a personal note, I know how pivotal the Executive Director position is, and I can't think of a better person to fill the position of SAGE Executive Director, since Rob is the co-founder of the LISA conference and was instrumental in the early success of the SANS conferences as well.

Certainly not new is the desire of board members for input and feedback from the membership regarding any of the issues facing the organization. I know I'd personally love to hear from you if you have thoughts about what we're doing right, what we're doing wrong, or what we're not doing! Even if you've never contacted a board member before, go ahead, try something new!

;login:

EDITORIAL STAFF

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

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

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

COPY EDITOR:
Steve Gilmartin

TYPESETTER:
Festina Lente

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

high availability

by Hobbit

Hobbit espouses a straightforward, no-frills approach to infrastructure design and risk mitigation, which he has brought into environments spanning small home networks to large ASPs. He currently develops methodologies and tools for secure, scalable network and host deployment. He is perhaps best known as the author of Netcat, a useful tool that has found its way into many open-source operating system distributions.

hobbit@avian.org

EDITOR'S NOTE: THIS ARTICLE WAS FORWARDED TO ME, SO I ASKED HOBBIT IF WE MIGHT PUBLISH IT. I FOUND IT QUITE PROVOCATIVE.

I find myself wondering why so many organizations get their panties in such a bunch over high availability and failover. Many organizations that insist on it arguably don't need it, at least in most people's sense of HA as it's sold into the industry today. Your average small-to-medium business can get by just fine without it, which is a good argument for them to use something like IPF that doesn't do HA but solidly serves all of their other needs. A sensible management strategy will go a lot further than any of these vaunted HSRP/VRRP toys.

What does full HA buy you? In the *very rare* event of total firewall failure, session states aren't lost and the load quickly shifts to the standby box. Whoo, whoo. Most people are just surfing the Web, making new connections over and over, and wouldn't notice if they lost a TCP state or two. People who lose an SSH session will simply blame Sprint's half-million dollar piece of crap in Pennsauken and reconnect after the network starts working again. Streaming media will likely pick right back up again. In general, users are much more resilient than any of the HA proponents give them credit for. For this large community of users, some form of non-transparent failover is *fine*, and I really wish the HA vendors would get off their high horse about how *everyone* needs 5 nines of uptime. They don't. Get over it.

Besides, in the event of "seamless" failover, the firewall administrator is unlikely to realize that it has happened either, and won't know that box A had some problem until box B finally fails too. At which point everyone is *hosed* because now someone has to go and rebuild *both* firewalls and, well, the config probably never got properly backed up because the admin thought, "Oh, well, it's all this swoopy HA stuff, so I don't have to really worry about backups." Oops.

Some from the financial sector might point out how when their networks go down, they start losing millions of dollars per minute. (Losing to where, I wonder, if the bits representing value cannot be transferred in the first place?) Okay, so maybe they need seamless HA for their current business model, but let us also consider that if the financial industry has painted itself into a corner badly enough to wholly depend on continual transfer of data, there exists a much deeper problem beyond the scope of this rant. If I had a buck for every stupid, unrealistic SLA that has been drafted without taking such contingencies into account, I wouldn't have to worry much about finances anymore.

What is possibly a much more realistic (and cheaper) strategy is the warm or cold standby – a second box configured just like the first one, ready to go any time, with "failover" involving a junior NOC monkey walking out to the right rack and moving two or three clearly labeled wires. Then someone knows that box A has failed and can unrack it for repairs at leisure. Box B will likely hold up the load for the next year and a half without a hitch, so the priority of fixing box A becomes very flexible. Maybe it was time to upgrade to a faster machine for that spot anyways, eh?

I'm sure that more "automated" failover can be done with IPF and a little bit of scripting to run around and ifconfig interfaces and ping a few surrounding devices with the hope of forcing ARP and switching tables to update. The commercial HA products often have trouble convincing neighboring switches and things that no, this MAC address is over *here* now, really, at which point that multi-thousand-dollar HA setup still needs a swift kick. A successful semi-automated failover combined with user

resiliency gives you the same *long-term* net effect as the much more expensive "HA product." And a closer look inside a lot of HA products often reveals a nest of grotty shell scripts that do just that too, so don't be fooled.

So if you have an HA setup, go figure out how many times over the last year it has done its job in such a way that it even came close to paying for itself. Include all the factors such as administrative overhead, having to learn about the product, the engineering time spent integrating it into your network environment in a way that genuinely works, the effect of network design compromises you may have had to make to do so, etc. Don't forget to consider the increased *risk* that the lame JavaScript-laced HA management GUI has caused you to bear all that time, since it required you to open more avenues into the firewalls themselves. And were you ever able to get your run-of-the-mill NOC guy to understand the thing?

If your "HA" setup consists of swapping the hard drive and network cards into a new chassis and powering back up, be thankful and count those nice crisp hundreds you saved. And if you've read this far, go back up your nice simple little text-based rule set, if it's been a while, just in case it's the drive that craps out instead. But you might wait another two years for that to actually happen.

USENIX and SAGE Need You

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

The SAGEwire and SAGEweb staff are seeking:

- Interview candidates
- Short article contributors (see *http://sagewire.sage.org*)
- White paper contributors (for topics like these):

Back-ups	Emerging technology	Privacy
Career development	User education/training	Product round-ups
Certification	Ethics	SAGEwire
Consulting	Great new products	Scaling
Culture	Group tools	Scripting
Databases	Networking	Security implementation
Displays	New challenges	Standards
E-mail	Performance analysis	Storage
Education	Politics and the sysadm	Tools: system

- Local user groups: If you have a local user group affiliated with USENIX or SAGE, please mail the particulars to *kolstad@sage.org* so they can be posted on the web-site.

;login: is seeking attendees of non-USENIX conferences who can write lucid conference summaries. Contact Tina Darmohray, *tmd@usenix.org* for eligibility and remuneration info. Conferences of interest include (but are not limited to): Interop, SOSP, O'Reilly Open Source Conference, Blackhat (multiple venues), SANS, and IEEE networking conferences. Contact *login@usenix.org*.

;login: always needs conference summarizers for USENIX conferences too! Contact Alain Hénon *ah@usenix.org* if you'd like to help.

computer forensics

by Erin Kenneally

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

erin@sdsc.edu

Beyond the Buzzword

What do the Chandra Levy disappearance, Enron/Arthur Anderson collapse, Danielle Van Dam murder case, Microsoft antitrust trial, former President Clinton sex scandal, and tracking of al Qaeda terrorists all have in common? In each instance, computer forensics figured prominently in investigating the questions at hand. Simply put, computer forensics has reached prime time. It is no longer the stuff of back-office geeks and techno-wizards but has been embraced by both law enforcement and the private sector as a technique to reconstruct crimes, conduct digital discovery, and, generally, uncover the electronic traces that help prove or disprove accounts of historical events.

The negative corollary to the consciousness-raising effect of being in the limelight is that "computer forensics" has become something of a buzzword among profit-savvy businesses seeking to market their "advanced capabilities." This has resulted in a dilution and misrepresentation of the evolving discipline of computer forensics. At the risk of having no agenda save for battling ignorance, this article is meant as a primer on the essence of computer forensics so that one can better appreciate and expect accurate, reliable, and scientifically based standards when encountering digital evidence issues.

Computer Forensics Defined

Odontology, structural engineering, pathology, serology, or analysis of computer systems are all methods used in forensic science. Since forensic science is the application of a scientific discipline to the law, the essence of all forensic disciplines concerns the principles applied to the detection, collection, preservation, and analysis of evidence to ensure its admissibility in legal proceedings. Computer forensics refers to the tools and techniques to recover, preserve, and examine data stored or transmitted in binary form. The application of forensic techniques to digital analysis, therefore, can be viewed as the new kid on the block more commonly populated by the likes of DNA fingerprinting and hair and fiber analysis. And instead of Quincy, ME, examining a corpse to determine cause of death, we're dealing with digital examiners conducting machine autopsies to recover evidence of a crime.

Analogizing Computer Forensics to Traditional Forensic Sciences

The fundamental principles of computer forensics are the same as that of traditional forensic disciplines. All start with intense variability among a large number of attributes and advances are aimed at enhancing the identifying, characterizing and correlative properties of the evidence source. Whereas an MD-5 hash may identify a digital document to the exclusion of all others, the remnants of a deleted Netbus application found in unallocated space may help correlate a suspect to victim's firewall log data of scans on port 12345 coming from the suspect's IP address.

Forensic techniques are designed to uncover these identifying, characterizing and correlative properties more precisely, more accurately, faster and with less evidence.

For instance, a comparison of analysis development between digital data versus biological data (blood) would illustrate how A/B/O typing gave way to Rh factors, which was supplanted by DNA typing via RFLP (restriction fragment length polymorphism)

and PCR (polymerase chain reaction) – which resulted in the same evidence source being used to characterize and then positively identify persons to the exclusion of all others. Similarly, forensic analysis techniques for digital evidence has yielded hash libraries (to identify data files), file signatures (to characterize files by matching filename and file type), and mirror imaging software (to copy larger amounts of evidence without altering the original evidence).

Regardless of whether the discipline is computer forensics or fingerprinting, the driving question is not whether evidence exists but, rather, can investigators uncover and contextualize the evidence. Therefore, the challenges are: Where to look? What techniques will make the evidence apparent? And is the evidence admissible?

Regardless of whether the discipline is computer forensics or fingerprinting, the driving question is not whether evidence exists but, rather, can investigators uncover and contextualize the evidence.

Just as a pathologist may deduce by observing the lack of water in a person's lungs that he was already dead when his car sank to the depths of a lake, computer forensic examiners may analyze file modification/access/creation times to determine if intellectual property was transferred after an employee was fired. And in the same way that sources of biological evidence may be blood, saliva or hair shafts found on clothing, cigarette butts, and weapons, digital evidence can be found on any number of media sources (hard drive, floppy disk, CD-ROM, PDA) and in locations such as print spooler files, hidden partitions, registries, system logs, bad clusters, and/or metafiles. In the biological realm, techniques such as PCR, RFLP, and STR (short tandem repeats) exist to identify DNA in a drop of dried blood that is not visible to the naked eye. In computer forensics, techniques exist to recover deleted data; recover passwords; analyze file slack, unallocated space, and swap files; reconstruct user and application activity on a system; and search email for source and content information.

Finally, in terms of admissibility hurdles, the technology to recover deleted data has been accepted, but what is contested is the inclusiveness of the software that undertakes to recover it – in other words, are there measurable error rates for the software that address the likelihood of missing potentially exculpatory evidence? Likewise, insofar as DNA fingerprinting technology has been accepted in the courtroom, certain techniques (like STR) remain open to challenge.

Contrasting Digital Evidence with Physical Evidence

Despite core similarities, the differences between computer forensic analysis and the more traditional forensic sciences bear reflection. From a historical perspective, computer forensics is a burgeoning discipline compared to traditional forensic sciences, many of which are rooted as far back as the early 20th century. One prominent difference lies in the diametric evolution of computer forensics as compared to traditional forensic sciences. Computer forensics originated in "cop shops" rather than clinical laboratory settings. Electronic tools and techniques have been developed to solve specific problems on known platforms within given parameters, rather than the more traditional application of scientific rigor and controlled testing to derive facts for crime solving to investigations for legal proof[1]. DNA analysis, for instance, was developed for non-forensic purposes and was only later applied for judicial purposes, unlike the forensic analysis software employed by digital technicians today.

To be sure, a grounding in scientific rigor is increasingly being recognized and applied to computer forensic tools and techniques to ensure the reliability and admissibility of digital evidence. However, unlike physical evidence, digital evidence poses novel challenges to computer forensic analysis. The mutable, fleeting, and intangible nature of digital evidence stands in contrast to persistent physical features used in other disci-

1. See generally, David Goodstein, "How Science Works" *Reference Manual on Scientific Evidence* 2nd Edition, Federal Judicial Center (2000) *http://air.fjc.gov/public/fjcweb.nsf/pages/74*

plines – i.e., ridge patterns for fingerprinting, polymarkers for DNA analysis, and bone characteristics for forensic anthropology. This fosters the advantage of conducting comparisons with known exemplars to uncover those identifying, characterizing, or correlative properties. However, the variables involved in complex computer network activity and software/hardware that produces digital evidence are dynamic and not as conducive to reproduction.[2]

Digital Evidence – Search and Seizure Challenges

Digital evidence has shifted paradigms in collecting, preserving, and analyzing evidence, as illustrated by the unique legal challenges facing computer forensic professionals. Specifically, these shifting paradigms can be appreciated by understanding the resource challenges, attempting to define "reasonableness," and paying heed to modification challenges presented by digital evidence.

RESOURCE ISSUES

The traditional approach when investigators would encounter a crime scene with a computer was to seize everything. That approach may have worked at a time when the ratio of computers to employees was 1:1 or greater, or when there was a stand-alone computer at a domestic crime scene. However, this approach is no longer feasible in a society where computers and their appendages dominate the landscape and information is increasingly being stored, transmitted, and created in digital form. Insofar as our ability to collect far outweighs our ability to analyze, the "seize everything" mentality is simply economically infeasible, both for budget-constrained public law enforcement as well as for private sector responders whose work is not part of the corporate profit center. Indeed, the cost of storage media has declined appreciably, yet the man-hour resources and capabilities to image and cull through hundreds of gigabytes worth of data on a compromised network is no small task. Relief does not appear imminent, as technologies such as FMD-ROMs, which store 140GB, may soon supplant CD and DVD media, and consumer grade hard drives are shipping at 75GB and up.

To put this resource challenge into context, imagine that a 1.44MB floppy disk holds the equivalent of a novel. Now, a standard 20GB home computer would produce the paper equivalent of a stack of books roughly as high as a fifteen-story building. Placed in the context of paper-based evidence, it is easier to appreciate how the nature of digital information and the relevant data contained therein strains the resources of forensic professionals who must uncover, collect, preserve, and analyze these electronic haystacks.

DEFINING REASONABLENESS

A fundamental right guaranteed by the Constitution is protection from unreasonable searches and seizures by the government. Courts have applied this protection by ensuring that search warrants are only issued upon a showing of probable cause, which is grounded in "reasonableness" and defined in terms of narrowness and particularity of scope. However, the time and scope variables (narrowness and particularity) that affect the reasonableness of the search and seizure take on a different dynamic. For instance, judges oftentimes authorize a search warrant with narrow time limits to minimize business disruption. In doing so, there is a faulty assumption that the scope of the search will be narrowed by decreasing the time allotted to conduct the search and seizure. In actuality, a narrow time frame will result in a wider scope of data seizure – thus increasing the chances of capturing irrelevant and overbroad data.

2. See Randolph Johnkait, "Forensic Science: The Need for Regulation," *Harv.J.L. & Tech.*, vol. 4, no. 109 (1991), 133–34.

Furthermore, search warrants often authorize authorities to search anywhere that the evidence in question can "reasonably" exist. So, a warrant for a gun would preclude investigators looking in a cell phone case. Digital evidence, however, is not bound by those same physical limits, so notions of what is reasonable must be put into a new context. For instance, large amounts of data can be hidden or compressed in a very small area, and file extension labels do not necessarily reflect the underlying data type (i.e., a strategic diagram in the form of a .jpg can be named "anything.txt").

EVIDENCE MODIFICATION CHALLENGES

Finally, the mutability of digital evidence facilitates legal challenges grounded in chain-of-custody and evidence-tampering arguments. Whereas DNA analysis is performed on the original blood evidence, maintaining the sanctity of original evidence is a tenet of computer forensics, and analysis must be conducted on a copy of the original media (with a few, notable exceptions where circumstances preclude a copy being made). Perhaps because the 33rd copy of a file is indistinguishable from the original and it is trivial to change bytes without leaving a trace, benign actions when handling digital evidence may have probative consequences upon which guilt or innocence hinges. Setting aside the wholesale substitution of blood evidence, if a serologist contaminates a blood sample, there is little risk that the DNA of another person will be created. Rather, the blood sample will not conclusively identify the culprit. With digital evidence, for example, merely turning on a Win95 system opens roughly 8% of the files on the hard drive just to boot the system. The consequence is that 417 access dates, some of which may have been crucial to proving guilt or innocence, have been altered .

So what? The criticality of timestamp data associated with file modification, access, or creation can make or break a case. For example, take the case where the digital evidence found on a defendant's computer was a large collection of adult porn (legal) and a smattering of kiddie porn images (illegal). Now, the defendant may claim that he downloads adult porn via IRC, and the kiddie porn must have been unintentionally downloaded at the same time, unbeknownst to the defendant. If this were true, computer forensic analysis might reveal access dates on the adult images well after the creation dates (initial download), but the child images had creation and access times that matched the creation times for the adult pictures. This would support the defense that he did not view or distribute the child porn. However, if the seizing officer booted the suspect machine and started rifling through the images, he would have changed the timestamps and quashed potentially exculpatory information.

The criticality of timestamp data associated with file modification, access, or creation can make or break a case.

Conclusion

In our increasingly electronic society, digital evidence promises to continue to permeate crime scenes and civil disputes, thus rendering computer forensics an increasingly vital discipline in the resolution of disputes. The danger is that computer forensics will be driven by industry and market forces that lose sight of the need for scientific underpinnings regarding computer forensic tools and techniques. Hopefully, this primer has served to raise awareness about the similarity in principles between computer forensics and the traditional forensic sciences, as well as highlighting the unique nature of digital evidence, so that the collection, preservation, and analysis of digital evidence will advance the search for truth.

wide characters

by Glen McCluskey

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

glenm@glenmccl.com

We've been looking at some of the new features in C99, the standards update to C. In this column we'll consider features added to the language and library in support of wide characters, as typically used with foreign languages.

Character Sets

Several terms are used in the standard to describe C character sets. The first of these is "basic character set" and refers to a set of single-byte characters that all C99 implementations must support. Roughly speaking, this character set is 7-bit ASCII without some of the control characters. It consists of printable characters like A–Z and 0–9, along with tab, form feed, and so on.

The basic character set is divided into source and execution character sets, and these differ slightly. For example, the basic execution character set is required to have a null character (\0), used as a string terminator.

The extended character set is a superset of the basic character set, and adds additional locale-specific characters. It, too, is divided into source and execution character sets.

A wide character is a character of type wchar_t, and is capable of representing any character in the current locale. In other words, a wide character may be a character from either the basic character set, such as the letter A, or a character from the extended character set.

Wide Character Constants

Let's look at some actual examples of wide character usage. The first demo program prints the size of wchar_t on your local system:

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

int main()
{
    printf("sizeof(wchar_t) = %u\n", sizeof(wchar_t));
}
```

When I run this program on my Linux system, the result is:

```
sizeof(wchar_t) = 4
```

wchar_t is a signed or unsigned integral type big enough to hold all the characters in the local extended character set, and is a 32-bit long on my system.

wide character constants are specified similarly to normal character constants, with a preceding L before the constant:

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

int main()
{
    wchar_t wc1 = L'a';
    printf("%lx\n", wc1);

    wchar_t wc2 = L'\377';
    printf("%lx\n", wc2);

    wchar_t wc3 = L'\x12345678';
    printf("%lx\n", wc3);
}
```

When I run this program, the result is:

```
61
ff
12345678
```

In the first two cases, the wide character is stored in the least significant byte of the long, while in the last case, all four bytes of the long are used to represent a single wide character.

This example illustrates a confusing point about wide characters – the idea of multiple representations. For example, wc3 is initialized with a wide character constant, a constant that requires 13 bytes to express in the source program. The constant itself is stored in four bytes during execution (in a 32-bit long). And a little later on, we'll see examples of what is called "state-dependent encoding," a mechanism used to encode wide characters as a stream of bytes (1–6 bytes per wide character, on my system). This encoding is used for writing wide characters to a file.

The term "multibyte character" is defined to be a sequence of one or more bytes that represents a single character in the extended source or execution environment. A character from the extended character set can have several different representations. These representations may appear in source code, in the execution environment, or in data files.

Here's another example, showing how wide character strings are specified:

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

int main()
{
    wchar_t* wstr1 = L"testing\x12345678";
    wchar_t wstr2[] = L"testing\x12345678";

    assert(*wstr1 == 't');
    assert(*(wstr1 + 7) == 0x12345678);

    assert(wstr2[0] == 't');
    assert(wstr2[7] == 0x12345678);
}
```

String Operations

Many familiar operations are supported on wide character strings. For example, here's a demo that implements a function to convert to lowercase:

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

wchar_t* tolower(wchar_t* str)
{
    wchar_t* start = str;

    // convert each wide character to lowercase

    for (; *str; str++) {
        if (iswupper(*str))
            *str = tolower(*str);
    }

    return start;
}

int main()
{
    wchar_t* str = L"TESTing";
    wchar_t buf[8];

    wcscpy(buf, str);

    tolower(buf);

    printf("%ls\n", buf);
}
```

Note that the definition of an uppercase character may be locale specific.

Wide Characters and I/O

Suppose that you have a wide character string and you'd like to write it to a file and then read it back. How can you do this? Here's one approach:

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

int main()
{
    // write a wide character string to a file

    FILE* fp = fopen("outfile", "w");
    assert(fp);
    fwprintf(fp, L"string is\377: %ls\n", L"TESTing\x1234");
    fclose(fp);

    // read the characters of the string back from the
    // file

    fp = fopen("outfile", "r");
    assert(fp);
    wint_t c;
    wchar_t buf[25];
    size_t len = 0;
    while ((c = getwc(fp)) != WEOF)
        buf[len++] = c;
    fclose(fp);
    buf[len] = 0;

    // check results

    if (wcscmp(buf, L"string is\377: TESTing\x1234\n")
                == 0)
        printf("strings are equal\n");
    else
        printf("strings are unequal\n");
}
```

Much of this code is identical to what you would use when reading and writing regular strings of bytes.

wint_t is a type that is related to wchar_t in a way similar to the relationship between int and char; it can hold all possible wchar_t values, as well as one distinguished value that is not part of the extended character set (WEOF).

The only other tricky thing in this example is stream orientation, something that's implicit in the code. A file stream can be either byte or wide oriented. The orientation is determined by the first operation on the stream, or explicitly via the fwide() call. Since the first write operation in the demo is fwprintf(), and the first read operation is getwc(), and these are wide character functions, the streams are marked as having wide orientation.

Why does stream orientation matter? The reason is that an encoding may be applied to wide characters written to a file. Suppose you are programming with wide characters, you need to do wide character I/O, and your wide characters are four bytes long when using the wchar_t representation. One way of writing such characters to a file is to actually write four bytes for each character.

But what happens if, most of the time, the values of your wide characters are within the range of 7-bit ASCII? In such a case, three zero bytes will be written for each character. And the resulting files will not be readable by tools that expect ASCII. This problem exists today, for example, with tools that write 16-bit Unicode to a file. One solution to this problem is to encode characters such that 7-bit ASCII is represented as itself, that is, a single byte, while other character values are encoded using multiple bytes.

But if an encoding is applied, then it no longer makes sense to mix byte and wide-file operations. This is especially true given that an encoding may be state dependent, and dipping into a byte stream in the middle of a multiple-byte encoding of a wide character has no meaning.

Encodings

Let's look a little deeper into the encoding issue, with another example. This demo converts wide character values into sequences of encoded bytes:

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

int main()
{
      char buf[MB_CUR_MAX];
      int n;

      // convert a single-byte character to a multibyte
      // character

      n = wctomb(buf, L'a');
      printf("len = %d\n", n);
      for (int i = 0; i < n; i++)
          printf("%hhx ", buf[i]);
      printf("\n");

      // convert another single-byte character

      n = wctomb(buf, L'\377');
      printf("len = %d\n", n);
      for (int i = 0; i < n; i++)
          printf("%hhx ", buf[i]);
      printf("\n");

      // convert a wide character

      n = wctomb(buf, L'\x12345678');
      printf("len = %d\n", n);
      for (int i = 0; i < n; i++)
      printf("%hhx ", buf[i]);
      printf("\n");
}
```

The output is:

```
len = 1
61
len = 2
c3 bf
len = 6
fc 92 8d 85 99 b8
```

The character a is encoded as itself, while the character \377 is encoded as two bytes 0xc3 and 0xbf. The constant L'\x12345678', internally represented as a four-byte long, is encoded using six bytes.

Here's another example of encoding and decoding:

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

int main()
{
      char buf[MB_CUR_MAX];
      wchar_t wc1 = L'\x12345678';
      wchar_t wc2;
      int n, nn;

      // convert a wide character to a multibyte
      // character

      n = wctomb(buf, wc1);
      printf("%d\n", mblen(buf, n));

      // reverse the process

      nn = mbtowc(&wc2, buf, n);

      // check result

      if (wc1 == wc2 && n == nn)
          printf("equal\n");
      else
          printf("unequal\n");
}
```

The wctomb() function encodes a wide character into a stream of bytes, and mbtowc() reverses the process.

Restartable Functions

Consider the second part of the last example. The processing of the first part of the example – a wide character encoded into a buffer of one or more bytes – was reversed, by taking the buffer and converting it back into a wide character.

In a real-world example, things might not be quite as simple. For instance, you might have an application where bytes are coming in across a network one at a time, and several of the bytes put together represent a wide character. You'd somehow like to keep track of the state of the decoding as each byte

comes in, and when a valid wide character is detected, process it.

As part of support for wide characters, C99 has a set of what are called restartable functions. The idea is that you initialize a state object used to keep track of the encoding or decoding state, and then you pass this object to the functions. Let's see how this idea works in practice:

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

int main()
{
    // convert a wide character into a byte stream

    char buf[MB_CUR_MAX];
    wchar_t wc1 = L'\x12345678';
    int len = wctomb(buf, wc1);
    printf("len = %d\n", len);

    // initialize mbstate_t object

    wchar_t wc2;
    mbstate_t mbstate;
    memset(&mbstate, 0, sizeof(mbstate_t));
    size_t retval;

    // convert the first len - 1 bytes of the byte stream

    retval = mbrtowc(&wc2, buf, len - 1, &mbstate);
    printf("retval = %d\n", retval);

    // convert the last byte of the byte stream

    retval = mbrtowc(&wc2, &buf[len - 1], 1, &mbstate);
    printf("retval = %d\n", retval);

    // compare with original

    if (wc1 == wc2)
        printf("equal\n");
    else
        printf("unequal\n");
}
```

In the first part of the example, a wide character is encoded into a stream of bytes. We then initialize an mbstate_t object and convert the stream of bytes back to a wide character. But in the first call to mbrtowc(), we omit the last input byte, implying that the conversion cannot be completed during this function call. The state object captures an intermediate state of conversion. The state object is then passed to the second mbrtowc() call, and the conversion is completed.

The result of running the demo is:

```
len = 6
retval = -2
retval = 1
equal
```

The initial -2 return value from the first mbrtowc() call indicates that a valid partial encoded wide character was found in the input byte stream.

Wide character support is especially useful if you're working with foreign languages. C applications often assume English and ASCII are being used, and the wide character type and library functions add support for other character sets.

practical perl

Programming with Iterators

by Adam Turoff

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

ziggy@panix.com

Most programs deal with examining a sequence of values at some point. In this column, we investigate iterators, a way to simplify processing of a computed sequence of values. We conclude by revisiting a common problem: parsing a configuration file, this time by using iterators.

Introduction to Iterators

I recently worked on a project where I needed to use the iterator design pattern. Design patterns are common structures and behaviors that occur frequently in many programs and are used across programming languages and application domains. The iterator pattern describes a behavior where an object offers sequential access to a data structure composed of many individual elements.

Iterators are more common in strongly typed programming languages. Java, for example, contains an Iterator interface as part of the core language definition. Classes that use this interface provide next() and hasNext() methods to enable programs to examine a series of values sequentially, one at a time. One particularly interesting use of iterators involves traversing a binary tree. One way to examine each node in the tree would be to code up a breadth-first traversal algorithm every time you need it. An easier way to examine the tree is to make successive calls to a next() method to retrieve each item from the tree in the proper order.

You may not have heard about iterators before just because they're not particularly necessary in Perl. Because Perl has a generic list data type already, there isn't a pressing need to create generic interfaces or design patterns to access list-type objects in a sequential manner.

Here are a few common examples of how iterators are commonly used in Perl. Built-in operators like foreach, map, and grep can work with any kind of data in a list, because lists are generic containers:

```
foreach (@ARGV) {
    ## process items in a list
}
## transform a list of values
## into a list of squares
my @squares = map {$_ ** 2} 1..10;
## reduce a list of files
## to a list of writable files
my @writable = grep {-w $_} </htdocs/*>;
```

Iterating over the lines in a file using a while loop is another common technique:

```
while (<FILE>) {
    ## process each line of FILE
}
```

Recall that foreach will examine lists one element at a time, but while loops execute until the test condition evaluates to false. That's why the while(<FILE>) idiom is shorthand for this construct:

```
while (defined($_ = <FILE>)) {
    ## process each line of FILE
}
```

The point here is that a line is read from FILE each time the block is executed. Looking at the while loop this way shows that we're not examining a sequence of values in a list, but examining a sequence of values generated dynamically. It just so happens in this case that these computed values are lines from a file. We could just as easily iterate over a series of rows coming from a database query using the DBI module:

```
use DBI;
my $dbh = DBI->connect("dbi:SQLite:dbname=my_data", "", "");
    my $sth = $dbh->prepare("SELECT * FROM books");
    $sth->execute();
    while (my @row = $sth->fetchrow_array()) {
        ## ... process each row
    }
```

Iterators in Perl

In order to iterate over a sequence of computed values (like the ones returned by <FILE> or $sth->fetchrow_array()), it is necessary to return a series of values followed by some false value when the sequence is exhausted. One easy way to signal the end of a sequence is to return undef or an empty list. This is sufficient when examining a series of strings (lines from a file), a series of lists (rows from a database), or a series of numbers.

Generating a sequence of computed values involves maintaining some state variables so we can tell when the sequence is

exhausted. This is generally done with an object, but it can also be done with a closure. Closures are anonymous subroutines that maintain some private-state variables. They're like objects, except that they've been turned inside out. Where objects are pieces of data (like a hash) with some subroutines attached, closures are subroutines with some data attached.

