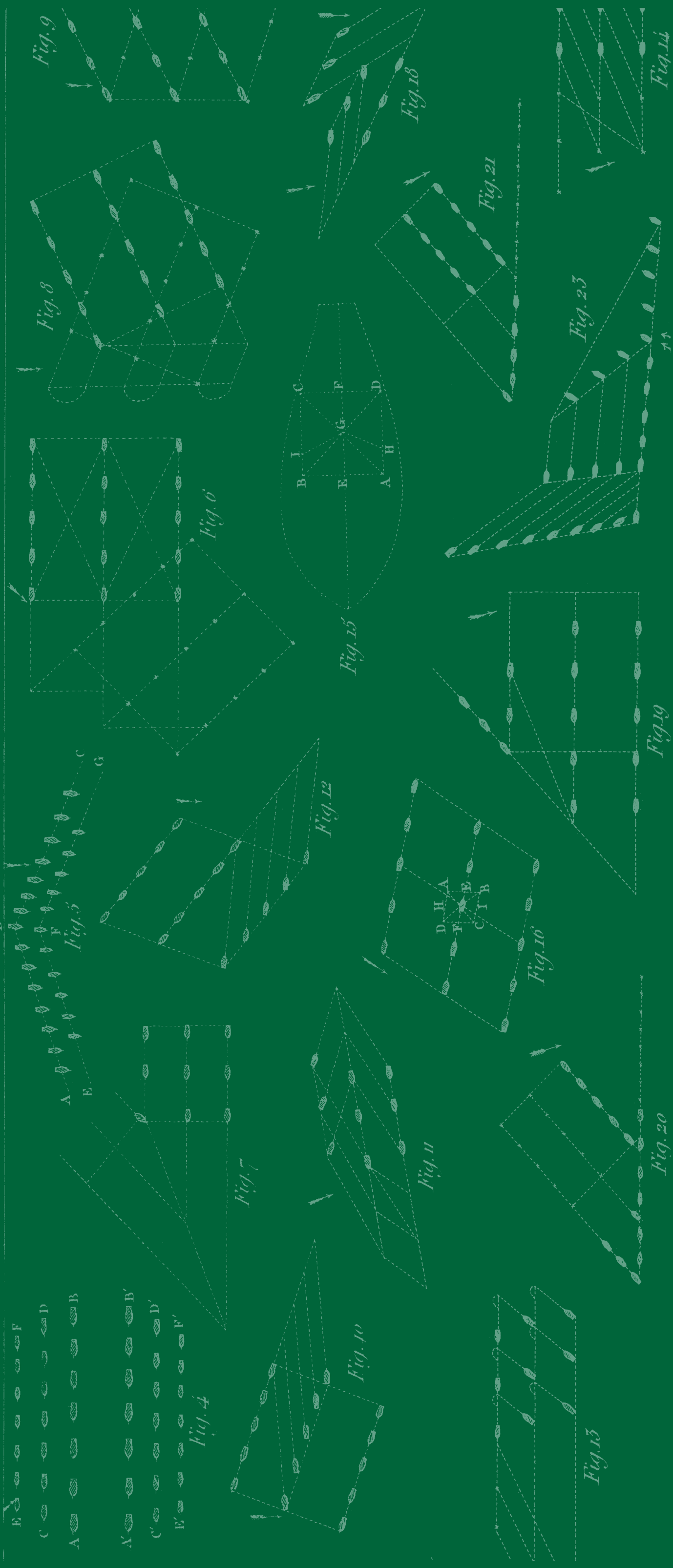
FEBRUARY 2006 VOLUME 31 NUMBER 1

;login:

THE USENIX MAGAZINE



USENIX

The Advanced Computing Systems Association

USENIX Upcoming Events

7th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2006)

Sponsored by IEEE Computer Society in cooperation with USENIX

April 6–7, 2006, Semiahmoo Resort, WA, USA
http://research.ihost.com/wmcsa2006

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

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

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

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

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

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

2006 USENIX Annual Technical Conference (USENIX '06)

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

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

Sponsored by IEEE Computer Society and USENIX

June 13, 2006, Dublin, Ireland
http://www.aqualab.cs.northwestern.edu/HotACI/
Paper submissions due: March 3, 2006

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

Sponsored by ACM SIGPLAN in cooperation with USENIX

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

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

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

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

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

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

15th USENIX Security Symposium (Security '06)

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

7th Symposium on Operating Systems Design and Implementation

Sponsored by USENIX, in cooperation with ACM SIGOPS

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

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

November 8, 2006, Seattle, WA, USA
http://www.usenix.org/usenix06
Paper submissions due: July 15, 2006

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

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

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

contents

;login:

VOL. 31, #1, FEBRUARY 2006

EDITOR
Rik Farrow
rik@usenix.org

MANAGING EDITOR
Jane-Ellen Long
jel@usenix.org

COPY EDITOR
Steve Gilmartin
proofshop@usenix.org

PRODUCTION
Rob Carroll
Casey Henderson

TYPESETTER
Star Type
startype@comcast.net

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

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

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

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

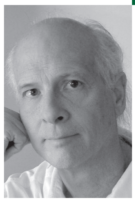
$85 of each member's annual dues is for an annual subscription to *;login:*. Subscriptions for nonmembers are $115 per year.

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

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

RIK FARROW

musings



rik@usenix.org

One of the advantages of being disorderly is that one is constantly making exciting discoveries.—A.A. Milne

I AM NOT ONE OF THOSE PEOPLE whose desk is neat, or whose office belongs in a house prepped in a manner that makes real estate agents smile. Hardly. I subscribe to the notion that such signs of orderliness speak of rigidity of mind. This extends to my writing and speaking, with various degrees of success.

For writing papers, disorderliness makes the process painful and slow. You want order when you are writing papers. On the other hand, with public speaking I have found that a certain lack of order leads to more interesting presentations. I was once tasked with presenting the same talk, twice a day, in twelve cities over the space of several weeks. I quickly found myself becoming bored, and challenged myself to tailor each presentation to the audience in front of me. And I succeeded, and was recognized by the sponsor of the road trip for my creativity while still covering the points outlined in the slides.

To be honest here, I decided that I could be a public speaker and lecturer after watching Timothy Leary perform in a bar in Berkeley. Professor Leary used a slide projector as both a visual aid and a crutch that kept him more or less on track. I like to think I am better organized than Leary, but not rigid, like a speaker reading from a prompter.

Management

But there are situations where disorderliness is anything but a creative solution. Anyone who has ever attempted to manage a large number of systems which share few characteristics other than the name of the operating system knows this for a fact. A measure of orderliness and a means for maintaining it become a life and sanity saver.

I became aware of configuration management in three phases. In phase one, I had researched patching solutions, reading through Proceedings of past USENIX conferences, looking for common threads in the many papers. I quickly spotted a common thread, and it went something like this: "We [the paper's authors] spent some time [6–18] months working to get all systems up to the same patch level before we could start using X [their solution]." Wow, that sounds like real pain, enough to convince most people that it was not even worth attempting the project.

What if you could avoid the pain of that half-year to eighteen months of work? In the next phase, sysadmins created solutions for building systems in standard ways. Instead of having a multitude of installs, you could have several basic installs, depending on how the target system would be used: desktop, Web server, mail server, file server, laptop, etc. Whether you use NFS-mounted installation packages or a set of CDs, you have at least solved one problem. And you can instigate a patching program that takes advantage of the homogeneity of your classes of systems.

My awareness of the final phase jelled during a conversation with Adam Moskowitz during LISA '04. Adam tried to convince me of the importance of configuration management, and that Paul Anderson should write a Short Topics Booklet about this. I countered by saying that it seemed that Mark Burgess's cfengine was a fine solution to configuration management, but Adam carefully explained that cfengine is one solution to configuration management, but not *the* solution.

In this edition of *;login:*, you can read three articles about configuration management. The first is an Opinion piece by Paul Anderson, who has written a Short Topics Booklet which will be published this spring. Paul makes a strong case for better configuration management tools judged on three criteria: reliability, security, and correctness. My personal favorite is the second, security. I have long asserted that a well-maintained system is also likely to be a secure system. If you wonder about that, just imagine the security of a poorly maintained system for a moment. There is a very real correlation.

Two articles which describe different configuration management solutions are presented, not with the intent of promoting either solution, but in the hope of making sysadmins aware that the configuration management tool space continues to grow. The problem is nontrivial, and much research and practice will be involved before we can come close to the solution that Paul advocates. Narayan Desai and his co-authors discuss Bcfg2 and their motivation for taking the time to set up a configuration management system, while Luke Kanies describes Puppet, the configuration management software he has been developing.

Onward

Marc Wallman provides a case study of implementing a campus-wide spam handling system using open source software. Tom Limoncelli talks about using external services as a method for delegating work, while Dustin Puryear provides useful advice about becoming a sysadmin consultant, based on his many years of experience. On a slightly different note, Thomas Sluyter lectures us on managing our time better (more organization).

Regular columns now have their own section, Columns, to make them easy to find. David Blank-Edelman introduces his new column on Perl by discussing (what a nice coincidence) handling of configuration files. Robert Haskins takes a look at a different type of spam than you may be used to dealing with.

In Technology, Dave Brown has boldly gone where others have also stumbled. Dave takes us through his adventures in using MythTV to build a DVR (Digital Video Recorder). Dave's journey reveals the travails of someone new to Linux (but not to programming and operating systems) as he accomplishes this complicated task, while managing to educate and entertain us at the same time.

In the Security section, we have a return engagement by Mike Rash. Mike wrote about port-knocking in the 2004 Security issue, and has returned to write about a new technique that provides authorization with only a single packet before starting a service like SSH. And the trio of Bill Cheswick, Steve Bellovin, and Angelos Keromytis take a deep look at how the vastly larger IPv6 address space will affect future Internet worms. Their article provides what is likely a prophetic view of a future Internet that will still be plagued with cleverly spreading malware.

The Book Reviews section of this issue is larger than usual, partially because there were no reviews in the December issue (note that some of these book reviews do appear in the online version of the December 2005 *;login:*). The missing reviews represented an embarassing oversight, in that the many eyes that read page proofs all failed to notice something that wasn't there. It has taught all of us a lesson about noticing what is *not* there.

Organized

There! I managed to describe the entire issue. Well, almost. I left out something near the beginning, where Alain Hénon, former managing editor of *;login:*, provides advice about improving the image of computer science practitioners. While image isn't everything (I am a great fan of substance), we really cannot ignore the image we present to the world, as it has a tremendous (and sometimes inadvertent) effect upon those we come into contact with. Now, if I could just remember where I left my tie . . .

PAUL ANDERSON

why configuration management is crucial

Paul Anderson has a background in pure mathematics and over 20 years of experience in system administration. He is currently a principal computing officer with the School of Informatics at Edinburgh University. He is the primary author of the LCFG configuration system and the organizer of the LISA configuration workshop series. His homepage is http://www.homepages.inf.ed.ac.uk/dcspaul/

dcspaul@inf.ed.ac.uk

THE PAST FEW YEARS HAVE SEEN an increasing interest in "Configuration Management." Some of us believe that the lack of good tools and procedures in this area is rapidly becoming *the* major barrier to the deployment of reliable, secure, and correct systems. I am going to try to define the configuration problem more clearly and to explore some possible reasons for these difficulties. However, it is worth starting with a few simple examples:

- Reliability—If we decommission a server, can we be certain that nothing else depends on this server in any way? Perhaps it might have been the only DHCP server on some little-used subnet, and problems will only become apparent when some host on that subnet fails to boot.
- Security—Can we be certain that the configuration files on a group of machines are set up so that there are no unexpected trust relationships between the machines? What if a supposedly secure machine installs new versions of an application from a remote file system on a less-secure server?
- Correctness—Can we be certain that every compute node in a cluster is running the required (new) version of a particular library, before starting a critical job?

An ideal configuration management system would prevent such problems by design. Current tools, used with best practice, should at least make it possible to identify and avoid them. However, sites with less-developed configuration management would even have difficulty in deciding whether or not such problems existed! For example, the information may only be available on the remote nodes themselves, and a certain percentage of these will always be unavailable at any one time.

Most sites have probably used some form of "configuration management" tool as a way of coping with large numbers of very similar "clients." This certainly addresses a whole class of configuration problems, such as the last of the above examples. However, modern computing installations form a complex web of related services. Managing the "servers" and the relationships they imply is much more difficult—this is the root cause of the deeper problems illustrated by the first two examples. The increasing scale, and particularly the complexity, of modern sites means that manual approaches to configuration of these relationships are no longer adequate; human system administrators simply cannot manage the complexity of the interactions, or foresee the full consequences of individual configuration changes.

Achieving high reliability in complex systems also requires the capability for fully automated reconfiguration. An *autonomic* system must have the ability to reconfigure some other machine as replacement for a failed server. In a multi-tier Web service, for example, this is likely to involve extensive reconfiguration of related services.

What Is "System Configuration"?

The basic system configuration problem is quite simple to describe:

- Starting with:
  - A large number of varied machines with empty disks.
  - A repository of all the necessary software packages and data files.
  - A specification of the functions that the overall system is intended to perform.
- Load the software and configure the machines to provide the required functionality. This usually involves a good deal of internal infrastructure, e.g., DNS, LDAP, DHCP, NFS, NIS services.
- Reconfigure the machines whenever the required service specification changes.
- Reconfigure the machines to maintain conformance with the specification whenever the environment changes—for example, when things break.

In practice, the task of "configuring the machines" probably involves editing configuration files (or perhaps supplying the configuration information via some API or GUI). However, it is not the mechanics of this process that is important; the real difficulty is in determining a suitable configuration for each service on each host that will make the overall system behave according to the specification.

To solve difficulties such as those in the first two examples, a configuration system must have a model that can represent the relationships implied by the configuration of the individual machines. Conceptually, we can think of a configuration tool as a type of "compiler," whose input is a set of requirements for the entire system and whose output is a set of configuration parameters for each service on each host in the system. Of course, a real configuration tool involves a lot more practical details, such as formatting and distributing the configuration information, but these parts of the process are comparatively straightforward.

Ultimately, we would like the input language for our configuration compiler to be at a very high level. For example, we might specify a Web service with certain behavioral properties, and the compiler would generate the appropriate configurations for all of the individual services, on all of the participating hosts. Unlike most programming languages, these configuration specifications need to be *declarative*—i.e., we want to specify the required properties of the resulting configuration and have the tool automatically work out the procedures for achieving the end result.

"Copy this disk image onto these machines"

⇓

"Put these files on these machines"

⇓

"Put this line in sendmail.cf on this machine"

⇓

"Configure machine X as a mail server"

⇓

"Configure machine X as a mail server for this cluster"
(and the clients will automatically be configured to match)

⇓

"Configure any suitable machine as a mail server for this cluster"
(and the clients will automatically be configured to match)

⇓

Configure enough mail servers to guarantee
an SMTP response time of X seconds

FIGURE 1

Present-day technology is some way from being able to translate such high-level specifications automatically; current best practice involves a combination of manual procedures and automatic tools that provide a smooth translation of the service requirements into implementable configuration details. Many factors, including the capabilities of the specific tool, will affect the level of detail at which the configuration needs to be manually specified. Figure 1 shows some possibilities, starting with very low-level tools which require all configuration decisions to be made manually, to very high-level tools which accept more abstract service requirements.

The final example defines a required behavior, and this is ultimately the type of specification that we would like to be able to make. However, this requires dynamic monitoring of performance levels, and the ability to do this in any general way is not yet part of any common configuration tool.

So What's the Problem?

The current situation with system configuration tools has many similarities with the early days of computer programming:

- Vendors sold mutually incompatible hardware. Changing platforms required a significant investment of time and resources.
- It was not possible to share code between machines without rewriting.
- The basic principles of programming had not yet been developed. Programs were created in unstructured ways that made them error-prone and difficult to verify or maintain.
- The low-level nature of the program code made it difficult to implement clear high-level objectives.

It was only the advent of high-level languages, with their underlying theory and portable compilers, that enabled this situation to improve.

In the system configuration field, there is a real need for new tools, based on sound theory and targeted at a much higher /level of configuration description. Programming-language development required a new generation of specialists to achieve a similar evolution, and it seems likely that real progress with configuration tools is not possible without a comparable development. This will require new specialists with a good understanding of theory, software development, and system administration practice.

As with programming-language development, the resulting systems will demand from working system administrators a significant change in approach. Worrying about which physical host is running a particular service should be as rare as worrying about which machine register is holding a particular Java variable!

ALAIN HÉNON

extreme makeover

Alain Hénon was the managing editor of *;login:* for a while; he is now providing tactical support for the chief executive of a major technical organization.

ah@usenix.org

THERE IS A CRY HEARD IN THE LAND of the computer industry: where are all the engineers-to-be? According to reports, students in the United States and abroad are shunning computer science courses in favor of—well, no one seems to know. Bill Gates complains that there are not enough graduate engineers to fill the halls of Microsoft. Other computer poobahs proclaim that graduate programs are going begging for Ph.D. candidates. Even the Chinese are concerned about the same trend, we are told. Various organizations, including USENIX, have been called upon to provide a solution to the problem. Why is the field not more appealing to young people?