Here is a function that creates closures, each of which will count from 1 to 10:

```
sub make_counter {
    my $i = 1;
    return sub {
    return if $i > 10;
    return $i++;
    }
}
```

In this example, a new variable $i and a new anonymous sub are created each time we call make_counter. Each closure we create maintains its own private value for $i. We can then call the closure 10 times to get the values 1 through 10. After that, we'll always return a false value. This satisfies the requirements for an iterator, so we read values from it one at a time, almost as if it were a file:

```
my $iterator = make_counter();

while(defined($_ = $iterator->())) {
    print; ## 12345678910
}
```

Combining Iterators

The iterators that are created by make_counter() are very simple and may not seem very worthwhile at first. But it is easy to combine iterators to produce more interesting results. Here is an iterator that filters values from our simple counter iterator and emits only the odd values:

```
sub odd_numbers {
    my $next = shift;
    return sub {
    my $i = $next->();
    while (defined($i) and ($i % 2) == 0) {
        $i = $next->();
    }
    return $i;
    }
}

my $iter = make_counter();
my $odd = odd_numbers($iter);

while (defined($_ = $odd->())) {
    print;  ## 13579
}
```

First, we ask for a value from the iterator $odd. Within $odd's closure, we ask for a value from its $next iterator until we find an odd value or the end of $next's sequence of values. The result, as expected, is a sequence of odd values from 1 to 10.

This example shows another property of closures. Not only do closures turn objects inside out, but they turn logic inside out as well. Instead of skipping even values within the while loop, we weed them out beforehand, simplifying the while loop down to a single statement.

Note that we created the $odd iterator by modifying another iterator. This process can be extended, transforming a sequence of odd numbers into a sequence of odd numbers squared:

```
sub make_squares {
    my $next = shift;
    return sub {
    my $i = $next->();
    return unless $i;
    return $i ** 2;
    }
}

my $iter = make_counter();
$iter = odd_numbers($iter);
$iter = make_squares($iter);

while (defined($_ = $iter->())) {
    print;
}
```

We can go even further, adding another filter to transform this sequence of odd numbers squared into a running total of odd numbers squared, a running average of odd numbers squared, or something entirely different. No matter how we build the iterators up, the process of examining the final result remains the same: a simple while loop.

Parsing Configuration Files

Now that you understand the basic ideas behind iterators, it's time for a more practical example: parsing a configuration file. Let's start with a few simple requirements:

- Configuration files consist of a series of name-value pairs and are stored in a hash.
- Comments start with the # character and continue until end-of-line; all comments should be ignored.
- Lines consisting of nothing more than space characters should be ignored.

The first few requirements seem simple enough to implement with a standard while loop. It might look something like this:

```
while (<CONFIG>) {
    s/#.*$//;    ## delete comments until end-of-line
    ## skip blank lines
    while (m/^\s*$/) {
        $_ = <CONFIG>;
    }

    my ($name, $value) = m/^(.*?)=(.*?)$/;
    $config{$name} = $value;
}
```

If you look closely, there are some bugs caused by the inner loop. Only the first line's comments are deleted; after we've found a blank line (or a line with nothing but a comment), then the next non-blank line's comment will be kept. There are a lot of ways to fix this bug. If we had used iterators, these bugs would be easier to avoid.

First we need to read lines from a file using an iterator. Once that is done, we can then transform that stream of values by stacking one iterator on top of another until we're left with a stream of name-value pairs:

```
sub make_file_iterator {
    my $filename = shift;
    open(my $fh, $filename);
    return sub { return scalar <$fh>; }
}

sub strip_comments {
    my $next = shift;
    return sub {
        my $line = $next->();
        $line =~ s/#.*$//;
        return $line;
    }
}

sub skip_blanks {
    my $next = shift;
    return sub {
        my $line = $next->();
        while(defined ($line) && $line =~ m/^\s*$/) {
            $line = $next->();
        }
        return $line;
    }
}

my $config = make_file_interator("my.config");
$config = strip_comments($config);
$config = skip_blanks($config);

while (defined($_ = $config->())) {
    ## process name=value pairs
    my ($name, $value) = m/^(.*?)=(.*?)$/;
    $options{$name} = $value;
}
```

In this example, we start with three generic subroutines that create iterators. If another portion of our program needed to skip blank lines or strip comments, we could reuse these subroutines to generate iterators for that task. This allows us to maintain and debug code in one spot, rather than maintaining and debugging a few repeated lines in many places.

Another benefit is that our main program consists of three lines of initialization to set up the $config iterator, and a simple while loop that only sees valid values and operates on them. As requirements change over time, this main loop would need very little modification. Most of the changes could be handled by adding filters to the $config iterator to perform more pre-processing.

Conclusion

Iterators are a very powerful construct for processing a series of values. The kinds of iterators described here use closures for a simple and effective way to create and transform a series of values generated one at a time. Iterators simplify programming by separating out the pre-processing from the main processing for a series.

the tclsh spot

by Clif Flynt

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

clif@cflynt.com

One of the most useful features of modern GUIs is the little pop-up help window. Whenever I end up with a new application, and no time to actually read a manual, I'll let the cursor rest on a bizarrely named button and hope I get a hint for what it will do.

Tk does not include a pop-up help widget as one of the basic widgets, but one can be created with just a few lines of code. The help balloon described in this article was submitted to the Tcler's Wiki by Daniel Steffen (*http://www.maths.mq.edu.au/~steffen/tcltk*).

The code for a help balloon is fairly short, but not trivial. Creating a help balloon requires interacting with a several aspects of the window manager and Tk interpreter, and it's not always obvious how to gain access to the feature you need. Knowing what types of information are controlled by the window manager, and which are controlled by the Tk interpreter makes it a bit easier.

The first trick with a help balloon is that we need to know when the cursor has entered a window that has a help balloon associated with it.

Tcl handles linking an action to an event with the bind command. The bind command links a Tcl script to a window and event. When that window has focus, and that event occurs, the registered script will be evaluated.

The command looks like this:

Syntax: bind *window event script*
This causes script to be evaluated if event occurs while window has focus.

window The name of the window to which this script will be bound

event The event to use as a trigger for this script

script The script to evaluate when the event occurs

The events that will trigger evaluating the script are defined as zero or more modifiers, followed by an event-type descriptor, followed by a detail field. You must have at least a type or detail field in the event descriptor. Depending on the event, more fields may be required. The fields can be separated by white-space or dashes.

The event types include all the events supported by the X Window System:

Activate	Enter	Map
ButtonPress, Button	Expose	Motion
ButtonRelease	FocusIn	MouseWheel
Circulate	FocusOut	Property
Colormap	Gravity	Reparent
Configure	KeyPress, Key	Unmap
Deactivate	KeyRelease	Visibility
Destroy	Leave	

A simple event would be something like <H>, which would trigger on someone typing an uppercase H. In this case the event descriptor is just a detail field, with an implicit type of KeyPress.

The detail field describes the event in more detail. For example <KeyPress-H> would also describe the event when someone types an uppercase H, and <ButtonPress-1> describes the event when someone clicks the leftmost button.

The modifier field adds information about events that must happen simultaneously (like Control, Alt and Delete being held down together), or sequentially, like mouse double clicks.

Modifiers include Control, Shift, Lock and Alt, to describe a key that must be depressed when the event occurs, or Double, Triple, and Quadruple to describe how many times the event must occur: <Double-ButtonPress-1> describes the event when someone double-clicks the left mouse button. We could watch for someone triple-clicking while holding the Control key with <Triple-Control-ButtonPress-1>.

To make a help balloon, we want to know when the cursor enters or leaves a widget. The Enter and Leave events are generated when a cursor enters or leaves a widget, so a pair of lines like the following would display and destroy a balloon when the cursor enters and leaves a widget named .needsHelp:

```
bind .needsHelp <Enter> "create Balloon"
bind .needsHelp <Leave> "destroy Balloon"
```

Since a help balloon should appear below the widget that it relates to, the code that will create a balloon needs to know where that window is. Though the window name sounds like a line from a bad fantasy novel, knowing it allows you to learn its location.

The bind command will let us pass certain runtime values to the script that is evaluated when the event occurs. These values are defined in the script as a percent-item, which will be substituted for the actual value just before the script is evaluated.

The bind command supports many percent-items, including:

%b	The number of the button that was pressed to generate this event. Valid only for ButtonPress or ButtonRelease.
%d	The detail field from the event.
%h	The height field from the event. Valid for Configure and Expose.
%k	The key that was pressed or released. Valid only for KeyPress or KeyRelease.
%x %y	The X or Y coordinates for the event. Valid for events such as mouse events that have an X or Y field.
%R %S	The root or subwindow identifier for the event.
%W	The window for which this event is being reported.

The %W option lets us tie a help balloon to the window that created it. A command to bind help-balloon creation to a widget might look more like this:

```
bind .needsHelp <Enter>
                "createBalloonProc %W $helpMessage"
```

Help windows should appear, not immediately, but a second or two after the cursor enters a widget. This means we need to have a way to schedule an event to occur in the future.

The Tcl after command enables a script to react to a timer event or idle condition. This command has several subcommands that will let an application interact with the queue of scripts waiting for a chance to happen, but for a help balloon we only need the simple form of:

Syntax: after *milliseconds script*

after	Schedule a script to be processed in the future.
milliseconds	The number of milliseconds to pause the current processing, or the number of seconds in the future to evaluate another script.
script	The script to be evaluated after the number of *milliseconds* have elapsed.

Given a procedure to create the balloon named balloon:show, the beginning of a procedure to add a help balloon to a widget looks like this:

```
proc balloon {w help} {
    bind $w <Enter>
        "after 1000 [list balloon:show %W [list $help]]"
```

We can't leave that balloon up forever, so we need to be able to destroy the balloon when the cursor leaves the target widget.

The Tcl command to destroy a window is destroy.

Syntax: destroy *windowName ?window2...?*

Destroy one or more Tcl widgets.

windowName	The name of the Tcl widgets to be destroyed.

We can decide to name the help balloon the .balloon child of the window it relates to. This makes the entire balloon registration procedure look like this:

```
proc balloon {w help} {
    bind $w <Any-Enter>
        "after 1000 [list balloon:show %W [list $help]]"
    bind $w <Any-Leave> "destroy %W.balloon"
}
```

The balloon:show procedure will create and display the help balloon. There are a few steps in this process.

1. Confirm that the cursor is still inside the window that is associated with this help balloon.
2. Destroy any previous balloon associated with this window.
3. Create a new window with the appropriate text.
4. Map this window to the screen in the appropriate place.

Several of these steps require information from the window system: finding the location of the cursor, the location of a widget, etc.

Tk provides for interaction with a windowing system via two commands: the winfo command that returns information about the windows Tk controls, and the wm command that interacts with the window manager.

These commands have many subcommands, most of which aren't needed for this application. I'll just discuss a few of them as they become necessary.

The first step, confirming that the cursor is still within the window, can be done with two winfo commands. The pointerxy subcommand will return the coordinates of the cursor, and the containing subcommand will return the name of a window that encloses a pair of coordinates.

Syntax: winfo pointerxy *window*

Return the X and Y location of the mouse cursor. These values are returned in screen coordinates, not application window coordinates.

window The mouse cursor must be on the same screen as this window. If the cursor is not on this screen, then the coordinates will each be -1.

Syntax: winfo containing *rootX rootY*

Returns the name of the window that encloses the X and Y coordinates.

rootX An X screen coordinate (0 is the left edge of the screen).

rootY A Y screen coordinate (0 is the top edge of the screen).

The containing subcommand requires two separate arguments, while the pointerxy returns a pair of arguments. If we tried to write this code:

```
winfo containing [winfo pointerxy .]
```

the Tcl interpreter would throw an error.

The return from winfo pointerxy . would be substituted into the command as a single unit. The command evaluated by the Tcl interpreter would resemble:

```
winfo containing {120 300}
```

instead of

```
winfo containing 120 300
```

The solution to this is to use the eval command to evaluate the string.

Syntax: eval *string1 ?string2...?*

Concatenate the arguments into a single string and evaluate that string as a command.

*string** Strings that will compose a command.

Because lists are concatenated onto the end of the previous data, the eval command loses one level of grouping information. If you need to maintain the grouping of some sets of data, use the list command to make a list of it.

Using a string match command to compare the window that currently has the cursor with the window that requested the help balloon, we get code like this:

```
proc balloon:show {w arg} {
    if {![string match [eval winfo containing
                    [winfo pointerxy .]] $w]} {
        return
    }
```

The next step is to destroy any previous existing balloon. This might seem unnecessary – after all, a cursor has to leave one window before it can enter another.

However, there are circumstances when a cursor *can* enter a second window without leaving the first. For example, if one widget is contained within another, the cursor can enter the inner widget without leaving the outer widget.

The example below shows an unlikely example of this situation:

```
# Create and display a canvas
canvas .c
pack .c

# Create and display a label within the canvas
label .c.l -text label
.c create window 50 50 -anchor nw -window .c.l

# Add bindings to report when the mouse enters
# and leaves the windows.
bind .c <Enter> {puts {in .c}}
bind .c <Leave> {puts {out .c}}
bind .c.l <Enter> {puts {in .c.l}}
bind .c.l <Leave> {puts {out .c.l}}
```

As a mouse cursor enters the canvas, then the label, and then leaves the label and canvas, the following output is generated:

```
in .c
in .c.l
out .c.l
out .c
```

To make the balloon help code a bit more readable, the name of the new balloon help window is saved in the variable top.

If the window does not exist, it can't be destroyed, and Tcl will throw an error. A script can catch an error with the catch command, which will evaluate a script in a safe way, and return the results and status of the script separately.

The syntax is:

Syntax: catch *script ?varName?*

The catch command returns the status from evaluating the script, and optionally places the results of evaluating the script in the variable *varName*.

In this case, we don't need the results from the destroy, so we can destroy any previous balloons associated with this window with:

```
set top $w.balloon
catch {destroy $top}
```

The next step is to create the new window. Tcl supports two types of windows:

- Windows that are managed within a Tk window
- Windows that are managed by the window manager

A window managed within a Tk window (like most buttons, labels, scrollbars, etc. that your script creates) must fit within the parent window. Windows that are managed by the window manager (called top level windows) can appear anywhere on the screen and may have decorative borders set by the window manager.

For a help balloon, we want a top level window (in case the widget this balloon is associated with is at the bottom corner of the application), and we want the window to not have any decorations. Our script will place a message widget inside this top level to hold the help text.

The command for creating a new top level window is toplevel.

Syntax: toplevel *windowName ?-option value ...?*

The options include setting the border width, relief, background, class, etc.

This application wants a very simple top level with a one-pixel-wide border.

```
toplevel $top -borderwidth 1
```

A help balloon window should not have the decorations added by the window manager – we don't want the user to be able to move this window, iconify it, etc. The decorations are added by the window manager, not managed by Tk, so removing the decorations is done with the wm command. The subcommand that handles this is override-redirect.

Syntax: wm override-redirect *windowName boolean*

	Sets the override-redirect flag in the requested window. If true, the window is not given a decorative frame and can not be moved by the user. By default, the override-redirect flag is false.
windowName	The name of the window for which the override-redirect flag is to be set.

boolean A boolean value to assign to the override-redirect flag.

The wm override-redirect command should be given before the window manager transfers focus of a window. Unlike most Tcl/Tk commands, you may not be able to test this subcommand by typing commands in an interactive session.

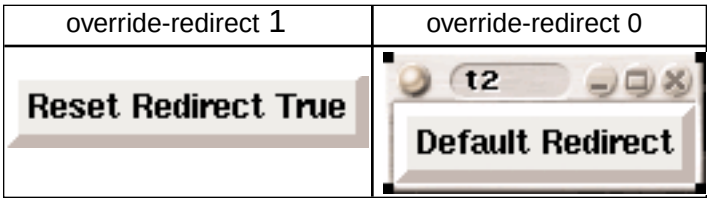
The difference between override-redirect true and false looks like this:

```
catch {destroy .t1 .t2}
toplevel .t1 -border 5 -relief raised
label .t1.l -text "Reset Redirect True"
pack .t1.l
wm override-redirect .t1 1

toplevel .t2 -border 5 -relief raised
label .t2.l -text "Default Redirect"
pack .t2.l

wm geometry .t1 +300+300
wm geometry .t2 +300+400

raise .t1
raise .t2
```



Creating the new top level and getting rid of the borders looks like this:

```
toplevel $top -borderwidth 1
wm override-redirect $top 1
```

The next step is to add the help message. Tk supports three widgets for displaying textual information:

label	Displays a single line of text.
message	Displays one or more lines of text.
text	Displays one or more lines of text with support for editing, multiple fonts, tagged areas, etc.

Any of these widgets could be used for the help message, but the help message may be longer than can fit on a single line, and the text widget is a bit heavyweight for this application. The message widget combines some of the features of the text widget and some features of the label widget, making it the best widget for this application.

Syntax: message *name ?options?*

message	Create a message widget.
name	A name for the message widget. Must be a proper window name.
?options?	Options for the message include:

-text	The text to display in this widget.
-textvar	The variable which will contain the text to display in this widget.
-aspect	An integer to define the aspect ratio: (Xsize/Ysize) * 100
-background	The background color for this widget.

When a widget creation command is evaluated, it returns the name of the widget that was just created. This can be used with

the geometry managers to make a single-command create and display command like this:

```
pack [message $top.txt -aspect 200
      -background lightyellow \
      -font fixed -text $arg]
```

The final step is to place the new window just under the widget that requested the help balloon.

The window that requests the help will be a window managed by Tk, so we can use the winfo command to determine its height and X/Y locations.

The subcommands for these data are:

winfo height*winName*	Return the height of a window in pixels.
winfo rootx *winName*	Return the X location of this window in screen coordinates.
winfo rooty *winName*	Return the Y location of this window in screen coordinates.

These two lines of code set variables for the X coordinate to be the same as the left edge of the window requesting the help balloon, and the Y coordinate to be just below that window.

```
set wmx [winfo rootx $w]
set wmy [expr [winfo rooty $w]+[winfo height $w]]<
```

Placing a top level window on the screen is a task for the window manager, so the wm geometry command gets used.

Syntax: wm geometry *windowName ?geometry?*

	Query or set the geometry for a window.
windowName	The name of the window to be queried or set.
?geometry?	If this is present, it's a geometry string following the X windows convention of *width*x*height*+/-*Xposition*+/-*Yposition*. The x and + or - separators are required. If this field is not present, the wm geometry command returns the current geometry of the window.

For most X Window window managers, we could just provide the X and Y locations for the new window:

```
wm geometry $top +$wmx+$wmy
```

But, to be completely safe on multiple platforms, with different window managers, we should provide a complete geometry specification with the width and height of the window included.

The requested width and height is known by the Tk interpreter, and is returned by the winfo reqwidth and winfo reqheight commands.

A better geometry command resembles this:

```
wm geometry $top \
    [winfo reqwidth $top.txt]x[winfo reqheight
            $top.txt]+$wmx+$wmy
```

The final step is to make sure the new window isn't hidden behind other windows. The raise command places one window above another, or above all other windows, if no other window is defined.

```
    raise $top
}
```

Wrapping all these code fragments together, the balloon:show procedure looks like this:

```
proc balloon:show {w arg} {
    if {![string match [eval winfo containing
                [winfo pointerxy .]] $w]} {
        return
    }
    set top $w.balloon
    catch {destroy $top}
    toplevel $top -borderwidth 1 -background black
    wm overrideredirect $top 1

    pack [message $top.txt -aspect 200
                -background lightyellow \
                -font fixed -text $arg]
    set wmx [winfo rootx $w]
    set wmy [expr [winfo rooty $w]+[winfo height $w]]
    wm geometry $top \
        [winfo reqwidth $top.txt]x[winfo reqheight
                $top.txt]+$wmx+$wmy
    raise $top
}
```

This code, with a tweak for Macintosh platforms, is available at *http://mini.net/tcl/534.html*.

securing FTP

by Gary Cohen

Gary is the CEO and co-founder of Glub Tech, Inc. and is concurrently employed by Adobe Systems. In the past he has worked for the San Diego Supercomputer Center and IBM.

gary@glub.com

In the summer of 1999, fellow classmate Brian Knight and I teamed up to take part in a senior project at the University of California, San Diego (UCSD). Both Brian and I worked as interns in the Computer Security department at the San Diego Supercomputer Center (SDSC) and were quite aware of some of the inherent security problems that lie within the FTP protocol. Under the leadership of Sid Karin and Tom Perrine, we worked with SDSC to build a safer FTP client. Just prior to starting this project, SDSC had made a move to disable all services that transmitted a user's password in the clear and to allow a user to transfer files only via scp (a file-transfer wrapper provided with SSH), FTP via an SSH tunnel, anonymous FTP, or Kerberos-authenticated FTP. Brian and I felt that these options each had their shortcomings.

For example, scp was widely available only on UNIX, and to create a user interface for it on other platforms made little sense since Secure Shell v2.0 was in beta and bundled an FTP emulator, sftp, with it. We looked at writing an FTP client that supported sftp, but the daemon was not very stable, which provided some hurdles, and we needed to finish the project within the 10-week quarter.

The second file-transfer method sanctioned by SDSC was FTP via an SSH tunnel. Unfortunately, this method requires an end-user to perform a fairly complex setup process. Even though SDSC provided documentation on how to configure a tunnel and connect to an FTP server securely, most end-users had difficulty completing all the steps or felt it was too inconvenient to do so.

The third method, anonymous FTP, was probably the most popular way to transfer files between systems at SDSC, but the workflow was disjointed and could sometimes cause problems. By using anonymous FTP, a user would upload a file into a write-only directory (or dropbox). Once the file was uploaded, the user would have to log in to a shell, take ownership of the file, and move the file where it needed to go. In order to download the same file, the file would have to be moved to a separate, read-only directory before it could be retrieved by FTP. To say the least, the process was not very efficient.

The fourth method was the only cross-platform option: use FTP with Kerberos authentication. (SDSC also supplied the means to FTP via S/Key, but the number of users for this service was even smaller than the number of those who used FTP with Kerberos.) However, most of the users at SDSC did not use Kerberos, and the ones who did were mostly UNIX folks who preferred the secure file transfer capabilities of scp. Additionally, there were only a few clients that supported FTP via Kerberos.

Weighing our options, Brian and I felt implementing a secure client that did true FTP seemed like a smart decision. FTP had been proven to work since its inception, and people were already familiar with the client interface. But we were debating on how to secure the control channel. Prior to this project I had used an SSL wrapper to secure POP and IMAP for SDSC's mail system. It was fairly easy to set up on the server side (although we encountered a few issues using certificates generated by the Netscape Certificate Manager), and it seemed to work well with the available SSL-enabled mail clients. Keying off that success, we decided SSL looked like a viable security solution

for FTP as well. Our goal was to write an SSL-enabled FTP client that could communicate with an ordinary FTP server through an SSL wrapper.

Brian and I started planning the project (only as well as a couple of college seniors could). After reviewing the design with our advisors, we began writing our Secure FTP client. To ensure our program had the greatest reach across many platforms, we wrote it in Java as both applet and application. I took the role of designing and building the user interface while Brian implemented the networking. Most of the project went smoothly until we reached the phase of adding the SSL layer. Rather than reinvent the wheel, we looked for a Java library that would provide SSL functionality. There were a few to choose from but most were expensive (or at least it seemed that way to us poor college students). Fortunately, a couple of weeks before the project was due, Sun Microsystems released a reference implementation of SSL called the Java Secure Socket Extension (JSSE). The documentation was limited, and the only form of a demo was via Sun's HotJava Web browser. But we were able to get the library to behave as expected.

A screenshot of the Secure FTP client

We had planned to release the source code for the client once we were finished, but with this being our first large Java program, and with a deadline of 10 weeks, the finished code was anything but pretty. Both of us quickly agreed that nobody should ever see this code. It was embarrassing, but it worked as designed. We had written a client that could connect to an SSL wrapper that acted as a proxy to an existing FTP server.

Using our client with an SSL-wrapped FTP server provided a means to encrypt the control channel (which protects the username and password, the main motivation behind the project), but it didn't allow for encryption of the data channel. Although there were SSL wrappers available on UNIX, there wasn't a wrapper for Windows. Shortly after releasing the first revision of the client (aptly named Secure FTP v1.0), we started looking at writing a complementary wrapper in Java. At first this wrapper would only address the same issues as the UNIX-based ones (encryption of the control channel) but with multiplatform support. Furthermore, we wanted to simplify the wrapper configuration and the generation of certificates. We would address data encryption later. Secure FTP Wrapper v1.0 was released to the public in February 2001.

With the initial releases of the client and wrapper under our belts, we decided to look at data encryption, starting with the wrapper first. Since FTP uses two channels, one for the commands and one for the data, using a wrapper to encrypt both is nontrivial. Essentially, we would need to write a wrapper that understood the FTP protocol to handle data encryption. So that's what we did – we wrote a smart wrapper.

Version 1.0 of our wrapper acted as a transparent proxy, or "SSL translator," between an SSL-savvy client and a non-SSL FTP server. We got a secure connection from the client and forwarded the plaintext "conversation" to the server. When the user issued a command that required a data transfer, we just passed on the information without intervening. For the most part this worked fine. However, this caused problems for some servers that saw this behavior as a possible port theft. Port theft occurs when an FTP client connects to the server's control channel from a certain IP address but initiates a data transfer from a different one. The problem with our setup was that the FTP server saw the wrapper as the client when handling commands, but when a data transfer was started, the data was being sent to the true client (instead of the wrapper). This

There is no good reason to transmit a password in the clear.

conflict led the server to think the data was being stolen and, in some cases, the FTP server denied the request.

When we started working on the next version of the wrapper, the added support for data encryption had the positive side effect of removing any possible port theft. Instead of the wrapper just passing commands onto the server, it now acted as both a mini-client and mini-server.

Because of the nature of file transfers in FTP, an FTP client can act as both client and server. If the client sends the command PASV, it is requesting the server to open a listening socket to deal with the data transfer; this is known as a passive transfer. On the other hand, if the client sends the command PORT, it is requesting the server to connect to a socket that is listening on the client machine. Because of this dual nature, our wrapper had to deal with both possibilities.

To do this, instead of merely passing on commands to the server from the client, we now checked them against a list of known FTP commands. If we saw a PASV command issued by the client, our wrapper would still pass on that request to the server. But instead of sending the server's response back to the client as is, we create a new server socket, "duct" our new socket with the server's old socket, and rewrite the response to reference the new socket. Depending on the type of data connection, this new socket may be SSL enabled. Handling a PORT connection is similar to the PASV workflow, except the roles are reversed; we create a new server socket, have the server connect to our new socket, create another socket which we use to connect to the client, and "duct" our two sockets together. It sounds complicated (which it is), but this is why we decided to tackle data encryption at a later date.

Version 2.0 of our wrapper worked, but after we had originally started on our Secure FTP crusade, the official spec for FTP security extensions changed. Originally there were two supported mechanisms to handle FTP over SSL – explicit and implicit – but version 8 of the draft inexplicably dropped support for the implicit options. An explicit connection occurs when an FTP client connects to the standard FTP port (port 21) and, to enable security, it issues the command AUTH SSL. If the server supports this command, it would convert the control channel from an insecure to an SSL-enabled, secure channel. The other option, implicit, occurs when the client connects to the IANA-specified port for ftps (port 990). This port already is SSL enabled much the way https is enabled on port 443; when connecting to a secure Web server, an AUTH command is not required since the connection is secure to begin with.

It remains a mystery to us what problem the standards makers were trying to solve by dropping the implicit option. We tried to find a case where the control channel should allow user information to be transmitted in the clear. We can understand why one would not want to force encryption on the data channel due to speed issues, but there is no good reason to transmit a password in the clear.

While we do not agree with this change, we decided to handle it nonetheless by adding support for an explicit connection (Secure FTP Wrapper v2.5). However (due to our stubbornness), we have kept the implicit connection on by default. Additionally, we added a flag to our configuration that makes a secure control (and data) connection a requirement. When an explicit connection is made, and the requirement of a secure control connection is set, we will not allow the user to send a password without first sending the AUTH command.

Unfortunately, adding support for explicit SSL complicates the wrapper concept. Having a wrapper listen on the standard FTP control port (port 21) can cause configuration issues since the existing FTP server may already be listening on that port. Due to this obstacle, we do not enable the explicit connection if the wrapper IP address and port are the same as those of the destination server. There are a couple of solutions to make an explicit connection possible. One option would be for the server to listen on a different port with the same IP address as the wrapper. Another option would be to bind the server to a different address (such as localhost).

Separating the SSL support from the server might add complexity to a system administrator's job, but we think the advantages outweigh the disadvantages. The major advantage is you can continue to use your legacy FTP server. The last thing a busy administrator wants to do is upgrade an FTP server and make sure the users can still get their work done. In addition, our wrapper can be configured to listen on a border router (straddling the DMZ) and allow a secure connection from the Internet into a server that exists in your intranet. This kind of flexibility cannot be done if the encryption is taking place in the FTP daemon.

Now that there was a wrapper that supported encryption on both channels, we added the much-requested support for data encryption in the client (Secure FTP v1.6).

What's next? We are planning on externalizing our FTP via SSL implementation into a Java bean so others can incorporate it into their own Java programs. We also have some ideas to make our wrapper even smarter, but for now we'll keep that secret.

For more information on Secure FTP, or to download the client and wrapper, we invite you to take a look at the Secure FTP Web site (*http://secureftp.glub.com*) and the Secure FTP Wrapper Web site (*http://wrapper.glub.com*).

This paper was written with input from Brian Knight, CTO of Glub Tech, Inc.

We are planning on externalizing our FTP via SSL implementation into a Java bean so others can incorporate it into their own Java programs.

musings

by Rik Farrow

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

rik@spirit.com

I often take a look at a past column before ramping up the old rant engine. Exactly a year ago, I was complaining about how a default install of Linux as workstation left your system wide open, with many unnecessary network services running. Now, I can cheerfully report that the opposite is true, that a more recent release of Linux (RedHat 7.2) not only left services disabled, but also included optional installation of netfilter (*http://netfilter.samba.org*), configured to protect your workstation. How nice.

Not that everything has come up roses. I also ranted (last year) about weaknesses in IE and XP. These issues have not gone away. IE still continues to be exploited on a regular basis. I have been getting several copies of the Klez worm, an Outlook virus, every day. And I port scanned a default install of Windows 2000 Professional Server and found that it had 25 TCP ports running services. This scan showed the simple services, good for denial of service, the ever present SMB share everything services, plus some new ones that are security related (Kerberos, ldapssl, and kpasswd5). Something shows up at port 6666/tcp, nominally IRC, but I don't believe that Win2K comes with IRC by default.

I have a useful security suggestion and a rant for this month as well. First the suggestion, then the rant.

Backfire

I spend a lot of time researching and thinking about attacks. The Honeynet Project (*http://project.honeynet.org*) is a really great source for learning about opportunistic attacks. Opportunistic attacks are those without any particular target in mind – any vulnerable system will do. These usually come about through scanners running on previously rooted boxes. The average length of time between when Project members install a default configured system and it gets rooted appears to be two days, maybe less. Something to think about, especially if you are in a network without any sort of firewall or controlled update program.

The other really interesting thing is what attackers do when they root a UNIX/Linux box. The standard sequence of events goes something like this:

1. Clean up signs of the attack.
2. Install new user accounts (one normal, one root).
3. Login via new user account, `su root`.
4. Download patches.
5. Download tools.
6. Install tools.
7. Logout.