I do no doubt that this dismal news is true. But somehow I am not surprised.

I believe that the general public has very mixed impressions of what the computer industry is all about. To begin with, the computer world has managed to gain a terrible reputation among ordinary people, including students and, more important, their parents. (Reading some of the reports mentioned above, you would think that students make career decisions in a vacuum devoid of any parental pressure.)

Put yourselves in the shoes of the average, non-engineer, naive computer user (most people, in other words). Their image of the people who work in the computer industry is that of folks who produce an appliance that is difficult to use; is still unreliable years after its introduction; becomes obsolete within three to five years; requires frequent updates to its software, at some considerable cost; has instruction manuals that run to hundreds of pages, if they are available at all other than online; is serviced by people who are often just one chapter ahead of you in the aforementioned manual or who are disdainful of your inability to understand how the escape key differs from the enter key; is susceptible to "invasion" by "viruses" and "bugs" from which, once again, you have to pay for protection. (Sounds like some sort of Mafia arrangement: you buy my gizmo or we'll invade your machine.)

And aren't they the same folks who shouted for all to hear that a revolution was at hand, everybody could start a business in their garage and make a million bucks, and why didn't you invest in their cleverly named new company which, any day now, would produce—uh, we're not sure what to call it right now, but it'll be great! And the whole thing collapsed and those guys walked away with your money and any confidence you might have had that they knew what they were doing.

But, you say, that's not fair, that's not what the industry is like. We're serious engineers doing important, exciting work. Why don't you join us?

Well, for one thing, there's the media image of the typical computer engineer. He (there are so few shes that it's hard to be politically correct here) is a nerd, a geek, a hacker, someone who will "crack" your computer and make it "crash" and "corrupt" your data, "steal" all your private information, and produce endless amounts of "spam." In the average B-movie, he is hirsute, ill-dressed, and typing madly on a keyboard in front of a flickering screen—you can almost smell him. Or else he is the nerdy high school student, with thick glasses and no friends, who giggles as he breaks into the FBI's computers and endangers our safety. Here's a quote from today's *New York Times*, talking about a "computer programmer": "He was straight out of central casting: nerdy-looking, glasses, pocket protector." Just so.

Unfortunately, some of those images ring true. Wander in the halls at technical conferences (something I have often done) and you will see that the media is actually on to something. I don't mean to suggest that all computer people should wear suits, but really, people, it is no longer 1968 and we are not marching down Telegraph Avenue anymore.

In short, do you really think the proverbial mother who wishes her darling daughter would marry a nice doctor, or perhaps an attorney, will wonder if the nice geek from across the street could be induced to take darling Judy out to the movies? Let alone that she will want little Judy to become a geek herself.

The industry needs an extreme makeover, to coin a phrase. If you want people to become like you, you first have to gain their respect. So I humbly suggest that all those who wonder about why young people are not going into this field get together and hire themselves the best public relations company in the country and be prepared to spend a lot of money trying to undo the damage. It probably can be done, but not by wringing your hands and offering a few scholarships to students who would rather be accountants. Not that I have anything against accountants.

NARAYAN DESAI, RICK BRADSHAW, AND JOEY HAGEDORN

system management methodologies with Bcfg2

Narayan Desai is a programmer and system administrator in the Mathematics and Computer Science Division of Argonne National Laboratory. His current research interests include system management and HPC system software issues.

desai@mcs.anl.gov

Rick Bradshaw is a system administrator in the Mathematics and Computer Science Division of Argonne National Laboratory. He helps to maintain HPC resources, experimental computing resources, and general UNIX infrastructure.

bradshaw@mcs.anl.gov

Joey Hagedorn is a student in Computer Science at the University of Illinois at Urbana-Champaign. When not studying, he spends time working on several programming projects.

hagedorn@mcs.anl.gov

AS UNIX NETWORKS CONTINUE TO grow in size and complexity, system management methods must evolve as well. In this article we discuss a typical deployment of Bcfg2 and describe its sophisticated configuration management capabilities. We present information about the environment at Argonne National Laboratory's Mathematics and Computer Science Division and the common tasks we in the systems group must perform, providing an overview of the tools used in our implementation of Bcfg2. We then discuss the procedural and qualitative impact that Bcfg2 has had on the way we manage our systems. Our aim is to describe what an environment with a comprehensive configuration management infrastructure looks like and to explain why one might want to invest the time needed to set it up.

Background

Configuration management is an area of intense interest in the system administration community. Although this area has seen substantial effort over the past 15 years, consensus on configuration management methods has not yet been reached. While a limited form of configuration management is widespread, relatively few organizations have adopted a comprehensive approach. Similarly, few practical accounts of tool adoption and results are available.

During the past year, the systems group in the Mathematics and Computer Science Division of Argonne National Laboratory redefined its methods for building, maintaining, and reconfiguring UNIX machines. The deployment process was quite involved, including substantial input from all 12 members of the systems group, and it altered the way many tasks are accomplished. Two aspects of the process proved especially interesting: the social aspects of tool adoption, and the technical aspects resulting from changes in systems management. In a companion paper, published at LISA this year (see Resources, below), we discussed the social issues, focusing on the nontechnical problems we faced. This article focuses on the technical issues, particularly the architecture we deployed, the changes we made in our management process, and the tools we chose.

BCFG2

Bcfg2 provides a declarative interface to system configuration. It was designed and implemented in-house at Argonne but has matured to the point that external sites have begun using it. Its configuration specifications describe a literal configuration goal state for clients. In this architecture, the Bcfg2 client tool is responsible for determining what, if any, configuration operations must occur and then performing those operations. The client also uploads statistics and client configuration state information.

All complicated processing occurs on the Bcfg2 server. It uses an abstract, aspect-based classing system to represent patterns in system configuration. These abstract classes typically correspond to functional characteristics of the configuration. For example, a class may contain a description of the configuration needed to produce a Web server, a Samba server, or an ntp client. Other, more abstract classes can also be created. These tend to be more site-specific—for example, “desktop” or “user-login.” The configuration needed to fulfill these goals will vary greatly from site to site. This classing system allows administrators to employ a cookbook-style approach to building new configuration profiles, once various classes are built. Administrators can decide to include features on a profile-by-profile basis.

The other main function of the Bcfg2 server is to provide a reporting system that describes details about client execution. Several different types of statistics are collected during each client execution. Also recorded are overall client configuration state and lists of configuration entries that were either modified or remain incorrect. Timestamps are stored so that inactive clients can be detected. The reporting system has a major impact on how Bcfg2 can be used. It provides sufficient feedback for administrators that they can solely use Bcfg2 for deploying changes on all machines.

ENVIRONMENT

In our division we have about 100 researchers, with large numbers of collaborators who frequently need access to our machines. During the summer, we have numerous temporary research aides and co-op students. The requirements of these collaborators and visitors strongly affects our network configuration:

- Many users access our resources from offsite. This access is vital for collaboration, but it means we cannot depend solely on a firewall for security.
- Our desktop environment is constantly in a state of flux because of constant staffing changes. Such changes are particularly pronounced at the beginning and end of the summer, when students arrive and leave. For this reason, the machine build process must be streamlined and easy.
- Our management system must continue to work and remain secure independently, in order to allow system administrators to focus on more pressing issues.

All said, our environment is fairly typical of most academic and research environments. The principal exception is that we have slightly more stringent security requirements than many sites, because of our government affiliation.

Deployment

Deploying Bcfg2 took substantial time and effort, including work by nearly all members of our systems group. Adopting a new set of tools and methodologies was a challenge, both technically and socially.

Most of the social issues had technical issues at their root, many of which were tool-specific. Administrators were not comfortable that Bcfg2 would *do the right thing* when reconfiguring systems. What followed was a six-month process of identifying what the *right thing* was and ensuring that Bcfg2 did it.

The other main task during deployment was the construction of a configuration specification that Bcfg2 could use to generate proper client specifications for our network. This task moved in jumps; some configuration aspects were quickly transcribed, while other, more subtle ones took much longer to get right.

TOOL REQUIREMENTS

During our group discussions about system management strategy, several key issues emerged. Administrator confidence in Bcfg2 was the most important issue. Administrators need to trust a tool, in terms of both generating proper configurations and performing correct reconfiguration operations on the client. Without such trust, administrators won't use a tool for anything important.

To address this issue, we chose to make Bcfg2's behavior as observable as possible. Specifically, we implemented a comprehensive dry-run mode in the Bcfg2 client. This allowed our administrators to experiment with the tool without undue pressure; once they were comfortable that the pending changes were reasonable, they were willing to commit to adopting the tool. Similarly, high levels of debug output were added, documenting all decisions the client makes while determining what operations should be performed. The availability of this information fostered confidence in the client, because the administrator could watch the tool in operation and understand why it performed the way it did.

Another issue we addressed was management of client configurations. Bcfg2 had to be able to handle all aspects of client configuration and reconfiguration without manual intervention. It also needed to be robust in the face of manual client reconfiguration. (Who hasn't made several changes debugging a problem, only to cause a new problem later?) To this end, we designed Bcfg2's reporting system so that it can describe all aspects of Bcfg2's actions and can provide salient information about client configuration. This reporting system gives administrators the ability to consider client configurations in a class-based way, using the Bcfg2's configuration specification for all nodes. The reporting system then reports all deviations from that specification. We augmented the Bcfg2 client to detect configuration elements on the system that weren't specified in its configuration. These *extra* configuration elements are also reported back to the server.

The third issue we considered was convenience. We streamlined several common tasks, including the machine build process, and we made the configuration profile selectable from the boot disk menu. These small measures typically reduced the interactive time substantially. Most important, they were vital in convincing administrators that it was worthwhile to spend time learning how Bcfg2 works.

CONFIGURATION SPECIFICATION

In parallel with our technical discussions, we devised a Bcfg2 configuration specification that describes our network. We started with the desktop system. This is, by far, the largest basic type of system in our division and thus has the most uniform configuration. Building a specification for our desktop systems consisted of identifying services and software on each machine, recording these in the configuration specification, and then testing this configuration in stages.

Next we turned to the servers. This process took much longer, for a number of reasons. Server configurations varied much more than desktop systems. Desktops had been managed in an organized way, while servers were managed in an ad hoc fashion. Servers also had much more complicated service definitions. Many of these systems had specific owners who had performed manual modifications over time. Most important, these machines provided a large number of user-visible services and represented the infrastructure on which the entire division functioned. Servers were one of the primary drivers for many of our technical discussions. Once these issues were resolved, however, the specification process was quite similar to that of the desktop process.

The basic procedure for incorporating new classes into the specification comprises writing a description of all the interrelated configurations that provide a service, and collecting relevant configuration information, such as configuration file contents and permissions. This process can be expedited by using the Bcfg2 client in dry-run mode. Once all configuration information has been integrated, the Bcfg2 client can be used to detect whether any reconfiguration on the system is needed.

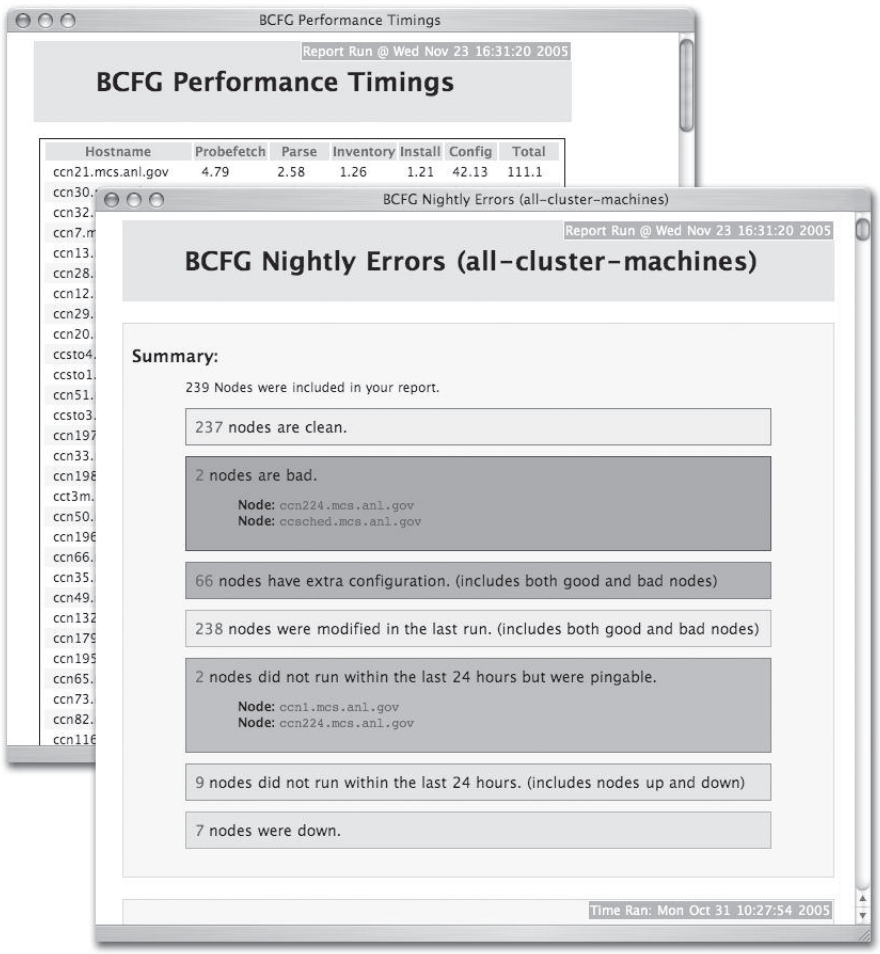
Another Bcfg2 feature that allowed smooth migration was the ability to incorporate information about unmanaged hosts. Bcfg2 stores statistics about all aspects of a client configuration that do not match the configuration specification. If the Bcfg2 client is run on a machine, statistics describing its configuration deviations are uploaded to the server and included in system reports, even if no changes occurred. These reports can be used to find areas where the configuration specification is incomplete.

REPORT-BASED CONFIGURATION MONITORING

Bcfg2 configuration reports provide an impedance-matching mechanism between the configuration specification and the actual configuration state of all clients. Discrepancies between the two cause a variety of latent management problems. Most important, if a service-providing machine has a running state that does not match the configuration specification, it cannot be rebuilt or duplicated. Its state also cannot be reasoned about by Bcfg2.

This system also allowed us to administrate our servers in a more interactive fashion. We run the Bcfg2 client on each server in dry-run mode. The client inventories the local machine state, determines what operations should happen, and uploads this information to the server. Administrators view the resulting reports daily, and can supervise the execution of the Bcfg2 client on critical servers when it is needed.

The reporting system provides a bird's-eye view of the overall configuration state of all clients. This view exposes configuration specification problems, allowing their repair before they cause problems. We now feel confident in our understanding of all machines that are properly described in the configuration specification, show a clean configuration state, and have no extra configuration detected. *All* of our administrators now have a deep


BCFG Performance Timings

Report Run @ Wed Nov 23 16:31:20 2005

BCFG Performance Timings

```
Hostname          Probefetch  Parse  Inventory Install  Config  Total
ccn21.mcs.anl.gov   4.79      2.58     1.26      1.21   42.13   111.1
ccn30.
ccn32.
ccn7.m
ccn13.
ccn28.
ccn12.
ccn29.
ccn20.
ccsto4.
ccsto1.
ccn51.
ccsto3.
ccn197
ccn33.
ccn198
cct3m.
ccn50.
ccn196
ccn66.
ccn35.
ccn49.
ccn132
ccn179
ccn195
ccn65.
ccn73.
ccn82.
ccn116
```

BCFG Nightly Errors (all–cluster–machines)

Report Run @ Wed Nov 23 16:31:20 2005

BCFG Nightly Errors (all–cluster–machines)

Summary:

239 Nodes were included in your report.

237 nodes are clean.

2 nodes are bad.

Node: `ccn224.mcs.anl.gov`
Node: `ccsched.mcs.anl.gov`

66 nodes have extra configuration. (includes both good and bad nodes)

238 nodes were modified in the last run. (includes both good and bad nodes)

2 nodes did not run within the last 24 hours but were pingable.

Node: `ccn1.mcs.anl.gov`
Node: `ccn224.mcs.anl.gov`

9 nodes did not run within the last 24 hours. (includes nodes up and down)

7 nodes were down.

Time Ran: Mon Oct 31 10:27:54 2005

understanding of all of our machines' configurations, or clear indicators for situations where they do not.

Impact on Administration

Rebuilding our management infrastructure had a dramatic effect on our daily lives. Many everyday procedures were simplified, and powerful new mechanisms for automation became available. Moreover, the system administration process was changed in a qualitative way that transcends particular tasks.