The installation of patches really surprised me at first. Who would have thought that attackers would be so kindhearted and altruistic as to patch the security hole just used to root the box? The truth is much simpler – if the attacker got in this easily, so can anyone else, so the box must be patched at once. If they don't patch, someone else will soon "own" the system.

The tools downloaded vary a lot. Commonly, rootkits get downloaded. You can learn a lot about rootkits by visiting *http://www.chkrootkit.org/*, as well as by finding a script for detecting installed rootkits. Scanning and attack tools are also popular, turning the recently rooted system into yet another tool for global domination. Distributed denial

of service (DDoS) agents or handlers may be installed. The system may also be used to run a bot, such as eggdrop (*http://eggdrop.org*), and to keep sysop privileges on some IRC channels.

What does all of this activity have in common? It all requires downloading software to the rooted system. Attackers most commonly use FTP, although they have been known to use TFTP (especially on Windows boxes), and could use other mechanisms, like Lynx (which version 1 of the lion worm did) or wget, as well. No matter what the attacker uses, downloading software requires an outgoing network connection.

If you have a public Web server sitting behind a firewall, you can easily prevent outgoing connections from that Web server. Web servers accept requests, they don't make outgoing connections to the Internet. A Web server could be performing DNS lookups, but this is generally not done, since it is slow and can be done offline when logs are analyzed (if anyone even cares about the FQDN of all visitors).

I suggest that you use firewall rules to block outgoing connections from public Web servers.

I suggest that you use firewall rules to block outgoing connections from public Web servers.

I suggest that you use firewall rules to block outgoing connections from public Web servers. Even better, set the firewall up so that you get paged when the Web server attempts to make an outgoing connection. At the very least, send yourself an email from the firewall. An outgoing connection should NEVER happen, so if it does, something very bad has happened.

This firewall rule also blocks scanning of Internet addresses from a public Web server, another common behavior seen from successfully attacked systems. Firewall rules like this would have done a lot to slow down the various versions of Code Red, but not Nimda, which used other attack vectors (JavaScript that executed the virus code, README.EML or README.EXE).

Using this simple trick with other public servers will not be quite as effective. SMTP servers make lots of connections to other SMTP servers, and if an SMTP server gets hit with an SMTP worm (how could anyone forget the Morris Worm?), these connections will also go to other SMTP servers, on port 25/TCP. Same thing with DNS servers, who naturally make connections to ports 53 TCP and UDP. Still, only allowing connections on these ports and sending out a page or email when something like FTP gets used from your public SMTP relay or DNS server would catch some attackers.

If you really want to go further, do this for your entire internal network. You cannot, practically speaking, only permit connections to specific remote hosts or services. You can, however, block access to services forbidden by policy, and use this to set off alarms. The popular Windows Trojan horse SubSeven (and others as well) will use a connection to an IRC channel, or ICQ, to announce that it has been successfully installed. If your policy forbids the use of IRC and/or ICQ, then block them, and alarm on them.

In May and June of 2002, two sites were quietly compromised, and the configuration scripts for software had a handful of lines added to them. In each case, the configuration script now compiles a program, executes it, and removes the source and executable while someone is running ./configure. The short program makes an outgoing connection to a site, and if the connection succeeds, execs a shell at the victim's end. Think of this as a reverse telnet, but using (in this case) port 6667/tcp. The shell runs as the user running configure. A firewall rule that blocks IRC (which just happens to be the port used) would have blocked this backdoor. Too bad that one of the packages backdoored was the Bitchx IRC tool (meaning that the choice of the IRC port almost guarantees the port would be open).

Ever read the End User License Agreement that comes with any software or operating system you have ever acquired? Its only warranty is for the media it comes on.

Other great examples are the Microsoft everything sharing ports, 137/udp and 139/tcp. Certain attacks against IE encourage the victim to connect to a remote file share, which gets treated as part of the Local Zone for IE security purposes. Always block outgoing connections to port 139/tcp.

Marcus Ranum has talked about these types of tricks a lot in the past, and probably will in the near future (he is teaching a class about Honeypots at the USENIX Security Symposium). Ranum, in a past life, installed burglar alarms, and has talked about tricks like putting an alarm on a bogus jewelry box (one that the owner knows never to open). Having your firewall set off an alarm when something occurs that should never occur works as well, and perhaps better, since the alarm is on a separate system, not the one that has just been rooted.

DoR

A conservative think tank, the Alexis de Tocqueville Institution (ADTI), published a White Paper this summer entitled "Opening the Open Source Debate." The paper does not really debate anything but instead contends that open source "opens the gate to hackers and terrorists." I certainly consider this a bogus and misleading statement, and Microsoft has admitted that they give money to ADTI but will not say if they funded this report.

Jim Allchin, group vice president of Microsoft, in testimony in the MS antitrust case, had a lot to say about his own company's software (*http://www.eweek.com/article/0,3658,s%253D701%2526a%253D26875,00.asp*). Allchin said that "sharing information with competitors could damage national security and even threaten the U.S. war effort in Afghanistan." Wow. If Microsoft were to share their "secret," proprietary extensions to Kerberos Five, national security will be weakened? I don't think so.

The truth comes out a bit later in the same article. There Allchin acknowledged that some Microsoft code was so flawed it could not be safely disclosed. Included in this was the part of the Message Queuing protocol that contains a coding mistake which would threaten the security of enterprise systems using it if it were disclosed.

Well, that is certainly interesting news. Microsoft wants people to trust them and use their OS software for enterprise-level systems, yet knows it has fatal errors within it. If this were open source it would simply be patched, but because it is Microsoft it will remain hidden, at least until someone discovers and discloses it – perhaps with the Queuing worm? And which type of software are you going to trust now?

Andy Oram, of O'Reilly, has coined a term for this behavior. He called it "Denial of Responsibility (DoR)" in *http://www.oreillynet.com/cs/user/view/wlg/1500*. Oram's DoR attacks are about vendors shipping code that they know has killer bugs in it (see above). DoR is also about organizations and agencies that require the use of buggy software, so that information that should be protected gets exposed instead.

I'd like to take the concept further and consider DoR in terms of licensing. On the one hand, we have the GNU copyleft, open source, and the BSD license which describe the rights and privileges of the users of open source (or some related variant) software. Pundits argue that when open source software is used, no one is responsible for errors, omissions, or bugs in the code.

And what about proprietary software? Ever read the End User License Agreement that comes with any software or operating system you have ever acquired? Its *only* warranty

is for the media it comes on. You use the software, it is your responsibility. Now, there is a real DoR attack. And, it is standard operating procedure today.

I like to compare the state of the current software industry to the automobile industry in the past. Do you know when automobile manufacturers started using safety glass in windshields? I really didn't appreciate this factoid until I watched a series on safety and liability issues on PBS. I also was made aware of the issue when my daughter told me that her husband was adding safety features, including modern brakes and safety glass, to the Model-A Ford he was turning into a hot rod.

You might even see Model-A Fords these days; they are really popular as restorations and hot-rod conversions. Their flat windshields are very distinctive and were once also lethal weapons. The ordinary plate glass used in the windshields could explode into razor sharp shards anytime a rock hit the windshield. It wasn't until the mid-'50s that safety glass became common in windshields. That's right. The cars your parents might have been driving (mine were) had plate glass windshields.

That was 50 years ago. Today, when Ford designs an SUV that rolls over quite easily, it has a big battle with a tire manufacturer, trying to place the blame for a dangerous design somewhere else. Newer Ford Explorers, however, have a wider wheelbase, making them less prone to rollovers (something pointed out to Ford many years ago). But do software vendors even bother? Nope. You use the software, therefore you are to blame.

With this level of responsibility, it is a wonder that *anyone* pays for software. Vendors take as much responsibility for their software, and the potential damage it might do, as does the open source community. If you are forced to use non-open source software, from a vendor who publicly declares that revealing information about the protocols and APIs involved is definitely dangerous, I think you are making a big mistake.

It's like driving a car with a plate glass windshield, just hoping that that truck in front of you doesn't toss a rock at you.

It is a wonder that *anyone* pays for software.

identifying and tracking unauthorized 802.11 cards and access points

by Robert Foust

Robert Foust has been experimenting with Linux since version 0.95. He is currently employed as a system administrator where he enjoys working in a heavily mixed hardware and software environment.

rfoust@interlinknetworks.com

A Practical Approach

Introduction

The object of this paper is simple: to determine whether an individual, using inexpensive off-the-shelf components and free software, could detect when an unauthorized 802.11 card or access point was powered up and began broadcasting within range of a local WLAN. Furthermore, could the event be tracked, activity monitored, and the offending card or access point physically located?

Tracking unauthorized external accesses, is considered by many to be the biggest concern. Midnight parking-lot attacks are a real, tangible threat to many people, and especially frightening because the attacker could do almost anything. It's the unknown and uncontrollable risk that frightens many security professionals.

Realistically, though, the biggest threat in most organizations may come from inside. Through maliciousness or plain old carelessness, an internal employee has the ability to inflict major damage on a local network or open up large, gaping holes in the local network.

For example, many users would think nothing of starting up Microsoft's Internet Information Server (IIS) on their Windows laptop, but most security professionals would shudder at the thought of an untracked, unpatched IIS server running open to the world. The user may bring their machine home at night, connect it to their ISP, get infected with a worm or virus, and the next day, bring that machine on the local network to wreak all kinds of havoc.

The 802.11 specification opens up a more interesting and far more dangerous possibility: a power user could simply bring an access point to work because they want the convenience of a wireless network without the bother of the IT department's delays in deployment. Being a power user, they know that they can simply assign the access point an address via DHCP, plug their own wireless card into their laptop, and then walk around the office with their laptop. With proxying and NAT software, this kind of activity might even go totally unnoticed by security personnel or automated intrusion detection systems. Little does this user know that the IT department's concerns are well founded and that the user has unwittingly opened a gaping hole in the local network, such that any drive-by attacker could simply hop on the local network and do anything they wish. To address these concerns, we assembled the readily available necessary pieces into a usable detection and tracking tool.

Hardware and Software Used

All software used is freely available and can be downloaded and used by anyone.

- Generic x86-based PC laptop.
- Cisco Aironet 340 802.11 card.
- Cisco Aironet 340 access point, used as an experimental rogue access point.
- Lucent Orinoco "Silver" 802.11 card, used as an experimental unauthorized wireless card.

- SuSE Linux 7.1.
- Development version of libpcap (libpcap current-cvs-2001.07.20), a sniffing library used under Linux. The latest development version was used because it understands the 802.11 frame format (obtained from *http://www.tcpdump.org/*).
- Development version of ethereal (v0.8.19), a very complete sniffer and protocol analysis tool. This was used because it was one of the only tools available which could dissect 802.11 frames in a human-readable format (obtained from *http://www.ethereal.com/*).
- Linux kernel 2.4.9. This was used because it had an updated driver for the Cisco Aironet 340 wireless Ethernet card. There had been numerous changes to the driver since 2.4.0 (one of the standard SuSE 7.1 kernels), and we wanted to be sure the driver supported the 802.11 frame format, offered complete reporting, supported RF Monitor mode (which allowed "promiscuous" recording of 802.11 packets), and supported the iwspy ioctls, which were necessary to gather signal strength statistics on arbitrary wireless clients and access points. The Aironet driver included with the 2.4.9 kernel supported all of these.
- Octave v2.0.16, a Matlab-like mathematics package. This was used to solve some relatively complex simultaneous equations encountered while trying to track down the physical location of unauthorized cards/access points. This package was included with *SuSE 7.1 but can also be downloaded from Octave's Web site, *http://www.octave.org/*.
- Gnuplot, latest CVS snapshot as of September 7, 2001. Gnuplot was used to find a best-fit curve instead of Octave. Octave did not have any pre-canned best-fit functions, while Gnuplot did, and a best-fit curve ended up yielding better results than finding the simultaneous solution of three equations with three data points (the first attempt).
- The GIMP, for modifying and manipulating various graphics.
- Emacs, Xfig, TeX, LaTeX, Ghostview, xv.

Note: We would *very* much liked to have used snort, a free IDS. This tool is quite remarkable and extensible, but we couldn't find any easy way within the scope of this project to define or act on events that happened at the 802.11 frame level, which is essential for detecting unauthorized cards or access points. Perhaps in future versions snort will offer a combination of robust high-level intrusion detection along with the complete protocol analysis tools offered by ethereal. For this project, the command-line version of ethereal was used to track unauthorized wireless clients.

Background

The 802.11 specification refers collectively to a family of IEEE standards for a new wireless networking protocol designed to be compatible with previous 802 specifications; it covers everything from the physical medium (microwave radio and IR) and modulation techniques to the link-layer protocol. The 802.11 specification can be obtained from the IEEE Web site, *http://www.ieee.org/*. As implemented by most vendors, 802.11 operates over microwave frequencies. In North America, it operates in the 2412 to 2462MHz range over 11 channels. When used over radio links, 802.11 makes use of a technology called "spread spectrum," summarized in an article by Schilling, Pickholtz , and Milstein:

> Spread-spectrum radio communications, long a favorite technology of the military because it resists jamming and is hard for an enemy to intercept, is now on the verge of potentially explosive commercial development. The reason: spread-spec-

trum signals, which are distributed over a wide range of frequencies and then collected onto their original frequency at the receiver, are so inconspicuous as to be "transparent." Just as they are unlikely to be intercepted by a military opponent, so are they unlikely to interfere with other signals intended for business and consumer users – even ones transmitted on the same frequencies. Such an advantage opens up crowded frequency spectra to vastly expanded use.

In a nutshell, spread spectrum makes it difficult to distinguish legitimate signals from normal background noise and makes 802.11 less susceptible to noise from the environment. This is especially important considering that the microwave frequencies used by 802.11 (~2.4GHz) are unlicensed, and so are susceptible to interference from many sources, most notably microwave ovens.

Unfortunately, spread spectrum turns out not to offer much in the way of security of communications, since the spreading algorithm used by 802.11 is well known and implemented in every 802.11-capable card. However, spread spectrum still complicates efforts to locate the source of unauthorized wireless cards and access points, since normal fixed-frequency receivers may not be able to easily distinguish 802.11 transmissions from normal background noise. Using a traditional microwave receiver and directional antenna to locate the source of unauthorized transmissions would probably be difficult.

Detection

The first goal is detection. Can we tell when someone powers on a card within range of the local network? This can be done with off-the-shelf components and free software. The Cisco Aironet driver included with the more recent Linux kernels supports "RF Monitor" mode, which permits promiscuous monitoring of 802.11 packets – specifically, monitoring raw 802.11 frames to detect if there are any telltale frames broadcast by a rogue access point or card.

As outlined in the original 802.11 specification, there are three classes of 802.11 frames. With the goal of detecting rogue access points and unauthorized wireless Ethernet cards, we are primarily interested in class 1 and 2 frames. Class 1 frames are the only frames allowed in state 1, the unauthenticated state, and are largely management frames used for authentication, beacons, and probe requests. Class 2 frames are allowed in both states 1 and 2 and are used for association and re-association. From access points, we would expect to see a large number of beacon frames (class 1). From unassociated ad hoc clients scanning in active mode, we would expect to see a large number of probe requests (also class 1). To test this hypothesis, a method of monitoring all 802.11 management frames is needed, which the Cisco card and Linux driver are capable of in RF Monitor mode.

SETUP

To put the card into RF Monitor mode, use any BSS (use `Mode: r` for plain RF Monitor mode):

```
# echo "Mode: y" >   /proc/driver/aironet/eth0/Config
#
```

Then, start logging packets with tcpdump, saving them to a file for later analysis with ethereal:

```
# tcpdump -i eth0 -s 0 -w capturefile
#
```

UNAUTHORIZED AD HOC NETWORK

The first test was to confirm the ability to detect a WLAN card being powered on. A Lucent Orinoco card was configured in ad hoc mode on a Win2K laptop, and turned on to find out if there were any characteristic frames sent out by the Orinoco card when it was put into ad hoc mode.

After the card initialized, tcpdump was stopped, ethereal started, and the capture file opened. A large number of probe requests from the Orinoco card were found, confirming that it was indeed possible to detect when someone within close range had powered up a wireless Ethernet card in ad hoc mode.

The dissected frame was as follows:

```
IEEE 802.11
    Type/Subtype: Probe Request (4)
    Frame Control: 0x0040
      Version: 0
      Type: Management frame (0)
      Subtype: 4
      Flags: 0x0
        DS status: Not leaving DS or network is operating in AD HOC mode(To DS: 0 F...)
        .... .0.. = Fragments: No fragments
        .... 0... = Retry: Frame is not being retransmitted
        ...0 .... = PWR MGT: STA will stay up
        ..0. .... = More Data: No data buffered
        .0.. .... = WEP flag: WEP is disabled
        0... .... = Order flag: Not strictly ordered
    Duration: 0
    Destination address: ff:ff:ff:ff:ff:ff (ff:ff:ff:ff:ff:ff)
    Source address: 00:02:2d:1b:51:ca (Agere_1b:51:ca)
    BSS Id: ff:ff:ff:ff:ff:ff (ff:ff:ff:ff:ff:ff)
    Fragment number: 0
    Sequence number: 118
IEEE 802.11 wireless LAN management frame
    Tagged parameters (19 bytes)
      Tag Number: 0 (SSID parameter set)
      Tag length: 15
      Tag interpretation: roguepeertopeer
      Tag Number: 1 (Supported Rates)
      Tag length: 4
      Tag interpretation: Supported rates: 1.0 2.0 5.5 11.0 [Mbit/sec]
0000  40 00 00 00 ff ff ff ff ff ff 00 02 2d 1b 51 ca @...........-.Q.
0010  ff ff ff ff ff ff 60 07 00 0f 72 6f 67 75 65 70 ......`...roguep
0020  65 65 72 74 6f 70 65 65 72 01 04 02 04 0b 16    eertopeer......
```

Indeed, it is possible to tell if someone starts an actively scanning card in ad hoc mode, and quite a bit of useful information can be gleaned from a single frame. Most relevant are the SSID and the MAC address, since they can be used to track down a particular card and/or person.

UNAUTHORIZED ACCESS POINT

The next test was to confirm the possibility of detecting a rogue access point. A tcpdump session was started, and then a Cisco Aironet 340 access point was turned on. After the access point had finished booting, the dump was examined with ethereal, and a large number of beacon frames sent out by the access point were found. The following is one such frame, again dissected by ethereal:

```
IEEE 802.11
    Type/Subtype: Beacon frame (8)
    Frame Control: 0x0080
      Version: 0
      Type: Management frame (0)
      Subtype: 8
      Flags: 0x0
        DS status: Not leaving DS or network is operating in AD HOC mode (To DS: 0  From DS: 0) (0x00)
        .... .0..   = Fragments: No fragments
        .... 0...   = Retry: Frame is not being retransmitted
        ...0 ....   = PWR MGT: STA will stay up
        ..0. ....   = More Data: No data buffered
        .0.. .... = WEP flag: WEP is disabled
        0... ....   = Order flag: Not strictly ordered
    Duration: 0
    Destination address: ff:ff:ff:ff:ff:ff (ff:ff:ff:ff:ff:ff)
    Source address: 00:40:96:36:88:23 (Telesyst_36:88:23)
    BSS Id: 00:40:96:36:88:23 (Telesyst_36:88:23)
    Fragment number: 0
    Sequence number: 0
IEEE 802.11 wireless LAN management frame
    Fixed parameters (12 bytes)
      Timestamp: 0x0000000000019274
      Beacon Interval: 0.102400 [Seconds]
      Capability Information: 0x0021
        .... ...1   = ESS capabilities: Transmitter is an AP
        .... ..0.   = IBSS status: Transmitter belongs to a BSS
        ...0 ....   = Privacy: AP/STA cannot support WEP
        ..1. ....   = Short Preamble: Short preamble allowed
        .0.. ....   = PBCC: PBCC modulation not allowed
        0... ....   = Channel Agility: Channel agility not in use
        CFP participation capabilities: No point coordinator at AP (0x0000)
    Tagged parameters (31 bytes)
      Tag Number: 0 (SSID parameter set)
      Tag length: 18
      Tag interpretation:
      Tag Number: 1 (Supported Rates)
      Tag length: 4
      Tag interpretation: Supported rates: 1.0(B) 2.0(B) 5.5 11.0 [Mbit/sec]
      Tag Number: 3 (DS Parameter set)
      Tag length: 1
      Tag interpretation: Current Channel: 11
      Tag Number: 5 ((TIM) Traffic Indication Map)
      Tag length: 4
      Tag interpretation: DTIM count 1, DTIM period 2, Bitmap control 0x0,
                          (Bitmap suppressed)
0000  80 00 00 00 ff ff ff ff ff ff 00 40 96 36 88 23   ...........@.6.#
0010  00 40 96 36 88 23 00 00 74 92 01 00 00 00 00 00   .@.6.#..t.......
0020  64 00 21 00 00 12 00 00 00 00 00 00 00 00 00 00   d.!.............
0030  00 00 00 00 00 00 00 00 01 04 82 84 0b 16 03 01   ................
0040  0b 05 04 01 02 00 00                              .......
```

UNAUTHORIZED CLIENT

The final condition tested for was unauthorized clients. The first scenario considered (the more likely scenario) is that someone brings a foreign card and powers it up with the wrong SSID. If the card was actively scanning, probe requests would be seen from this card as it attempted to find an access point. The second scenario is that someone brings a foreign card and powers it up with the correct SSID. This one turns out to be a little more problematic to detect, in that there will be only a few 802.11 management frames to trigger an alert, and then more "normal" traffic. This is problematic primarily because of the way RFMON_ANYBSS mode on the Cisco card works – despite its name, the card cannot receive packets simultaneously from all BSSes in range, espe-

cially if those BSSes use different frequencies. The consequence is that it takes some manual intervention to sniff traffic from a particular BSS – see "Problems and Complications," below, for more details on this problem and how to work around it. This problem was ignored and instead the focus was on the few 802.11 management frames that do show up readily in the sniffer; both scenarios turned out to produce similar probe requests, so both scenarios are treated as identical.

The dissected probe request sent out by this card:

```
IEEE 802.11
    Type/Subtype: Probe Request (4)
    Frame Control: 0x0040
        Version: 0
        Type: Management frame (0)
        Subtype: 4
        Flags: 0x0
            DS status: Not leaving DS or network is operating in AD HOC mode (To DS: 0  From DS: 0)
(0x00)
            .... .0.. = Fragments: No fragments
            .... 0... = Retry: Frame is not being retransmitted
            ...0 .... = PWR MGT: STA will stay up
            ..0. .... = More Data: No data buffered
            .0.. .... = WEP flag: WEP is disabled
            0... .... = Order flag: Not strictly ordered
    Duration: 0
    Destination address: ff:ff:ff:ff:ff:ff (ff:ff:ff:ff:ff:ff)
    Source address: 00:02:2d:1b:51:ca (Agere_1b:51:ca)
    BSS Id: ff:ff:ff:ff:ff:ff (ff:ff:ff:ff:ff:ff)
    Fragment number: 0
    Sequence number: 1
IEEE 802.11 wireless LAN management frame
    Tagged parameters (13 bytes)
        Tag Number: 0 (SSID parameter set)
        Tag length: 9
        Tag interpretation: roguehost
        Tag Number: 1 (Supported Rates)
        Tag length: 4
        Tag interpretation: Supported rates: 1.0 2.0 5.5 11.0 [Mbit/sec]
0000  40 00 00 00 ff ff ff ff ff ff 00 02 2d 1b 51 ca   @...........-.Q.
0010  ff ff ff ff ff ff 10 00 00 09 72 6f 67 75 65 68   ..........rogueh
0020  6f 73 74 01 04 02 04 0b 16                       ost......
```

PROBLEMS AND COMPLICATIONS

A few problems came to light with the Cisco card and driver that need to be mentioned.

The first problem is that the Cisco card, by default, even in RFMON and RFMON_ANYBSS modes, does not actively scan for traffic on all channels at all times. The following are the conditions under which it will rescan for BSSes:

- When the card is first inserted
- When the interface enters or leaves promiscuous mode
- When synchronization with the current BSS is lost (due to interference, moving out of range, or anything else that would cause the loss of a few beacon frames)
- When the /proc entry /proc/driver/aironet/eth0/BSSList is opened for writing (use touch /proc/driver/aironet/eth0/BSSList)

All of these conditions will "kick" the card into rescanning. To build a practical detection device, the card should be kicked at regular intervals, perhaps every minute. A simple script to touch the BSSList file every minute will do the trick.

Second problem: not all the BSSes in the range showed up reliably in the file /proc/driver/aironet/eth0/BSSList.

When the card is put into RFMON mode, transmitting is disabled, so the card cannot scan actively for BSSes by sending out probe requests. Therefore, the card must use passive scanning instead – instead of sending out probe requests, the card listens for beacons. Passive scans use a timer – the card will listen for beacon frames until the timer expires and then will move to another channel. The problem with the Cisco card is that this timer is set too low. The default value is 40ms, which was insufficient on our test network to notice all BSSes, regardless of the range or relative signal strength of the access points. The solution was to add this line to the card initialization routine, setup_card, in airo.c:

```
cfg.beaconListenTimeout = 120
```

Tripling this timeout made BSS detection work reliably. Consequently, all of our access points showed up in BSSList, all the time.

Third problem: despite its name, even putting the card in RFMON_ANYBSS mode did not cause the card to receive traffic from all of our access points, which were all using different frequencies and were probably synchronized differently.

The card itself chose a BSS to synchronize to based on its own algorithm (probably on its assessment of the relative signal strength). The problem with this is that we *want* to see traffic from all BSSes in range, not just those that happen to have the strongest signals. A way could not be found to disable this feature on the Cisco card, but there is a workaround — the Linux driver provides a /proc interface to set a preferred AP. Once the list of BSSes in range of the scanner is found (/proc/driver/aironet/eth0/BSSList), choose the one to monitor and enter the MAC address in the file /proc/driver/aironet/eth0/APList. This will force the card to synchronize with that BSS and switch to that channel, after which traffic from that BSS can be received and used for signal strength assessments or monitoring for suspicious activity.

CONCLUSIONS

These simple tests confirm that there are 802.11 frames that are characteristic of typical rogue access points and unauthorized ad hoc networks, and that these frames can be detected and analyzed using off-the-shelf components and free software.

Using these concepts along with a database of trusted access points and cards and the fingerprints of suspicious frames, ethereal could be used as a fundamental building block in a full-blown 802.11 intrusion detection system.

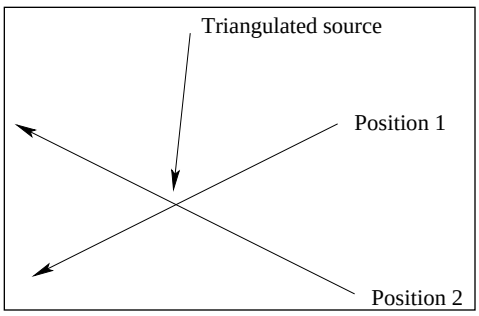


Fig. 1

Tracking

While detection of rogue access points and unauthorized ad hoc networks is certainly useful, physically tracking the offender is more interesting.

THEORY AND TECHNIQUES

Most triangulation done today uses directional techniques. A dish, yagi, or other directional antenna with a narrow range is attached to a receiver, and an area is scanned for the strongest signal. The station is moved some distance away, the area is scanned again, and then the position is found using simple trigonometry.

In practice, there is usually more involved than this. Radio propagation is affected by terrain, obstructions, heat and atmospheric effects, reflections, refraction, antenna

design, and many other factors that could affect the apparent location of the transmitter. The apparent location of the strongest signal may not actually be the location of the transmitter. A full discussion of these effects is beyond the scope of this paper.

Regardless, the basic theory is sound and can be a very effective technique to locate a transmitter.

Another technique, not quite as common today, is to locate the transmitter using only the relative signal strength at various well-placed locations around the area and, if known, the free-space propagation losses and the power of the transmitter.

If we know the output power of the transmitter, the gain of the receiving and transmitting antennas, power drop-off, and the received signal strength, the location of the transmitter can be narrowed down to two possible places with only two samples.

At point A, from the signal strength and power drop-off equation, we'd be able to determine the distance from the transmitter. At point B, we'd also know that, so the location of the transmitter will be found at the point(s) where both d_A and d_B meet:

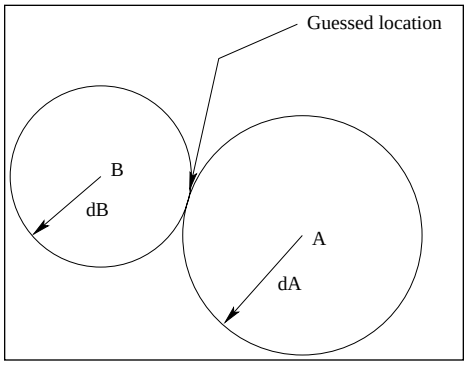


Fig. 2

If we don't know the strength of the transmitter or the gain of any of the antennas involved, we just have to introduce one more unknown, a signal strength ratio; the problem then turns into a simultaneous equation with three unknowns. Thus, a minimum of three data points are needed. If these data points are well placed, there should only be one or two realistic solutions to the equations.

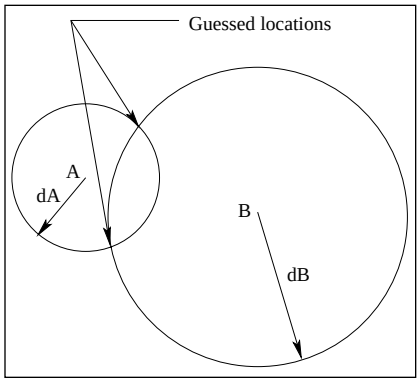


Fig. 3

PRACTICAL APPROACH

The approach taken is a mixture of experimentation and extremely simplified indoor propagation theory. In "A Study of Indoor Radio Propagation," available at *http://www.sss-mag.com/indoor.html*, researchers have come up with an approximation for losses at 2.4GHz inside buildings.

What was needed was a solution that would work well in any office environment. Instead of meticulously accounting for every possible loss or gain, "A Study of Indoor Radio Propagation" provided a generic model for indoor radio propagation that, while not perfect, would work well in most indoor environments.

TEST ENVIRONMENT

Testing was done in a typical office suite, an area of about 30 by 20 meters. There were a large number of offices, and some larger open rooms with no more than six cubicles each. Structurally, the office is largely concrete and steel. This is fairly typical of most offices, and is probably similar to other offices that use wireless networks.

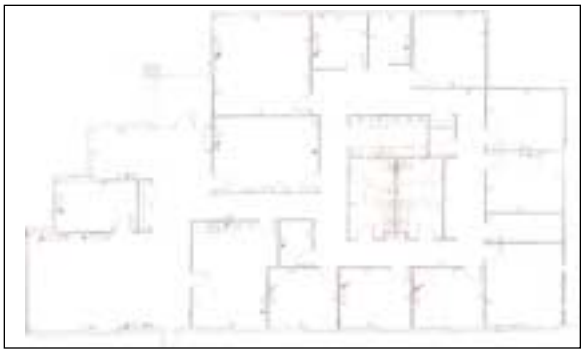


Fig. 4. The office used for testing

Three separate access points were used for testing, placed at random locations around the office. The first two were Cisco AP350s. One was equipped with two antennas. The second did not have any antennas. The third was an Orinoco AP-1000 with a small Lucent antenna.

DATA GATHERING

The first goal was to make some plots of distance vs. relative signal strength as reported by the card's driver, so that the hypothesis that the signal strength would drop off at some predictable rate could be tested. The program iwspy was used to gather signal strength statistics on a particular MAC address:

```
# iwspy eth0 0b:0b:0b:0b:0b:0b
#
```

Then, successive calls to "iwspy eth0" returned the signal strength of packets received from that transmitter.

The tester then simply walked around the office, taking and recording readings at random locations around the office. After some processing, three tab-delimited datafiles containing the distance from the transmitter in the first column and the recorded signal strength in the second were generated.

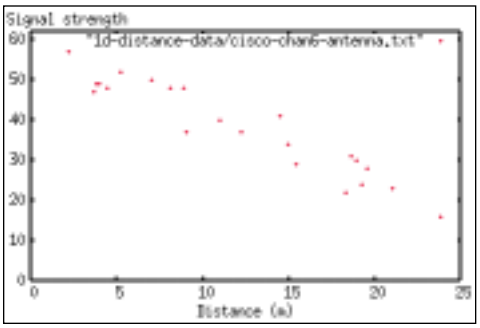


Fig. 5: Cisco access point – antenna

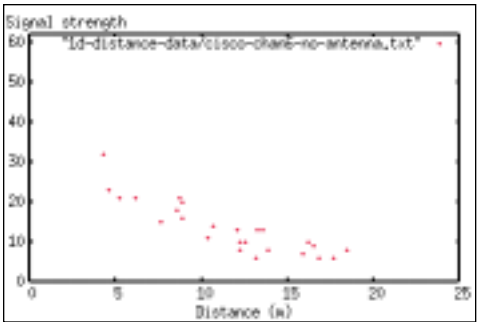


Fig. 6: Cisco access point – no antenna

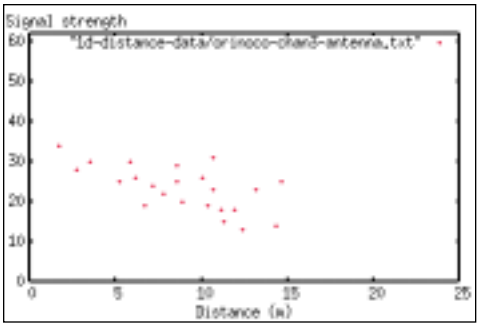


Fig. 7: Orinoco access point – antenna

DISTANCE–SIGNAL STRENGTH TRENDS

The distance–signal strength plots for each of the three access points are shown here.

DISTANCE–SIGNAL STRENGTH CURVE FITTING

The next step was to determine whether or not the data could be fitted to an equation approximating indoor losses at 2.4GHz. A simple least-squares fitting present in Gnuplot was used to verify this. If the curve fit the data "well," one would assume that the equation was a good approximation.

First, signal strength values returned by the iwspy interface (RSSI values, or Received Signal Strength Indicator) needed to be converted to dBm in order to use canned indoor loss equations. dBm is a logarithmic measure commonly used when dealing with low-power radio equipment.

The values returned by iwspy were most likely already logarithmic, to make them easier to read. If they were logarithmic, they should have some linear relationship to the signal strength in dBm.

The relationship between the values returned by iwspy and dBm was found to be linear (determined experimentally using the Windows interface, so this is not exact):

$$P_{dBm} = 1.205\, P_{non\text{-}normalized\ \%} - 101.07$$

$$P_{non\text{-}normalized\ \%} = .83\, P_{dBm} + 83.891$$

If these values are "normalized" (this is an option that can be turned on in the card), the equation becomes:

$$P_{dBm} = .602\, P_{normalized\ \%} - 101.07$$

$$P_{normalized\ \%} = 1.66\, P_{dBm} + 167.782$$

Currently, the iwspy ioctl for the Linux driver returns non-normalized values (incidentally, the Cisco client for Windows returns normalized values), so the first two equations should be used to convert between dBm and iwspy percentages, but this may change at some future date.

Rather than pre-process the datafiles, one of the above conversion functions was included in the curve-fitting function.

For the curve-fitting function, an equation was used from "A Study of Indoor Radio Propagation," which presents a modification to the traditional free-space loss equation to account for indoor attenuation, reflection, refraction, and interference. The researchers experimentally found that the following equation could be used to approximate losses indoors at 2.4GHz in most typical buildings, where D is the distance between the transmitter and receiver in meters:

$$\textit{Path loss (in dB)} = 40 + 35\log_{10}D$$

All of the unknown constants as well as the 40 term were gathered into one constant, C, where C is some unknown constant representing the output power and static cumulative gains and losses due to attenuation, the antenna, and other factors:

$$\textit{Received signal strength (dBm)} = C - 35\log_{10}D$$

Using the equations above to convert to non-normalized signal strength percentage (to match the output from Linux iwspy):

$$\textit{Received signal strength (non-normalized \%)} = .83(C - 35\log_{10}D) + 83.891$$

To the right are the resulting plots. They fit fairly well, so we can assume that the above equation is a suitable approximation for indoor propagation losses.

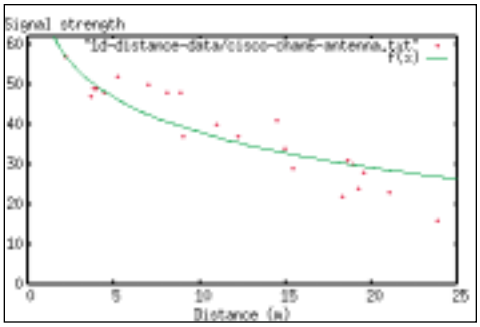


Fig. 8: Cisco access point – antenna

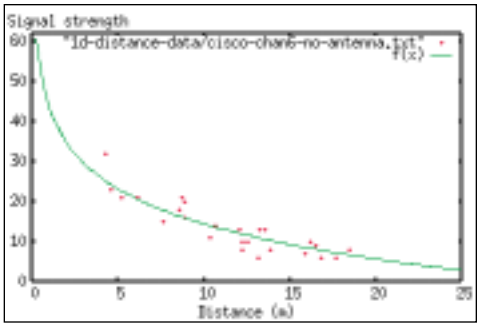


Fig. 9: Cisco access point – no antenna

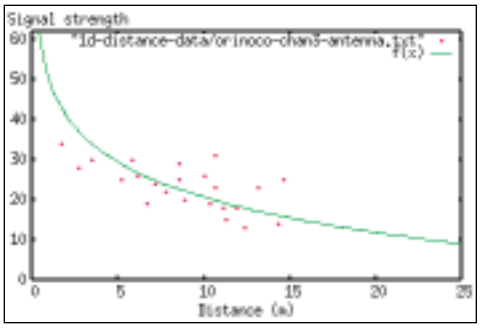


Fig. 10: Orinoco access point – antenna

3-D CURVE FITTING

The final step in tracking was to write the distance-power approximation equation as a function of a position (x,y) and three unknowns, the unknown coordinates (A,B) of the transmitter and an unknown constant C. The resulting equations look like this:

$$\textit{Received signal strength (dBm)} = C - 35\log_{10}\sqrt{(A-x)^2 + (B-y)^2}$$

$$\textit{Received signal strength (non-normalized \%)} = .83(C - 35\log_{10}\sqrt{(A-x)^2 + (B-y)^2}) + 83.891$$

This equation could then be plugged into Gnuplot for fitting. Below is one sample plot. The initial parameter file, used to provide initial guesses to the best-fit algorithm., consists of a = 10, b = –10, and c = –20. It's best to guess physically reasonable values, so coordinates near the center of the office space were chosen, and an initial value of -20 for c. The results are summarized in the next section.

RESULTS

The Gnuplot program, when run, will display a graph containing the parameters of the best fit found and give an estimate of the error. The following is a table of actual values and the best-fit values found from the data sets.

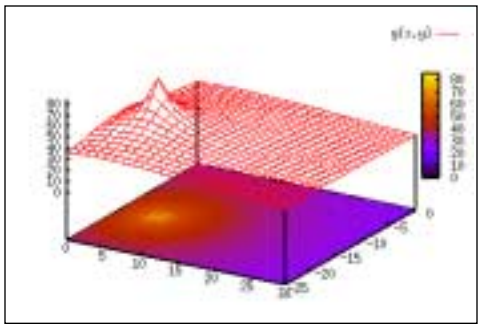


Fig. 11

Two results were very encouraging; one was not. While the first two were placed within a couple meters of the actual location of the transmitter, the results for the Orinoco antenna placed the transmitter about 8 meters south of its actual location.

	Actual location (x)	Actual location (y)	Guessed location (x)	Guessed location (y)	Guessed constant C (related) to the transmitter's output power
Cisco with antenna	5.18	-14.94	5.15986 +/- 0.9969	-14.847 +/- 0.7456	-20.1008 +/- 1.706
Cisco without antenna	12.50	-18.29	11.3324 +/- 0.2842	-16.2561 +/- 0.55	-50.4432 +/- 0.6815
Orinico with antenna	14.17	-13.11	14.6243 +/- 1.3	-21.1688 +/- 2.547	-34.1176 +/- 2.639

The causes for this discrepancy could be many. Among the primary suspected causes:

The location of the samples is of utmost importance, as are the number of data points.

- The distance–signal strength trend was not very strong for the Orinoco antenna. There was a large cluster, but not many points from very near the antenna or very far, essential for getting a proper fit.
- The transmitter was centrally located, so sampling a wide range of distances was not possible.
- The transmitter is located in a room with a lot of metal equipment, probably scattering the transmissions a great deal.
- Some "hot spots" near the southern end of the building pulled the fit too far south. The east-west coordinate was a good fit.

Regardless, the basic theory is sound and more careful data gathering and more data points would probably yield better results.

This has driven home an important lesson, however – the location of the samples is of utmost importance, as are the number of data points. If the sensing stations are stationary, there should be as many as possible and they should be as far away from each other as possible, probably in the furthest corners of the building, as long as reception is still possible.

LIMITATIONS

There are a number of limitations and restrictions to keep in mind when deploying a system like this.

- Using these techniques, it is impossible to trace a card or access point that is not transmitting. It would not be possible to track down the location of a truly passive sniffer using these methods.
- The Cisco driver and card are limited in how fast and how reliable their scans are. Therefore, it is possible for intermittent traffic to be missed completely.
- The Cisco card can only reliably sniff traffic from one channel at a time. If something interesting shows up, all cards should stop scanning and pay attention to the particular channel of interest until enough data is gathered to track down a location.
- If there is more than one tracking station, they should use identical hardware (antenna included) and driver revisions.
- The more data points, the better.
- Tracking stations, if stationary, should be in well-placed locations, so fit errors can be kept to a minimum. Ideally, they should be placed far away from each other, in open locations so path losses and attenuation can be kept to a minimum, but also placed such that all stations can detect traffic from a card or access point anywhere in the building.
- Administrators should also consider an active scanning station. If an unauthorized transmitter is found, one station might be selected to send pings, probe requests, or anything that might prod the transmitter into sending responses, from which signal strength information could be gleaned.
- A high-quality, high-gain antenna is important.
- No effort was made to handle the case of a mobile transmitter.
- The methods used here can occasionally yield results with extremely large errors, especially if the input set is not rational. It's a good idea to pick meaningful initial guesses for the fitting algorithm. It may also be a good idea to reject physically unlikely results — for instance, if the fitted constant C seems to indicate that an unauthorized transmitter is transmitting at 1000 watts, the result can probably be

discarded out-of-hand. Likewise, it's unlikely to receive transmissions from a transmitter miles away. It is possible, but unlikely.

- These methods could not, by default, detect cards or ad hoc networks that happen to be running on channels other than the ones reserved for use in North America.

IDENTIFIED ISSUES AND AREAS FOR IMPROVEMENT

- Make traffic detection more reliable and less dependant on channel scans. Ideally, a truly promiscuous 802.11 sniffer that could monitor traffic on all frequencies and all spread-spectrum synchronizations at the same time would be preferred.
- Be aware that the relationship between dBm and % signal strength from the Cisco card is not exact and may not be an exactly linear relationship.
- Integrate a GPS unit so as to allow for active processing and simplified data gathering.
- Build a working prototype system involving at least three stationary monitoring stations. Test it with mock attacks on the local network.
- Find a more robust IDS than ethereal and some scripting glue. Hopefully, snort's author(s) will integrate ethereal's protocol analysis engine.
- Write a protocol to link monitoring stations so that they can compare data with each other and cooperatively locate rogues – part of the working prototype.
- Consider adding some real path-length and wall-attenuation algorithms to reduce the amount of scatter.
- Consider full-blown physical modeling of the office.
- Gather more data to verify that $35 \log_{10}(D)$ really is a good approximation.
- Build some profiles of expected power outputs of various vendors' access points and cards with and without antennas, in an attempt to minimize the unknowns.
- Experiment with "real" triangulation using directional antennas.
- Gather hardware documentation on the Cisco Aironet cards. RFMON and RFMON_ANYBSS are still confusing, especially with synchronization and channel scanning.
- Get info on new Cisco RIDs and add them to the driver.
- Experiment with monitor mode on other cards and chipsets.

REFERENCES

Donald L. Schilling, Raymond L. Pickholtz, and Laurence B. Milstein, "Spread Spectrum Goes Commercial," *The IEEE Spectrum*, August 1990.

"A Study of Indoor Radio Propagation" – http://www.sss-mag.com/indoor.html. Thanks to the folks at Spread Spectrum Scene for this article, as it provided a good approximation for indoor radio propagation losses. This was essential to feed to the curve-fitting algorithms, and it yielded good results.

"Cisco – Wireless Point-to-Point Quick Reference Sheet"

http://www.cisco.com/warp/public/102/wwan/quick-ref.pdf

"Peter Mikulik's gnuplot page"
http://www.sci.muni.cz/~mikulik/gnuplot.html#pm3d

GNU Octave Documentation
http://www.octave.org/doc/octave_toc.html

Linux Aironet driver – comments and source were invaluable.
http://sourceforge.net/projects/airo-linux/

Ethereal home page – great documentation and source for disassembling all sorts of network protocols.
http://www.ethereal.com/

IEEE – lots of great collected information on 802 standards.
http://www.ieee.org/

I am indebted to V.N. Padmanabhan and P. Bahl for their article "RADAR: An In-Building RF-Based User Location and Tracking System," available at *http://research.microsoft.com/~padmanab/*. This article was not a reference, per se (in fact, it was discovered after this paper was nearly finished), but it was interesting to see that some of my work had been a duplication of the efforts of someone else, and it provided independent confirmation that this method can work in an office environment. My approach was not nearly as elegant or as scientifically rigorous as theirs, but it was encouraging to know that the results were similar.

Special thanks to Richard Stallman for GNU.

combating the perils of port 80 at the firewall

by Avishai Wool

Dr. Avishai Wool is a co-founder and chief scientist of Lumeta Corporation, and an Assistant Professor at the Department of Electrical Engineering Systems, Tel Aviv University, Israel.

yash@acm.org

Last summer, the Code Red worm and its relatives hit Web servers all over the Internet. The worm spreads by requesting a Web page from a Web server running an un-patched version of Microsoft's Internet Information Server (IIS). The request is issued using the ubiquitous HTTP protocol and is sent to the server's default port 80. However, the requested page's URL is carefully and maliciously crafted to trigger a buffer overflow on the Web server, thus infecting the server with a copy of the worm. The newly infected server then turns around and does the same to other Web servers.

In the October 2001 "Inside Risks" column,[1] Somogyi and Schneier describe the worm and point out the general susceptibility of the Internet, and all who connect to it, to such worms. They argue that "http has become Internet-connected computers' lingua franca," yet popular Web servers have not been properly engineered to eradicate remotely exploitable vulnerabilities, so companies and customers are assuming increased risk in deploying and using the Web.

While the issues that Somogyi and Schneier raise are valid, the article points the finger only at software vendors such as Microsoft. The trouble with this approach is that it absolves other corporations and their network security staff of any responsibility. The article makes it sound as if the only way you can fight back and protect your network is to wait for, and install, the latest patches from Microsoft.

Installing security patches is important, and getting software vendors to improve the security of their products is indeed an excellent idea. It just requires time, effort, and money. In the meantime, though, there is something very simple that you can do today that will greatly decrease the ability of Code Red and its ilk to spread, and significantly reduce the risk to your internal network and to public Internet sites in general.

You need to ensure that your Web servers are properly quarantined by a firewall. The operative word here is "properly": practically all Web servers are placed behind firewalls, which are supposed to shield the servers from attacks. Unfortunately, these firewalls are not configured to do everything they could to combat HTTP-based worms such as Code Red. Evidence collected from firewall configurations run through the Lumeta Firewall Analyzer shows that HTTP traffic is often allowed through firewalls unhindered.[2] This policy is too liberal.

The point to remember is that a Web server is supposed to serve. It is passive. Under normal circumstances, a Web server waits for HTTP requests and serves the requested pages. A healthy Web server does not initiate requests to other Web servers. Only Web browsers actively request pages. However, once a Web server has been infected with a Code Red worm, it starts behaving like a Web browser – actively sending its maliciously crafted HTTP requests to other Web servers, either on your internal network or on the Internet. There is no reason to let your Web server initiate HTTP requests like this.

So here is the recipe. If you have a modern (stateful) firewall, you need two firewall rules to protect a Web server, in this order:

1. Allow the HTTP service from anywhere to YourWebServer.
2. Drop any service from YourWebServer to anywhere.

Rule 1 allows Web browsers on the Internet to request pages from your Web server and allows the server to serve the pages; a modern firewall can match the server's responses to the browsers' requests. This is what those state tables are for. (Technically, the firewall keeps track of the TCP three-way handshake so it can distinguish between the computer that initiated the HTTP session (the browser) and the computer that responds (the server).)

You probably already have something like rule 1 in your firewall's rule set. What you need to add is rule 2, which prevents your Web server from turning around and starting to actively request pages. Once the Web server is blocked from behaving like a Web browser, it will not be able to spread HTTP worms.

Now, actually, rule 2 prohibits the Web server from initiating any traffic. Taken literally, rule 2 may be too restrictive for the Web server to function, e.g., the Web server may need to initiate domain name queries. Also, Microsoft's "Windows Update" feature works by having the computer access Microsoft's own Web site using a Web browser that is embedded into Microsoft's operating systems. You'll need to add rules dealing with such exceptions before rule 2 – just make them specific only to those Web sites your server needs access to.

Note that the above recipe will not prevent your externally-visible Web server from getting infected in the first place. Get a patch from your software vendor for that. What the recipe will do is make sure your Web server is properly quarantined. It will prevent your Web server from infecting your internal networks, partners and clients. And, it will reduce the spread of the next HTTP worm that comes along, even before Microsoft issues a patch, and even if the next worm targets, say, Linux-based Apache Web servers.

So if you run a Web server, don't just passively wait for your software vendor to issue security patches. Take action. Review your firewall rules and make sure that your Web server is not allowed to behave like a browser. It's good for your network's security and it's important for the Internet as a whole.

Notes

1. S. Somogyi and B. Schneier, "Inside Risks: The Perils of Port 80," *Communications of the ACM*, vol. 44, no. 10, October 2001, 168.

2. A. Wool, "Architecting the Lumeta Firewall Analyzer," *USENIX Security Symposium*, Washington, D.C., August 2001, 85–97.

ISPadmin

Stopping Spam: Part 1

by Robert Haskins

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

rhaskins@usenix.org

Introduction

This edition of ISPadmin covers the methods used by ISPs to stop email spam on the server side. While the focus is on service providers, any enterprise can use the methods outlined below. The term "spam" will be used interchangeably with "unsolicited bulk email" (UBE) and "unsolicited commercial email" (UCE). If you ever wondered, the use of the word "spam" comes from episode 25 of *Monty Python's Flying Circus*, recorded on June 25, 1970, and broadcast sometime later in 1970.

Different people define spam in different ways. To be sure, the individuals and corporations who generate UCE do not consider their communications to be spam. On the other side of the spectrum, some recipients view any message that is commercial in nature which goes to more than two people to be UBE. Here are some attributes which may cause a message to be considered spam:

- Commercial content
- Large recipient list and/or use of BCC
- Concealing or forging message headers
- Numerous messages sent in a short period of time
- Use of recipient addresses without the owners' explicit approval
- Use of an open mail relay to send messages
- Invalid/Unresolvable To or From address header(s)

How Big Is the Problem?

It is difficult to come up with exact figures, but a reasonable estimate is 30% of all email messages could be considered spam. Some of the methods outlined below will catch 50% to 80% of UCE going through a provider's network. Depending on the exact situation, these numbers can vary significantly.

Where Can Spam Be Stopped?

There are two places to catch spam: inbound (messages on the recipient's, or recipient provider's, network) and outbound (messages on the sender's, or sender provider's, network). Both will be covered in detail: inbound in this installment and outbound next time. Usenet spam will be covered only briefly, as it is a much easier problem to solve than email spam.

INBOUND SPAM

Inbound UCE can be controlled utilizing the following methods:

- Source identification (including Realtime Blackhole List, Open Relay Database)
- Source analysis (Spamassassin)
- Source analysis services (Brightmail, Postini, etc.)
- Distributed key methods (Vipul's Razor and Rhyolite Software's Distributed Checksum Clearinghouse, or DCC)
- Electronic coin methods (camram, which is actually a combination of inbound and outbound methodologies)

Each method will be examined in detail. The newest and most promising of the bunch are the distributed key (Vipul's Razor and DCC) and electronic coin (camram) methods. Brightmail and Postini are very effective but are costly.

OUTBOUND SPAM

Outbound UBE can be controlled using the methods below:

- Log analysis
- MTA controls
- Authentication before sending (POP before SMTP)
- Mail message metering

Each one of these methods will be examined in detail. Disclaimer: this author developed a pending patent for the "mail message metering" method. (An article by this author titled "Mail Message Metering" in the December 2000 issue of *;login:* covered this solution in detail.) This article covers each method outlined above as it pertains to inbound spam. The next installment of ISPadmin will cover methods for dealing with outbound UBE.

Sendmail

Sendmail began implementing anti-spam relaying mechanisms as of about version 8.8. While most of the focus is on stopping outbound spam, version 8.8 included support for Realtime Blackhole List (RBL).

Here is a list of a few Sendmail features that can help control inbound spam:

- Access control database mechanism (/etc/mail/access)
- Connection rate throttle
- Limited number of recipients per message
- Disallowing connections from open relays, dial-up IP addresses, or black hole lists (e.g., RBL)

THE ACCESS DATABASE

All Sendmail databases are created from simple text files with key/value pairs. A program converts the text file into a quickly searched database format. Values and their meanings include:

OK	Allow message to pass; overrides other checks.
REJECT	Refuse connections from that host.
DISCARD	Silently delete message.
<message>	Reject message, informing sender of <message>.
RELAY	Allow domain/IP address to relay mail through this server (more on this in next installment's discussion of outbound spam).

For example, the entry below would reject any mail coming from the domain "knownspammer.com" but allow mail from "friend@knownspammer.com", and silently discard mail from "spammer@spamhaus.com":

knownspammer.com	REJECT
friend@knownspammer.com	OK
spammer@spamhaus.com	DISCARD

All Sendmail databases are created from simple text files with key/value pairs.

Note that the configuration file must be built with the FEATURE(access_db) in your sendmail.mc and converted to sendmail.cf utilizing the m4 utility. Also, the Sendmail daemon must be restarted to re-read the new configuration.

SENDMAIL CONFIGURATION TWEAKS

A connection rate throttle will limit the number of connections per second from a given server. The line below (in sendmail.mc or its equivalent) enables this feature:

```
define('confCONNECTION_RATE_THROTTLE',3)dnl
```

Limiting the number of recipients per message can be a good way to limit some spammers. (Enabling this feature may cause problems with legitimate bulk emailers.) The restriction can be accomplished by adding the following line to the sendmail.mc:

```
define('confMAX_RCPTS_PER_MESSAGE',25)dnl
```

For the purposes of this article, "simple filter service" means a service that requires nothing more than an existing message transport agent, or MTA (e.g., Sendmail). Other filter services that require additional machines (e.g., certain Brightmail offerings) will be covered below.

The Mail Abuse Prevention System, or MAPS, maintains several IP databases for "black holing" spammers. When subscribing to a service like RBL, the service essentially routes traffic from sites in the list to the bit bucket. As a result, the end subscriber will never be able to receive packets (i.e., email) from a machine on the list. MAPS is a commercial service, except for "Individual/Hobby" sites. The databases it maintains are as follows (note that MAPS controls what IP addresses are placed into these lists):

RBL+	combines RBL, RSS and DUL
RBL	Spammers, the granddaddy of them all
RSS	Relay Spam Stopper, list of open mail relays
DUL	Dial-Up User List, contains addresses that shouldn't be originating mail (for example, dial-up IP pools, DSL/cable modem customer IP pools, etc.)
NML	Non-Confirming Mailing List, mailing lists that do not verify email addresses of subscribers

More information is available on the MAPS Web page. Such lists can be activated within Sendmail by adding line(s) similar to the following to the sendmail.mc file:

```
FEATURE(dnsbl, 'blackholes.mail-abuse.org',
    'Rejected - see http://www.mail-abuse.org/rbl/')dnl
```

Note that depending upon what type of services desired, changes to the DNS configuration need to be made in order to properly activate the MAPS services.

One other black hole service bears mentioning: the "OR" family of open relay black hole services. This service, in its various iterations, has been the subject of much debate within the anti-spam community. This is due to the ultra-strict checking that some of these open relay checkers perform. Several have been shut down due to litigation and/or threats of litigation. The Open Relay Database (ORDB) is one of the more well known ones. The OsiruSoft site contains many others, as well as the ability to check names/IPs to see if such names/IP addresses appear in some well-known open relay databases. This can be a very useful tool. Care must be taken, however, as some of these services are considered by some to be overly restrictive.

What to Do With Email Tagged as UCE?

One problem common to many solutions that attempt to filter spam (including the distributed key method as well as services like Brightmail) is what to do with the message once it is identified as UCE. The possible dispositions of such messages are:

- Deletion
- Tagging
- Sidelining

For a service provider, deletion of a potential spam message is most likely not an option. Paying customers do not appreciate having their mail deleted, given the lack of a common definition of spam as well as possible false positives (tagging messages as spam that are, in fact, not spam).

Tagging (adding a header – e.g., X-Spam-Score: – to a possible spam message) is a good idea, but, unfortunately, the only common mail client that supports arbitrary header filtering is Eudora 5.1. Neither Netscape Mail nor Outlook Express currently support such filters. (Mozilla 1.0RC2 as well as Netscape 7 Preview Release 1 both support this type of filtering.)

Sidelining involves sending a potential spam message to an alternate email box (usually a Webmail machine). Once the message is sidelined, the recipient needs to check the sidelined mailbox and dispose of the mail. Hopefully, it is all indeed spam. As a practical matter, until a perfect anti-spam solution is available, some non-spam mail will probably get tagged as spam. Sidelining requires additional hardware, as well as added complexity (and associated costs) for the organization running the email domain.

As a practical matter, until a perfect anti-spam solution is available, some non-spam mail will probably get tagged as spam.

Mail Filtering and UCE

Several programs exist which attempt to filter mail on their own, without external intervention. Such mail filters (and services) are a good method to stop spam.

STAND-ALONE MAIL FILTER PROGRAMS

Spamassassin can be used to identify spam by various attributes in the mail header and body. It has hooks for DCC and Razor (see below) and supports black holing via MAPS and other methods. Another such program (which is also a virus checker) is MailScanner. Figure 1 illustrates how these mail filter programs might be implemented in an ISP's mail infrastructure.

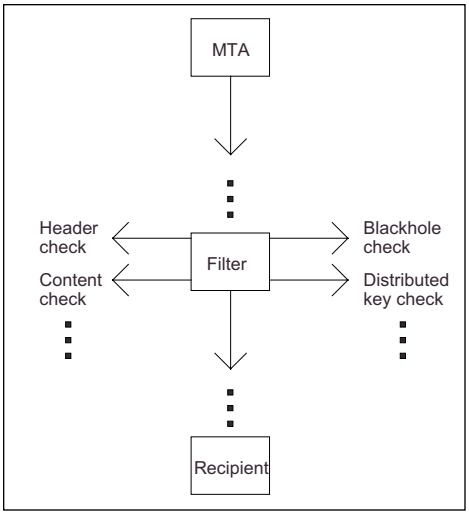


Figure 1: Message flow through static filter

The MTA box in Figure 1 represents Sendmail, qmail, etc. It has hooks (via procmail, milter or other mechanisms) to call an external program to process a message. In this case, the program is indicated by the box labeled "Filter." The filtering program then makes a number of checks against the header and body of the message, utilizing internal static filters, or external filter sources (such as black hole lists or distributed key hosts). If the message is identified as spam, a score is assigned to the message. The program then adds a header indicating that the message has been processed, along with the score it received. The message then is sent along to the rest of the mail infrastructure until it reaches the recipient. The recipient's mail client is responsible for determining what to do with messages tagged as spam.

The limitations of such static filters are such that they will never tag 100% of spam, and they also have false positives. Another downside is the requirement that such fil-

ters be constantly updated, as spammers are always creating new ways for their messages to escape filters.

DISTRIBUTED KEY METHODS

One of the most promising anti-spam ideas of late is the concept of exchanging keys summarizing mail messages with a mail server's peers. Essentially, these are static rules which count the number of times a server has processed a version of a message. The message checksums are then exchanged with other servers so that everyone knows to block a certain message based on its checksum.

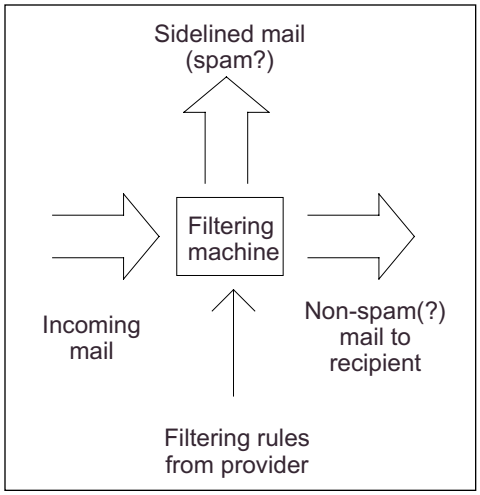


Figure 2: "Filter Service" message flow

The downside to such systems is the requirement the mail server keep "white hat" lists of legitimate bulk email, as such mail will have the same high hit counts that UBE will. If such mailing lists are not placed into the white hat list, then the legitimate bulk email will be tagged as spam. Also, the fuzzy algorithms which the programs use to identify spam need to be tweaked periodically as spammers adjust their methods to avoid detection systems.