PROCEDURAL CHANGES

Our new management infrastructure enabled several categories of procedural changes. Some tasks disappeared altogether. Many more changed in basic character. The fundamental unit of automation in our old environment was the venerable shell script. Scripts are useful for a variety of purposes but are lacking in one major way: scripting multi-machine processes is fault-prone, and error handling is difficult. Moreover, these scripts generally have a lot of local topology information hardcoded inline. This approach prevents them from being portable across sites.

Bcfg2's model—specifically, the existence of a central, declarative configuration specification that can be programmatically modified—makes simple scripts considerably more powerful. Administrative applications need only calculate final results and can leave all error handling to the Bcfg2 client. These applications can be adapted to the Bcfg2 server plug-in interface, which is called during client configuration generation. This plug-in interface has access to add or alter configuration elements on any managed client. We have adapted several administrative applications to use this interface:

- Controlling user access to batch-scheduled nodes
- Managing SSH keys and creating a correct ssh_known_hosts file
- Balancing virtual hosts across several Web servers

In each of these cases, the plug-in logic needed encapsulated a near-literal transcription of policies or configuration generation rules. The previous implementations of each were uniformly complicated and non-portable. In all cases, conversion to this API reduced the code volume by 75% or more. This reduction occurred because much of the heavy lifting is now handled by the Bcfg2 client.

Good experiences with basic automation have led us to attempt much more complicated workflows. For example, we are adapting our IP/host management application to directly feed Bcfg2 with DNS and DHCP configurations. Once this system is integrated, it will be easy to correlate with any effects from other plug-ins. We are confident that such efforts will automate many of the remaining daily reconfiguration operations requested by our users.

QUALITATIVE CHANGES

More important than the procedural changes, several qualitative changes affected the administration process overall. These transcended the performance of particular tasks and changed the character of system administration in our division.

The most striking change was that configuration management tasks became a proactive part of the environment. At regular intervals, all clients check against the central specification for configuration changes, and may (depending on their settings) apply configuration changes. In any case, statistics describing their current state are uploaded. Bcfg2 serves as a steady-state deployment engine that can detect and correct configuration inconsistencies.

This change in model allows administrators to focus on changing the configuration specification and inspecting reports describing the results without having to worry about deployment details. Having a comprehensive deployment engine also greatly reduces the cost of individual reconfiguration operations. The availability of cheap reconfiguration operations expands the range of options open to administrators. We found that our environment now has some daily churn of configuration changes. Automated scripts can make changes based on external stimulus, such as the release of software updates, and deploy appropriate configuration changes across clients.

These changes resulted in an environment where we can make reasoned judgments about how we want changes to propagate to our environment. The time freed up by deployment automation allowed us to design a system with a more measured approach to change management and testing. Our changes now migrate to desktop machines first; on these systems we value security more than anything, because of the large number of clients.

In contrast, we are willing to wait for administrators to run the Bcfg2 client on servers, so that they can ensure that everything is still running properly. This approach costs more than the one employed for desktop machines, but we think that it is worthwhile for server machines. Most important, we were able to make a local determination about how we wanted changes to propagate and implement that exact system. This is a policy matter that will vary greatly from site to site, and one size will never fit all.

Overall, these changes resulted in a much more deliberate system management process in our environment. Administrators were freed from repetitive tasks, and we were able to exploit the new tools to make the decisions that only experts can make effectively.

Conclusions

Configuration management needs to be adopted globally. It can dramatically reduce the time spent performing repetitive tasks. Initial efficiency gains can be fed back into improving configuration management capabilities. The net effect is a large time savings for system administrators—time that can always be used to improve services for users.

Bcfg2 provides a good framework for automating complex workflows. This infrastructure offers an interface with simple and reliable reach throughout your environment. This enables easy automation at a scale not previously possible. Complex, network-wide policies can be implemented from a central location. Moreover, the central configuration specification and statistics can be mined for a variety of information.

This redesign of our infrastructure took a substantial amount of time and effort. Nevertheless, we recommend that others attempt the same. The long-term benefits far overshadow any short-term costs.

RESOURCES

The Bcfg2 Web site: http://www.mcs.anl.gov/cobalt/bcfg2 (information about Bcfg2, including a manual, mailing list archives, and sources).

Narayan Desai et al., “A Case Study in Configuration Management Tool Deployment,” *Proceedings of the Nineteenth System Administration Conference (LISA ’05)* (Berkeley, CA: USENIX Association, 2005). This paper describes the social issues encountered by a large system administrator group during the adoption of new tools and management procedures. The paper provides the social counterpoint to the technical account of this process provided here.

Paul Anderson, *Configuration Management* (Berkeley, CA: USENIX Association, forthcoming). This book provides a primer in configuration management, as both a practice and a research area.

The lssconf mailing list: http://homepages.informatics.ed.ac.uk/group/lssconf/. This mailing list provides vigorous discussion of configuration management tools and the techniques they employ. Many configuration management tool developers subscribe to this list.

ACKNOWLEDGMENTS

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38.

The submitted manuscript has been created by the University of Chicago as Operator of Argonne National Laboratory ("Argonne") under Contract No. W-31-109-ENG-38 with the U.S. Department of Energy. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.

5th System Administration and Network Engineering Conference

SANE 2006

15–19 May 2006

Aula Congresscentre, Delft, The Netherlands

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

www.sane.nl/sane2006

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

LUKE KANIES

Puppet

NEXT-GENERATION CONFIGURATION MANAGEMENT

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

luke@madstop.com

IN MY DECADE-LONG CAREER AS A system administrator, I have always done my best to use tools to lower my workload. I like to think one of my goals is to be the laziest person around, and great tools help me to reach that goal by allowing me to get more done with less effort.

I've spent the past few years as a consultant, integrating existing tools and developing new ones when the need arose. I generally restricted my development to smaller projects, because my revenue model did not allow me to take years or even months off for long-term development, which meant that I largely had to rely on existing tools as the primary solutions I provided. Experimentation with different ways of getting better tools, including contributing to existing projects and working within a larger organization, finally led me to attempt to create the tool that I really wanted to use to do my job. I call this tool "Puppet"; I have released it under the GPL, and I am building a company, Reductive Labs, around developing and supporting it.

Puppet is being developed with two purposes in mind. The first and most obvious purpose is to be the best configuration management tool available, such that I can build a company and community around making sysadmins' lives easier. Less obviously, I am developing Puppet to be an operating system abstraction layer (OSAL), something that functions as a cross-platform API into the features of the OS without forcing you to delve into the messy details that come with each separate distribution and release. I believe that providing this OSAL is a required step in providing the best automation tool, and I hope that other tools can also begin writing to this OSAL instead of coming up with a new way of handling each OS's messy details.

To download Puppet or the Puppet source code, go to http://reductivelabs.com. There you will also find links to the blog I maintain about Puppet's development, mailing lists, and everything else you've come to expect from open source projects.

A Low-Level Abstraction Layer

At some point all automation tools must isolate their users from detail, else those tools would take more effort than they saved. The point at which they hide detail, though, varies widely among tools and has a large impact on how those tools are used and developed. Tools like cfengine hide almost no detail at all, enabling the user to choose exactly how to interact with the system but requiring that the user know too much about each supported platform, such as where the crontab command is or what command to use to add users. Conversely, tools like SmartFrog and LCFG use large, coarse-grained modules responsible for large swathes of functionality; these hide almost all detail, but they leave the user only as much room for customization as the module developer thought appropriate.

Puppet could be said to be either lower- or higher-level than cfengine, depending on how you stack it. On the one hand, Puppet is designed to hide details such as file locations and the differences between "useradd" and "adduser," which makes it higher-level than cfengine, but it also provides hooks to directly modify a much larger range of detail on a given operating system, so it could be said to be lower-level. I like to think of it as just being different: cfengine provides a simple API to the elements that the operating system cares about (files and file contents, packages, processes, etc.), while Puppet exposes an API to the elements that humans care about (users, groups, cron jobs, virtual hosts, etc.). There is some crossover, such as humans sometimes caring about file permissions, but more often than not two otherwise equivalently functional cfengine and Puppet configurations will look quite different.

I have ambitious goals for Puppet's OSAL. Package management systems could write directly to it, instead of each pre- and post-install script having its own idea of how to create a user or cron job. System administrators could use it to replace unstable or outdated tools without affecting the core functionality, as long as the OSAL could configure each tool equivalently. In the end, I hope that Puppet's OSAL will be the standard repository for all of the details necessary to configure each of the different operating systems.

A Better Automation Tool

Puppet is more than an abstraction layer, though. It is a whole declarative configuration management framework. In addition to altering the level and type of detail that administrators must handle, Puppet also raises the bar in terms of expressiveness and communication. It includes a simple but powerful language (LISP taught us that language power often comes through simplicity rather than complexity or variety) capable of expressing the relationships between the different elements of an operating system, along with a set of clients and servers meant to make it easy to get information into and out of your network.

A LANGUAGE BASED ON RELATIONSHIPS

Puppet's language is declarative, meaning that you specify the "what" (objects and their values) but not the "how" (how to turn those objects into configuration state). The OSAL library takes care of the "how" for us, so the language can focus entirely on the objects, their values, and how they all relate.

Puppet's language has only one goal: to get your network to provide the features you need in the way you need them. Given an abstraction layer that handles all the details of the different operating systems and applications, service provision turns into a process of collecting the list of objects you need, marking how those objects are related, and filling in the details. To provide an Apache service, you need to collect the package, the content to provide, the configuration for the service, the service itself, and maybe an IP address and a file system.

Puppet's language provides a classing mechanism to indicate that this collection of objects functions as an Apache service, and it goes one further by allowing you to specify the relationships between these objects, such as the obvious fact that the service can only actually be started when all of the other objects are correctly in place. It even allows you to go further still and specify whether the service should be restarted if the package gets upgraded or the configuration gets modified.

Puppet provides basic abstraction mechanisms in the language, so you can vary individual details based on other details (e.g., different groups for different operating systems) or vary the work itself based on details (e.g., provide different elements on different operating systems).

REUSABLE CODE

There is nothing resembling a CPAN for system administration tools, because we have never had tools that could separate these relationships that we care about—the elements that make up a service, how those elements relate to each other, and what details the elements should have—from the specifics of how to implement those relationships on a given platform and in our environments. The existence of an OSAL provides us the opportunity for that separation, so Puppet's language has been written with a focus on reusability. If you know exactly what it takes to make a Solaris 10 server secure or to provide Apache plus mod_perl, you can write a server class that does this and then share that class. When I have development time, I plan on creating community space online to facilitate this sharing, but I think Puppet's simple language will encourage it whether I facilitate or not.

COMMUNICATION IS THE KEY

Every sysadmin knows that tools cannot be silos, meaning they cannot be cut off from the rest of the tool fabric, yet for some reason most of our tools aren't very fond of talking to each other. Each tool seems to want its own user and host database, and it is often prohibitively difficult to get data from one application to another. This extra overhead usually just means that we hack up our own, simpler tools that all talk to each other but are too specific (and too embarrassing) to share, rather than using the well-known tools.

I will not let Puppet fall victim to that. I am building simple APIs into both the client and the server, using XML-RPC over SSL, so if you want to write a querying tool, you can, or if you want to replace some portion of Puppet with a much better tool that responds to the same interface, you can. In addition, I am doing everything I can to get data back out of Puppet—if you provide Puppet with information, it will do everything it can to take advantage of it, and you should certainly never have to tell Puppet the same thing twice.

I plan direct integration with the different tools in the sysadmin fabric, although I have been so focused on the core functionality (which is now release-worthy) that I have had little time to spend on this. Puppet will directly open tickets and configure your monitoring and trending systems, rather than assuming that you will just do that yourself, and when it does so it will provide everything it can to make your job easier. You have already given Puppet enough information for it to manage your network; it would be downright offensive if Puppet did not use that information to provide context to the information it gives you back. Every network has a somewhat constant stream of failures; the context of the failure, such as the services it affects and the overall failure rate, is what determines whether it is a critical failure or not, and Puppet will do everything it can to provide that context.

Puppet already has a built-in log centralization mechanism, for instance, and the logs that Puppet produces include the configuration path to the specific element that emitted the log, so you know whether a package installation failure is affecting DNS or GDM, whether it's on a workstation or server, and whether it is on your main LAN or in your DMZ. You do not have to explain to the log server where your LAN is versus the DMZ, because you already explained that to Puppet, which just sends the information along.

By the way, Puppet does use SSL certificates for all authentication and encryption, but because of the inherent complexity in managing those certificates, Puppet includes a simple certificate manager to help you. By default, the central Puppet server creates a new certificate authority, each new client asks that authority to sign its certificate request, and there is a simple command-line tool you use to sign those requests.

The Syntax

Although there is a lot more to Puppet than its language, and it is expected that mechanisms other than the language will eventually be supported for input, the language is currently the only way to speak to the OSAL.

I have tried to keep Puppet's language as simple as possible. It looks more like a data dump than a language, and I plan on keeping it that way as long as I can. It only supports a few statement types, mostly centered around describing and aggregating objects, along with a few simple operator-like constructs for greater abstraction.

VARIABLES AND ASSIGNMENT

One of the statement types you will immediately recognize is assignment:

```
$variable = value
```

Simple words do not need to be quoted in Puppet (strictly speaking, words which match /[-\w]+/ in Ruby do not need to be quoted). Variable scopes work as one might expect (well, at least they work as I expect)—variables are visible when defined in the current scope or any enclosing scope. The only twist is that, in an attempt to be declarative, variables cannot have their values set more than once in a given scope.

When retrieving its configuration, a Puppet client collects a configurable set of facts about itself, and these facts are defined as variables in the top-level scope. The most useful facts are things like $operatingsystem (usually the output of uname -s), $ipaddress, $domain, and $hostname. These facts are all retrieved using a separately maintained library imaginatively named Facter (which you can get independently from http://reductivelabs.com), and Puppet converts them all to lowercase.

ELEMENT SYNTAX

Puppet's elements can be thought of as a kind of named hash, or named associative array. Any specification of low-level elements requires the element type and the element name, and all element types support a fixed list of arguments. When applied appropriately, the following snippet will verify (and fix, if necessary) the metadata of /etc/passwd and make sure that the latest version of the sudo package is installed:

```
file { "/etc/passwd":
    owner => root, group => root, mode => 644
}
package { sudo: install => latest }
```

You could use the stand-alone puppet executable to apply this snippet, and it would check that /etc/passwd is owned by user and group root, that its mode is

644, and that the sudo package is installed and is the latest version available (via whatever mechanism is defined to retrieve packages). Puppet will automatically fix any deviations it finds, although this can be set to just log deviations, rather than fixing them.

The "package" statement is a bit special, because Puppet knows what the default package type is for every platform on which it runs. For those platforms like Debian and RedHat that support automated retrieval of packages and their dependencies, this statement would be enough. For those platforms like Solaris and AIX that do not, you would need to provide additional information (e.g., a URL) on how to retrieve the package, but not how to install it or any of the messy details of putting it into a /tmp and so on.

You can easily specify multiple objects at a time, either separating them with semicolons or just using an array as the object name:

```
file {
    "/etc/fstab": owner => root, group => root, mode => 644;
    "/etc/named.conf": owner => root, group => named, mode => 644
}
```

This will check both of these files as though they had been specified in separate file blocks. This type of syntax is useful for those cases where you have many objects of the same type but with entirely different details.

```
file { ["/etc/shadow", "/etc/sudoers"]:
    owner => root, group => root, mode => 440
}
```

This will check that the two specified files have the exact same metadata. This is obviously useful for those relatively rare cases where you have many objects of the same type and with the same details.

You can, of course, combine them:

```
file {
    ["/etc/shadow", "/etc/sudoers"]:
        owner => root, group => root, mode => 440;
    "/etc/fstab": owner => root, group => root, mode => 644;
    "/etc/named.conf": owner => root, group => named, mode => 644
}
```

This is just a combination of the other two snippets, in the tersest (and, thus, not necessarily the most readable) form. Actually, you could get even more terse if you desired, but you'll have to see the documentation for how to do that.

White space does not matter, so any differences in it in this code are to help you, not Puppet.

CLASSING

Puppet supports three main encapsulation constructs: classes, components (created using the define keyword), and nodes. All three constructs are just named collections of objects, with some use-appropriate behaviors tacked on.

The simplest and most common construct is a basic class; it is useful for collecting a set of related elements which all get applied to provide a certain service:

```
class apache {
    package { apache: install => latest }
    service { apache: running => true, requires => package[apache] }
}
include apache
```

This class contains two statements, one that makes sure that an Apache package is installed and another that verifies that the Apache service is running (Puppet defaults to using init scripts to start, stop, or check services) and also specifies that the service depends on the package. Once this class is created, it can be applied using the include keyword in any host's configuration. Classes also support inheritance, although this is more useful for defining server classes than for providing a specific service:

```
class base {
    package { sudo: installed => latest }
}

class webserver inherits base {
    package { apache: installed => latest }
}

class dnsserver inherits base {
    package { named: installed => latest }
}
```

This defines three classes, each of which verifies that a package is installed. Including any of these service classes also includes the base class, just as you'd expect inheritance to work.

NODE CONFIGURATION

Puppet provides a class-like structure for specifying a given node's configuration:

```
node kirby {
    include $operatingsystem, webserver
}
```

The include function can handle variables just fine, and kirby happens to be a Solaris x86 server in my basement, so when kirby connects, the central Puppet server will return a configuration containing all the work associated with the sunos and webserver classes.

CODE REUSE

The last organizational structure in Puppet is analogous to a function in other languages; I alternately call them components or definitions, and they are created using the define keyword. You would use it to specify a chunk of work that will be applied multiple times in the same configuration. For instance, I store many of my configuration files on a central server, so I have created a remotefile keyword that encapsulates all of the details that get repeated with each copy:

```
define remotefile(source, mode => 644) {
    file { $name:
        owner => root, group => root, mode => $mode,
        source => "/nfs/files/$source"
    }
}

remotefile { "/etc/sudoers": mode => 440, source => "sudoers" }
```

This defines and then uses the component remotefile, which is just a wrapper for a simple file statement. This file statement verifies that the local copy of the specified file is the same as the remote copy of the same file (I have chosen to use an nfs-mounted file, but Puppet supports some remote protocols, too) and then verifies that the local file has all of the correct metadata.

Component prototypes define the arguments that they accept, just like other functions, and Puppet components can have defaults. The only quirk in Puppet is that the value before the colon in a Puppet statement (e.g., /etc/sudoers in the example) is an implicit argument and is set as the $name variable inside the component.

ABSTRACTIVE OPERATIONS

Puppet only has one operator in the strict sense of the word, and it only has one construct that resembles a control statement. The operator resembles a C-like trinary operator, modified (some would say butchered) slightly to fit in a bit better. It is useful for using one value to determine another value:

```
# remember $operatingsystem is set by Puppet for you
$fstab = $operatingsystem ? {
    sunos => "/etc/vfstab",
    default => "/etc/fstab"
}
```

Notice the use of the default value here as a fall-through option. These statements can also be used inline anywhere a value is needed, and multiple, comma-separated values can be provided for a given option. The other statement type useful for abstraction is a switch-style statement like many others; see the documentation for details.

Conclusion

Puppet is an ambitious new open source configuration management framework. It provides an extensible operating system abstraction layer (OSAL), along with a language that writes to that language and a set of clients and servers for sharing information such as configurations and file contents across the network. Puppet is still in its early stages and does not yet handle the majority of the elements that you need to manage, but it is easy to extend and has a keen focus on supporting and encouraging community involvement.

MARC WALLMAN

spam filtering for the enterprise

Marc Wallman is Senior System Administrator at North Dakota State University.

Marc.Wallman@ndsu.edu

THE EVER INCREASING VOLUME OF spam causes ubiquitous frustration for end users. In the spring of 2005, North Dakota State University (NDSU) made a commitment to do something about this problem for its users. What follows is a presentation of our response: a spam filtering system combining existing open source software and homegrown applications. The resulting system was designed to have a minimal impact on our existing mail system, to be easily scalable, and to be modular enough for us to be able to perform maintenance on particular components without disrupting the others.

The spam filtering system presented here is not a stand-alone system. It was assembled from existing open source products and integrated using homegrown connectors into an existing enterprise email system. Because this is an integrated solution, the spam filter and the mail portion of the system will both be presented here. The email system, and to a certain extent the spam component, developed organically along the lines of the spiral model of software engineering (although much more informally). The focus of this article will be on the design of the system as a whole and the implementation of the spam filter.

Why create a spam filtering system when there are hundreds of open source products that address the problem of spam? Why not just pick one and use it? Here's why: there is no single open source product that is appropriate in and of itself at the enterprise level. Existing open source spam products are partial rather than comprehensive in their approach. They provide specific solutions to particular problems. For instance, SpamAssassin [1] is a very effective tool, but making it directly available to end users is not appropriate. System administrators may be comfortable with Bayesian filtering, editing rule sets, and configuring delivery rules, but end users are not. SpamAssassin is useful, but incomplete without help from other applications.

An important caveat exists here. Deploying solutions like SpamAssassin may be possible in a straightforward way if end users do not need to be given a choice of whether or not they use it or how it behaves. Consider the case of EduTech, the state-funded IT organization that provides email services for K–12 institutions in the state of North Dakota. Clearly, much of the spam that floats around the

Internet is inappropriate for minors to receive; there is no need to give them an option about how the spam they receive is handled. EduTech system administrators may decide what is spam and what is not spam. Anything that is determined to be spam, they may delete. No flexibility is required on the part of the EduTech system administrators.

NDSU hosts approximately 18,000 IMAP mail accounts for six schools in the North Dakota University System plus the University System Office. It accepts approximately 135,000 messages on a given weekday for approximately 210,000 recipients. A typical message takes 18 seconds or less from the time it is sent to the time it is read by an actively checking recipient. A middleware solution called Hurderos was developed in-house for integrating mail for the aforementioned institutions. Unfortunately, this middleware system is beyond the scope of this article. Those who are interested may read more about the GPL-licensed Hurderos at http://www.hurderos.org. For the sake of simplicity, the rest of this article will treat this spam solution as a single-institution solution.

A Failed Attempt and Lessons Learned

This section will briefly describe a first attempt at a SpamAssassin-based spam solution and some of the lessons learned. An attempt had been made to provide a central solution several years prior. However, the implementation of a centralized SpamAssassin-based solution was largely ineffective, because insufficient thought was given to maintainability and scalability. This opt-in service worked by piping mail through SpamAssassin before final delivery on the IMAP mail servers where users' mailboxes resided. Two main problems existed with this system. First, SpamAssassin was invoked once per piece of mail, resulting in lots of expensive forking. Batch processing was not possible. Processing the bounces from a bulk mailing to the entire campus would cause noticeable slowness on the server that housed the sender's mail account. Second, as SpamAssassin aged, it became difficult to update. The solution was implemented using a 2.x release of SpamAssassin, which required Perl 5.6. A little over a year later, SpamAssassin 3.0 was released, which required Perl 5.8. The IMAP mail servers ran RedHat Enterprise Linux 2.1, which provided only Perl 5.6. Suddenly, we were in a situation where we would have to maintain our own version of Perl and associated modules if we wanted to keep SpamAssassin up to date. SpamAssassin requires frequent updates and architectural changes to the system it runs on, while the University of Washington IMAP daemon has the opposite requirements.

As the SpamAssassin-based solution aged and lost its effectiveness, some users turned to the built-in spam filtering capabilities of their email clients. While possibly effective on a case-by-case basis, this approach was never successful at an enterprise level, for two main reasons. First, we did not have, and still do not have, established mail client standards; therefore, our help desk ends up supporting everything from Outlook to Eudora to Pegasus Mail. This is an unfortunate situation, but one that many other higher-education institutions and ISPs find themselves in. Client diversity presented a serious obstacle to providing effective end-user support for client-side spam filtering. Further, client-side spam filtering is not universally present in these various mail clients and is complicated by users who use multiple mail clients (e.g., Thunderbird in the office and a Webmail client on the road). Second, this strategy places the work of catching spam on the end user. For at least some users, this is a difficult thing to do. It is not uncommon for people to misconfigure their mail client to block messages from everyone on campus.

System Requirements

The requirements for this system came from a committee of users led by NDSU's IT Security Officer (ITSO). This committee established general requirements for this new spam filtering system. This non-technical component was critical to the success of this project. The recommendations were just that—recommendations. No particular solution was specified (commercial or open source). The recommendations were to be fulfilled insofar as was possible. The rest of this section summarizes these recommendations.

Mail should be grouped into three categories: (1) obvious spam, (2) potential spam, and (3) not spam. Identifying what is spam is not an exact science. The intent of these three categories is to recognize this fact. Messages falling into the category of obvious spam are those that are most clearly identifiable as spam and most likely to be considered spam by everyone. Potential spam consists of those messages that are probably spam. It is a more aggressive category than obvious spam and thus encompasses a larger corpus of messages. The not spam category is self-evident.

Using these categories, different levels of service were established. At all levels of service, mail that is considered not to be spam is delivered to users' inboxes. Obvious and potential spam are handled differently, depending on the desired level of service. A matrix summarizing the various levels of service is depicted in Table 1.

Level	*Obvious Spam*	*Potential Spam*
Disabled	Deliver	Deliver
1	Quarantine	Deliver
2	Quarantine	Quarantine
3	Delete	Quarantine
4	Delete	Delete

TABLE 1: RECOMMENDATIONS FOR HANDLING SPAM

By default, new users have the spam service disabled, making this an opt-in solution. All mail is delivered to users' inboxes, regardless of the category it falls into. At level 1, obvious spam would be quarantined and potential spam would be delivered to the user's inbox. Ideally, the quarantine would appear as another folder under the user's mail account. Both potential spam and obvious spam are quarantined at level 2. At level 3, obvious spam is simply deleted with potential spam being quarantined. Finally, at level 4, all spam is deleted. In addition to these varying levels of service, the committee asked that users be able to specify lists of senders who were considered safe. Mail from addresses on this safe sender list was always to be delivered. A corresponding list of senders to block was also requested. Mail from addresses on this list was always to be deleted.

Open Source Spam Filtering

The open source solution deployed at NDSU was based on two products already in use, MailScanner and SpamAssassin. This section will present the resulting solution.

Spam filtering for the enterprise was developed within the context of a preexisting open source email infrastructure. The mail system is an IMAP mail solution built on Linux. Each of the components—sending, delivery, and retrieval—will be briefly examined. It is illustrated below in Figure 1.

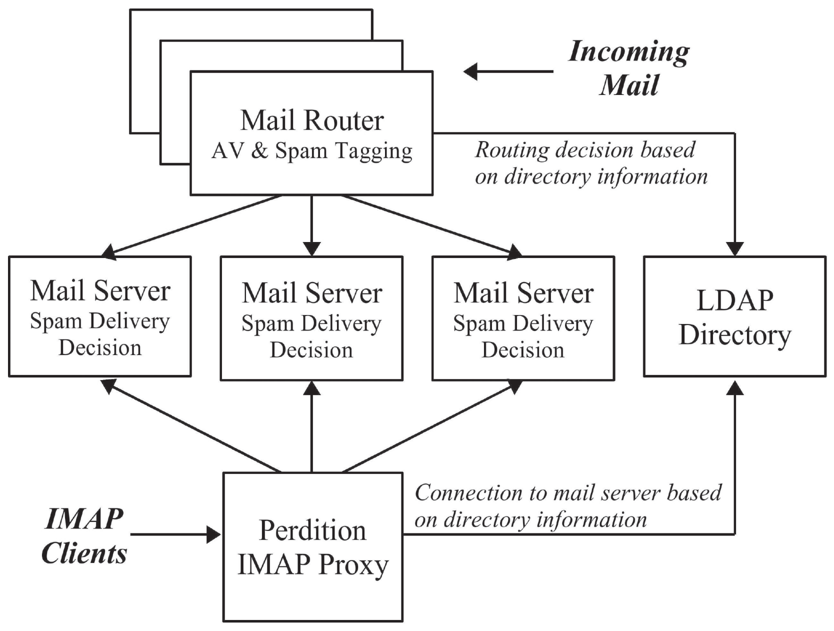


FIGURE 1: EXISTING MAIL INFRASTRUCTURE

All outgoing mail is handled by a single SMTP server running Sendmail on Linux. This portion of the mail system was unchanged throughout this entire process. To date, no spam, or even virus, filtering takes place on this server. All such filtering occurs only on mail delivery. (This is all that is necessary to protect our own users from spam and viruses.) It is likely that this will change in the future. Spam originating from our institution and other institutions within the North Dakota University System has caused no end of headaches for campus security officers who handle spam complaints and oversee the cleanup of compromised desktops, which are almost always the source of these problems.

The process of mail delivery begins with the mail routing servers that run Sendmail and MailScanner [2]. Mail routers are depicted in Figure 1 in a stacked configuration because, conceptually, there need only be one, but multiple configurations may be used to distribute the processing load of the incoming mail. At NDSU this is done by using multiple MX entries in our DNS [3]. One could also accomplish this by using a load balancer, such as the Linux Virtual Server [4].

Routing information used by the routing servers is stored in an LDAP directory using the standard Sendmail schema. MailScanner controls virus scanning and spam tagging. The delivery process works as follows:

A Sendmail daemon running on the incoming mail servers receives a connection from an MTA requesting that mail be delivered to a particular user or users. Sendmail looks up the address(es) in the LDAP directory to determine where the mail should be routed. Routing rules appear in the directory like this:

```
mailLocalAddress: Marc.Wallman@ndsu.edu
mailRoutingAddress: mwallman@imap3.ndsu.edu
```

Making a routing decision at the point of entry is very important. Spammers routinely use brute force as a method of delivering spam. They will crawl through a name dictionary constructing possible usernames with the hope of finding one that will be accepted by the targeted domain. If the incoming mail servers are relay only, they do not know what usernames are valid for hosts or domain they serve and must blindly accept all messages for these hosts and domains. Much of the spam coming from brute force ends up stuck in the queues. The destination hosts reject all the messages for unknown users and the spammers won't take the bounces back. Queues become bloated with accumulated bounces. When the server gives up on delivery for these bounces, new ones quickly take their place in the queues. Initially we had our incoming mail server set to relay and not route. It was not uncommon to have a total of 15,000 bounces stuck in our queues.

Messages that Sendmail accepts are delivered to a special queue directory that is processed by MailScanner, which in turn controls virus scanning and spam flagging. MailScanner is highly configurable and scalable. It uses one or more external virus scanners (there are many to choose from—we use McAfee [5]) to catch viruses and SpamAssassin to filter for spam). MailScanner is highly configurable and very efficient. Virus scanning and spam filtering happen in batch mode, not on a per-message basis. The system administrator may determine the batch size and scan interval.

MailScanner may make a decision on how to handle mail based on a message's spam score as rated by SpamAssassin. The score is assigned based on the application of many weighted spam tests. The point value of all tests that fail (e.g., a forged sending IP address) are added up and the message is given a total score. We chose to only tag mail with X headers. A decision about what to do with the mail is not made at this level, because it cannot be made on a per-user basis.

Messages were tagged to match the recommendations of our user community. Obvious and potential spam were each given their own tags:

```
X-NDUS-SpamFlag: Obvious Spam
X-NDUS-SpamFlag: Potential Spam
```

Mail given 8.0 or more points by SpamAssassin was tagged as obvious spam. Mail assigned between 5.0 and 8.0 points was tagged as potential spam. Mail scoring less than 5.0 was not tagged. The raw score is also embedded in the header. Category tags are added at this level for later use in deciding how to deliver mail based on an individual user's chosen level of service. Using category tags instead of a raw SpamAssassin score allows flexibility in how aggressive we wish to be in categorizing mail. (A small contingent of GroupWise users exist at NDSU. A special tag, `X-SpamFlag: Yes`, was also added to all messages that were considered spam, i.e., both obvious and potential spam. GroupWise is able to make use of this special tag with its client-side junk mail processing.)

Once MailScanner completes scanning and tagging, it re-queues the mail for final delivery by Sendmail. The final delivery point is the routing address specified in the mailRoutingAddress attribute LDAP directory. We happen to allow users to opt out of our mail solution altogether and have their mail routed to any address they choose. Common destinations are hotmail.com, yahoo.com, and gmail.com. Mail bound for these and other off-campus destinations is still tagged. If they choose, users may do client-side filtering based on the headers we add.

After going through the incoming mail router, messages are passed to their final delivery point: a server running the University of Washington IMAP daemon [6]. Here, procmail is invoked to look for the spam tags and make a delivery decision. Four different templates exist, corresponding to the four levels of service outlined above. When users select a level of service, the appropriate template is installed in the user's `.procmailrc` file in their home directory. Based on the recipe in `~/.procmailrc` and the categorization of the message (obvious spam, potential spam, or not spam), incoming mail is either delivered, quarantined, or deleted (i.e., delivered to `/dev/null`). The quarantine is simply an IMAP folder that exists in the user's email account. We have chosen to name this folder SPAM-Quarantine. Safe-sender and block-sender lists as described above are also encoded in procmail rule sets. The following is an example of a procmail recipe for level 2 (quarantine/quarantine):

```
INCLUDERC=.SafeSenderList
INCLUDERC=.BlockSenderList
```

```
:0 W
* ^X-NDUS-SpamFlag: Potential Spam
|/usr/local/sbin/dmail +SPAM-Quarantine
:0 W
* ^X-NDUS-SpamFlag: Obvious Spam
|/usr/local/sbin/dmail +SPAM-Quarantine
```

As indicated in this recipe, the safe-sender and block-sender lists are stored in separate recipe files. The following is an example of an entry from a block sender list:

```
:0:
* ^From:.*foo@example.com
/dev/null
```

Readers may learn more about procmail at http://www.procmail.org. The dmail mail delivery agent referenced above in the procmail recipes is bundled with the University of Washington IMAP daemon. It is used instead of the native delivery agent in procmail in order to allow us to use MBX-style indexed mailboxes. Documentation on dmail is included with the UW-IMAPd.