However, a distributed key filtering method can block an impressive amount of spam with minimal work and is certainly worth considering. The best way to implement distributed key solutions is via something like Spamassassin (mentioned above).

FILTERING SERVICES

Services (such as Brightmail and Postini) are similar in nature to the DCC and Vipul's Razor applications. These services will, for a fee, maintain filter "lists" which identify certain messages as potential spam. These filtering rules are applied to incoming mail before hitting a customer's mail infrastructure. Figure 2 is an example of how such a filter service might be implemented.

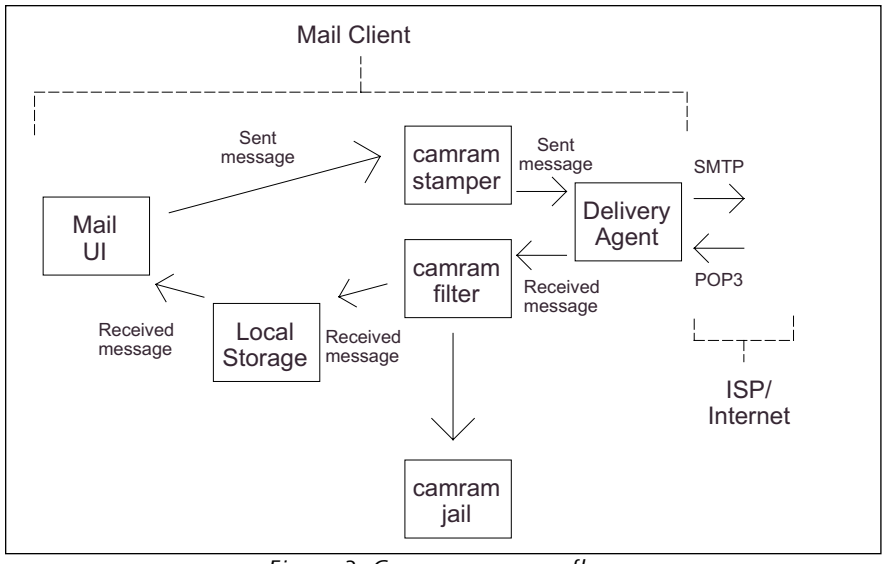


Figure 3: Camram message flow

Mail arrives at the filtering machine from the provider's mail relay. This filtering machine has human-built filters applied to it in a bulk manner from the anti-spam service provider, such as Brightmail. These rules are applied to incoming mail, and messages which the filters identify as UCE are sidelined to be viewed by the recipient at a later date. If the message is not tagged as spam, it continues through the provider's mail system.

Such anti-spam filtering services differ from the distributed key mechanisms in a two important ways:

- They use human-based rather than machine-based filters.
- They usually require additional hardware running in front of a customers existing mail relay(s).

These commercial services will catch a lot of spam. However, they can be costly (start-up costs range in the tens of thousands of dollars, plus a per-user per-month fee) and while good, are not perfect.

Electronic Coin Method

At least one solution, camram, is based on the idea of electronic "coins," also known as HashCash. Money is not really exchanged; rather, computational time is required on the sender's machine in order to generate the "coin." Figure 3 diagrams how camram would be set up in an ideal situation. "Ideal situation" refers to a mail client with embedded camram support.

It is important to realize that under this scheme, camram support must be on the same machine that is running the user interface. As a result, supporting a solution such as camram will require client-side changes. These changes are required so that it is relatively easy for one machine to generate a small number of coins; however, it is very time-consuming for a machine to generate a *large* number of coins.

One way to minimize such changes is to run camram in a proxy mode whereby camram functionality is implemented separately from the mail UI. This will likely be a "transitional" mode, as once code is placed in all widely used mail clients, the need for such proxy programs disappears.

In Figure 3, inbound mail comes into the client machine and passes through the camram filter, which determines whether or not appropriate "postage" exists for the message. If there is no or not enough postage, the message goes to a "jail," which is similar to sidelining. If there is enough postage, the message continues on its journey, eventually making it to the recipient. It is important to note that exception lists can be made for legitimate bulk mail that doesn't have appropriate postage.

Outbound mail passes through a camram "stamper" which "affixes" enough postage to the message and sends the message on its merry way. The recipient's mail client would need a camram decoder, if they wanted to filter spam. Otherwise, if they didn't use a camram decoder, the special headers would simply be ignored by the non-camram-aware mail headers. Of course, the recipient could choose to treat messages that haven't been signed by camram as junk mail, and deal with it at a later time.

A side effect of coin generation is the ability to easily add some level of public key encryption. With camram, you already know a lot about the sender, and adding a digital signature would be an easy modification to the protocol.

The camram concept's big appeal is its ability to significantly slow persons generating large amounts of spam. However, it still requires a white list for legitimate bulk emailers, so in that respect it is very similar to the distributed checksum method. A spammer would likely not use a solution such as camram, as the sender would quickly run out of "coins." The issue with an idea such as camram is simply one of adoption: the wider its use, the more useful it is. Time will tell if a HashCash-based solution works effectively or not.

Conclusion

I wish to thank Eric Johannson for his input into this article (Eric is one of the people behind camram, who is looking for help on the project; check the camram site for more information). Next time, I'll continue my look at spam by examining how to stop it at the outbound side. Until then, please send me your comments.

REFERENCES

abuse.net's anti-spam pages for admins – *http://spam.abuse.net/adminhelp/mail.shtml*

Blackmail – *http://www.jsm-net.demon.co.uk/blackmail/blackmail.html*

Brett Glass, "Stopping Spam and Malware with Open Source" – *http://www.brettglass.com/spam/paper.html*

Brightmail – *http://www.brightmail.com/*

Camram – *http://www.camram.org*

Dan Garcia's Spam Homepage – *http://www.cs.berkeley.edu/~ddgarcia/spam.html*

Eudora – *http://www.eudora.com/*

HashCash –*http://www.cypherspace.org/~adam/hashcash/*

Kai's spam shield – *http://spamshield.conti.nu/*

Mail Message Metering – *http://www.ziplink.net/ziplink/solutions/mmm/*

MailScanner – http://www.sng.ecs.soton.ac.uk/mailscanner/

MAPS/RBL/DUL – *http://www.mail-abuse.org/*

Milter – *http://www.milter.org/*

Monty Python Spam – skit *http://www.ibras.dk/montypython/finalripoff.htm#Spam*

Obtuse System Corp's Juniper firewall –*http://www.obtuse.com/smtpd.html*

ORDB – *http://www.ordb.org/*

OsiruSoft Open Relay black list – *http://relays.osirusoft.com/*

Postini – *http://www.postini.com/*

Procmail – *http://www.procmail.org/*

Qmail – *http://www.qmail.org*

Rhyolite Software's DCC – *http://www.rhyolite.com/anti-spam/dcc/*

Sendmail – h*ttp://www.sendmail.org/*

Sendmail's anti-spam pages *http://www.sendmail.org/antispam.html*

Spamassassin – *http://spamassassin.org/*

The Complete Monty Python's Flying Circus: All The Words (Pantheon Books, 1989); ISBN: 0679726470.

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

the problem of PORCMOLSULB

Can Root Be Controlled in Engineering Environments?

by Howard Owen

Howard Owen has been a tech junkie since Conroy's Life appeared in Life Magazine. He's been a professional geek since 1984. Howard loves Systems Administration because it sits at the interface between human beings and computer technology, and that's where the action is.

hbo@egbok.com

Introduction

The other day I received a call from a user who was having trouble checking in her latest changes to CVS. Since she was using a Linux box that was not supported by our group, I could have refused to look at the problem. But I'm pathologically interested in making sure our users get the most out of the computing environment. Besides, it could have been an issue with the CVS server, whose health as a system I am responsible for.

So I strolled over to her cube and had a look. Our CVS check-in process has hooks that cause email to be sent. I soon determined that the problem had to do with the fact that the Sendmail binary on her system was owned by her.[1] Her boss, a senior engineer, had installed Linux for her when she started. He didn't ask the systems group to do the work, because we would have set the root password to something he didn't know and have given the user sudo instead. (More about sudo shortly.)

I walked over to his cube and asked him if he knew what was up with Sendmail being owned by the user. "Sure," he said, "I did a `chown -R <user> /usr` so she wouldn't have permission problems."

I'm slightly ashamed to say I laughed out loud before telling him, "Well, you are going to have to reinstall Linux."

He got annoyed and asked, "Why can't you just do `chown -R root /usr`?" I told him why[2] and handed him the Linux CDs. They were back on my desk within 20 minutes, so I knew he had decided to implement his "solution" rather than reinstalling the system.

This would solve her email problem, but other problems would surely be created. I told the user that I wouldn't touch her system until Linux was reinstalled. I knew that the senior engineer in this case was no dummy, despite the incredibly boneheaded mistake he had made. He was, in fact, a very bright guy, engaging and witty in conversation and trusted with a critical role in designing the software our small startup was betting everything on.[3] What could account for the extreme wrong-headedness he displayed? Why did he resist the reasonable restrictions we asked our users to accept in order to receive support on their personal workstations? How could I reconcile my certain knowledge that this was an extremely sharp and competent senior engineer with the apparently abysmal lack of wit his actions showed?

PORCMOLSULB

The above problem is an example of PORCMOLSULB: Proliferation of Randomly Configured, More-or-Less Screwed-Up Linux Boxes. It's been showing up more and more in environments I work in. This is partly due to the increasing popularity of Linux, but the main cause is software engineers' desire to control root on their personal workstations. This desire conflicts with the system administrator's imperative to maintain systems in a supportable condition and to prevent anonymous damage to other systems on the same network from inexperienced root-enabled users.

1. On most UNIX systems, the Sendmail binary has the "setuid" bit set. This means that when it is run, it takes on the system identity of the user who owns the binary. Usually this is the root user, or some other system account that has permission to write to the mail spool, where temporary mail files are kept. Since the user didn't have this permission, and since the Sendmail binary was owned by her, Sendmail couldn't write to the spool and mail was broken.

2. In any UNIX-like system, the files stored in /usr will be owned by a variety of users. There is usually a reason why a particular file would be owned by a particular user. The example of Sendmail being owned by root given above is one case in point. Once the ownership is changed through a global chown as in this case, it's very hard to set things back the way they were. It's easier just to reinstall the system.

3. I work for a small startup company in Silicon Valley. The company wants to keep their name out of this article. Since I am telling potentially embarrassing stories about our engineers, I wholly agree with this position. The arrangement also allows me to be more frank about certain things, as long as I remain circumspect about others, such as names.

The conflict is based not only on differing goals but on real differences in the competencies and enthusiasms of the two groups. PORCMOLSULB adds an interesting new twist to the struggle that tends to shift the balance of power toward the users in the ongoing battle. In this paper I will describe the battle in a little more detail, then ask and answer the question, "Can root access on engineering workstations be controlled in the face of PORCMOLSULB?"

The Conflict over Root Access

I've worked as a system administrator for 18 years, in academia, for government contractors, and in private industry. In each of those environments I have found a peculiar local version of low-intensity warfare between the computer users and sysadmins. I hasten to add that this conflict was rarely the only characteristic of relations between the two groups, or even the defining one. Nonetheless, the conflict was always present in some form. The most common form I have seen this conflict take is the struggle over root access. There are compelling business, psychological and technical reasons why this should be so.

The Role of Business Imperatives

From the perspective of the business employing them, system administrators and technical computer users such as software engineers come to work for the same reasons. That is, to make widgets, grommets, yo-yos or whatever else the enterprise is producing. However, looking a little deeper reveals differences in the business roles played by the two types of employees. Generally speaking, businesses hire systems staff to ensure that their computing environments are maintained in a state fit for maximizing the productivity of the enterprise.

Software engineers generally are employed to design and write products for sale. It is their productivity that the systems staff must maximize.[4] This difference in business imperatives colors a lot of the interaction between the two groups. Specifically, it shows up when a software engineer demands root access to get her job done. Granting the access may in fact help the engineer to be more productive, at least until she shoots herself in the foot with her rootly power. The sysadmin is bound to see a threat to the stability and security of the systems under his care, and to discount the possibility that any benefit might accrue from granting the access that couldn't also be accomplished with a less sweeping grant of privilege.

Before I examine that in more depth, I'll tackle the most difficult-to-characterize cause of conflict between system administrators and their technical users: the personalities of the people themselves.

The Role of Personalities

I've always thought that the conflict over root access was particularly strange in the context of UNIX software startups in Silicon Valley. It seems to me that UNIX system administrators and UNIX software engineers have a lot in common. However, the difference in business roles described above, plus differing enthusiasms and capacities, tends to lead bright people with an interest in computer technology down different paths.

APOLOGIA

Since I'm a system administrator, I can't avoid telling this part of the story from that perspective. I've tried hard to understand my users, and I've gotten pretty good at it

4. This is a sweeping generalization. There are plenty of systems engineers directly contributing to product, just as there are many software engineers working on the productivity of others. However, I'll stand by the generalization for the purposes of this discussion, since my experience tells me it's true more often than not.

UNIX operating systems generally provide a rather primitive model for distributing privilege to system users.

over the years, but the coloration my own place in the scheme of things will lend to my discussion of the personalities involved in this conflict is unavoidable. With that warning issued, I hope you will forgive the personal nature of the discussion that follows.

MY CHOICES

Why am I drawn to system administration? Why not be a software engineer, for instance? I do a fair amount of programming in my work. I can code some Perl for several hours, enjoying all the things you must do to program effectively, such as holding several dozen details in your mind at once. Best of all, I love integrating all those details into a finished solution that actually *does* something.

However, I don't like to wait too long to get to that point. I'm impatient. I also get burned out quickly doing that sort of thing. Finally, I get bored really really easily. Fortunately, as a sysadmin I am compelled to do a lot of other things. I have to deal with other human beings, frequently under difficult circumstances. I work with computer hardware a lot, racking up systems or diving under desks to replace bad components.

And best of all, I get to work with computer systems: UNIX, Linux, Windows, Palm, it doesn't matter. I love systems. I can make them stand on their heads or dance the two-step. I love the feeling of control and accomplishment that going to the exact center of a difficult problem in complex systems gives me.

MY USERS' CHOICES

How is all that different from what a typical software engineer does? This is a hard question for me to answer, because I have to try to put myself in the place of an engineer, and I tend to just assume that he thinks exactly the way I do.

However, there are some clues in the experiences I've had with such engineers that have helped me make the leap of imagination. First, I've noticed that these folks seem to have powers of concentration that are rather absurd, by my standards. Whereas I need to take a break after a couple of hours of coding, these folks stay glued to their screens and keyboards throughout their 14-hour days.

Second, I've noticed that their technical knowledge tends to be less broad than mine, but deeper. Both of these observations start to add up to a (perhaps) obvious conclusion: *software engineers are specialists*. Another conclusion I've drawn has taken a lot longer to arrive at, because it cuts so directly against my own stance toward technology. *Software engineers are generally not enthusiastic about computer systems*. Instead, they are enthusiastic about *software*! They view systems as a vehicle for software, a means to an end.

I view systems as ends in themselves. Once this idea struck me, I marveled at how long it took me to see it. It seems that both system administrators and software engineers are constitutionally suited for the differing roles they are asked to play in the enterprises that employ them.

Now that we've introduced the players, let's set the scene: UNIX in all its common permutations, including Linux.

The Role of Technology

UNIX operating systems generally provide a rather primitive model for distributing privilege to system users. The power to control all system processes and resources is

granted to the single all-powerful user: root. Other users may be granted varying levels of access, depending mainly on which UNIX group they belong to and on how group access permissions are set on various objects in the system. However, root (or any user with UID 0) is the only user who can arbitrarily change access permissions. As a result, when non-root users encounter a restriction in access permissions, they must call upon the power of the root user to rearrange permissions so that they may continue their work.

C2

There are exceptions to this monolithic permissions model among various proprietary and free UNIX implementations. Many OS vendors, including most UNIX vendors, have applied for and received DOD Orange Book C2 certification for one or more of their products. (For an exhaustive list, see *http://www.radium.ncsc.mil/tpep/library/fers/tcsec_fers.html*.) However, these vendors generally do not ship their systems with C2 security enabled.

Even Microsoft, whose Windows NT code base implements many of the facilities that C2 requires, such as Access Control Lists, doesn't do that. Since Microsoft, at least, has had to submit a version of NT with network access disabled in order to get certification, that's not entirely surprising. And though I'm not an expert on the topic, I suspect that the reason even those vendors who may be able to run C2 while on a network don't ship with it enabled is because C2 access controls are fairly burdensome to users and administrators alike.

Regardless of the real reason, the fact remains that the UNIX systems found in most commercial environments, from vendors like Sun, HP, IBM, as well as Linux and xBSDs, come configured by default with an antiquated permissions model.

Working Within the Model

Even given the monolithic UNIX permissions model, it is possible to give users most of what they want without unleashing the full power of root. Issues that concern shared access to files can be dealt with by judiciously adjusting group membership and permissions. If a user needs to open low-numbered TCP/IP ports, for example, it's possible to setuid root as just a particular binary, though that carries with it all sorts of other security implications.

There are many strategies that help users to "work within the system." But each of these has in common one fatal flaw: if the model needs to be adjusted because of an unforeseen condition, root (aka the sysadmin) needs to get involved to make the adjustment.

In a rapidly changing environment like a software startup, this has several impacts on the user. First, it slooows her down. An overworked and harassed sysadmin has to be located by an at least equally overworked and harassed engineer to make the change. According to Murphy, this will always happen at 3:00 a.m. before a critical demo. I really really hate to have my pager go off at that hour! Even if that apocalyptic scenario doesn't get played out, resentment may be fostered on both poles of the struggle.

The user may start to see the sysadmin as a power-mad tightwad, jealously guarding root access for his own nefarious purpose. The sysadmin may feel put out by the fact that the user isn't willing to learn enough about UNIX to get around her problem. He may also be blind to any benefit that might accrue to the user and the enterprise from allowing the access.

System administrators often complain (with justification) that what they do is never visible until something breaks.

Because working within the system is so troublesome, the ever-inventive UNIX community has produced many tools that try to add finer-grained control to the monolithic UNIX permissions model.

5. A comprehensive list of such tools is maintained at *http://www.courtesan.com/sudo/other.html*.

There's a bit of irony here. System administrators often complain (with justification) that what they do is never visible until something breaks. This is a consequence of the natural outcome of great sysadmin: quietly working systems. In a similar way, the sysadmin is unlikely to see any benefit from giving a user root, because those benefits short-circuit trouble calls to the sysadmin! Before moving on, I'd like to note that neither of the characterizations presented above is fair, and they rarely play out in such an extreme form in the real world. But their flavor is correct, at least in the places I've been.

Tools That Try to Help

Because working within the system is so troublesome, the ever-inventive UNIX community has produced many tools[5] that try to add finer-grained control to the monolithic UNIX permissions model. One of the most popular is sudo, familiar to many sysadmins. It allows users to invoke specific commands with root privilege. It uses the user's own password to authenticate access, thus protecting the root password. It also logs each command invocation with the name of the user, thus providing an audit trail of root access.

Typically, tools like sudo are deployed to meet a specific user need for root access, such as to mount a CD-ROM drive. Used in this way, the tools add a little to the risk of root compromise, but it's usually manageable. The main issue is unintended privilege that the sudo-enabled command might offer to the user. For example, the `mount` command that can make a CD-ROM available could also allow an arbitrary file system to be mounted. That file system (or even the CD) could contain a setuid root shell binary.

One way around this would be to wrap a script around the `mount` command that disallowed setuid mapping. But then you have to worry about the security of shell scripts running as root. In fact, in a relatively open environment like a software startup, there is no sure way to protect yourself from malicious misuse of privilege in all cases. You end up having to fall back on trust, treating abuse of that trust as a personnel problem.

The problem gets even worse as more and more commands are added to the suite of those offered to sudoers. Each new command brings its own particular set of security holes. The problem, once again, is that UNIX assumes a monolithic permissions model that tools like sudo can only work around, not cure.

This shows up again as weaknesses in programs like sudo that have nothing to do with the quality of the code, and everything to do with the fact that hacks like sudo are necessary in the first place. For example, sudo has difficulty with I/O redirection:

```
hbo@egbok > ls -l /tmp/foo
-r--r--r-- 1 root    other          1464 Mar 25 13:10 /tmp/foo
hbo@egbok > sudo ls >>/tmp/foo
bash: /tmp/foo: Permission denied
hboegbok > sudo ls | sudo cat >>/tmp/foo
bash: /tmp/foo: Permission denied
```

This problem occurs because I/O redirection is implemented by the shell before the command (sudo) is executed. The monolithic UNIX permissions model leads the shell to assume that the identity that does the I/O redirection is the same as the one that will result from the execution of the command. This is false in the case of sudo, which violates that permissions model. The following trick gets around the problem:

```
hbo@egbok > sudo ls | sudo tee -a /tmp/foo >/dev/null
```

But it's not very intuitive. This also works:

```
hbo@egbok > sudo sh -c "ls >>/tmp/foo"
```

But as previously noted, if you allow shell access with sudo, you might as well give out the root password.

Globbing is broken too:

```
hbo@egbok > mkdir fff
hbo@egbok > chmod 700 fff
hbo@egbok > touch fff/foo
hbo@egbok > sudo chown root fff
Password:
hbo@egbok > cd fff
bash: cd: fff: Permission denied
hbo@egbok > sudo cd fff
sudo: cd: command not found # cd is a bash builtin!
hbo@egbok > sudo rm fff/*
rm: cannot remove  fff/*': No such file or directory
```

The "globbing" expansion requested by the use of the asterisk fails because, once again, the shell tries to do it before executing the sudo command. We also see in this example the problem of trying to "cd" into a protected directory. Since "cd" is a bash builtin, sudo doesn't know what to do with it and you are out of luck

Of course, you could put code to solve either problem in a script and pipe to that. But if you let your users run Perl with sudo, what's to stop them from writing something like this?

```
#!/usr/bin/perl
exec "/bin/bash";
```

Once again, there goes your audit trail! In fact, if your users have successfully agitated for sudo access to more than a handful of commands you will almost certainly face an impossible number of holes in your security policy.

PORCMOLSULB, Again

So far, we've seen two groups of professionals, apparently similar on the surface, engaged in a struggle for control of root access on personal workstations. Each group is trying to carry out the goals that their respective business imperatives demand. The software engineer wants root so that she can get around restrictions in the UNIX system in order to get her work done. The system administrator is trying to ensure that the user's system stays functioning. What are some possible outcomes of this struggle?

Complete victory by either side is unlikely. To borrow a concept from chemistry, a more plausible outcome is that some sort of "dynamic equilibrium" will be reached as managers in support and engineering struggle to balance competing business interests. When the struggle concerns root access on servers, the business imperatives lean more toward the sysadmin's view of things, because the technical problems of sharing root on a server are less tractable. On engineers' personal workstations, however, the business case for allowing unfettered root is more compelling, because the workstation is a primary tool enabling the engineer's productivity.

If the balance of power shifts toward the sysadmin, we start to see the phenomenon of PORCMOLSULB showing up. This occurs when support departments can't keep up

with the demands of their user base for development "playpens," or when they put restrictions on those playpens beyond what the users are willing to accept.

It turns out that engineers are increasingly able to convince their managers that a completely uncontrolled Linux box would be a boost to their efforts in the rush to meet insane deadline pressure.[6] The sysadmin crew is probably feeling the pressure too, so they are in worse shape than normal to resist this trend. Indeed, they may not even become aware the box exists until it shows up in the critical path for some important milestone. But even if they know the box is being deployed, and lack the power to prevent it, they can still be stuck with fixing the box under killer time pressure, with the business on the line and with no advance idea of how the box was configured by its amateur sysadmin.

6. Managers generally feel bad about asking their people to work 12+-hour days to meet unreasonable deadlines, no matter how brave a face they put on the matter. Giving their engineers the tools they need is therefore not only good sense from an organizational standpoint, it lets the managers hand out a perk or two.

What Is to Be Done?

That nightmare scenario didn't actually happen to me in the case I opened this paper with. But we were facing a killer deadline, and the mere possibility of it happening made me nervous. I had faced similar situations before, so I knew that arguing for the "right" way of doing things wouldn't lead me anywhere useful. I'd also recently had my epiphany regarding the surface similarity and deep difference between sysadmins and software engineers. Here's how this particular comedy did play out.

DO YOU SUDO?

About 10 days after the senior developer got his engineer working again by doing a `chown -R root /usr`, she showed up in my cube asking for the Linux disks. I was mildly surprised that it had taken that long for a side effect of that solution to convince her that she needed to reinstall. But I tried not to act smug, and handed her the disks without asking why she wanted a reload of Linux. But I did ask her if she wouldn't rather that I do the install. I'd set her up so that her home directory on her workstation would automount underneath her when she went out to the network. I'd also arrange for it to be backed up regularly, and I'd support it so that she could come to me if she had problems. She allowed as how that might be a good thing. So I delivered the punch line: "All you have to do is give up root and use sudo. It takes a little getting used to, but I'll help out."

Well, she readily agreed to that too, and I was in a self-congratulatory mood when she came back in 10 minutes saying her boss had nixed the idea. He said she had to have root instead of sudo. I actually took a short time out before going over to his cube. My question to him was rather sharp, but nothing like it could have been. "Do you really think backups, the automounter and support are worth having the root password?"

"Yes," he said.

"Why fer ——ssake??" I politely asked.

"Because you won't let us run shells with sudo!"

I proceeded to tell the story of 27 eight-and-a-half by ten colored glossy audit trails.

He said, "Stop right there! What good would an audit trail do you if someone did `chown -R <user> /usr`?"

Well, he had a point. But I had an answer: "Because the audit trail would tell me right away that I had to reload Linux, rather than some less drastic solution. And besides most problems aren't caused by thoroughly boneheaded moves like that one!"

He laughed and said, "OK. Give her sudo."

I felt pretty good after that. It could have turned out differently, but it didn't. Despite the sharpness of the exchange, I felt like I'd made a critical connection with this guy.[7] In addition, I had a toehold in his group with a supported Linux box that would not be randomly configured, and would be less, not more, screwed up. And his new engineer would be using sudo! I would work hard to make sure that she had as good an experience with it as I could manage. In fact, over the next couple of days they came to me several times with things they couldn't do with sudo, and could I please just run the command with root? Each time it had nothing to do with sudo, and each time I cheerfully fixed it for them, or pointed them in the right direction. Soon, I had a couple of converts.

7. I probably neglected to mention that I'm new on the job.

BEYOND SUDO

Now it turns out that the senior developer, and all his colleagues, were resistant to using sudo because we restricted shells. This is an area where a sysadmin can argue unto blue-facedness about the lack of a need for a shell when you have more-or-less unrestricted sudo access. Indeed, since we had such unrestricted access, escaping from sudo and its audit trail was a trivial exercises. Given those facts, I decided to just accept that despite the technical arguments, sudo alone was not a workable solution for these senior engineers on their workstations. In half a day, I whipped up a pair of Perl scripts that used `script(1)` and a FIFO to provide an audited root shell using sudo.[8] This gave them practically nothing they didn't have already with our open sudo policy, and preserved our audit trail. All the senior engineers accepted a support regime that included these scripts.

8. The result of considerably more than half a day's effort is available at *http://www.egbok.com/sudoscript*.

Caveats and Conclusions

DANGER [WIJ]ILL SYSADMIN!

There are big problems with this solution. Having root on a workstation that mounts NFS shares is tantamount to giving the user root on the NFS server! Most NFS servers can and should be configured so that any access to an exported file system by UID `0` is mapped to a user with no privilege whatsoever. But that's not the whole answer. With root, a user can assume any UID in the `passwd` map. This means that on the NFS server, other users' files and system files not owned by root are at the mercy of root on the NFS client! The approach I've described works best when the workstations are NFS servers, not clients. There is still an issue with other systems mounting shares from the workstation. If the NFS client implementation doesn't enable you to disallow setuid binaries, a root user on the server could place, for example, a setuid root bash binary on the exported file system, then execute that binary on the client and get root privs.

PHILOSOPHY 101

This is not an exhaustive list of the security problems such a setup could raise. However, in my small shop, I can look each of my users in the eye every day if I choose. There is not a single unteachable idiot in the bunch. I also don't hand out my scripts to everyone. In short, I rely on the good faith of my users. I give them the tools they say they need, and I try to give them the benefit of the doubt on the question, despite my technical knowledge to the contrary. If they shoot themselves in an extremity with their privilege, I triage and fix the damage, with the benefit of a recent audit trail.

Documentation. Nobody likes to write it, and nobody likes to read it.

<tirade mode="self righteous" color="purple">
I trust my users in this regard because of one argument in favor of Democracy: if you give people more choices, some will make bad ones; many more will make good ones, yielding a net benefit.
</tirade>

This principle may well be applicable beyond my environment. How it plays out in yours is up to you and your users.

Finally, Documentation

Once I've fixed any problems caused by inexperienced root users blasting off their toes, I try to leverage the occasion to get them to read my documentation. Ah, yes. Documentation. Nobody likes to write it, and nobody likes to read it. I write lots of documentation, and, perversely perhaps, I enjoy doing it. What I find hard to take is the indifference most of my users show toward what I write.

My epiphany regarding the differences between sysadmins and software engineers has provided me with an explanation for that conundrum as well. What's relevant to me and to the systems under my care is not directly relevant to my users' concerns! If I were to write the best-ever UML manual, then they might notice. But when a pretty bright engineer has made some embarrassing error that has clearly resulted in a hit on his productivity, or worse, that of his colleagues, then the docs I write may seem more relevant.

You have to be tactful and swift in exploiting these opportunities for education, however. Tactful because these folks are proud, and their pride has just been wounded. Swift, because they'll have their heads completely stuffed full of Java before long, with no room for anything else.

an introduction to dependability

Definitions and Examples

by David A. Patterson

David Patterson is Professor of Computer Science at the University of California, Berkeley. He implemented one of the first RISC microprocessors and invented, along with Randy Katz, the Redundant Arrays of Inexpensive Disks (RAID),

patterson@cs.berkeley.edu

To improve dependability of systems, the Recovery-Oriented Computing (ROC) project is creating technology that will let systems recover more quickly from failures [Patterson et al. 2002]. We are especially interested in services accessed over a network, such as Internet sites and enterprise data centers. Since system administrators are the ones called when systems fail, we want to start a conversation about dependability problems in the hopes of developing technology that will really help.