Mail retrieval is mediated by the Perdition Mail Retrieval Proxy [7]. Perdition mediates all communication with the back-end IMAP servers (it also supports POP, which we do not use). When a user initiates an IMAP login, Perdition takes the supplied username and looks in LDAP to determine the server on which the user's mailbox resides. All subsequent communication from the client is simply proxied over to the real mail server. Similarly, responses from the mail server are proxied back to the client. In this way, many mail servers may be used to deliver IMAP service, while the end user is blissfully unaware of this as they only connect to the proxy. Perdition is not a resource-intense application. A single 2-CPU Linux server with 1GB of RAM easily proxies all IMAP communication for our system. Typical weekday traffic is over 110,000 logins from just under 9,000 unique users. If there were ever a need, this portion of the system could easily be scaled by adding servers running Perdition in a load-balanced configuration.

The remaining portion of this system is the mechanism by which users enable/disable their spam filter and populate their safe- and block-sender lists. This was integrated with a preexisting Web application that allows users to enroll for new services and do some limited management of services they are subscribed to (e.g., users wishing to have their @ndsu.edu email forwarded to their Hotmail account would use this site to do so). The design of the user interface was taken almost verbatim from the recommendations of the user community. The presentation of the form that allows users to enable the various levels of spam filtering service described above is basically the same as what is shown in Table 1. Simple Web-based lists allow the population, or depopulation, of the safe and block lists. Submission of the spam filtering service level form or safe/block sender lists triggers a program to execute on the mail server that hosts the submitter's mail account: the appropriate procmail template is placed in the user's home directory, the SPAM-Quarantine folder is created if it does not exist and is added to the list of visible folders if it is not already there, and the safe/block lists are recreated with the appropriate values. Since all of these items represent files in the user's home directory, they are all susceptible to being deleted. The decision to have these Web forms recreate rather than update the files was made so that repair of a broken spam filter was something the end user could do. (The spam filter most commonly gets broken when users delete their SPAM-Quarantine folder.) This strategy results in many fewer support tickets being sent from help desk staff to system administrators.

Measures of Success

From the perspective of the system administrator, the successful implementation of this spam solution arises from two key sysadmin-friendly traits: maintainability and scalability.

The system's maintainability derives from a number of factors. First, it is modular. The first attempt at implementing a spam filtering solution co-located the mail accounts with the spam filter's intelligence. This created a functional dependency that impaired our ability to perform upgrades. SpamAssassin required frequent upgrades, whereas the IMAP daemon required few. Now these applications reside on separate systems and the use of the "potential" and "obvious" tags further the insulation of these applications by establishing a kind of API. The end result of the spam filtering system is only to assign category tags to pieces of email. The mail servers process spam simply by examining the tags contained within the email messages.

Second, the system is simple. The modularity not only removed functional dependency, it also removed complexity. Upgrades to the mail server may now be done largely without consideration for how spam is identified. Upgrades to the spam solution may be done largely without consideration for how the mail servers are configured. Another respect in which this system exhibits simplicity is in its functionality. Only four levels of service exist (five if you count the disabled state). All are clearly defined and sufficiently abstracted from the process of categorization to allow a great deal of flexibility with regard to how this categorization is accomplished. This flexibility should also be of benefit as new techniques become available to combat spam. There is a reasonable chance that the MailScanner/SpamAssassin solution could be swapped out with another technology at some point in the future if the need arose.

Third, upgrades to the incoming mail servers which tag spam do not require downtime. As previously mentioned, multiple mail routers may exist and they are redundant. More may be added or existing systems may be removed. As long as sufficient throughput exists to run in a degraded mode—that is, with at least one routing server offline—it is possible to take systems out of production for upgrade and place them back when the upgrade is complete. Being able to do maintenance on production systems is a very good thing for system administrators. There are enough other opportunities for work in the early morning hours and on weekends.

In addition to maintainability, this system is also scalable. If the load processing for incoming spam becomes too great for existing routing servers, just add more. A side benefit is that more servers means not only increased throughput but more redundancy. At NDSU, we currently have three mail routers in production. We almost never see high load on all three servers simultaneously. In those rare instances that we do get alerts about load across all systems, they have invariably recovered by the time the situation is investigated. All other parts of this system are scalable too. Perdition mail proxies and outgoing mail servers may be added in a load-balanced configuration. IMAP mail servers may also be added, with the caveat that user accounts must be migrated; methods exist, however, for doing this with minimal disruption to the running systems. We have developed code that automatically coordinates the transfer of users' mail accounts between back-end mail servers with the population of updated mail routing information in our LDAP directory.

With all this said, how well do we do at catching spam? How well can we do? Figure 2 shows the amount of spam identified per week as a percentage of our total email.

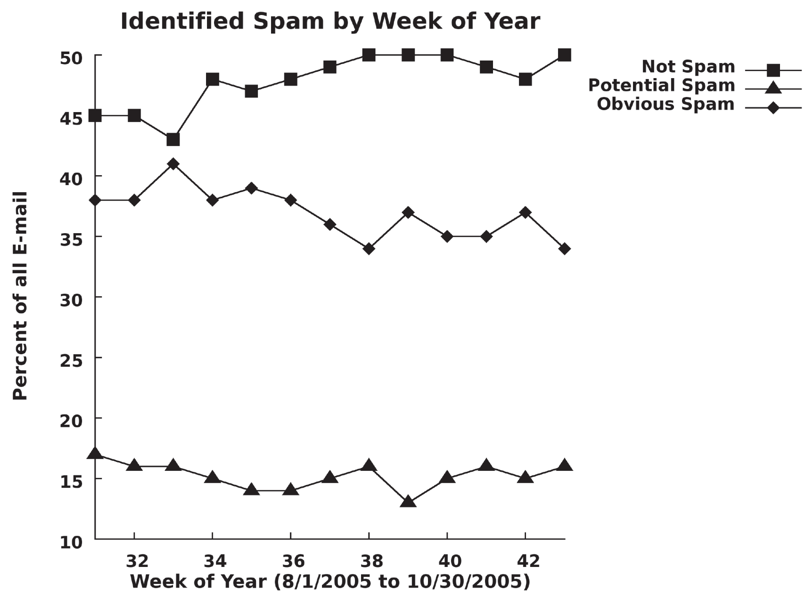


FIGURE 2: IDENTIFIED SPAM BY WEEK

The data depicted here is for the week beginning August 1, 2005 (week 31), through the week beginning October 24, 2005 (week 43). The graph has been aggregated by week in order to make it more readable. Legitimate mail decreases over the weekend, while spam seems to remain constant. As the graph shows, we routinely identify just over 50% of our incoming mail as spam. If measured against some reports, the spam filter does not appear to be very effective. Symantec recently reported that 61% of all email is spam [8]. Others have reported the percentage of spam to be much higher. User feedback has been quite positive (see below); so what might account for this apparent discrepancy in the numbers? We may simply be below average in the amount of spam that we receive, or the positive user response may be mainly due to the relative improvement offered by the new system. They may be happy that they are getting less spam, but we could be doing better. Third, the data sample shown in Figure 2 shows a slight decrease in the effectiveness of our spam filter since the beginning of August. SpamAssassin 3.1.0 was released on September 14, 2005, and, as of this writing, has not been put into production here at NDSU. SpamAssassin's effectiveness decreases with age. Spammers actively work to find ways to get their messages through without being identified. This implies that the numbers for September 2005 represent a low point of effectiveness. It is likely that the effectiveness of this solution will continue to decrease until SpamAssassin is upgraded. Last, we have not been overly aggressive in our attempts to tag spam. We have not used SpamAssassin's rules du jour, attempted to develop our own rule sets, or adjusted the thresholds at which we tag potential and obvious spam. It is likely we could do far better, but our users have been satisfied, so we have focused on other things.

During the summer of 2005 we had about 50 testers who used their accounts to quarantine all mail tagged as obvious or potential spam. Feedback from this group was VERY positive. All were extremely impressed at how well the system functioned. Testers also reported that there were almost no pieces of legitimate mail that were being tagged as spam, while a small amount of difficult-to-catch spam did get through. We could be more aggressive in our filtering.

The system was officially launched on September 19, 2005, supported by a considerable amount of PR. Oddly, far fewer people than expected signed up. Subscription rates jumped to about 7% after the initial launch, with a very minor growth trend. Feedback from the user community continues to be good. There has been quite a bit of unsolicited praise, which is very unusual. Unsolicited complaints or silence is more in line with the norm. Does most of our spam go to a small portion of the user community, leaving the majority unaffected? Are more people using client-side tools for addressing spam than we suspect? At this point, we do not know.

Future Directions

Two main questions confront us with this system. The first has already been mentioned: Why are there so few subscribers? The second is: Can we keep up with the spammers? Spam filtering differs greatly from Web or calendar service, for example, in that people are actively attempting to make spam filters obsolete. I believe this system has been built with sufficient flexibility so that it will be able to adapt, but that remains to be seen.

REFERENCES

[1] SpamAssassin, from the Apache Software Foundation: http://spamassassin.apache.org/.

[2] http://www.sng.ecs.soton.ac.uk/mailscanner/.

[3] For more information on how MX records work, see section 5 of RFC 2821 (Simple Mail Transfer Protocol).

[4] http://www.linuxvirtualserver.org/.

[5] The North Dakota University System has a licensing agreement with McAfee, which made this antivirus scanner an easy choice. ClamAV represents an open source alternative to commercial antivirus software: http://www.clamav.net/.

[6] http://www.washington.edu/imap/.

[7] http://www.vergenet.net/linux/perdition/.

[8] "Symantec Internet Security Threat Report Identifies Shift Toward Focused Attacks on Desktops," September 16, 2005: http://www.symantec.com/press/2005/n050919a.html.

TOM LIMONCELLI

delegating to the Web

Tom Limoncelli is the author of *Time Management for System Administrators* from O'Reilly and is co-author with Christine Hogan of *The Practice of System and Network Administration*, which received the 2005 SAGE Outstanding Achievement Award.

tallogin@everythingsysadmin.com

I WAS TOLD A BOOK ON TIME MANAGEMENT for system administrators was impossible, since a lot of time management is about delegation and "that's impossible for a system administrator." Oh, ye of little faith.

One of the problems with being a system administrator is that we enjoy our jobs so much that we want to do everything ourselves. (It can't possibly be that we're control freaks.) We want to do everything our way. Because of this, we rarely learn to delegate. It helps that there is rarely someone to delegate to.

The kind of delegation we are capable of usually consists of using pre-written software rather than writing it ourselves. We find the right open source or commercial package for the task at hand and spend our time doing "integration and deployment" rather than writing code. It's easier to buy backup software than to write a new program from scratch. Integration is more powerful than invention.

Lately I've been using a lot of Web-based applications to make my life easier. There's no software to install—it's all at their Web farm. Usually there is a "dashboard" that I can log into to control and configure the service I'm receiving. The data is all kept on their servers, which brings up a huge number of confidentiality, privacy, and reliability problems. However, I have been happily surprised to find that many of these services, though less than my ideal, provide more security than I could provide myself. So, they win.

The real difficulty for me has been control or fear of letting go. It's difficult to get out of the habit of trying to do everything yourself. Looking back on all the times I've moved to a hosted solution, I've never regretted it.

Here are some examples.

Email and File Sharing

This first example came out of necessity rather than planning. I run a lot of email lists (first Majordomo, now Mailman) for nonprofits that I work with. The burden on my server was getting to be very heavy. It was a victim of its own success. Soon they wanted more features, like a way to share files. I was dealing with accounts, security issues, training issues, and so on, and didn't have the time to handle all the requests. Then I discovered Yahoo! Groups (http://www.yahoogroups.com). They give you a free email list, which has an associated file storage area, calendar, and all sorts of features. It was more efficient to spend my time training people how to use those services rather than run them myself. Teaching someone to use a Web form–based file upload is a lot easier than installing an FTP client, no matter what OS you use. In addition, the training was leveraged over many organizations. More and more people already know how to use Yahoo! Groups, and once a person has used Yahoo! Groups for one non-profit, they are able to use their knowledge with other organizations.

I found the following formula useful for most of the small nonprofits that I volunteer with: Each organization usually needs two groups: one named after their organization, which includes all their members, and another just for their board members and/or volunteers. If the group is called "Save the Foo," we might have STF-announce@yahoogroups.com as the inclusive group and STF-workers@yahoogroups.com as the private group. The two file areas give the organization the ability to have private and public documents. "Organizational memory" is helped by the fact that documents are stored in and accessed from a central location. No more reinventing a form because the only copy is on the ex-president's PC.

Obviously, there are privacy concerns. Two organizations that I help can't use Yahoo! because of the terms of service. However, it's a trade-off. At least there is a legally binding terms-of-service document that can be evaluated. The only assurances I could give people were, "It's as private as it can be" and "If I'm around, I'll try to fix it."

Sales Management

The next example is sales management ("customer relationship management," or CRM) software. After battling sales management products such as ACT! and Goldmine for nearly a year, my then CEO proposed we look at a hosted solution. I was shocked. How could we put important sales tracking data on someone else's server? In this case, it was off to the legal department for analysis while the CEO and upper management evaluated the risks. I was surprised when they decided to take the risk of a hosted solution. Privately, however, I sighed with relief, because the IT team was overcommitted with other projects. I couldn't even imagine having the time to set up the data backup/recovery system for a full-fledged CRM system, let alone the software itself. It was great to see this entire class of problems disappear from my plate. The sales group became self-supporting. I provided a Web browser; everything else was dependent on the ability to dial the vendor's toll-free number. If this sounds appealing to you, investigate Salesforce.com, Siebel's CRM OnDemand, or the up-and-coming SugarCRM. A Web search will turn up dozens more.

Email Security

In 2004 I realized that I was spending about four hours a week (half a day) dealing with the anti-spam/antivirus solution(s) we were using, and yet people didn't have cool features like a private dashboard they could log into to manage their quarantined emails. Beyond the usual daily issues, I was constantly evaluating new software, taking the system down to do upgrades (I'm sure this annoyed my users no end), and chasing RBL lists. Being able to eliminate these tasks would save me 10% of my week. (I pondered that if I could find nine more like it, I would never have to work again.) There is a lot of competition in this area, so the hosted solutions are very powerful and very feature-rich. There are many services that do anti-spam/antivirus on email. You simply point your DNS MX records at their servers and they do the work. Each user gets an account where they can log in to review their quarantine. Spam is no longer my problem, and the reduction in viruses has gained me a few hours each week, too. If we're ever unhappy, we can change to their competition very easily. Some products to look at include MessageLabs, Postini, and McAfee's Secure Messaging Service. The "enterprise" edition of these products usually includes outbound email queueing, which means I was able to eliminate most of the load of my outbound email queuing infrastructure. (Hosted email security might have some of the most obvious privacy issues, though if privacy is your concern, why are you using unencrypted email on the Internet?)

Email

Speaking of email, like many USENIX members I have long struggled to avoid MS Exchange. I now have an alternative answer to the new CEO who arrives and asks what it would take to bring Exchange to the company. I simply say, "$15 per user per month; I can have it set up for you in a month." That's about the going rate for hosted MS Exchange. With the huge competition in that area, hosted Exchange offers a very feature-rich service, including calendars. Compared to the cost of sending me to Exchange training, plus buying the hardware and data recovery costs, it's a bargain. A Web search for "hosted MS Exchange" returns so many results it's almost dizzying. DNS service providers like Register.com and NetSol provide this service, but the one with the coolest name has got to be ElephantOutlook. As a disclaimer, I should point out that I've never actually used any of these services. While $15 per user per month sounds like a bargain to anyone who knows how difficult it is to run any kind of email service, a CEO typically thinks that email is free as the wind. Hearing $15 per user per month scares them away from any future thoughts of switching out of our legacy system. Dance, puppet! Dance!

Antivirus

While most antivirus products for Windows include some kind of dashboard application that lets you see who is out of date, this ties up a server. McAfee's Managed VirusScan provides the same service in a hosted product. Since the software and signature updates come from their hosted service, finally road warriors get updates on a timely basis.

DNS, Domains, and Web Sites

Running my own DNS servers was great when I was learning DNS. I must have homed 30 domains just for friends. For external DNS servers (on the public Internet), I now let my DNS register do it for small domains (the more expensive DNS registrars do it "for free," i.e., Register.com), or use various hosted DNS secondary services like BackupDNS, Ultra DNS, or any of the dozens that show up in a search engine. Their prices range from free to extremely costly.

Running a Web server used to be exciting and new. Now for static Web content you can get a lot better service for $10/month just about anywhere.

Membership and Registration

Friends who run conventions asked me to set up a Web-based registration system. While getting ready to learn the PayPal APIs, I found that Mollyguard had already built a better system than I would ever have time to create. Best of all, it's so easy to use that the nonprofits I used to help with this kind of thing are now completely self-sufficient.

Nonprofits I work with also have huge membership database problems, especially when one membership chair leaves and another comes on board. Most can't process online renewals or credit cards. Lately, I've been recommending companies like 123Signup, which do all those functions for them.

Others

Using hosted applications is great but how far can this go? I've seen payroll departments eliminate the need for servers by using Paychex. TriNet gives you an entire Human Resources division, including benefits and payroll, all from a Web browser.

There are plenty of hosted solutions for SLA monitoring, from simple ping systems to systems that are commensurate with Nagios or BigBrother. Other system administration tools, such as request tracking and wiki, are products I look forward to.

Where is all this leading? Will I be out of a job? Obviously not. All of these products simply help me leverage my time for more interesting pursuits, such as directly interfacing with customers or helping my company improve business processes through better uses of IT. While hosted solutions work well for small to medium businesses, they are not yet appropriate for large companies. I couldn't imagine a giant like IBM or McDonald's trusting all their email to flow through someone else's process farm or being able to move their highly specialized sales processes to a hosted provider.

When you outsource a function to another company, your job becomes quality assurance. Moving to a hosted anti-spam solution doesn't mean you can forget about spam, it just means that you have to monitor the quality of the service you are receiving. You need to maintain a relationship with the vendor so they understand your changing needs. You need to make sure they maintain the quality of service they promised during the sales pitch.

Best of all, the more services I move to hosted solutions, the more time I have to search for new hosted services! Joy!

DUSTIN PURYEAR

consulting for fun and profit

Dustin Puryear works with businesses requiring UNIX and Windows expertise to get the best return from their technology investments. Corporate projects range from the design and management of email solutions to integrating Directory servers and applications. In addition, he is the author of *Integrate Linux Solutions into Your Windows Network* and *Best Practices for UNIX and Linux Management.*

dustin@puryear-it.com

THERE HAS BEEN A LOT OF TALK recently in various USENIX/SAGE venues on how to become an independent consultant. (Well, let's be honest: when hasn't there been a lot of talk on this topic?) Most of the questions are concerned not only with the business side of consulting (e.g., taxes, whether to incorporate, insurance for the self-employed), but also with the nuts and bolts of how to make it as an independent consultant.

So, to help prepare you, I'll cover what I've learned in my years as an independent consultant. Of course, it's reasonable to wonder how I'm qualified to offer advice on this topic. My only credentials are that, well, I'm successful. Outside of that, feel free to take my advice with a grain of salt.

Keep in mind, though, that this is serious business. There is a lot of money to be made in offering expertise to those in need, but there is also a lot of heartache and headache for those who don't properly prepare themselves, their families, and their bank accounts for the adjustment. Take notes, keep your chin up, and welcome to the world of consulting.

Consulting Defined, Kind Of

You'll notice that I often say "independent consultant" rather than simply saying "consultant." So what is the difference between the two?[1]

Generally speaking, a consultant works with a consulting firm. These firms are commonly structured in one of two ways. First, the firm is structured around several or more partners, where each partner usually has equal footing within the company. Second, a firm may be more hierarchical in nature, where partners are at the top of the pyramid, and the lower ranks are filled with many relatively low-paid worker bees who are billed out as "consultants." There are many people who happily work in both types of environments because they enjoy the project-to-project lifestyle of a consultant but crave the stability of working within a firm.

Independent consultants are different. They work for themselves. An independent consultant must organize his or her own business entity, find clients, decide how accounting should be done, speak directly with lawyers on business and contract issues, and, by and large, be anything and everything that is needed.

1. Many people are also confused by the difference between a contractor and a consultant. A contractor almost always works on assignment through a placement firm and is assigned a specific task in a larger project, such as programming the interface to a new program or maybe installing Linux servers in a database roll-out project. Consultants often work at a higher level (e.g., designing a new network architecture and then implementing it), but not always. Admittedly, the line between consulting and contracting tends to blur at times.

Of the two types of consulting, independent consulting offers the greatest amount of stress and the greatest reward. You can go very far if you are bright, technically capable, and have good business sense. Or you can go bust.

Moonlighting

If you make the decision to take the leap into consulting, you need to time it properly. In my opinion, the best way to start is with moonlighting. To "moonlight" means that you work on outside projects after your day job. That is, you have consulting clients, but you still have a part-time or full-time job with an employer. This is an excellent way to learn the business. You have the stability of being employed while learning the ins and outs of consulting, such as billing, client relationships, and time management.

Moonlighting is no panacea, however. First, many employers have strict rules against it. This is especially true if your employer is a consulting firm. Naturally, they don't want their employees competing with them for clients. So be sure to read your employment agreement. Second, many clients don't want a consultant who is moonlighting, and this is understandable. When you moonlight you are telling the client, "My first priority is my real job. I'll work with you when I have time." But if you can swing it, by all means moonlight for at least a year or more before taking the dive.

Another benefit of moonlighting is that it develops a client base for you to use after you quit your day job. In most cases, if you don't have a well-funded bank account from the start, you are going to fail if you start your business without clients. It takes years to build a stable, well-paying client list, so don't expect to become an overnight success.

Organizing Your Business

Whether moonlighting or going full-time, it's important that you consider how you want to organize your business. However, keep in mind this point: the business of being in business is making money, not organizing or running your business. Make a few key decisions early on, implement them, and get on with building your client list and billing out hours.

With that said, let's talk about an important decision: business type. Your state of residence is going to define the various business types available to you, and you must pick the one that has the most advantages for you. Generally, the types available are the sole proprietorship, partnership (which we won't discuss), corporation, and LLC.

The "default" business type is the sole proprietorship. In the U.S., it's very easy to create a business. You say, "I'm in business," you get an occupational license, and then, basically, you automatically become a sole-proprietor. The advantages of being a sole proprietor are that taxes are relatively easy to prepare, and there is very little paperwork to maintain. The disadvantages are that you may be missing out on some tax advantages of other business types and you maintain full liability.

Corporations are more complex than sole proprietorships but have some advantages. The main advantage is a transfer of liability from you to the corporation. The main disadvantage is more complicated tax filings, much more onerous requirements for record-keeping, and an overall increase in the complexity of how you handle your business.

A Limited Liability Company (LLC) is often an excellent compromise between a sole proprietorship and a corporation. An LLC that is structured as a "pass-through" is basically a sole-proprietorship with limited liability. The taxes tend to be the same, and there is very little additional paperwork. A more advanced form of the LLC mimics a corporation, including the fact that you become an employee of the LLC. A major advantage of this type of LLC is that you can write off what is known as "self-employment tax." With a sole proprietorship and a "pass-through" LLC, you have to pay the full 15% Social Security tax, instead of an employee's 7.5% contribution. In the second form of an LLC, you still pay the full 15%, but the LLC gets to write off its portion of the social security payment, lowering overall tax liability. Quite often this write-off can justify the additional bookkeeping required for the second form of the LLC.

I suggest that you begin your business as a "pass-through" LLC. After your business has become successful, and you have a large enough income ($80,000 or more is a good rough guess), convert it to the second form of an LLC. Naturally, you'll want to discuss this with both a lawyer and a CPA.

Speaking of lawyers and CPAs . . .

You Aren't That Smart

I'm amazed by consultants who spend hours and hours working on their accounting and trying to figure out contracts, when they should be spending their time working on billable hours. The math is simple: for every hour that you spend working on your business, you lose one hour of billable time. Add to that the fact that you are less productive than a specialist (just as your clients aren't as productive doing the services that you offer, otherwise they wouldn't need you), and you have a no-win situation.

What I've found that works—and this seems to be the pattern of many successful consultants that I know—is to maintain your own day-to-day bookkeeping (e.g., putting receipts into your accounting system), but to leave major work to the professionals. This includes quarterly and year-end taxes and all legal work. Do not try to write your own contracts. Let a lawyer do it for you. It's not cheap, but then again neither is your time.

You Aren't That Rich

Billing is critical to your success. Or, to be more accurate, cash flow. Every consultant, especially when they first start, has cash flow problems. And normally it has nothing to do with how much work you have. It's a problem in billing. The Golden Rule of billing is to bill early and to bill often.

By billing early, I mean that you should not wait until a project is complete to send a bill. Let the client know that you will be billing either monthly (or perhaps bi-weekly—this is the idea of "billing often") or on a percentage-completed basis. If nothing else, if you wait until the project is complete, the client has no incentive to pay you on time.

As already mentioned, every consultant falls into the trap of not billing properly. Time and again I have let my billable hours stack up until I finally get around to sending invoices. And after sending invoices, those billable hours are merely converted into Accounts Receivable (A/R), which is to say "virtual money." Sure, clients owe me money, but that doesn't pay the lease on my office. After a few years you develop a sixth sense for which clients will pay late, how far you can let your accounts fall, and other billing-related issues. But the best bet is to take no risk: bill regularly and monitor your cash flow. Remember, it doesn't matter what you have in your bank account today, it's what you will have in 30, 60, and 90 days that matters.

Be Good, Do Good

Being an independent consultant is entirely and without question a reputation-based occupation. I am entirely referral-based. You will be too, if you are good. Prove to your clients that you are the best person for the job, and they will recommend you no end. That's just how it works.

Now, this isn't to say that you won't have failed projects. This happens regardless of whether you are a consultant, a project sponsor, or even a normal employee. The difference between a good consultant and a bad consultant is that a good consultant knows how to pick up the pieces, how to properly communicate what happened, and, hopefully, how to maintain the client's trust.

Also, keep in mind that other consultants are watching you. While being an independent consultant means that you work by yourself, it doesn't mean you live in a self-contained microcosm. Consultants working for your clients will notice the work that you do and how you communicate with them and with the client. If you earn the respect of other consultants, they will refer work to you. If you earn their ire, then they will (rightfully) warn potential clients about you.

Network, Network, Network

Keeping in mind that other consultants are watching you, also keep an eye out for other consultants. Small consulting companies almost never compete with one another. We help each other out. It's kind of weird and cool at the same time. I can't tell you how many projects I've worked on that were brought to me by another consultant. You need to establish expertise and a good relationship for this to really kick in. For example, I haven't worked with many people in USENIX/SAGE and sage-members@sage.org, but I know who is out there and we say hi off-list every now and then. One day I'll see a project and forward it to someone. One day someone may do the same for me. That's the business.

This brings us to the larger issue of networking. Networking is absolutely crucial to a consultant. It's how you develop a lot of new business. Having a Web site won't do it, and neither will a billboard or, heaven forbid, the yellow pages. Your success depends almost exclusively on handshakes and the respect of your peers and clients. So, be sure to meet fellow consultants and prospects at events such as user group meetings, technical mailing lists, and conferences, as well as at client sites. Smile, have a business card ready, be prepared to describe your business in only a few seconds, and follow up with new contacts with an email or letter a few days afterwards. (Of course, don't badger people, either.)

Identify Your Expertise

Become an expert in one or two areas. In other words, don't try to be a jack-of-all-trades. The reason is simple: being an expert means that you can charge premium rates. Being a jack-of-all-trades works at first, and chances are that you won't have much of a choice when you first start your business, but being a jack-of-all-trades will keep your billing rate low. So, try to specialize.

On the other hand, even I still do jack-of-all-trades work at times, if for no other reason than that it leads to more specialized projects with clients. There is no right or wrong way to figure this out. Over time you will develop a feel for the exact kind of work at which you excel and for which you get paid the most. As you identify these areas, start to hone those skills and present yourself to clients as a specialist in that area.

Obviously, you should never oversell yourself. If you are not an expert in a given area, don't claim to be one. The client will eventually find out the truth. That's a given. If you don't present yourself as an expert and the client still takes you on as a consultant, consider tht an opportunity to increase your skills in that area and to prove to the client that they made the right choice. (Alternatively, refer the work to an associate who is an expert, and so establish a relationship that may lead to referrals for you later on.)

On a side note, once you become expert and are well established, begin charging premium rates in your area. If you bill average rates, then, by golly, you must be an average consultant.

Identify Your Clients

At first you are going to take whatever clients you can. No matter what you say, every consultant does this. After a year or two, though, you are going to learn that some clients are just not good for you or you for them. Over time you will let certain clients go (consultants can "fire" clients if the relationship is bad enough, but you should focus on recommending consultants with whom those clients will, you feel, have a successful working relationship). But the better solution is to learn how to avoid clients who aren't right for you. It's hard to say "thanks, but no thanks" at first, but over time you will learn that it's the right course of action in many situations.

Learn from every relationship. Even when you have a great relationship with a client, there will be bumps in the road. Learn from these situations.

Learn How to Communicate

Finally, learn how to communicate with people. All too often, technically adept people have shortcomings when it comes to communicating effectively with nontechnical clients. But a lot of the work that you get will be commissioned by upper management, and these people do not tend to be technically savvy. If you can't describe your solution to them, then don't expect to get the project.

Furthermore, it's important to be an effective communicator throughout a project. A little-known career-saver is being able to convey why a project is having problems or delayed, and to work out a solution with the client. Do not try to hide important information from a client. They will find out eventually.

As a consultant, you must be a better communicator than a typical employee or even a contractor usually is. It's your job to describe both problems and solutions. Additionally, documentation is going to be very important to how clients view your work.

To improve your communication skills, try to write for industry journals (e.g., *;login:*), present at local user groups, and speak at conferences. This will not only improve how you share concepts, but it will also increase your exposure as a consultant. Remember, you need to establish yourself as an expert.

Conclusion

I hope these tips will prove useful to you. This article addressed a wide range of issues that affect independent consultants, but, as always, there is more information out there. Be sure to find these other resources and learn what you can, because running your own business, while fun and exciting, comes with its own set of headaches. In the end, however, most people find those headaches well worth it.

And, finally, smile. Life is short. Don't get too caught up in work.

Thanks to USENIX & SAGE Supporting Members

Addison-Wesley Professional/
Prentice Hall Professional

AMD

Asian Development Bank

Cambridge Computer Services, Inc.

EAGLE Software, Inc.

Electronic Frontier Foundation

Eli Research

FOTO SEARCH Stock Footage and
Stock Photography

GroundWork Open Source Solutions

Hewlett-Packard

IBM

Intel

Interhack

The Measurement Factory

Microsoft Research

MSB Associates

NetApp

Oracle

OSDL

Perfect Order

Raytheon

Ripe NCC

Sendmail, Inc.

Splunk

Sun Microsystems, Inc.

Taos

Tellme Networks

UUNET Technologies, Inc.

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

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

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

THOMAS SLUYTER

keeping track of time

In daily life Thomas (a.k.a. Cailin) works for Snow, a UNIX consultancy bureau in the Netherlands. He took his first steps as a junior UNIX sysadmin in the year 2000. Thomas part-times as an Apple Macintosh evangelist and as a board member of the Dutch J-Pop Foundation.

tsluyter@kilala.nl

WAY BACK IN 1999, WHEN I STARTED my second internship, I was told to do something I had never done before: create and maintain a plan of my activities.

At the time this seemed like a horribly complex thing to do, but my supervisor was adamant. He did not want me to shift one bit of work before I had taken a stab at a rough plan. So I twiddled in Word and I fumbled in Excel and finally, an hour or two later, I had finished my first project plan ever. And there was much rejoicing! Well, not really, but I felt that a Monty Python reference would be welcome right about now.

So now it's six years later and I still benefit from the teachings of my past mentor. However, I see people around me who appear to have trouble keeping track of all of their work. Which is exactly why I was originally asked to write this article.

I have always worked in large corporate environments with several layers of management between the deities and me, which always seems to obfuscate matters needlessly. However, the ideas outlined in the next few paragraphs will be applicable to anyone in any situation.

Juggling Egg Shells

"*They [hackers] tend to be careful and orderly in their intellectual lives and chaotic elsewhere. Their code will be beautiful, even if their desks are buried in 3 feet of crap.*"

—*The New Hacker's Dictionary*

I realize that keeping a plan is definitely not one of the favorite activities for most people in IT; they seem to abhor the whole task, or fail to see its importance. Also, most are of the opinion that they don't have enough time as it is and that there is absolutely no way that they can fit in the upkeep of a personal plan.

Now here's a little secret: the one thing that can help you keep your workload in check is a plan. By keeping a record of all of your projects and other activities, you can show management how heavily you're loaded and when you will be available for additional duties. By providing management with these details, you are allowing them to make decisions like lowering your workload or adding more people to the workforce.

Personal Time vs. Project Time

A personal plan is what dictates your day-to-day activities. You use it to keep track of meetings, miscellaneous smaller tasks, and time slots that you have reserved for projects. You could say that it's your daily calendar, and most people will actually use one (a calendar, that is) for this task. In daily life your colleagues and supervisor can use your personal plan to see when you're available for new tasks.