One persistent difficulty with the general topic of making computers systems that can survive component faults has been confusion over terms. Consequently, perfectly good words like reliability and availability have been abused over the years so that their precise meaning is unclear.

Clearly, we need precise definitions to discuss such events intelligently. As a first step in a conversation about dependability, we define the dependability terminology: fault, failure, reliability, availability, mean time to failure (MTTF), and mean time to repair (MTTR). We also show how to calculate MTTF of a system given the MTTF of its components.

This paper is derived from Chapter 7 of Hennessy and Patterson [2002]. It provides a simplified version of definitions used by the IEEE Computer Society Technical Committee on Fault Tolerance and the IFIP working group 10.4. This paper is the first in a series; future papers will talk about issues relevant to the system administration community using these definitions.

Defining Dependability, Reliability, and Availability

The research community picked a new term – dependability – to have a clean slate to work with: computer system dependability is the quality of delivered service such that reliance can justifiably be placed on this service. Each component of that system also has an ideal specified behavior, where a service specification is an agreed description of the expected behavior. A system failure occurs when the actual behavior deviates from the specified behavior. The failure occurs because of a fault, a defect in that component.

We can now explain reliability and availability. Users may see a system alternating between two states of delivered service with respect to the service specification:

1. Service accomplishment, where the service is delivered as specified
2. Service interruption, where the delivered service is different from the specified service

Transitions between these two states are caused by failures (from state 1 to state 2) or restorations (2 to 1). Quantifying these transitions lead to the two main measures of dependability.

Reliability is a measure of the continuous service accomplishment (or, equivalently, of the time to failure) from a reference initial instant. Hence, the mean time to failure of disks is a reliability measure. The reciprocal of MTTF is a rate of failures. Service interruption is measured as mean time to repair. The related term mean time between failures (MTBF) is simply the sum of MTTF and MTTR. Although MTBF is widely used, MTTF is often the more appropriate term, as repair times may be harder to predict.

Availability is a measure of the service accomplishment with respect to the alternation between the two states of accomplishment and interruption. Module availability is statistically quantified as:

Availability = MTTF/(MTTR+MTTF)

Note that availability and reliability are now quantifiable metrics, rather than synonyms for dependability. Availability ranges from 0% to 100%, with 100% being perfect; reliability as measured by MTTF ranges from 0 to infinity, with infinity being perfect.

Calculating MTTF and Availability

If we assume that the age of the component is not important in its probability of failure and that failures are independent, the overall failure rate of a subsystem is just the sum of the failure rates of the modules. Let's do an example. Assume a disk system with the following components and rated MTTF:

- 1 SCSI controller, 500,000 -hour MTTF
- 1 power supply, 200,000-hour MTTF
- 1 fan, 200,000-hour MTTF
- 1 SCSI cable, 1,000,000-hour MTTF
- 5 SCSI disks, each rated at 1,000,000 -hour MTTF;

We can compute the MTTF of the system as a whole by adding the failure rate of each component, which is the inverse of its MTTF.

$$\begin{aligned}\text{Failure rate}_{system} &= 1/500{,}000 + 1/200{,}000 + 1/200{,}000 + 1/1{,}000{,}000 + (5 * 1/1{,}000{,}000)\\ &= (2 + 5 + 5 + 1 + 5)/(1{,}000{,}000 \text{ hours})\\ &= 18 / (1{,}000{,}000 \text{ hours})\end{aligned}$$

The MTTF of the system is just the inverse of the failure rate of the system:

$$\text{MTTF}_{system} = (1{,}000{,}000 \text{ hours}) / 18 = 55{,}555 \text{ hours}$$

If the average MTTR is one day for this system, the estimated availability would be

$$\text{Availability}_{system} = 55{,}555 / (55{,}555 + 24)$$

which is about 99.96%. Marketing departments have shorted availability from the actual percentages to the number of leading 9s in the percentage. Thus, 99.96% can be called "3 nines" of availability. Well-run servers achieve 2 or 3 nines of availability, or 99% to 99.9%.

Failures vs. Faults

The difference between faults and failures aren't as obvious as you might think.[1] Here are some examples of the difficulties.

- Is a programming mistake a fault or a failure? Does it matter whether we are talking about when the program was designed, or when it is run? If the running program does not exercise the mistake, is it still a fault or failure?
- Suppose bits on disk in a RAID system change due to a problematic sector in a disk. Did a fault or failure still occur if the error correction codes (ECC) of the sector delivers the corrected value to the processor? Is it a fault or failure if it was an uncorrectable fault according to the disk, but the RAID system corrects it?
- The same difficulties concerning data change, latency, and observability arise with a mistake by a human operator.

NOTE

1. The dependability community makes the subtle distinction between a defect that does not change anything and a defect that does change the state [Gray and Siewiorek 1991, Laprie 1985]. They call the former a fault and the latter an error. An example is an Alpha particle hitting a DRAM cell. That collision is a fault, and it is only an error if it changes the value in the DRAM cell. Although this distinction is more precise, it is often confusing, resulting in debates on whether something is a fault or error. In this paper we concentrate on the differences between defects and service outages, which we call faults and failures.

Initially, a fault is considered latent and becomes effective when it is activated. For example, a programming mistake is a latent fault until that code is invoked by the system. If the fault actually affects the delivered service, a failure occurs. The time between the occurrence of a fault and the resulting failure is the latency. Thus, a failure is the manifestation on the service of a fault. Reviewing the properties of fault:

- A latent fault becomes effective once activated.
- An effective fault often propagates from one component to another, thereby creating new faults.

Thus, an effective fault is either a formerly latent fault in that component or it has propagated from another fault.

Reviewing the fault-failure sequence, the steps are latent faults, then effective faults, and finally, if it disrupts the delivered service, a failure.

Let's go back to our motivating examples above. A programming mistake is a fault; upon activation, the fault becomes effective; when this effective fault produces erroneous data which affect the delivered service, a failure occurs. For the disk example, the flaw in the sector is a fault. If the ECC corrects the fault, the RAID system would not observe it. If the disk could not correct it, and thus has a failure, then RAID system would see a fault. If the RAID system corrected it, the operating system would not see a fault. A mistake by a human operator is a fault; it is latent until activated; and so on as before.

These properties are recursive and apply to any component in the system. That is, a defect is either a fault or a failure depending on your perspective. For example, the specified behavior of a disk is to deliver correct sectors when requested. Thus an uncorrectable read fault is a failure from the disk perspective, but it is a fault from the perspective of the RAID system. Confusion between faults and failures often depends on how you draw the boundaries around the system and hence what is the expected service of that system.

Categorizing Faults and Ways to Handle Them

The purpose of this section is to familiarize you with some terms that you may see when looking at systems that claim greater dependability. There are many ways to categorize faults. We show two ways – by duration or by cause – to give you some intuition about how to talk about faults. Classifying by their duration yields three options:

1. Transient faults exist for a limited time and do not recur.
2. Intermittent faults cause a system to oscillate between faulty and fault-free operation.
3. Permanent faults do not correct themselves with the passing of time; they remain until repaired.

The classification above shows a hierarchical taxonomy of faults based on cause. The first split is whether it is physical or logical, where all software and operator faults are logical. Hardware faults are either due to problems in manufacturing, in operation, or in design. Manufacturing faults are either individual flaws or due to problems in the manufacturing process. Physical operation faults are either the result of wear or of environmental problems, such as power outages, high temperature, fire, flood, earthquake, and so on. Design faults may simply be bugs in hardware or software, or not designing-in sufficient margins in hardware to handle normal variations in, say, volt-

A programming mistake is a latent fault until that code is invoked by the system.

As many as half of disk failures are due to problems in shipping.

age. Finally, logical operation faults can be people breaking into the system or mistakes by operators, although poor design of the user interface and documentation leads to operator mistakes.

Just as there are many ways to categorize faults, there are many ways to categorize systems' handling of them. Laprie [1985] divides improvements into four methods:

1. Fault avoidance: how to prevent, by construction, fault occurrence – that is, preventing the creation of latent faults.
2. Fault tolerance: how to provide, by redundancy, service complying with the service specification in spite of faults having occurred or occurring – that is, preventing faults from becoming failures.
3. Fault removal: how to minimize, by verification, the presence of latent faults.
4. Fault forecasting: how to estimate, by evaluation, the presence, creation, and consequences of faults, and thus take preemptive action to prevent the fault from turning into a failure without necessarily using redundancy.

A final topic is repair. Some systems or modules are repair tolerant, in that you can safely repair them while the system continues to operate. For example, many systems allow disks to be hot swapped without shutting down the computer. Some systems and modules are repair intolerant. For example, you often must shut down the system before replacing the motherboard.

Drawbacks to MTTF and the Definition of Failure

One drawback of MTTF calculations is that they imply a comfort zone that is not merited in practice. First, although MTTF is just the inverse of the failure rate, it is not intuitive. For example, a million-hour MTTF means the mean time to failure is over 100 years. Does this mean the average disk lasts 100 years? No. Since manufacturers calculate disk lifetime as five years, it means that if you bought many disks and copied the data to a new drive every five years, on average you could do it 20 times before you saw a failure. Annual failure rates are a better match to human intuition. For example, if we make common assumptions about the distribution of independent failures, about 1% of components would fail in each of the first few years if each component was rated as a 100-year MTTF.

The second drawback is that MTTF assumes failures are independent and that they are based on MTTF numbers supplied by the manufacturer. The manufacturer supplies MTTF rates assuming the products were not damaged in shipping and that they were operating in nominal conditions of temperature and voltage. As many as half of disk failures are due to problems in shipping. High operating temperature due to fan failure, something blocking the air flow, or air-conditioning failure can severely shorten lifetimes. Such environmental problems can also violate the independent-failure assumption.

The purpose of MTTF calculations is to show relative reliability of different designs rather than to predict what you will see in practice. For example, the calculation above shows that MTTF is limited by the weakest link in the chain. To significantly improve the reliability of this subsystem, we would need a more reliable power supply and fan. If those two components were unchanged, even if we had perfect controllers, cables and disks, the MTTF would be capped at a tenth of the reliability of one disk.

As it is either expensive or impossible to replace components with more reliable versions, the primary way of coping with failures is redundancy. For example, one of the

long-standing guidelines in design is to have no single point of failure. We will talk about calculating the reliability of redundant systems in a future paper.

A final note about the definition of failure itself. The terminology takes the simplified view that the system is either accomplishing service or there is a service interruption. A more nuanced view sees a third state, service degradation, whereby the service is not interrupted but is performing poorly enough to be a problem. It also assumes a single service, although Internet sites like eBay and Yahoo offer a collection of services. We will tackle a more nuanced definition of failure in future papers.

Conclusion

The goal of this paper is to begin to pierce through the fog of dependability terminology. We distinguished a fault from a failure, showing the difference can simply be a matter of perspective. We also gave quantitative definitions of reliability and availability, and provided an example of how to estimate MTTF and availability. Finally, we warned to not be too comfortable with high MTTF, since the numbers can be misleading.

Future papers will talk about redesigning systems so that there are no single points of failure, statistics collected on why systems fail, the cost of downtime, and the goals of the Recovery-Oriented Computing project.

We hope to initiate a series of conversations about why current systems fail and how researchers can help create a new foundation for systems that are easier to operate.

Acknowledgements

I would like to thank Aaron Brown, George Candea, David Oppenheimer, and Mike Patterson for comments on earlier drafts of this paper. This work is supported by the National Science Foundation, grant no. CCR-0085899, the California State MICRO Program, Allocity, Hewlett Packard, IBM, and Microsoft.

REFERENCES

Gray, J., and D. P. Siewiorek. 1991. High-availability computer systems. *Computer* 24:9 (Sept.): 39–48.

Gray, J., and A. Reuter. 1993. Fault tolerance. Chapter 3 of *Transaction processing: concepts and techniques.* San Francisco: Morgan Kaufmann Publishers.

Hennessy, J. L., and D. A. Patterson. 2002. *Computer architecture: A quantitative approach*. 3d ed. San Francisco: Morgan Kaufmann Publishers.

Laprie, J.-C. 1985. Dependable computing and fault tolerance: Concepts and terminology. Fifteenth Annual International Symposium on Fault-Tolerant Computing FTCS 15. *Digest of Papers*. Ann Arbor, MI, USA (June 19–21), 2–11.

Patterson, D., A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, P. Enriquez, A. Fox, E. Kiciman, M. Merzbacher, D. Oppenheimer, N. Sastry, W. Tetzlaff, J. Traupman, N. Treuhaft. 2002. Recovery-oriented computing (ROC): Motivation, definition, techniques, and case studies. *U.C. Berkeley Computer Science Technical Report* UCB//CSD-02-1175, March 15, 2002 (see *http://roc.cs.berkeley.edu*).

monitoring strategies for high-availability web apps

by Neil Ashizawa

Neil Ashizawa is an SiteScope Product Manager responsible for System Monitoring solutions. Neil has been with Mercury Interactive since 1996. During his six years, Neil has held positions such as a Technical Consultant,a Technical Courseware Developer, and as a Manager of Program Development.

neila@merc-int.com

With the rapid proliferation and popularity of Web interfaces, companies everywhere have begun exposing internal data and methods to the network as never before. Sometimes, this is simply done as an in-house intranet, with a Web interface thrown on top simply to make the users happy. In other cases, this is a full-blown customer-interaction system. While this is all well and good, mission-critical network-enabled applications are difficult to test and maintain, especially as the network scales.

In a network with thousands of users, key network applications – even internal ones – can reside across several servers, in multiple databases, or even in many geographical locations around the world. A Web administration team has three primary goals:

1. Maintain reliable and secure access for authorized users (and deny access to others)
2. Maintain users' ability to perform business-related activities (for instance, in an e-commerce site, it's essential to keep the purchasing functionality working)
3. Provide users with a responsive, efficient, and effective interface

How to Monitor System Availability

Unless your site is static, not a source of revenue, or does not offer any value to your customers, you need to monitor it. In the Web world, "monitor" is a fairly nebulous term, basically including all the methods an administrator has at his disposal to verify that the site is functioning as expected.

Monitoring system availability might be one of multiple monitoring objectives for which an operations team might be responsible. Other objectives might include system-level monitoring for capacity planning and trending or application-level monitoring for diagnostics and optimization. This paper primarily focuses on availability monitoring through the use of network service monitors. These range from simple tools – such as ping, traceroute, and others that are included by default with most UNIX-based systems – to complex enterprise solutions that can perform in-depth, cross-platform monitoring.

However, just knowing whether or not all the servers involved are running is often not nearly good enough. In many cases, potential problems need to be highlighted as quickly as possible – preferably *before* the failure affects the users. For this, you need to track both key statistics on individual servers and the health of your entire network.

As an example, consider the network of XYZ Inc. The two sections of this network are:

- The external network that provides e-commerce, Customer Relationship Management (CRM) support, and knowledge-based resources for customers
- The internal network that provides interfaces to these same systems for employees

Both systems provide mission-critical services and are thus built with high-availability, fail-over-capable, hot-swappable equipment. A highly trained IT staff is on-site or on-call 24 hours a day. Everything is working perfectly… right?

Picture this: an executive, working late due to an upcoming conference call to a potential customer on the other side of the globe, discovers that the internal CRM systems are responding very sluggishly. She considers contacting IT, but the only technical person on site is on a different floor, and she doesn't have time to locate him. She figures

that the system is only slow, not completely broken, and that IT probably already knows about this problem.

In truth, the primary database server for the CRM application has crashed. A backup server is now carrying the entire load, and its performance is not quite keeping up. The backup server itself has a problem: one of its cooling fans has dust in the housing and is no longer moving much air. The backup's internal monitoring software notified the IT manager of the problem right as he was going home at 5:45 p.m. He figured that the backup could run on its other fans for the rest of the night – he would have someone fix the problem in the morning. Besides, this is only the backup we're talking about.

When the executive talks with the potential customer, she finds that they're unable to access the network at all. She completes their order from her end, commenting on the problems in the system. She promises to have someone look into it. After several minutes, she finds the IT guy on a coffee break and tells him about the problem. However, without knowing which server or servers are having problems, he has to check each one individually.

When another important overseas customer attempts to connect to the external network at 3:15 a.m., they cannot retrieve any data, either. The backup database server's second fan just failed, and the system overheated and shut down as a safety measure designed-in from the start. The customer sends an email to the administrator's personal address. Unfortunately, the administrator has been out sick for several days, and his email is piling up unread.

The IT person finds the crashed primary server and restarts it. Thinking the problem is solved, he doesn't check the backup server. With the primary running, the network is once again functioning – albeit a bit slowly. He sends an email to the administrator.

Do you see the problem? Several hours of costly downtime could have been avoided if the staff had been fully aware of the status of the key servers and if the IT manager had been alerted as soon as the problem escalated. When the primary server crashed, monitoring the stats of the backup became even more crucial, since it became a single point of failure.

WHAT IT NEEDS TO KNOW ABOUT SYSTEM AVAILABILITY

To determine the comprehensive status of a large network application, you basically need to watch three key metrics. First, is the server running and accessible? System crashes, hardware failures, and extended power interruptions do happen – as an administrator, you need to know about these as quickly as possible. Even if your mission-critical servers have fail-over redundancies in place, you still need to know immediately when a server goes down. Accessibility is a closely related factor: it doesn't matter that your server is running smoothly if it's unreachable from the rest of the network.

Second, you need to know if the servers are responding correctly. Even if the server is up, its data may be corrupted; in the case of a Web server, files could be out of sync or links broken, or its pages could have been defaced by hackers. You not only want to know that your server is serving *something*, you want to make sure it's serving the *right thing*. This same point goes for database, application, and other back-end servers – just because the database server responds to SQL requests doesn't mean that it's responding with the correct data.

You not only want to know that your server is serving *something*, you want to make sure it's serving the *right thing*.

Ping monitoring is the most basic way to keep track of your servers' status.

Third, you need to watch the server's response times. If your server responds correctly but takes several seconds longer than normal, it may mean that there are more serious problems waiting in the wings. You might need another server, or you might need to adjust what's running on the server. For diagnostic and trending purposes, it is best to watch metrics such as CPU utilization and memory and hard disk usage directly, so you can either isolate the root cause of performance degradation or even anticipate potential failures and correct the problem before it occurs.

Basic Monitoring

Ping monitoring is the most basic way to keep track of your servers' status. Sending an ICMP ECHO_REQUEST packet causes the server to reply with an ECHO_RESPONSE, indicating that it is functioning.

Actually, this simply indicates that its hardware is active and that the network layer of the TCP/IP stack is correctly processing ICMP packets. It does not mean that the server is fully functional or that it will produce the correct response to other types of requests.

Even so, ping monitoring is an important part of an overall monitoring strategy. Besides indicating that the server is capable of a response, pinging a computer can also reveal information about network traffic patterns. If it takes the server longer to respond than usual, it may mean that it is deluged with packets or that heavy traffic across the intervening network link is interfering with the transmission of ICMP packets.

On a UNIX system, it is a trivial matter to set up a cron job to ping key servers every hour or so and then email the administrator if the ping fails or if the response takes longer than expected. One way of doing this is shown in code sample 1 on page 72. However, coordinating dozens of ping scripts into usable data can be a very complex project.

Keep in mind, too, that this script only confirms that the server is accessible from whatever machine you use for monitoring. Unless this unit is outside your firewall – and preferably on the other side of the Internet – you might not be aware of problems caused by external network issues. Admittedly, such issues are often beyond your control, but having remote testing centers, perhaps outsourcing them, can be an important part of your monitoring strategy. Of course, email notification might be problematic from an external monitor if your network is down. Some sites go so far as cellular telephone modem communications for extremely critical notifications.

URL AND CONTENT MONITORING

On the other hand, just using ping is clearly not enough for the vast majority of network situations, particularly the large, mission-critical applications discussed at the outset. The second key metric then comes into play: we want to make sure that the server is responding correctly.

In the case of a Web server, the simplest way to do this is to establish an HTTP session, request key pages, and compare the results to an expected norm. For instance, you could use wget and grep in a shell script to confirm that a supplied regular expression occurs in the server's output as expected. As an example of this, see code sample 2 on page 73.

The example script is somewhat limited. For instance, you might need to track how the server responds to different types of browsers. The script in code sample 2 could be modified to change the USER_AGENT string sent by the client, so that you can retrieve pages separately for the major browsers.

An important point to remember, though, is that it is fairly trivial to submit a request for a certain document from a Web server, but in most cases, watching static pages on a server is not nearly as important as monitoring dynamically created content. Often we need to be able to see how the server responds to a submitted form, and this requires supplying CGI POST data and then observing how the server responds to given data. For an effective test, it might be necessary to try a handful of different data types or sizes.

A shell script implementation of this level of functionality is possible, but this type of program should ideally be written in a friendlier language. Additionally, the complexity of that kind of solution is far beyond the scope of this paper.

ADVANCED MONITORING

So far, we have discussed monitoring techniques that basically examine how the server is responding from the user's point of view. To gain insight into why the server responds the way it does – arguably the information most needed to solve a problem – we need to probe deeper, digging out information from the nether layers of your application.

A simple way to keep an eye on your server's performance is to measure response time and notify the administrator if response latency spikes. If the server suddenly takes substantially longer to perform a given task, your users would probably be able to see this same delay.

Another important way to watch the innards of your application is by monitoring key values in your database. By checking certain totals, important metrics, or even test values periodically, you can make sure that your application is interacting with data as intended. In fact, a comprehensive monitor could retrieve information from the database and then perform some quick calculations to ensure that the values are correct.

Implementing this kind of solution as a shell script is possible, but it would be wiser to work in a language more suited for database interaction and data handling – for quick-and-dirty scripting of this kind of system, Perl is often a good choice. The key would be to stage regular SQL queries and then compare the returned data to expected values or expressions.

For more advanced monitoring, it is important to keep track of resource usage inside the server. On most UNIX systems, you can use rsh or SSH to run standard system commands, such as df, free, and ps, or to retrieve kernel statistics from the /proc pseudotree.

THE PROBLEM WITH A SHELL SCRIPT SOLUTION

Although a shell script like those in the code samples is a good quick-and-dirty solution to this kind of problem, it has a number of significant limitations.

Scalability: A multitude of shell scripts may get the job done, but it can be a pain to track down the various options in dozens of configuration text files. A company with a large, complex Web site and IT department, or with an organizational structure that

For more advanced monitoring, it is important to keep track of resource usage inside the server.

calls for multiple notifications when a system enters a failure state, would need a solution far more robust than an odd assortment of scripts could provide.

Maintenance: If the data to be monitored changes frequently, the complexity of a do-it-yourself solution can scale quite rapidly. A commonly used solution is to use "meta-scripts" to administrate scores of worker scripts. This "solution" is kludgy at best – like a house of cards, it's fragile and requires continuous, careful maintenance.

Learning Curve: Additionally, without a unified, easy-to-use interface, nontechnical users cannot use the system to get information about system failures or to report new problems unless they can be trained to retrieve needed information. Generally, do-it-yourself solutions place the system in the hands of one highly skilled administrator, so if he or she left, even other technical staff members could require a great deal of time to figure out how the system works.

Accuracy: Finally, a homemade solution may overlook important statistics or monitoring techniques. Even if a certain solution meets your company's needs today, how much work will it take to retrofit it to interoperate with tomorrow's hardware? For enterprises in fast-moving vertical markets, maintaining a shell script solution could be a full-time job. Management now has to pick between monitoring accuracy and cost – and accuracy often gets the short end of the stick.

COMMERCIAL SOLUTIONS

In response to these issues, a number of companies offer commercial monitoring products.

Empirix offers a broad range of testing and monitoring solutions, including OneSight, e-Monitor, and the e-TEST suite. Combining measurements of user experience and server/network activity, Empirix provides monitoring solutions to track Web application performance from inside (OneSight) and outside (FarSight) the firewall. Empirix's e-Monitor can also perform end-to-end Web transaction monitoring evaluating the user experience of your site.

Keynote's performance management solutions are designed to enable you to take control of your Internet performance by effectively turning your data center into a fast, efficient triage environment. Keynote can monitor the overall end-to-end Web-based application for availability and can send configurable emails or pager alarms when TCP-enabled Internet connections, servers, and CGIs become inaccessible or return incorrect data.

NetIQ AppManager provides a centralized console to proactively manage virtually every component of a highly distributed Windows NT and Windows 2000 environment, from the physical hardware to business-critical server applications, such as Microsoft Exchange, SQL Server, Citrix MetaFrame, Lotus Domino, Oracle, SAP R/3, and Microsoft Internet Information Server. The latest version also supports monitoring certain UNIX systems.

BMC is currently rolling out version 7 of its PATROL monitoring software. This version has been extended to allow for more secure monitoring configurations and now supports new platforms, including SuSE Linux Enterprise Server (zSeries), Microsoft .NET, and Microsoft Windows XP. The installation routine has also been upgraded to make it a more straightforward process to set up or upgrade PATROL.

Freshwater Software's robust monitoring offering, SiteScope, is the only one from this list that not only can monitor but also runs natively on the three major server platforms: Windows NT, Sun Solaris UNIX, and Linux. SiteScope includes specialized monitoring tools for over 60 network protocols, system-level metrics, and enterprise software packages. A built-in Web interface allows users and managers to check the network's status and IT administrators to manage the entire system remotely. Versatile yet easy to use options allow for numerous configurations and a variety of alerting methods, including email, pager, SNMP, or customizable scripts. The optional SiteSeer module complements this system with outside-the-firewall monitoring from afar.

CHOOSING A MONITORING STRATEGY

Which monitoring solution is right for you? Only you can determine that. The key is to evaluate your network needs. If your network consists of only a handful of servers running just one or two Web-enabled applications with fairly simple interfaces, a shell script solution may work well for you.

In a highly homogeneous network, you'll want to find the solution that best fits your chosen platform. If, though, you plan to expand into other platforms in the near-to-mid future, or if your network is currently a best-of-breed composition of numerous products, you'll want to make sure that the monitoring solution you choose is flexible enough to deliver useful information about every server system you use.

You may also want to choose a solution that provides a wide breadth of monitoring types so that you can not only monitor the availability of your network components but monitor system utilization for capacity planning or application-specific monitors for diagnosing bottlenecks. To take it a step further, you may want a complete solution for all of your monitoring requirements, correlating all of the metrics together to gain a holistic view of the entire infrastructure.

Returning to our earlier example, XYZ Inc. will probably want a solution that includes both internal and external monitoring points and specific monitors for each of their mission-critical applications. The system ought to have customizable logic that can proactively respond to minor failures as well as signal a critical alert when several otherwise minor failures occur at the same time. Additionally, XYZ probably needs a system that can get the attention of its technical staff via several methods – emailing the manager, giving console alerts to the on-site staff, and paging the administrator when he's gone home.

Conclusion

Monitoring is a very important part of a company's information services, a key to good customer relations in the Internet-enabled world. Think through your options thoroughly, but don't delay in getting a solid monitoring system in place. If any of your IT systems even approximate "mission-critical" status, the cost of allowing them to fail unnoticed for hours is far greater than the deployment cost of any monitoring solution.

Good monitoring enables you to achieve the primary goals of any Web-enabled application: ensuring that users can reliably access needed functionality, keeping the doors open and the cash flowing in at your e-commerce storefront, and providing users with an enjoyable experience overall, free from crashes, hang-ups, or irritating sluggishness. If these are your goals, if your site contains dynamic content, if it is a revenue-generating project, if it offers value to your customers, or if it is mission-critical in any sense

Don't delay in getting a solid monitoring system in place.

of the word, you need monitoring software. But beyond that, you need a monitoring strategy – a cohesive view of what your network is, what you want it to be, and how monitoring it will help you get there.

CODE SAMPLE 1: pingmonitor.sh

```
#!/bin/bash
#
# pingmonitor.sh — simple monitoring with ping
#

# verify that the config file exists and is readable
if [ -r /etc/pingmonitor.conf ]; then

for machine in `grep -v '^#' /etc/pingmonitor.conf`; do
      # adjust -c and -w parameters as necessary for your network
      if ping -c5 -w10 $machine
      then : # do nothing, the host is up
      else # ping failed

# modify the message below as necessary
/usr/lib/sendmail -t -F 'pingmonitor.sh' << ENDMAIL
From: pingmonitor <admin@domain.net>
To: administrator <admin@domain.net>
Subject: Ping failed: $machine

Ping failed for "$machine".
Please verify that it is functioning correctly.
ENDMAIL

      fi
done

else # no config file
echo Sorry, it appears that your /etc/pingmonitor.conf file is missing echo or\
unreadable. No operations were performed.

exit 1
fi
```

To enable this monitor, mark it executable and add it as a cron job. On a RedHat Linux system, for example:

```
chmod 755 pingmonitor.sh
cp pingmonitor.sh /etc/cron.hourly/
```

You will also need to create a file named /etc/pingmonitor.conf. Every line in this file must either be a comment (starting with the # character) or an IP number or qualified domain name for the script to ping.

If the script can't read from its config file, it will print an error message and return a nonzero exit code, causing cron to then email the owner of the cron job (usually root). This could easily be modified to send email to an outside address, if preferable.

CODE SAMPLE 2: contentmonitor.sh

```
#!/bin/bash
#
# contentmonitor.sh
#

# verify that the config file exists and is readable
if [ -r /etc/contentmonitor.conf ]; then

tempfile="/tmp/contentmonitor.out"
while [ -e $tempfile ]; do
    # make sure we have a unique tempfile name
    tempfile = "$tempfile.$$.$SECONDS"
done

for lineno in `gawk '/^[^#]/ { print FNR }' /etc/contentmonitor.conf`
do

urltest=`sed -n "$lineno p" /etc/contentmonitor.conf | cut -f1`
testpat=`sed -n "$lineno p" /etc/contentmonitor.conf | cut -f2`

    wget —output-document=$tempfile $urltest

    if grep $testpat $tempfile
    then : # do nothing, the pattern was found
    else # failed, content has changed

# modify the message below as necessary
/usr/lib/sendmail -t -F 'contentmonitor.sh' << ENDMAIL
From: contentmonitor <admin@domain.net>
To: administrator <admin@domain.net>
Subject: Content change: $urltest

Content monitor test could not find "$testpat"
in "$urltest".
Please verify that the server is functioning and that the document
is correct.
ENDMAIL

    fi

rm -f $tempfile
done

else # error: no config file
echo Sorry, it appears that your /etc/contentmonitor.conf file is
echo missing or unreadable. No operations were performed.

exit 1
fi
```

Again, you will have to chmod this file to be executable and add it as a cron job.