A project plan, on the other hand, is an elaborate schedule that dictates the flow of a large project. Each detail will be described meticulously and will receive its own time slot. Depending on the structure of your organization, such a plan will be drafted either by you or by project managers who have been specifically hired for that task.

Tools of the Trade

"*Life is what happens to you when you're making other plans.*"

—John Lennon

KEEPING YOUR PERSONAL CALENDAR

I think it's safe to assume that everyone has the basic tools that are needed to keep track of their personal plan. Just about every workstation comes with at least some form of calendar software, which will be more or less suitable.

Microsoft Outlook and Exchange come with both a pretty elaborate calendar and a To Do list. These can share information transparently, so you can easily assign a task a slot in your personal plan. Each event in your calendar can be opened up to add very detailed information regarding each task. And you and your colleagues can give each other access to your calendars if your organization has a central Exchange server at its disposal.

One of the downsides to Exchange is that it isn't very easy to keep track of your spent hours in a transparent manner. It allows you to create a second calendar in a separate window, but that doesn't make for easy comparison. You could also try to double-book your whole calendar for this purpose, but that would get downright messy.

Looking at the other camp, all Apple Macintosh systems come supplied with the iCal application. It is not as comprehensive as the calendar functions of Exchange, but it is definitely workable. iCal comes with most of the features you would expect, like a To Do list and the possibility of sharing your calendar with your colleagues. However, this requires that you set up either a .Mac account or a local WEBDAV server.

One of the nice things about iCal is that it allows you to keep multiple calendars in one window, thus making it easier to keep track of time spent on projects. I use a green iCal calendar to contain all the events I schedule and a purple one to show how I really spent my time.

Finally, I am told that Mozilla's Sunbird software also comes with a satisfactory calendar. So that could be a nice alternative for those wishing to stick to Linux, or who just have a dislike of the previously mentioned applications.

KEEPING TRACK OF SPENT TIME

It's one thing to enter all of your planned activities into your calendar and another thing entirely to keep track of how you actually spent your time. Keeping tabs on how you spend your days gives you the following advantages:

- A way to report your progress to management
- A clear view of which activities are slipping in your schedule
- A clear view of which work needs to get rescheduled or even reassigned to somebody else

For some reason, however, there aren't any tools available that focus on this task, or at least I haven't been able to find them. Of course there are CRM tools that allow a person to keep track of time spent on different customers, but invariably these tools don't combine this functionality with the planning possibilities I described earlier.

As mentioned, it's perfectly possible to cram the time you spend on tasks into the calendar used for your personal plan, but that usually gets a bit messy (unless you use iCal). Also, I haven't found any way to create reports from these calendar tools that allow you to compare time planned against time spent. So for now the best way to create a management-friendly report is still to muck about in your favorite spreadsheet program.

REGARDING PROJECT PLANNING TOOLS

Most projects are of a much grander scale than your average work week. There are multiple people to keep track of, and each person gets assigned a number of tasks (which, in turn, get divided into subtasks, and so on). You can imagine that a simple personal calendar will not do.

That's why there is specialized software like Microsoft Project for Windows or PMX for OS X. Tools like these allow you to divide a project into atomic tasks. You can assign multiple resources to each task, and all tasks can be interlinked to form dependencies and such. Most tools provide professional functions like Gantt and PERT charts.

Making Guesstimates

In the next few sections I will ask you to estimate the time a certain task will take. Often sysadmins will be much too optimistic in their estimates, figuring that "it will take a few hours of tinkering." And it's just that kind of mindset that is detrimental to a good plan.

When making a guesstimate regarding such a time frame, clearly visualize all the steps that come with the task at hand. Imagine how much time you would spend on each step, in real life. Keep in mind that computers may choose not to cooperate, that colleagues may be unavailable at times, and that you may actually run into some difficulty while performing each step.

Do you have a good idea of how long the task will take? Good! Now double that amount and put that figure up in your plan. Seriously. One colleague recounts people who multiply their original estimates by pi and still find that their guesstimates are wrong.

One simple rule applies: it is better to arrange for a lot of additional time than it is to scramble to make ends meet.

Taking the Plunge

"*It must be Thursday. . . . I never could get the hang of Thursdays.*"
—Douglas Adams, *The Hitchhiker's Guide to the Galaxy*

Every beginning is difficult, and this one will be no exception. Your first task will be to gather all the little tidbits that make up your day and then to bring order to the chaos. Here are the steps you will be going through.

Make a list of everything you have been doing, are doing right now, and will need to do soon. Keep things on a general level.

Divide your list into two categories: projects and tasks. In most cases the difference will be that projects are things that need to be tackled in a structural manner that will take a few weeks to finish, whereas tasks can be handled quite easily.

Take your list of tasks and break them down into "genres." Exemplary genres from my plan are "security," "server improvements," and "monitoring wish list." The categorized list you've made will be your To Do list. Enter it into your calendar software.

For each task decide when it needs to be done and make a guesstimate of the required time. Start assigning time slots in your calendar to the execution of these activities. I usually divide my days into two parts, each of which gets completely dedicated to one activity. Be sure that you leave plenty of room in your calendar for your projects. Also leave some empty spots to allow for unforeseen circumstances.

Now proceed with the next paragraph to sort out your projects.

The Big Stuff: Handling Projects

"*Once I broke my problems into small pieces I was able to carry them, just like those acorns: one at a time. . . . Be like the squirrel!*"
—The White Stripes, "Little Acorns"

For each of your projects go through the following loop:

- Write a short project overview. What is it that needs to be done? When does it need to be done? Who are you doing it for? Who is helping you out?
- Make a basic timeline that tells you which milestones need to be reached in order to attain your goal (Fig. 1.1). For example, if the goal is to have all your servers backed up to tape, exemplary milestones could be: "Select appropriate software/hardware solution," "Acquire software/hardware solution," "Build basic infrastructure," and "Implement backup solution." For each milestone, decide when it needs to be reached.
- Work out each defined milestone (Fig. 1.2): which granular tasks are parts of the greater whole? For instance, the phase "Select appropriate software/hardware solution" will include tasks such as "Inventory of available software/hardware," "Initial selection of solution," "Testing of initially selected solution," and so on.
- Decide how much time will be needed to perform each of these atomic tasks (Fig. 1.3). Use the tips regarding guesstimates to decide on the proper figures.
- Put all the tasks into the timeline. Put them in chronological order and include the time you've estimated for each task (Fig. 1.4–5). You've now built a basic Gantt chart.

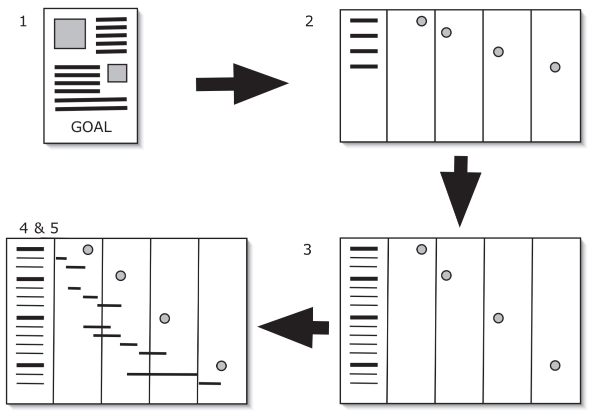


FIGURE 1: STEPS IN SETTING UP A PROJECT PLAN

Once you are done, go over the whole project plan and verify that, given the estimated time for each task, you can still make it on time. Discuss your findings with your management so that they know what you are up to and what they can expect from the project in the future.

Inevitable, Like Taxes and Death

"*Hackers are often monumentally disorganized and sloppy about dealing with the physical world. . . . [Thus] minor maintenance tasks get deferred indefinitely.*"

—*The New Hacker's Dictionary*

One of the vitally important facts about planning is that it's not a goal, but an ongoing process. Now that you have made your initial plan, you're going to have to perform upkeep. *Ad infinitum*. The point is that things change, and there's no changing that!

Projects fall behind schedule for many different reasons. Vendors may not deliver on time, colleagues may fail to keep their promises, and even you yourself may err at times. Maybe your original plan was too tight, or maybe a task is a lot more complicated than it seemed at first. All in all, your plan will need to be shifted. Depending on the project, it is wise to revisit your plan at least once a week. Mark any finished tasks as such and note any delays. Not only will this help you in your daily work, but it will also give management a good idea about the overall progress of your projects.

The same goes for your personal time. Projects need rescheduling, you may need to take some unexpected sick leave, or J. Random Manager might decide that doing an inventory of mouse mats really does need priority above your projects. It is best to revisit your calendar on a daily basis so you can keep an eye on your week. What will you be doing during the next few days? What should you have done during the past few days? Are you on track when it comes to your To Do list?

Final Thoughts

You may think that all of this planning business seems like an awful lot of work. I would be the first to agree with you. However, as I mentioned at the start of this article, it will be well worth your time. Not only will you be spending your time in a more ordered fashion, but it will also make you look good in the eyes of management.

As it says in the *Hitchhiker's Guide to the Galaxy*, you will be the "really hoopy frood, who really knows where his towel is," because when things get messy you will still be organized.

DAVID BLANK-EDELMAN

practical Perl tools

CONFIGURATION FILES

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

dnb@ccs.neu.edu

LET ME PUT DOWN THE "UNDER NEW management" sign for a moment and welcome you to the first article in a slightly different column than you are used to seeing in this spot. It would be very difficult to step into Adam Turoff's shoes, especially given his multiple years of great articles, so I'll be taking this column in a different direction.

It is most auspicious that the theme of this issue of *;login:* is "system administration" because that's my particular bent as well (having proudly been in the biz for about 20 years). As a sysadmin I've been interested in Perl for many of those years because it has been a good tool for lots of practical uses: hence the new column title. (To get the heresy out of the way early in my tenure: I don't think Perl is always the best tool. You should always use the best tool for the job.) So that's enough meta-yammering about the column; let's get on to the actual subject of today's article.

Let us consider the lowly config file. For better or worse, config files are omnipresent not just for a sysadmin but for anyone who has ever had to configure software before using it. Yes, GUI and Web-based point-and-click festivals are becoming more prevalent for configuration, but even in those cases there's often some piece of configuration information somewhere in a file that has to be twiddled before you can even get to that point in the setup of new software.

From the Perl programmer's point of view (ours), the evolutionary stages of a program usually go as follows.

First, the roughest, simplest of scripts (this stage may be skipped by senior programmers):

```
use strict; # assume this line for all of our examples

open my $DATA_FILE_H, '<', "/var/adm/data"
    or die "unable to open datafile: $!\n";
open my $OUTPUT_FILE_H, '>', "/var/adm/output"
    or die "unable to write to outputfile: $!\n";

while ( my $dataline = <$DATA_FILE_H> ) {
    chomp($dataline);
    if ( $dataline =~ /hostname: / ) {
        $dataline .= ".example.edu";
    }
    print {$OUTPUT_FILE} $dataline . "\n";
}
close $DATA_FILE_H;
close $OUTPUT_FILE_H;
```

That's quickly replaced by the next stage, the arrival of variables:

```
my $datafile      = '/var/adm/data';      # input data file name
my $outputfile    = '/var/adm/output';    # output data file name
my $change_tag = 'hostname: ';            # append data to these lines
my $fdqn          = '.example.edu';       # domain we' ll be appending

open my $DATA_FILE_H, '<', $datafile
    or die "unable to open $datafile: $!\n";
open my $OUTPUT_FILE_H, '>', $outputfile
    or die "unable to write to $outputfile: $!\n";

while ( my $dataline = <$DATA_FILE_H> ) {
    chomp($dataline);
    if ( $dataline =~ /$change_tag/ ) {
        $dataline .= $fdqn;
    }
    print {$OUTPUT_FILE} $dataline . "\n";
}
close $DATA_FILE_H;
close $OUTPUT_FILE_H;
```

Many Perl programs happily remain at this stage for the duration of their lifespan. However, more experienced Perl programmers recognize that code like this is fraught with potential peril when development continues and the program gets bigger and bigger, perhaps being handed off to other people to maintain. This peril manifests the first time someone naïvely adds code deep within the program that modifies $change_tag or $fdqn. All of a sudden the output of the program changes in an unexpected and unwanted way. In a small code snippet it is easy to spot the connection between $change_tag or $fdqn and the desired results, but it can be much trickier to find something like this in a program that scrolls by for many screensful.

One approach to fixing this problem would be to rename variables like $fdqn to something more obscure such as $dont_change_this_value_yesiree_bob, but that's a bad idea. Besides consuming far too many of the finite number of keystrokes you are going to be able to type in your lifetime, it wreaks havoc on code readability. There are a number of data-hiding tricks we could play instead (closures, symbol table manipulation, etc.), but they don't help with readability either and are more complex than is necessary. The best idea is to use something similar to the "use constants" pragma to make the variables read-only:[1]

1. Why not actually "use constants" instead? The Readonly module documentation points out a number of reasons. The three most compelling are: the ability to interpolate Readonly variables into strings (e.g., print "Constant set to $CONSTANT\n"); the ability to lexically scope the read-only variable (e.g., Readonly my $constant => "fred") so they can be present in only the scope you desire; and unlike "use constant," attempts to redefine a Readonly variable are rebuffed.

```
use Readonly;

# we've uppercased the constants so they stick out
# note: this is the Perl 5.8.x syntax; see the Readonly docs for using
#       Readonly with versions of Perl older than 5.8
Readonly my $DATAFILE        => '/var/adm/data';     # input data file name
Readonly my $OUTPUTFILE      => '/var/adm/output';   # output data file name
Readonly my $CHANGE_TAG   => 'hostname: ';           # append data to these lines
Readonly my $FDQN            => '.example.edu';      # domain we'll be appending

open my $DATA_FILE_H, '<', $DATAFILE
    or die "unable to open $DATAFILE: $!\n";
open my $OUTPUT_FILE_H, '>', $OUTPUTFILE
    or die "unable to write to $OUTPUTFILE: $!\n";

while ( my $dataline = <$DATA_FILE_H> ) {
    chomp($dataline);
    if ( $dataline =~ /$CHANGE_TAG/ ) {
        $dataline .= $FDQN;
    }
    print {$OUTPUT_FILE} $dataline . "\n";
}
close $DATA_FILE_H;
close $OUTPUT_FILE_H;
```

Now that we've seen the ne plus ultra of storing configuration information within the script,[2] we've hit a wall: what happens when we decide to write a second or third script that needs similar configuration information? Any readers who reached for the cut/copy function in their editor as an answer to that question are fired. Simple duplication of the same information into a second script may seem harmless, but it is the first step on to the road away from Oz and toward an unpleasant encounter with the flying monkeys and an unhappy lady with a broomstick. Don't do it.

2. There are some games we could play with the __DATA__ token, but, in general, keeping the configuration information at the beginning of the script is better form.

The right answer may well be some sort of config file.[3] Once you've decided to use a config file, the next question is, What format?

3. If your thoughts sped ahead to more sophisticated solutions, hold on—we'll mention them at the end of this article.

Answering that question is similar to the old joke "The wonderful thing about standards is there are so many to choose from!" Discussions of which formats are best usually become some mishmash of religion, politics, and personal aesthetic taste. Because I'm a flaming pluralist, we're going to take a look at how to deal with several of the most common formats and leave you to choose the best one for your application. I'll try to give you my humble opinion about each to help with that process.

Config File Formats

BINARY

The first kind of configuration file we're going to look at is my least favorite, so let's get it out of the way quickly. Some people choose to store their configuration data on disk as basically a serialized memory dump of their Perl data structures. There are several ways to write this data structure to disk, including the old warhorse Storable:

```
use Storable;

# write the config file data structure out to $CONFIG_FILE
store \%config, $CONFIG_FILE; # use nstore() for platform-independent file

# later (perhaps in another program), read it back in for use
my $config = retrieve($CONFIG_FILE);
```

I've also become fond of the module DBM::Deep, which has the benefit of producing data files that aren't platform-specific by default (though Storable's nstore method can help with that). For a pure Perl module, it is pretty spiffy.

```
use DBM::Deep;

my $configdb = new DBM::Deep "config.db";

# store some host config info to that db
$configdb->{hosts} = {
    'agatha'      => '192.168.0.4',
    'gilgamesh'   => '192.168.0.5',
    'tarsus'      => '192.168.0.6',
};

# (later) retrieve the name of the hosts we've stored
print join( " ", keys %{ $configdb->{hosts} } ) . "\n";
```

Files in this format are typically really fast to read, which can be quite helpful if performance is a concern. Similarly, there's something elegant about having the information stay close to the native format (i.e., a Perl data structure you're going to traverse in memory) for its entire lifespan versus transcoding it back and forth from another representation through a myriad of parsing/slicing/dicing steps.

So why is this my least favorite kind of config file? First, and least palatable to me, is the binary nature of the files created. I'd much prefer to have my config files human-readable wherever possible. I don't want to have to rely on a special program to decode the information (or to encode it, when the data gets written). Besides the visceral reaction, it also means I can't operate on the data using other standard tools such as grep. Luckily, if you are looking for speed, there are other alternatives we'll be discussing in a moment.