You will also need a configuration file in /etc/contentmonitor.conf. Comment lines (beginning with the # character) will be ignored. Every other line must begin with the URL to be monitored, followed by a tab, and then a word, phrase (in quotes), or UNIX regular expression for the monitor to verify it is present in the retrieved page.

ten reasons to monitor your systems

by John Sellens

John Sellens is the general manager for Certainty Solutions (formerly GNAC) in Canada, based in Toronto, and he is proud to be husband to one and father to three.

jsellens@certaintysolutions.com

Effective monitoring is an integral part of system and network administration. But not every site has a proper monitoring system (or systems) in place to watch over the site's systems, networks, and services.

To help motivate those of you who may not have all the system and network monitoring running that you should have, here are 10 (hopefully good) reasons to implement effective monitoring for your systems and networks:

1. You'll look like a hero when you prevent a problem before it starts.
2. Your boss (and your boss's boss) will like all the pretty graphs.
3. You'll have another good use for one of those cast-off PCs that are too slow to run a current version of Windows.
4. You'll have proper statistical backup to help justify your request for more bandwidth.
5. You'll be able to add paper to the color printer on the executive office subnet before your CEO starts complaining about his broken laptop.
6. Your disk and bandwidth utilization monitors will let you know when you've been cracked and your systems are being used to support a "warez" site.
7. You'll sleep better knowing that your machines are taking good care of each other.
8. You'll find out when your routers are flapping, so that you'll know when to yell at your upstream network provider.
9. You'll know if and when your site gets mentioned on Slashdot, because you'll be able to observe the "Slashdot effect" in real time.
10. You'll receive better annual performance reviews.

if systems were cars, would yours be double parked?

This article's title started out as "Optimizing Optimization," but that sounded much too formal for something conceived in a parking lot. Today's guideline for system administration takes on the notion that systems can be well maintained only by focusing on well-established order through standards and procedures.

by Steven M. Tylock

Steve Tylock has been managing infrastructures for the past 15 years in the Western New York area, and helped organize GVSAGE as a local SAGE for the Genesee Valley region.

Stylock@gvsage.com

First, to explain where this author sits on the spectrum between chaos and order, I'll admit, I've been a structure ogre on more than one occasion. One of my favorite mottos is, "If there's no request in the system, there's no work being done on it." I'd say that for the better part of my career, I have been in the majority group of sysadmins who believe the only way to manage systems is through order.

Time, Cost, and Quality

The basis for most project management philosophies dictates that you can (and must) prioritize any activity across three axes – time, cost, and quality. If time is king, costs and quality must necessarily be relegated to second and third class. Likewise, if cost or quality is most important, the others will suffer.

While accepting that specific cases exist where conditions require otherwise, for most sysadmins, cost is the single most important dimension controlling the operations of system support.

This focus on cost means system administrators operate with a perpetual shortage of resources. For some, that shows up as a lack of personnel; for others, a lack of funds for new equipment and software (or even maintenance of existing environments); and still others may find a lack of support from other functional areas of the company. As a result, system administrators have become resourceful, thrifty, and efficient – and focused on order as a cost-savings device.

You've Got to Have Someplace to Put Your Stuff

So imagine driving across a large, mostly empty, parking lot and thinking, "It's a good thing I can cut across these parking spots and roll through those stop signs." (I said it was conceived in a parking lot – I didn't say how.)

Everybody has to find a place to keep his or her car. (And whenever I say "car," I mean vehicle used to transport people and stuff from point A to point B, be it an automobile, SUV, truck, motorcycle, moped, motor home, bicycle, or what have you.)

They are very personal things – you probably have a few different types around your home. Businesses have to plan for them as well:

- A small-sized business may be located on a street without a parking lot – only space for a few cars along the road.
- A medium-sized business may have a modest-sized parking lot, but you might have to fight traffic to get in and out of the busy street.
- A large-sized business has ramps to and from the expressway, but all the company roads are one-way and you can't cut through the medians.

The point is not that any particular solution is flawless but that at each size an appropriate solution is sought. Cost is a significant factor – otherwise the smallest of businesses would have an acre of parking available.

Businesses have to worry about their changing needs as they grow. Relief can come through re-striping (how skinny can we make those spaces), re-paving, and amenities that help traffic flow (like signal lights and exit ramps). The eternal question is how to fit all of the needed cars into the given space while getting them in and out efficiently.

But Officer, They Made Me Do It

I submit that our companies' focus on the costs of system administration activities has blinded us to other potentially beneficial optimizations. We know that the way a small company operates cannot scale to a mid-sized company, and that the way a mid-sized company operates cannot scale to a large company. But when we scale up, we need to remain open to other ways of operating.

Event parking might evoke what this entails in a comparable way. While it is essential that a large number of cars get parked quickly and neatly, the solution is not a more rigid system but a more dynamic one. Plan where and how cars will be placed and then supply a mobile force to direct the stream of vehicles to the right places. But, when traffic is lighter, provide enough guidance in the form of signs and markings to enable the traffic to flow at that level as well.

For a similar situation in a technical vein, consider file system optimization. Optimize on time when file space is not a concern; optimize on space when it is scarce. This makes a tradeoff for scarcity of differing commodities – CPU cycles and disk space.

Personalized Service

How can a site that is optimized for cost give personalized service? Trick question – it can't. Not unless it can change from optimizing on cost to optimizing on service. In the same way that a parking facility changes modes from peak to off-peak, we need to identify other alternative optimizations.

Of course, "personalized service" is an area inundated with fraud. It is hard not to get bulk letters that appear "personal." Just a few encounters with "impersonal" personal correspondence are enough to heighten our levels of distrust.

Policy

If a small company tends to run with less policy and a closer relationship between sysadmins and users, large companies tend to be the exact opposite. Part of the process of "ensuring" quality is formalization. By taking some of the thought process out of how actions are carried out, individual capabilities are less significant. (This is old news for both the franchise market and the assembly line.)

When one individual knows what is happening on every portion of the network, there is little need for rules of conduct on that network – potential problems are already understood and resolved quickly (except when an individual lacks that specific experience).

When a great deal of network exists, it becomes essential that it be documented and understandable in pieces and as a whole. All parties to the operation of it must cooperate. The "cost" of this cooperation is less than the "cost" of conflicting activity on the network.

What is needed is a mechanism to tell which cost is most significant at any one time.

No Ticket, No Laundry

If you've ever read the fine print on your dry-cleaning receipt, you know that it is a reality – you lose that ticket (and the number) and they can't guarantee that you will be able to get your clothes back.

The trouble ticket system is an essential component on any sysadmin's tool belt. It should not be a surprise that this tool is found more frequently in larger sites. The mechanics of dealing with hundreds of problems without losing track becomes a burden.

The trouble ticket system is an essential component on any sysadmin's tool belt.

The trouble ticket is not so useful when:

- The trouble ticket system itself is nonfunctional.
- The computer one would use to report a problem is nonfunctional.
- The WAN connection to the central area is nonfunctional.
- Every system needs a specific action taken on it.

It is precisely for these reasons that sysadmins make sure there are redundant reporting mechanisms. In many such environments, email, pagers, and phone lines come into play. But what if the solution goes beyond ignoring the "system"?

The Alternative to Cost Optimization Is?

Here's where I'd like to present the "Tylock Theory" for optimizing on something other than less cost. Unfortunately, that insight hasn't been revealed to me yet. Perhaps in comparing notes we can find common ground. I'll offer these distinct situations where cost was less of an issue for me:

. . . .

Sorry, I can't name one. For every significant instance that I can recall quality as a driver, I paid the cost personally rather than in dollars from the company (example – working through the night or weekend to ensure an upgrade is problem free). For all of the issues where time was significant, cost was right there next to it (replacing broken equipment – yes, get it, but no, don't spend a lot on it).

So I'd like to hear from you. Please consider sharing your short story about optimizing on something other than cost. Drop me a note – with enough responses, I'll work up a composite of anecdotes. Without responses, I will of course have been proven right ;-).

The Sysadmin's Court: Liz Ation vs. Jack Trades

Transcribed by
Steven M. Tylock

Judge Daemon: I've read both the claim and the counterclaim. Please begin Ms. Ation.

Ms. Ation: Your Honor, by refusing to gain in-depth knowledge in any specific administrative domain, Mr. Trades is dooming the clients he supports to inferior service. He will be unable to properly assess and care for the critical situations that arise.

Mr. Trades: Your Honor, Ms. Ation is out of line when she suggests that my clients will receive inferior service when it is her environment that will suffer. By focusing on individual technologies, she and her associates will subject their clients to higher costs and incessant delays due to finger-pointing as each specialist disowns problems until proven otherwise. No problem will have a simple solution, and the overall architecture will suffer.

Judge Daemon: That will be enough. Ms. Ation, what specific issue do you have with Mr. Trades?

Ms. Ation: Mr. Trades refuses to delineate the specific nature of his administration. I present in support of this Figure 1, a "Brief Taxonomy of System Administration":

Brief Taxonomy of System Administration

<OS>ADMINISTRATOR (LINUX, SOLARIS, WINDOWS, ETC.)
<PRODUCT> ADMINISTRATOR (EMAIL, WEB, BACKUP, ETC.)
NETWORK ADMINISTRATOR
DATABASE ADMINISTRATOR
DESKTOP ADMINISTRATOR
SECURITY ADMINISTRATOR

SENIORITY SCALE:
JUNIOR
INTERMEDIATE
ADVANCED
SENIOR

Figure 1

Ms. Ation: If Mr. Trades were to focus his career in any one of these domains, he could be tested and certified as an expert. By not doing so, he loses the edge that the specialization would give him in both the workplace and the job market.

Mr. Trades: Your Honor, by learning enough about each of those domains to perform the generally required work, I am able to both design a robust environment and troubleshoot the entire computing and network infrastructure as needed.

Further, as an example, recall the botched rollout of the worldwide customer contact application? It was slow for every user not sitting in the headquarters. Each specific administrator said their domain was not at fault: servers were not loaded, the database was mostly idle, LAN and WAN traffic was clear, client systems worked normally.

The answer was noticing that each time the application wanted an update it had to make 120 round-trip calls to the server in headquarters. Each of these calls was dutifully and properly made, but the sum total of the time needed to make them had the application crawling along. The so-called domain experts provided graphs of proper service levels but missed the problem completely.

Ms. Ation: I'll concede that specific example, but there was the instance when our jack-of-all-trades was stumped by the nuances of protocol Z – and while he was scratching his head, the clients stopped working until an expert was brought in to clean it up.

Judge Daemon: Ms. Ation, what remedy would you propose?

Ms. Ation: That Mr. Trades be forced to declare a specialty and become properly trained in it.

Judge Daemon: And if the environment that Mr. Trades works in were to require expertise in three different specialties?

Ms. Ation: They would need a specialist for each of them.

Judge Daemon: So, If the organization had 30 system administrators and 10 domains of roughly equivalent needs, three admins would specialize in each?

Ms. Ation: Yes.

Judge Daemon: And if there was a budget and justification only for three system administrators for those same 10 domains?

Ms. Ation: (flustered) Well, they would have to live with inferior service as each admin takes on three or four domains of knowledge.

Judge Daemon: And Mr. Trades, what you are saying is that these three system administrators would be able to do quite well as generalists, with perhaps some minor differences within their abilities?

Mr. Trades: That is correct.

Judge Daemon: So, when the organization grows from those three admins to 30, do you think there is any cause for 10 of them to specialize in those specific domains?

Mr. Trades: (hesitating) Well, I suppose they would be able to at that point, as long as some of the generalists remained.

Judge Daemon: So we are agreed then that there is a body of knowledge required for the efficient operation of any organization's computing and network infrastructure.

The precise division of that knowledge into actionable units depends on a number of factors that will each affect the quality of service:

- Availability to the end user
- Level of expertise
- Knowledge of the overall environment
- Complexity of the overall environment
- Available budget for administration

I advise you to forget about being a specialist or a jack-of-all-trades and concentrate on making the infrastructure work according to the specific situation you find yourselves in.

Case Dismissed.

the bookworm

BOOKS REVIEWED IN THIS COLUMN

THE LANGUAGES OF EDISON'S LIGHT
CHARLES BAZERMAN
Cambridge, MA: MIT Press, 2002. Pp. 416.
ISBN 0-262-52326-4.

BEOWULF CLUSTER: COMPUTING WITH LINUX
THOMAS STERLING, ED.,
Cambridge, MA: MIT Press, 2001. Pp. 496.
ISBN 0-262-69274-0.

JIM SATO'S LEGO MINDSTORMS: THE MASTER'S TECHNIQUE
San Francisco: No Starch, 2002. Pp. 364.
ISBN 1-886411-56-5.

MAKING PROCESS IMPROVEMENT WORK
NEIL S. POTTER & MARY E. SAKRY
Boston: Addison-Wesley, 2002. Pp. 169.
ISBN 0-201-77577-8.

REQUIREMENTS BY COLLABORATION
ELLEN GOTTESDIENER
Boston: Addison-Wesley, 2002. Pp. 333.
ISBN 0-201-78606.

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

TCP/IP NETWORK ADMINISTRATION, 3RD ED.
CRAIG HUNT
Sebastopol, CA: O'Reilly & Associates, 2002.
Pp. 730.
ISBN 0-596-00297-1.

by Peter H. Salus

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

peter@matrix.net

I was going to devote this column to recent books on SQL, but I've spent the last four months traveling: in Scandinavia, Brazil, and the Netherlands (as well as the US). During this time a number of queries and interesting books have appeared, so I'm putting SQL off for the future.

I also want to express formal thanks to the folks running NORDU2002, to Michael Hejlskov Jacobsen and Sonny Larsen in Aarhus, to Gabriela Conceicao of the Sociedade Brasiliera de Computacao, to all those involved with the Forum Internacional Software Livre in Porto Alegre, and to Marielle Klatten and all the others who participated in SANE '02.

I was quite overwhelmed in Porto Alegre. There were 3500-plus at what I now understand is the largest software event in Latin America. I learned that the Brazilian state of Rio Grande do Sul has adopted open software and that the state of Minas Gerais was in the process of doing so. An anti-proprietary software bill has been introduced in Peru, too. The use of Linux and FreeBSD is amazingly widespread.

Interestingly, I seem to get asked many questions about the history of technology, not merely computing, so I'm going to begin with a history book.

Lux Fiat!

MIT Press has come out with a paperback edition of Bazerman's fascinating book on Edison. I recommend it to all of you. This is not a book about inventing electric light. It is about business and dealing with the public, with financiers (VCs?), and with government agencies. Bazerman is interested in symbols and sociology, and the electric light has become an incandescent and illuminating symbol.

Epic Poetry

Beowulf is the earliest epic poem in English, comprising a large part of the British Library MS. Cotton Vitellius A.xv. The use of Beowulf in computing has nothing at all to do with a mythic Germanic hero. *Beowulf Cluster: Computing with Linux* is a first-rate collection of essays supplying readers with a range of excellent tools for assembling a Beowulf cluster and for resource management.

LEGOs that Walk

Jin Sato is a major inventor. In *Jim Sato's LEGO Mindstorms*, there are toys and tricks to entertain nearly anyone. I was just thrilled to read about and examine the photos and diagrams of the robotic toys – are they really toys? – that Sato has developed. The chapter on "Making Mibo Walk" was really fascinating, if only because I can remember Marc Donner's brilliant walking robot (1983!) and his doctoral dissertation. I visited a lab at the University of Aarhus where LEGO robots were running quite impressively.

Management and Collaboration

Potter and Sakry have produced a small, easily read volume which I zipped through on a three-hour flight. *Making Process Improvement Work* has a concise approach to software process improvement, eschews a lot of the psychobabble and jargon that many books contain, and is full of good examples and concrete steps you can take. This may be a

book reviews

way of getting rid of those chronic software management problems.

Ellen Gottesdiener's somewhat larger (just about double the size) *Requirements by Collaboration* is about the human side of technology. More specifically, it's about how the quality of the various aspects of communication in a project, and most especially those employed when defining user requirements, strongly influence the success or failure of the project.

This isn't a book on building the product appropriately: it's a book about selecting the product to build.

Networking

If you really want to understand routing, I recommend Radia Perlman's *Interconnections* (Addison-Wesley, 1999), but for a smaller, more current view, I can't imagine a better volume than Malhotra's *IP Routing*. He assumes some knowledge of networking, of IP, etc., but he supplies the information that I imagine every network administrator needs.

I was really pleased to see the new third edition of Hunt's book on TCP/IP administration. Even if you've read the half-dozen or so parts of Comer and of Stevens, you need a contemporary handbook. Though it has gained a lot of weight over the years, it's still a fine piece of work.

HAVE YOU LOCKED THE CASTLE GATE?

BRIAN SHEA

Boston: Addison-Wesley, 2002. Pp. 193.
ISBN 0-201-71955-X.

REVIEWED BY CHUCK HARDIN
chardin@suchdamage.org

This is an insecurity book. If you follow its recommendations, you will be less secure in several respects than you were before. I'm not referring to merely running Windows; I assume the people who use the advice in this book start with default Windows installations with lots of viruses and exploits. The advice in this book can actually make you less secure than that.

The book incorporates a tedious analogy between a homestead-cum-village and a computer system in the evolution of their respective defenses. The concept is flawed, but it might have been excusable if the author had written a sufficiently compelling narrative to draw the reader through the book. He did not. The narrative is deadly dull and won't give the reader any real insights into the problems with computer security. It reads almost condescendingly, as if the reader herself were the peasant described in the narrative, to be led by the hand through the necessities of defending a computer.

Much worse, however, is the danger that the reader will buy into the author's analogy and adopt his security model, which was well-described by the IETF's Site Security Handbook (*http://www.ietf.org/rfc/rfc2196.txt*) as "the theory of a hard 'crunchy' shell and a soft 'squishy' middle." Shea emphasizes firewalls and physical security but ignores secure transport over the network for many kinds of data. The problem with pursuing this castle-like model has been well explored by the security community: once your single line of defense has been breached, you're wide open.

Shea's advice for defeating spammers is not much better. He recommends clicking on any provided link to be removed from the list. That's a great idea...for the spammers. Yes, the author warns you that you might have just verified the validity of your email address to the spammer, but offers no advice to avoid that problem. His advice is rather worse than useless.

He also presents important issues in illogical and inconsistent ways. One example is his Table 1-3, "Risk Numbers and Descriptions." It will badly confuse inexpert readers; more experienced readers will be annoyed that the author mixes attack mechanisms and attack goals with no apparent recognition of the difference. A virus is an attack mechanism, malicious destruction of data is an attack goal, but they should not be distinct items on the list — many viruses accomplish malicious destruction of data. Yet there they both are.

Roger Grimes' *Malicious Mobile Code* (O'Reilly, 2001) did much of what this book does, and does it much better. Buy that and skip this. It's horrible horrible horrible.

BUILDING SECURE SOFTWARE

JOHN VIEGA AND GARY MCGRAW

Boston, MA: Addison-Wesley, 2001. Pp. 528.
ISBN 0-201-72152-X.

REVIEWED BY NIINA KARHULUOMA
niina.karhuluoma@nokia.com

Software without security is a huge mistake. It is extremely easy to produce a buffer overflow type of attack on a system, and this could just as easily be stopped by taking some important design principles into account while programming.

Security is much more than just techniques or protocols like IPSec, SSL/TLS, and firewalls. It is an integrated process, which must be remembered in every phase of developing a system.

book reviews

Building Secure Software is one of the few books dedicated to examining the production or choice of software with security features.

In general, the book is excellent reading for anyone who wants to learn about, or who is already working with, software-related security issues. The book is understandable and well written – it can be read without deep security knowledge.

The book begins with an introduction to security awareness and gives some basic points as to where to get information about security vulnerabilities as well as describing the goals of security. These goals are useful to anyone who needs to know the different levels of security and why they are needed.

Entire books could be written on software risk management, but the writers manage fairly thorough coverage in their chapter on this broad issue. The chapter makes clear the importance of incorporating security into the software developing process from the beginning.

This book also takes up the pros and cons of open source versus closed source software. The best part of this chapter is that it provides guidance about the issues that should be taken into account in making this decision but does so without advocating for one side or the other.

The "Guiding Principles for Software Security" section supplies 10 important guidelines that can be used to avoid most potential problems in the context of software security. These guidelines are useful although they need to be considered case by case because enterprises frequently have restrictions on software features. Different kinds of rules (e.g., those applicable to customers) may cause difficulties in applying these guidelines literally. Still, the principles can help software developers to achieve more secure products.

Viega and McGraw take a close (and quite broad) look at software auditing. Auditing is an essential method of checking code's features and security. This should not be done merely by using code review. This book introduces some general baselines of how to audit software and what kind of tools to use, and it also provides some views about the effectiveness of auditing. The writers have a healthy attitude toward code and software checking – they are really pointing out that auditing is a method that needs to be part of the software process.

Viega and McGraw take a thoughtful look at security aspects that need to be considered in software design. Issues like buffer overflows, access control, race conditions, randomness, and determinism are described extensively in this book.

My own background is in cryptography research, and I was very pleased to see a book that describes secure software also considering cryptography-related programming issues like randomness (including views on the handling of entropy). This makes the book very good reading for anyone who needs to write code for cryptographic purposes. This area is very seldom described and quite often programmers end up learning by doing.

The cryptographic part of the book is extensive and provides an excellent overview, especially if the programmer is using some other, cryptography-specific, reference at the same time. The book also includes a short section about cryptography basics and can be used as an overview of the most essential issues concerning cryptographic software creation.

This book is excellent – useful for teaching at the university level and serving well as a handbook for those specialists whose main job is to develop software with security features. This book is clearly written and is a good introduction for people who want to get an overview of security in the software process.

From the President

by David Parter

President, SAGE STG Executive Committee

parter@sage.org

Have you heard the one about the cobbler's kids' shoes? We have – and fortunately, now we can laugh instead of getting angry. With a lot of hard work by the SAGE Online Committee (Trey Harris, chair; Gabe Krabbe; Josh Simon), staff and volunteers, SAGE has a new online presence. In fact, we now have a suite of Web sites:

SAGEwire (http://SAGEwire.sage.org) is our first new offering – a slash-based discussion and news site for sysadmins. We announced the public beta to the sage-members mailing list on July 3, and will continue to spread to word to wider audiences over the next few months. If you haven't been to SAGEwire yet, I encourage you to visit, make a login, and jump into the conversation. Please also pass the word to other sysadmins. SAGE members are among the brightest and best informed sysadmins – and your contributions (article submissions and discussion comments) will make SAGEwire the best sysadmin site it can be.

SAGEweb (http://SAGEweb.sage.org) will be the SAGE resource site for sysadmins. It will feature white papers, the SAGE short topics series, the SAGE jobs center, the SAGE store, and other resources for sysadmins.

We've also almost finished redesigning the entry page at *http://www.sage.org* – where the general public will go for information about SAGE. This is part of our new publicity and outreach campaign, headed by Rob Kolstad, our new SAGE Executive Director.

SAGEcert (*http://www.sagecert.org*) is the site for all things related to SAGE certification – news, information, study guides, discounts, etc.

With these four sites, SAGE has taken another big step towards serving our membership better, and expanding our reach to more sysadmins. Please give us feedback (email *sage-exec@sage.org*) and your ideas for how to continue to grow SAGE.

On another topic, elections for the SAGE Executive Committee will be held later this year. Details and a call for nominations will be posted on the Web and sent via email soon.

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

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

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

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

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

SAGE EXECUTIVE DIRECTOR

Rob Kolstad: *kolstad@sage.org*

SAGE MEMBERSHIP

office@sage.org

SAGE ONLINE SERVICES

list server: *majordomo@sage.org*
Web: *http://www.sage.org/*
http://SAGEwire.sage.org
http://SAGEweb.sage.org
http://www.sagecert.org

SAGE STG Executive Committee

PRESIDENT:
David Parter *parter@sage.org*
VICE-PRESIDENT:
Geoff Halprin *geoff@sage.org*
SECRETARY:
Trey Harris *trey@sage.org*
EXECUTIVES:
Bryan C. Andregg *andregg@sage.org*
Tim Gassaway *gassaway@sage.org*
Gabriel Krabbe *gabe@sage.org*
Josh Simon jss@sage.org

USENIX news

Summary of the USENIX Board of Directors Actions

by Ellie Young

Executive Director

ellie@usenix.org

The following is a summary of the actions taken by the USENIX Board of Directors at their meeting in Monterey, CA on June 12, 2002.

Finances

An internal audit for 2001 was performed for the Association by an outside accounting firm, Burr, Pilger and Mayer. See page 85 for more information.

The 2002 budget was discussed, and due to the decline in conference attendance and projected deficit of $1,500,000, the following actions were taken:

Transfers from the reserve funds to the operating funds from January, March and May 2002 were approved, and approval was given for $500,000 to be moved later this year.

It was agreed to publish 6 issues of *;login:* in 2002 (vs. 7).

Expenditures for Student Programs will be reduced by 50% as follows:

- Additional applications for Student Research Grant and Scholars program will not be entertained for the remainder of 2002.
- Fewer funds will be available for the Student Stipend Program which enables students to attend USENIX conferences.

The Board will cut back on discretionary and travel expenses.

No further expenditures on the E-Learning pilot program will be made.

Conference registration fees for technical sessions will be increased by $100.

Student registration fees for all conferences will be 50% of the regular tech session fees.

No other requests for funding good works beyond the two grants listed below will be considered this year.

The staff will prepare budget scenarios for 2003 for the Board's consideration this summer.

Grants

USENIX will support EuroBSDCon in 2002 with a grant of $6,000.

USENIX will again be a sponsor at the $10,000 level of the Grace Hopper Celebration of Women in Computing Conference in 2002.

SAGE

SAGE and SAGE Certification presented budget forecasts that would reduce some of their direct expenses for the remainder for 2002. The Board agreed to continue to support and subsidize both programs at a net deficit of $600K in 2002.

Committees and Liaisons

The following committees and liaisons were established:

COMMITTEES:

- Executive : Jones, Darmohray, Gilmore, McKusick
- Prizes & Awards: Hall (chair)
- SAGE Review Committee: Hume, Parter, Hall, Kolstad, Young
- Scholastic Services: Mary Baker, Darrell Long (chair), Rubin.
- STG Committee: Hume, Jones, Hall, McKusick

USENIX BOARD LIAISONS:

- Computing Research Association: Jones
- SAGE: Hall
- SAGE Interim Certification Board: Bennett

USENIX CONFERENCES LIAISONS:

- LISA, Philadelphia, Dec. '02: Rubin
- CARDIS, Nov. '02: Honeyman
- OSDI/WIESS, Boston, Dec. '02: Jones
- FAST, Mar. '03: Honeyman
- USITS, Seattle, Mar. '03: Honeyman
- Mobisys, SF, May '03: Jones
- USENIX Annual, San Antonio, June '03: Honeyman
- Freenix, Jun. '03: McKusick
- Security, Aug. '03: Rubin
- BSDCon, San Mateo, Sept. '03: McKusick

OTHER CONFERENCES (USENIX-RELATED):

- NordU: Hall
- EuroBSDCon: Honeyman

Young is the Staff Coordinator for each of the committees and is included on each mailing list.

THANKS

John Gilmore has made a donation of $15,000 to support the Association's Student Stipend Program in 2002. This program provides funds for travel, registration fees, and hotel expenses to attend USENIX conferences.

We hope that this generous donation will encourage others to do the same.

USENIX is most grateful.

Ellie Young, Executive Director
ellie@usenix.org

USENIX Association Financial Report for 2001

by Ellie Young

Executive Director

ellie@usenix.org

The following information is provided as an annual report of the USENIX Association's finances and represents the Association's statement of revenue and expenses for the year. Accompanying the statements are several charts that illustrate where your membership dues go, and what is spent on Good Works.

Audit

An audit was conducted by Burr, Pilger & Mayer, L.L.P. for the year ending December 31, 2001. The full financial statements and text of their report is available from the Association. The conclusion reached by the report is that "In our opinion, the financial statements present fairly, in all material respects, the financial position of the USENIX Association as of December 31, 2001 and the changes in its net assets and its cash flows for the year then ended in conformity with accounting principles generally accepted by the United States of America."

Financial Statements Summary.

These are challenging times, and USENIX is suffering from the overall downturn in the economy and, in particular, of the computer industry. In 2001, cash was down $1,735,000; the Reserve Fund was down $1,445,000; Net assets were down by 32%; Revenues down by 39%. This all translates into a very bad year financially. The USENIX Board has taken several actions (see above) to deal with this difficult situation.

USENIX MEMBERSHIP DUES & EXPENSES

USENIX averaged 8,300 members in 2001, and 58% opted for SAGE membership as well. Chart I shows the total USENIX dues income ($740K) for 2001, divided into membership types. Chart 2 shows where those dues were spent. Please note that all costs for producing conferences, including staff, marketing, and exhibitions, are covered by revenue generated by the conferences.

CHART 1
USENIX Membership Revenue Sources, 2001

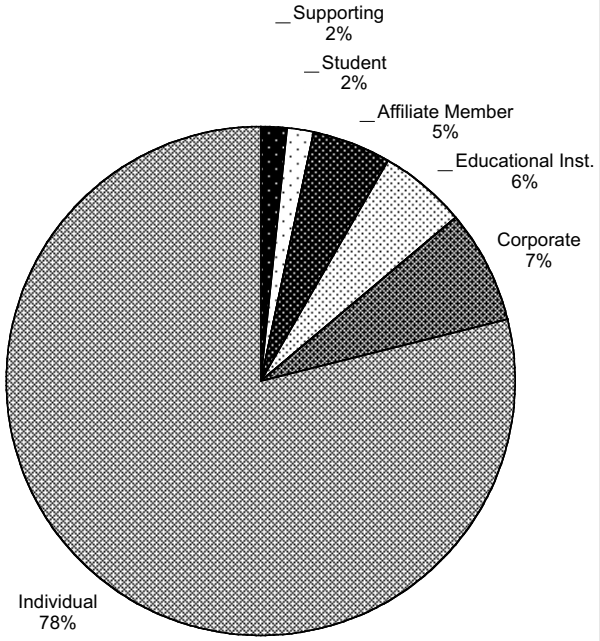
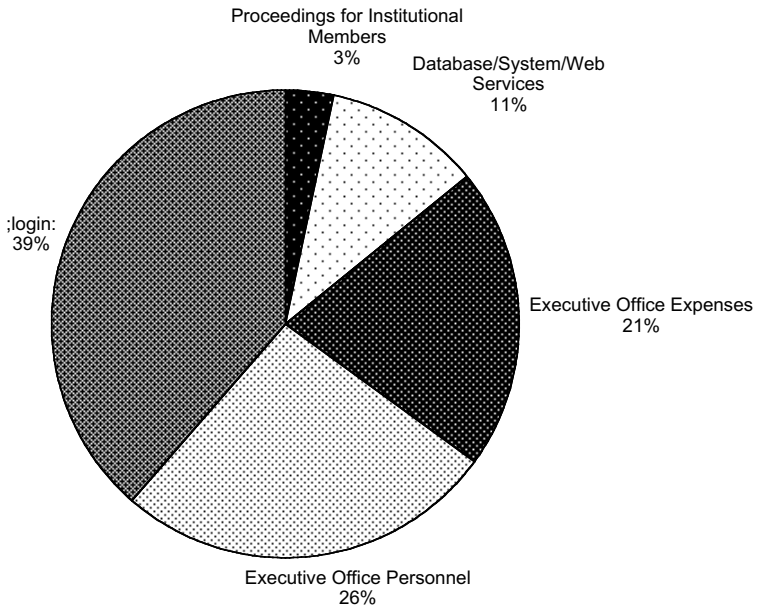


CHART 2
Where Your 2001 Membership Dues Went



SAGE

Chart 3 shows SAGE income and sources of support in 2001 ($455K). Chart 4 provides a breakout of SAGE expenses ($473K).

CHART 3 SAGE Revenue Sources, 2001

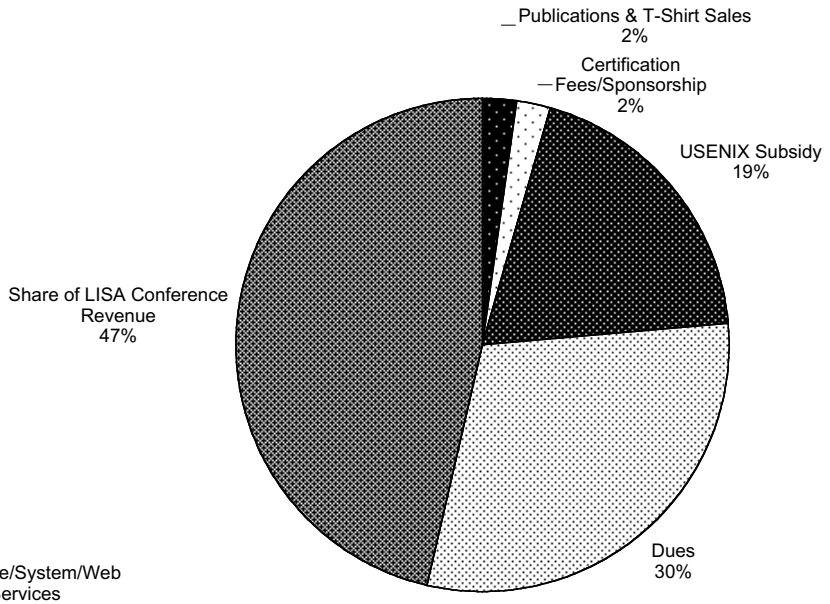
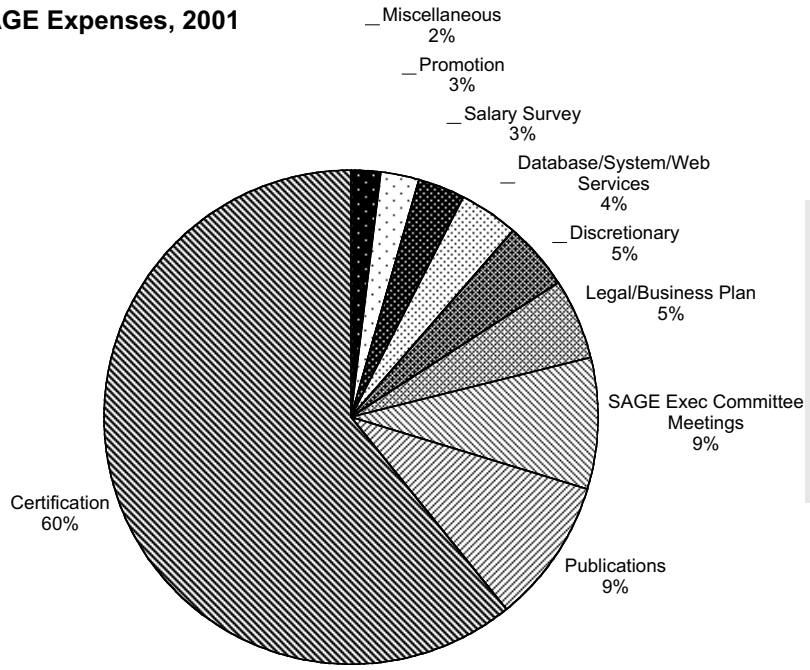


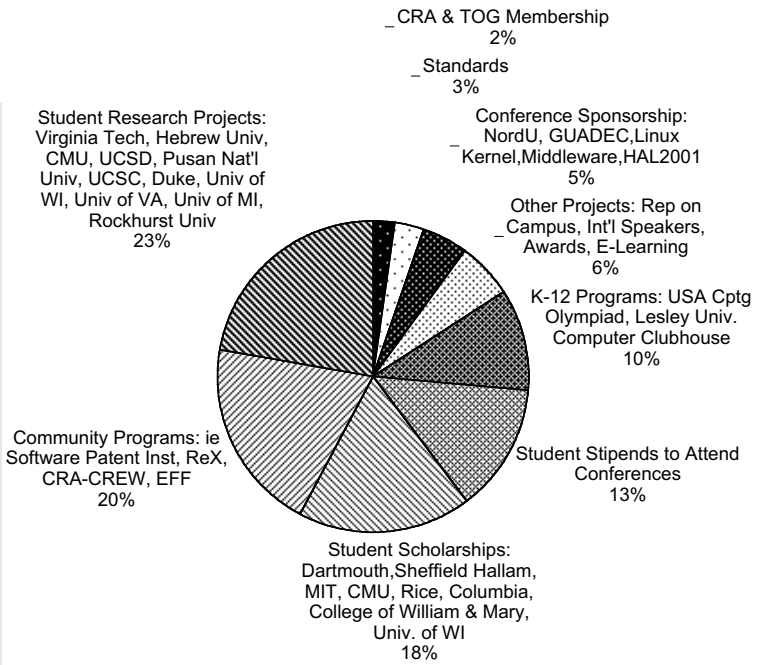
CHART 4
SAGE Expenses, 2001



USENIX PROJECTS AND GOOD WORKS.

Chart 5 describes how the money allocated to Good Works, and Projects ($496K) was spent in 2001.

CHART 5
Programs & Good Works Projects, 2001
Total Spent $967,192



USENIX ASSOCIATION
STATEMENTS OF ACTIVITIES
For the Years Ended December 31, 2001 and 2000

	2001	2000
Operating revenues:		
Conference and workshop revenue	$ 3,506,275	$ 6,005,509
Membership dues	739,856	947,846
SAGE dues & other revenue	151,820	255,294
Product sales	20,676	30,064
SAGE Certification	10,750	51,000
Total operating revenues	4,429,377	7,289,713
Operating expenses:		
Program services:		
Conference and workshop revenue	4,063,800	4,574,677
Programs and membership	629,833	658,447
Student programs, Good Works, and projects	981,806	1,044,583
SAGE	349,713	404,974
SAGE Certification	287,793	61,949
Total program services	6,312,945	6,744,630
Suppport services:		
Management and general	349,870	307,935
Fund raising	27,067	32,117
Total support services	376,937	340,052
Total operating expenses	6,689,882	7,084,682
Net operating (deficit) surplus	(2,260,505)	205,031
Net investment income and nonoperating activities		
Donations	532	50,000
Interest and dividend income	240,445	298,381
Net realized and unrealized losses on investments	(1,185,139)	(348,608)
Investment fees and costs	(94,171)	(115,966)
Net investment income and nonoperating activities	(1,038,333)	(116,193)
Change in net assets	(3,298,838)	88,838
Net assets, beginning of year	10,163,573	10,074,735
Net assets, end of year	$ 6,864,735	$ 10,163,573

USENIX ASSOCIATION
STATEMENT OF FINANCIAL POSITION
December 31, 2001 and 2000

ASSETS	2001	2000
Current assets:		
Cash & cash equivalents	$ 476,185	$ 2,212,063
Accounts receivable	66,936	364,982
Prepaid expenses	108,977	94,123
Inventory	31,225	20,149
Total current assets	683,323	2,691,317
Investments at fair market value -reserve fund	6638588	8,084,438
Property and equipment:		
Office furniture and equipment	422,576	497,378
Less: accumulated depreciation	(183,204)	(209,984)
Net property and equipment	239,372	287,394
Total assets	$ 7,561,283	$ 11,063,149
LIABILITIES AND NET ASSETS		
Current liabilities:		
Accounts payable and accrued expenses	$ 633,503	$ 860,225
Deferred revenue	63,045	39,350
Total liabilities	696,548	899,575
Net assets:		
Unrestricted net assets:		
Board designated	6,638,588	8,084,438
Undesignated	226,147	2,028,135
Total unrestricted net assets		
Temporarily restricted net assets		51,000
Total net assets	6,864,735	10,163,574
Total liabilitites and net assets	$ 7,561,283	$ 11,063,149

USENIX ASSOCIATION
STATEMENTS OF CASH FLOWS
For the Years Ended December 31, 2001 and 2000

	2001	2000
Cash flows from operating activities:		
Change in net assets	$ (3,298,838)	$ 88,838
Adjustments to reconcile change in net assets to net cash (used in)/provided by operating activities:		
Depreciation	77,455	67,545
Net investment income designated for long-term purposes	(94,289)	(69,304)
Realized and unrealized losses on investments	1,185,139	348,608
(Increase) decrease in assets:		
Accounts receivable	298,046	(249,427)
Prepaid expenses	(14,854)	(36,281)
Inventory	(11,076)	(1,605)
Increase (decrease) in liabilities:		
Accounts payable and accrued expenses	(226,723)	725,661
Deferred revenue	23,695	39,350
Net cash (used in) provided by operating activities	(2,061,445)	913,385
Cash flows from investing activities:		
Purchases of investments	(5,646,360)	(5,812,861)
Proceeds from sale of investments	5,646,360	5,812,861
Withdrawals from reserve fund	355,000	295,474
Additions to reserve fund		(903,933)
Purchases of property and equipment	(29,433)	(199,911)
Net cash provided by (used in) investing activities	325,567	(808,370)
Net (decrease) increase in cash and cash equivalents	(1,735,878)	105,015
Cash and cash equivalents, beginning of year	2,212,063	2,107,048
Cash and cash equivalents, end of year	$ 476,185	$ 2,212,063

USENIX ASSOCIATION
STATEMENT OF FUNCTIONAL EXPENSES
For the Years Ended December 31, 2001 and 2000

	Conferences and Workshops	Programs and Membership	Student Programs, Good Works and Projects	SAGE	Sage Certification	**Total Program**	Manage-ment and general	Fund Raising	**Total Support**	**2001 Total**	2000 Total
Operating Expenses											
Conference & workshop-direct	$ 2,656,037	$	$	$	$	**$ 2,656,037**	$	$ 10,099	**$ 10,099**	**$ 2,666,136**	$ 3,318,804
Personnel and related benefits:											
Salaries	783,827	116,359	6,716	73,539		**980,441**	130,109		**130,109**	**1,110,550**	1,025,320
Payroll taxes	58,227	8,644	499	5,463		**72,832**	9,666		**9,666**	**82,498**	70,536
Employee benefits	143,834	21,352	1,232	13,495		**179,913**	23,876		**23,876**	**203,789**	206,199
Membership/proceedings		40,102				**40,102**			**0**	**40,102**	45,613
Membership/login:		343,088				**343,088**			**0**	**343,088**	337,923
SAGE expenses				184,797		**184,797**			**0**	**184,797**	186,627
SAGE Certification expenses					287,793	**287,793**			**0**	**287,793**	61,949
Student programs, Good Works, and projects			967,193			**967,193**			**0**	**967,193**	977,038
General and administrative	421,876	100,288	6,166	72,419		**600,748**	186,219	16,968	**203,187**	**803,935**	854,673
	$ 4,063,801	$ 629,833	$ 981,806	$ 349,713	$ 287,793	**$ 6,312,945**	$ 349,870	$ 27,067	**$ 376,937**	**$ 6,689,882**	$ 7,084,682

Fifteen Years Ago in USENIX

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

At the USENIX Board meeting on March 26–27, 1987, the Board (Stephen C. Johnson, Marshall Kirk McKusick, Alan G. Nemeth, John S. Quarterman, Deborah K. Scherrer, Wally M. Wedel, and David A. Yost) unanimously approved the Business Plan proposed by Rick Adams and Mike O'Dell to found a service to be called UUNET.

I was authorized to meet with the Association's lawyer and account accountant and to sign checks for up to $35,000 for "the initial period."

The actual service began in mid-May. As I write this it has just celebrated its 15th birthday.

In retrospect, it's hard for me to be unemotional about this: I was an enthusiast when Rick made his first proposal to the Board in Monterey in October 1986. I was thrilled when UUNET was a clear success within a few months.

This was one USENIX project that was far more successful than anyone dreamt it would be, back in 1986–87.

Congratulations Rick and Mike...and the farsighted Board members.

USACO News

by Rob Kolstad
kolstad@sage.org

The USA Computing Olympiad (sponsored by the USENIX Association) has completed all but one phase of the 2001–02 season. After five Internet-based contests, 15 finalists were chosen to attend training camp at the University of Wisconsin-Parkside, home of Don Piele, the Olympiad's director.

Finalists were:

Seniors:

Adam D'Angelo	Phillips Exeter, CT
Jacob Burnim	Montgomery Blair HS, MD
Gary Sivek	TJHSST, VA
Steven Sivek	TJHSST, VA

Juniors:

Timothy Abbott, TJHSST, VA
Stephen Guo, Monta Vista HS, CA
Po-Ru Loh, James Madison Memorial HS, WI
Anatoly Preygel, Montgomery Blair HS, MD
Yan Zhang, TJHSST, VA
Yoyo Zhou, TJHSST ,VA

Sophomores:

Jongmin Baek, Cupertino HS, CA
Brian Jacokes, TJHSST, VA
Tiankai Liu, Phillips Exeter, NH

Freshmen:

Eric Price, TJHSST, VA

Alex Schwendner, Home school,TX

Long-time readers might note several familiar names, including the Sivek twins from Thomas Jefferson High School of Science and Technology. Again this year, TJHSST supplied the most students.

Freshman Alex Schwendner, a home-schooled student from Austin, Texas, was crowned this year's overall national champion, having placed high in all of the Internet contests, often against extremely difficult competition.

Alex Schwender

The training camp was the most competitive ever. Seniors are invited only if they have a significant chance of making the team. This year we had a record five seniors, all of whom were fighting for one of the four spots on the international traveling team. This year's big contest (the International Olympiad on Informatics – IOI) will be held in Seoul, Korea, on August 18–25.

Training camp included a "fun contest," which started the first night and continued as evening entertainment through the week (a challenging game-strategy program), and six programming contests throughout the nine-day event. Four of those contests were three hours in length; the remaining pair were a grueling five-hours long. The results of these contests determined the team of four that will represent the USA in Korea.

The coaching staff toiled long and hard to create a full year of contests to challenge the competitors through the week. Coaches included:

- Reed Barton, MIT freshman and last year's IOI world champion (and four-time gold medallist at the International Math Olympiad)
- Hal Burch, frequent USENIX speaker and Lumeta engineer
- Russ Cox, MIT grad student and Plan 9 release engineer
- Brian Dean, MIT grad student and Akamai employee
- Rob Kolstad, Executive Director of SAGE

In total, over 20 high-caliber problems were created, written up, solved multiple times, supplemented by test data, timed, and inserted into the contest-grading system (sometimes requiring a special program to check output from the finalist's entries). It takes about 8 to 12 hours to create a high-caliber problem that will pass muster at the elite level of competition these students were exhibiting . . . lots of work this year for the coaches.

Recreational activities (including Frisbee golf, the not-exactly-LISA Quiz Show, swimming, movie night, and bowling) kept the competitors busy every day from 8 a.m. to 10 p.m.

After seven days of camp, the IOI team selection came down to the final contest, with half of the finalists still in the running for the final four slots. After an agonizing discussion and repeated evaluation, the coaches chose four IOI representatives:

Jacob Burnim, a senior from Montgomery Blair HS in Silver Spring, MD
Adam D'Angelo, senior from Phillips Exeter Academy
Tiankai Liu, sophomore from Phillips Exeter Academy
Alex Schwendner, home schooled freshman from Austin, Texas.

Camp Director Don Piele kept operations running extremely smoothly. Don is also running the IOI in the USA for 2003 – contact him at *piele@uwp.edu* if you or your organization wish to assist in sponsorships for this event, which promises to attract competitors from around 80 countries. Don raised the bar this year for public relations by sending daily reports of camp activities to parents via email. He also posted a few dozen digital pictures every day.

The competitors had a great time as evidenced by letters from them and their parents directed to USENIX, the sole sponsor of the USACO. Jacob Burnim's mother wrote a particularly nice note (see sidebar).

The 2002–03 USA Computing Olympiad will start in October of 2002. Free training is always available at *http://train.usaco.org*; over 7700 students from around the world are currently registered.

Please join me in wishing the best for these outstanding students and encouraging any excellent pre-college programmers that you know to check out the USA Computing Olympiad at *http://www.usaco.org*.

SAMPLE TRAINING CAMP CONTEST PROBLEM: Sentence Finder (Parade Magazine)

The cows read Parade Magazine in the Sunday newspaper and really enjoy the sentence-find puzzles. Here's one:

```
C+E S-L M
R O T I A
A W H N F
S E A T A
B E S M R
```

The goal is to start at the C (to the left of the plus) and end at the S (to the left of the minus). Each move requires you to move to an adjacent, not yet used, letter by moving vertically, horizontally, or diagonally. As you traverse the letters, fill in this English-language sentence (more clues are given here than you will normally get):

```
C... ... ... .... ....
......S
```

In this case, the sentence is the standard cow-maxim taught to all the calves:

COWS ARE THE BEST FARM ANIMALS.

Given a puzzle and a dictionary of words, deduce the sentence that the puzzle represents. The dictionary should be read from a file named dict.txt. The dict.txt that will be used during grading can be downloaded for inspection. You will be allowed 1.0 CPU second on a 750MHz Pentium IV to find the answer.

Letters of Thanks

To Dan Geer
USENIX Board President

I am writing on behalf of myself and my husband Ira Burnim to thank you for the generous support USENIX provides to the USACO program run by Don Piele and his associates.

USACO has really made a different in our son Jacob Burnim's life. From the time he first discovered it on the Internet during his freshman year, it has provided him with the most challenging, stimulating, and enjoyable piece of his scientific and technical education. Even though he attended one of the best high school math, science, and computer science magnet programs in the country, his school could not provide him with the sophisticated learning experience he has enjoyed and is enjoying through USACO. His three camp sessions at the University of Wisconsin-Parkside were all great, and of course he is thrilled to be going to the IOI in Korea before he begins his studies at Caltech.

Jacob told us if he ever has money to give away, he would like to help support USACO – a good indication, I think, of how much the program means to students who participate in it.

Again, thanks very much for your support.

Sincerely yours,

/s/ Elizabeth Samuels

To Ellie Young:

As parents of a three-time USA Computing Olympiad finalist we would like to thank USENIX for its support. The USACO competition has enabled our son, Adam, to orient his interest in math and computer science. His focus on the contests became the most important part of his co-curricular high school experience. The friends and relationships he has established through USACO are wonderful. The advice and guidance from the USACO coaches, especially Rob Kolstad, is invaluable. Beyond programming information, he has helped Adam in his college search providing information about computer science departments across the country.

To illustrate how much the USACO competition means to Adam consider that the training camp this year in Wisconsin conflicts with his high school graduation. Without hesitation he chose the Olympiad week over the graduation ceremony. We are happy that he has the opportunity to compete for the international team.

The support your company provides for young computer programmers goes a long way in setting standards of interest and excellence. You should be commended for the support. As parents we feel both proud and fortunate to be a small part of this experience.

/s/ Susan and Raymond D'Angelo

Good Works

Mobility Support in a Publish/Subscribe Middleware

An abstract of work done with the Support of USENIX and Nlnet under the ReX exchange program. See http://www.usenix.org/XS/rex/ *for information and full reports on this program.*

by Mauro Caporuscio

mauro_caporuscio@katamail.com

This work focuses on the integration of a publish/subscribe middleware service with mobile components and applications. Publish/subscribe middleware is considered a good platform for the integration of loosely-coupled components on a large-scale. However, none of the implementations of publish/subscribe middleware available today is specifically designed to support mobile applications. Such applications are gaining popularity with the introduction of wireless data communication and portable computing devices such as PDAs or 3G cellular phones. Our idea is therefore to study how to design a publish/subscribe middleware capable of serving mobile, wireless applications. This effort consists of two parts: First, we studied the performance of an implementation of a publish/subscribe middleware built on top of a wireless network. Second, we studied the additional service-level requirements posed by mobile, wireless applications over the publish/subscribe middleware. In this paper, we present the results of our performance study, and the design and implementation of an auxiliary service-level support for mobile applications.

Thanks to USENIX

by Craig Soules

soules@ece.cmu.edu

My primary focus over the last year has been different two different research topics in the area of operating systems. The first project is called self-securing storage, and my focus has been on creating a space efficient versioning file system. The second project is online reconfiguration within an operating system. I have submitted papers on both of these projects to USENIX's OSDI 2002.

My work in self-securing storage has been on designing and implementing a comprehensive versioning system. This system uses a combination of file system techniques in novel ways to provide significant benefits in space utilization for versioned metadata while minimizing performance overhead. By combining a log-structured layout, multiversion b-trees, and a technique we call journal-based metadata, we were able to provide an increase in metadata space efficiency of over 80\%, reducing the overall space needed for versioning by nearly 40\%. This was work done with the help of my advisor and two other students, John Strunk, and Garth Goodson.

My work in online reconfiguration describes the benefits of having a single mechanism for reconfiguration within the operating system and describes our implementation of such a mechanism with IBM's K42 operating system. Once such a mechanism is in place, the system can easily support a number of well-known advances, such as application extensions, adaptive algorithms, and dynamic monitoring. We provide object hot-swapping and interposition within K42, and use it to implement a number of these benefits, concretely outlining the advantages and overheads of our approach.

I'd like to thank USENIX for the financial assistance I have received and I hope to have more interactions with the USENIX community as I continue with my degree.

This issue's reports focus on the Third International System Administration and Networking Connference (SANE 2002),

Our thanks to the Summarizer:

Diomedis Spinellis

conference reports

Third International System Administration and Networking Conference (SANE 2002)

Maastricht, The Netherlands

May 27–31, 2002

Summarized by Diomidis Spinellis

dds@aueb.gr

SANE, co-sponsored by USENIX and the NLnet Foundation, has evolved to be the European equivalent of the US-based system administration conference (LISA).

A lively and colorful crowd of systems-related attendees (including the obligatory UNIX elders), copious amounts of food, wireless Internet connectivity, interesting poster presentations, and a technical exhibition made the conference a fun place to be. Two parallel tracks of very interesting papers made the selection of presentations a real challenge. The following summaries are therefore only a subset of the conference's presentations. See the conference's Web page at *http://www.nluug.nl/events/sane2002/*.

The conference's keynote address was made by Bill Cheswick who described his Internet mapping work at Bell Labs that resulted in founding the Lumeta startup company. Mapping the Internet is becoming more and more difficult. Drawing routes from one point to another on a geographical map does not reveal any useful information in densely wired areas like North America and Europe: all that appears is a solid blob. More interesting are diagrams that depict the routes between different networks, arranged in a spring-like fashion, with well-connected networks appearing in the diagram's center and leaves at its periphery.

Color is used to distinguish different network providers, network addresses, or administrative domains (e.g., countries). Interestingly, directly representing the network IP address using a (red, green, blue) triple results in a map drawing where a mouse can directly determine the address of a given network by hovering over a particular color.

The applications of this research are numerous. Bill described how, after the events of 9/11, he ever more frequently finds himself at meetings in Virginia or Baltimore with individuals who refuse to identify themselves or the government branch they are working for! An animated map of Yugoslavia network connectivity during NATO's bombardments was especially interesting. We could see network links appearing and disappearing and total connectivity dropping as NATO started targeting Yugoslavia's infrastructure. "Son, you are making remote damage assessment from your basement," remarked one general about Bill's work.

Of course, viewing only Internet addresses and domain names has its limits. The excitement of a discovery of a group of Yugoslavian hosts (.yu) that proved to be extremely well connected, and resided somewhere in Virginia, was slightly tempered when he found out that he had only discovered the Yugoslavian embassy in the US.

Christine Hogan, co-author (with Tom Limoncelli) of the book *The Practice of System and Network Administration*, gave a talk on "Scheduled Maintenance Windows." This concept allows you to proactively plan your system maintenance (and thereby manage your users' expectations). The system administrator's role in such an exercise is similar to that of a flight director, the person you see in historical space-flight films managing the entire operation in the flight control center. The flight director knows

the mission details but does not participate in the actual operations, thus distancing herself from the task and keeping the clear head needed during the maintenance window's stressful hours. She is the one who will notice that the maintenance operations are running behind schedule and will command that the system should revert to its previous state (you did keep a backup, didn't you?), thus averting service disruption.

Mark Burgess, from Oslo University College, gave a thought-provoking talk titled "System Administration as Communication over a Noisy Channel." Mark believes that Shannon's communication theory can be used as the underlying foundation for explaining and predicting a number of phenomena related to system administration. And this is what science is all about. Specifically, Mark considers that a system's policy is communicated over a noisy channel, in which the users of the system represent noise. A significant result of this view is that error correction techniques are needed to create stable system administration tools.

Computer forensics, the study of the legal aspects of digital evidence, are increasingly important to system administrators who will be called to testify as experts in a court of law. Vlasti Broucek, an experienced system administrator who is currently researching this issue at the University of Tasmania, outlined in "Bridging the Divide: Rising Awareness of Forensic Issues amongst Systems Administrators" (co-authored with Paul Turner) the main challenges and techniques for preserving and effectively presenting forensic evidence.

Computer forensics differs from IT security in that it is typically conducted after an attack and its results will be presented to a non-IT-literate audience. Important aspects of digital evidence include its legal admissibility, its validity, and the conduct of the forensic analysis. System administrators faced with the task of collecting evidence should therefore minimize the handling of the original data, account for any changes, comply with the rules of evidence, and avoid embarrassment by not exceeding their knowledge and skills.

A paper co-authored by Giorgos Gousios of the University of the Aegean and your correspondent, "A Comparison of Portable Dynamic Web Content Technologies for the Apache Web Server," presented the main technologies for providing dynamic content on the Web (CGI scripts, PHP, mod_perl, mod_python and Java Servlets) and outlined the results of a series of benchmarks that measured their performance. FastCGI followed by mod_perl appeared to score best in moving data out of the server, but Java servlets proved to be the most resilient. The paper received the conference's best refereed paper award.

Diane Lark from Hewlett Packard presented work on a similar problem in a talk titled "Simulating Web Workloads." The major insight behind her and her colleagues' work was the similarity behind many Internet traffic patterns. They therefore used the SURGE network traffic generator to overcome the deficiencies of SPECWeb96 and Webstone. Through those tests they observed that Web serving is a memory-intensive operation that puts relatively less stress on the processor. They found out that a 1GHz server processor can serve the equivalent of 3000 users.

Your correspondent presented his work relating to the integration of home appliances, in a paper titled "The Information Furnace: User-Friendly Home Control." The Information Furnace is a basement-installed PC-type device that integrates existing consumer home-control, infotainment, security, and communication technologies to provide transparent user-friendly access and value-added services.

A modern home contains a large number of sophisticated devices and technologies. Access to these devices is currently provided through a wide variety of disparate interfaces. As a result, end-users face a bewildering array of confusing user-interfaces, access modes, and affordances. In addition, as most devices function in isolation, important opportunities to exploit synergies between their functionalities are lost. The Information Furnace distributes data, provides services, and controls an apartment's digital devices. Emphasis is placed on user-friendliness and on exploiting the synergies that inevitably come up when these technologies and services are housed under a single roof. The prototype implementation outlined integrates on a FreeBSD server the distribution of MP3-encoded music to DNARD/NetBSD thin clients, an answering machine, a burglar alarm, an Internet router, a fax server, a backup server, and intelligent control of a PBX.

A highlight of the conference, as indicated by the number of attendees, was the talk of Kirk McKusick (chief architect of the Berkeley UNIX and co-author of *The Design and Implementation of the 4.[3/4] BSD UNIX Operating System* books) titled "Running fsck in the Background." Kirk has a talent for simplifying the presentation of highly technical information. The fsck program verifies and fixes the integrity of UNIX file systems. Running such a program on a large disk (e.g., 100GB) can take hours, an unacceptable proposition for production servers. The problem was solved by taking a virtual snapshot of a disk (by temporarily suspending running system calls), maintaining the snapshot current by monitoring disk updates, and running fsck on that frozen snapshot. Snapshots could also be useful for

backing-up system state, and running a dump (backup) operation on a live system. Memorable quote: "I could write a special version of fsck, but I've already written fsck once, and I don't want to do that again."

Cor Bosman from XS4ALL gave a talk on installing and maintaining clusters of servers using PXE and Rsync. Installing software on a large number of servers can be a tricky proposition. Cor explained how he used the PXE remote booting standard supported by most modern Ethernet cards to transparently load and install FreeBSD on server clusters. The procedure is so smooth that having PXE booting enabled on the BIOS of a Windows machine will make it install FreeBSD on the fly — "a software upgrade," as Cor described it.

Mark Overmeer gave a talk on email processing with Perl. It turns out that many of the Perl modules that deal with mail are unsupported, buggy, and lack important functionality. In addition, correctly processing mail elements is a lot more difficult that what it appears to be. MIME encapsulation, multi-part messages, different presentation mechanisms, varying mail user agents, latitude in the mail header specification, and nonconforming implementations conspire to make the implementation of robust mail processing software a Herculean task. Mark worked on overcoming this situation by implementing a complete, robust, and reusable mail processing module (available on *http://www.cpan.org*) that other developers can import when building mail-handling applications. Thanks Mark!

The conference ended with an entertaining talk by Jos Visser titled "Welcome to the Tribe: Socio- and Anthropological Phenomena at UNIX Hacker Conferences," discussing the audience, conference, elders, mythology, economics, ethics, values, moral code, humor and entertainment, enemies, nutcases and outcasts, gadgets, women (section intentionally left blank), and status aspects of hacker culture. At the same time, Peter Salus, on a more somber note, gave a talk on "The Types of Internet Trauma: 1994–2002," where he showed how the Northridge earthquake, hurricane Floyd, fiber cuts, denial of service attacks, and the 9/11 WTC incident affected the Internet's connectivity. Overall the network fared well re-routing packets and compensating in real time.