NAKED DELIMITED DATA

Also in the category of formats I tend to dislike are those that are simply a set of data in fields delimited by some character. The /etc directory on a UNIX box is lousy with them: passwd, group, and so on. Comma or Character Separated Value files (CSV, take your pick of expansions) are in the same category.

Reading them in Perl is pretty easy because of the built-in split() operator:

```
use Readonly;

Readonly my $DELIMITER   => ':';
Readonly my $NUMFIELDS =>  4 ;

# open and read in a line from your config file here

# now parse the data
my ( $field1, $field2, $field3, $field4, $excess ) =
    split $DELIMITER, $line_of_config, $NUMFIELDS;
```

For CSV files, there are a number of helpful modules to handle tricky situations like escaped characters (i.e., using commas in the data itself, not anything from a prison break). Text::CSV::Simple, a wrapper around Text::CSV_XS, works well:

```
use Text::CSV::Simple;

my $csv_parser = Text::CSV::Simple->new;

# @data will then contain a list of lists, one entry per line of file
my @data = $csv_parser->read_file($datafile);
```

This data format is also on my "least-favored" list. Unlike the previous format, it has the benefit of being human-readable and standard tool-parseable. However, it also has the drawback of being easily human-misunderstandable and mangleable. Without a good memory or external documentation, it is often impossible to understand the contents of the file ("What was the 7th field again?"). This leaves it susceptible to fumble-fingering and subtle typos. It is field-order fragile.

KEY/VALUE PAIRS

The most common format around is the "key {something} value" style, where {something} is usually whitespace, a colon, or an equals sign. Besides the separator difference, there are often other twists like .ini "[sections]" names or configuration scopes (à la Apache's configuration file).

4. For more information on this, Barry Schwartz's book *The Paradox of Choice: Why More Is Less* is recommended.

Dealing with formats like this using Perl modules turns out to be initially hard because there are too many choices.[4] No, really! In my survey of CPAN for config modules for this article, I encountered at least 26 modules of interest that fall into this category. To winnow this down, there are a number of decision forks:

1. How complex do you want the configuration file to be: Will simple .ini files work for you? More complex .ini files? Apache style? Extended Apache style? Do you need sections? Do you need scoped directives? Want to write your own grammar representing the format?
2. How would you like to interact with the configuration information: Want to be handed back a simple data structure? an object representing the information? Prefer to treat things like magical tied hashes or Perl constants? Does the information you get back have to come back in the same order as it is listed in the config file? Would you be happy if the module figured out the config file format for you?
3. What else is important to you: Do you care how quickly the configuration is parsed or how much memory the parsing process takes? Should it handle caching of the config for fast reload? Do you want to be able to cascade the configs (i.e., have a global config with other configs for more specific information)? Should the config be validated on parse?

The answer to each of these questions will point at a different module or set of modules available for your use. We can't dive into all of the modules out there, so let's look at three you may not have seen:

Config::Std is Damian Conway's config parsing module. He's a smart guy and so his module in this space attempts to be the same. Unlike most configuration modules, his module lets you read and then update the configuration file while preserving the section order and the comments. The file format it uses looks much like .ini files, so it should be pretty easy for most people to understand on first sight. Here's an example of the module in action. Note: The examples in this section will be really boring because the modules are all designed to make the process of dealing with config files simple (boring).

```
use Config::Std;
read_config 'config.cfg' => my %config;
# now work with $config{Section}{key}...
# and write the config file back out again
write_config %config;
```

In Conway's book *Perl Best Practices*, he suggests that if you need something more sophisticated than his simple Config::Std format can provide, Config::General can oblige. It handles files in the Apache config file family and has a much richer syntax. Actual use of the module isn't any more complex than Config::Std:

```
use Config::General;
my %config = ParseConfig( -ConfigFile => 'rcfile' );
# now work with the contents of %config...
# and write the config file back out again
SaveConfig( 'configdb', \%config );
```

Config::Scoped gives you still more bells and whistles. It parses a similarly complex format that includes scoped directives (essentially the one used by BIND or the ISC DHCP server), can check the data being parsed, will check the permissions of the config file itself, and includes caching functionality. This caching functionality allows your program to parse the more complex format once and then quickly load in a binary representation of the format on subsequent loads if the original file hasn't changed. This gives us the speed we coveted from the first kind of file we looked at and the readability of the file formats discussed in this section. It doesn't, however, offer an easy way to programmatically update an existing configuration file like some of the other modules we've seen. Here's a small snippet for how to use the caching functionality:

```
use Config::Scoped;
my $parser= Config::Scoped->new( file => 'config.cfg' );
my $config= $parser->parse;

# store the cached version on disk for later use
$parser->store_cache( cache => 'config.cfg.cache' );

# (later, in another program...)
$cfg = Config::Scoped->new( file => 'foo.cfg' )->retrieve_cache;
```

If you are the type of person who likes to smelt your own bits, then there are also a number of other modules like Config::Grammar which allow you to define your own grammar to represent the configuration file format. I tend not to like creating custom formats, if I can help it, for reasons of maintainability, but if this suits your purposes these modules can oblige.

MARKUP LANGUAGES

The last format type we'll be looking at is becoming increasingly common as XML continues to pervade more and more of the IT space (largely due to its shininess). And the use of a markup language like XML to describe configuration information is becoming more and more prevalent.[5] Marketing potential aside, XML does have a few nice properties when used for config files. When kept simple (because a complex/convoluted XML document is as inscrutable as one in any other format), XML config files can be nearly self-documenting. The freedom to define almost arbitrary tags lets it be as descriptive as you'd like. If I write a simple XML file like this, you can probably understand the gist of it without needing a separate manual page:

5. There are in fact XML dialects such as DCML, NetML, and SAML which are gunning for parts of the configuration management space.

```
<config>
   <host>
      <name> agatha </name>
      <addr> 192.168.0.4 </addr>
   </host>
   ...
</config>
```

Another plus of this format is the well-defined syntax and optional validation mechanisms which are part and parcel of XML. At the very least, this means that all of your XML config files can share the same parser and validation mechanism independent of their actual content.

The easiest way to read an XML config file from Perl is the XML::Simple module. It allows you to write simple code like this to slurp an XML file into Perl data structure:

```
use XML::Simple;

my $config = XMLin('config.xml');

# work with $config->{stuff}
```

Turning that data structure back into XML for writing after you've made a change to it is just as easy:

```
... (data structure already in place)
open my $CONFIG_FILE_H, '>', $configfile
    or die "Can't write to $configfile:$!\n";
print {$CONFIG_FILE_H} XMLout($config);
close $CONFIG_FILE_H;
```

Now, some people aren't swayed by the sparkly nature of XML. They think that there's too much markup for each piece of content and would prefer something with fewer angle brackets. For these people there is a lighter-weight format called YAML (which stands for YAML Ain't Markup Language). YAML tries to strike a balance between structure and concision, and so it looks a little cleaner to the average eye:

```
---
name: agatha
address: 192.168.0.4
---
name: mr-tock
address:
    - 192.168.0.10
    - 192.168.0.11
    - 192.168.0.12
---
```

The Perl module to parse YAML[6] is called, strangely enough, YAML and is used like this:

6. One nice property of YAML is it is language-independent. There are YAML parsers and emitters for Ruby, Python, PHP, Java, OCaml, and even Javascript.

```
use YAML;
my @config = YAML::LoadFile('config.yml');
# @config now contains a list of references to hashes, one per record
# we now can use $config[N]->{address}
# (later...) dump the config back out to a file
YAML::DumpFile( 'config.yml' , @config );
```

If you'd prefer a more object-oriented way of working with YAML, Config::YAML can provide it.

There are an infinite number of possible formats for config files, but at least now we've hit the highlights.

All-in-One Modules

If all of this talk about picking the right module for config parsing has made your brain hurt, let me ease us toward the end of this article with a quick look at a set of modules which can help sidestep the choice.

Config::Context is a wrapper around the Config::General, XML::Simple, and Config::Scoped modules that allows you to use a single module for each of the formats those modules handle. On top of this, it also adds contexts à la Apache so you can use <Location> </Location> tags in those file formats.

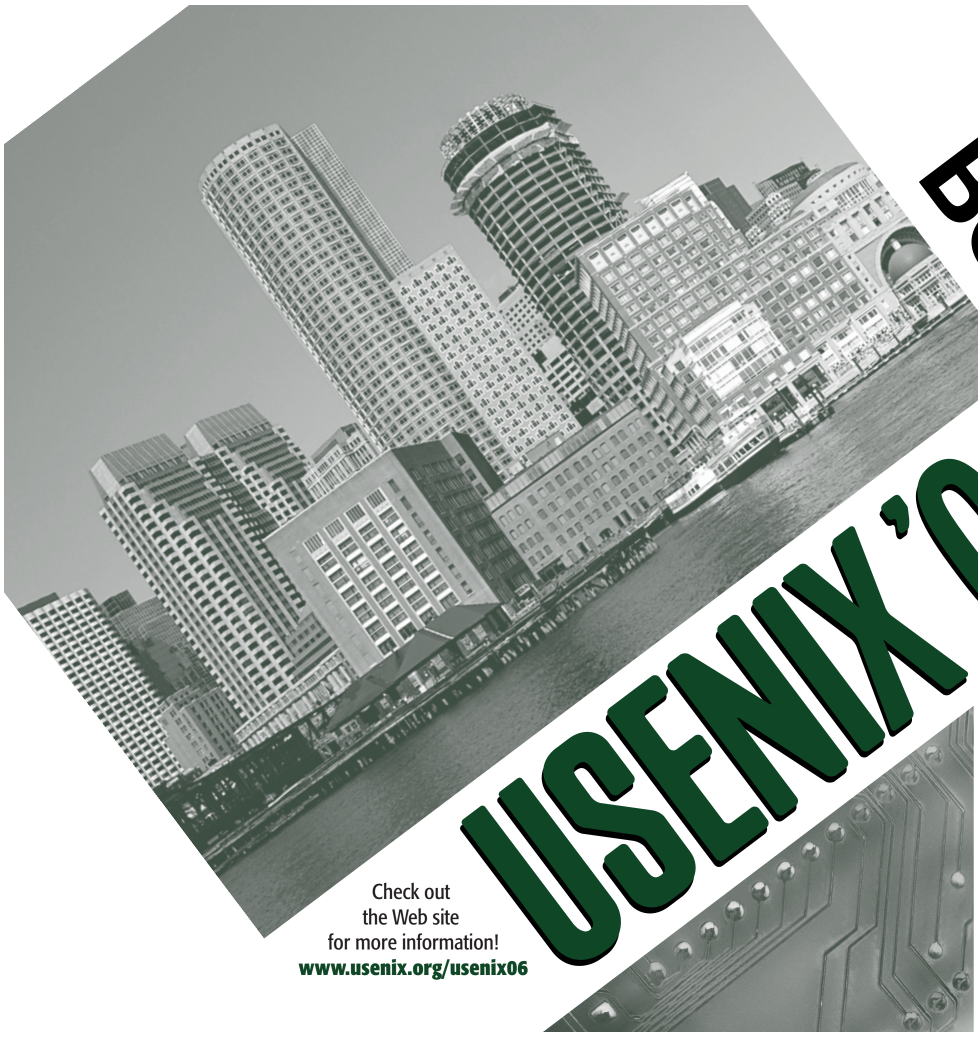
If you crave a module with a larger menu of config file formats supported, Config::Auto can handle colon/space/equals-separated key/value pairs, XML formats, Perl code, .ini formats, BIND9 style, and irssi config file formats. Not only that, it will (by default) guess the format it is parsing for you without further specification. If that's too magical for you, a format can be specified.

Epilogue

If you are sick of talking about config files at this point (I don't blame you), let's end with a brief mention of some of the more advanced alternatives. There are a number of other reasonable places to stash config information.[7] Shared memory segments can work well when performance is the key criterion. Many systems are now keeping their configuration in databases. Others have a specific network server to distribute configuration information.

These are all interesting directions to explore, but I'm afraid we're out of time. Take care, and I'll see you next column.

7. There are also a number of other unreasonable places—for example, hidden in image files using Acme::Steganography::Image::Png or in a play via Acme::Playwright.



Annual
Technical
Conference

Join us in Boston for 5 days of groundbreaking research and cutting-edge practices in a wide variety of technologies and environments.

Don't miss out on:

- **Extensive Training Program featuring expert-led tutorials**
- **New! Systems Practice & Experience Track (formerly the General Session Refereed Papers Track)**
- **Invited Talks by industry leaders**
- **And more**

Please note: USENIX '06 runs Tuesday–Saturday.

Check out
the Web site
for more information!
www.usenix.org/usenix06

Attention, Members:
Are You Getting the Most Out of Your Membership?

Become an active member of the Association. This is your community: get involved!

We are proud of our 30-year history of offering services to the advanced computing systems community. The support and participation of our members make us able to offer some of the most highly respected conferences and publications in the industry.

We have recently added the benefit of a Jobs Board for all USENIX and SAGE members, as well as additional benefits for our SAGE, Educational, Corporate, and Supporting members. We encourage you either to upgrade your membership or to talk to your employer about an institutional membership with USENIX.

In addition to the great benefits you already enjoy, we are offering these new benefits:

STANDARD USENIX MEMBERSHIP: INDIVIDUAL ($115 PER YEAR) AND STUDENT ($40 PER YEAR)

- The USENIX Jobs Board: Looking for a new job? USENIX members have direct access to offerings from top-notch potential employers. Members can also post resumes. For information on how to post, see http://www.usenix.org/jobs/.

SAGE MEMBERSHIP: INDIVIDUAL ($40 PER YEAR) AND STUDENT ($25 PER YEAR)

- Resume posting service
- The latest Short Topics in System Administration booklet for every member

USENIX EDUCATIONAL MEMBERSHIP ($250 PER YEAR)

- The USENIX Jobs Board (see above)
- Up to two additional copies of *;login:* per issue (email office@usenix.org with your request)

USENIX CORPORATE MEMBERSHIP ($460 PER YEAR)

- The USENIX Jobs Board (see above)
- Up to four additional copies of *;login:* per issue (email office@usenix.org with your request)
- Up to five conference registrations at the USENIX member price for your staff (email conference@usenix.org for a discount code to use in registering)
- Your company name listed on our Corporate Members Web page, http://www.usenix.org/membership/corporate.html.

USENIX SUPPORTING MEMBERSHIP ($2500 PER YEAR)

- The USENIX Jobs Board (see above)
- Up to four additional copies of *;login:* per issue (email office@usenix.org with your request)
- Tarballs of any USENIX conference Proceedings from the year *before* your membership term begins (email office@usenix.org with your request)

For a full listing of all benefits or to join online, please see http://www.usenix.org/membership.

ROBERT HASKINS

ISPadmin

BLOCKING NON-EMAIL SPAM

Robert Haskins has been a UNIX system administrator since graduating from the University of Maine with a B.A. in computer science. Robert is employed by Shentel, a fast-growing network services provider based in Edinburg, Virginia. He is lead author of *Slamming Spam: A Guide for System Administrators* (Addison-Wesley, 2005).

■ *rhaskins@usenix.org*

SPAM HAS BEEN AROUND FOR MANY years, since (at least) the infamous Canter and Siegel green card Usenet message in 1994. The bad news is that the problem of unsolicited commercial messages entered the SMS (mobile/cell phone), Web log (blog), and instant message (IM) space some time ago. However, the good news is that spam activity in these more modern forms of communication is currently much lower than traditional email spam.

Background

Before getting into what can be done about blocking these newer forms of spam, I'll briefly introduce the problem of these spamming methods. In most of these cases, existing anti-email spam methods can be and are applied by the provider and/or the end user to battle the new forms of spam. However, these traditional methods are much more dependent on the precise server and client software tools used by the provider.

The lack of a centralized, open source messaging server as there is in the email space (e.g., Sendmail) makes addressing the problem of non-email spam much more difficult. Another piece of bad news is that the available tools and techniques for addressing non-email spam messaging are in their infancy. However, as the perpetrators adjust their spamming techniques, this will undoubtedly change, as it did in the email spam area.

SMS (CELL PHONE) SPAM

In the case of cell phone spam, the perpetrators often guess the cell phone numbers of the lucky recipients. SMS spammers send their junk to unwitting subscribers using the publicly available email-to-SMS gateways provided by the cell phone service providers. It's a relatively easy and cheap way for the spammers to get their message out to lots of users in an immediate fashion.

The regularity of cell phone numbers (at least for most U.S. carriers) makes guessing recipients trivial. By using publicly available information for the area code and local exchange for the provider in question, the cell phone spammer must simply guess the last four digits for the subscriber.

Directory harvest attacks (where the spammer guesses the email address of the subscriber's phone) can be effective for identifying potential recipients for the SMS spammer. However, if the service provider has any sort of rate limiting in effect on the SMS-to-email gateway, it can be used to identify a spammer who exceeds preset message sending thresholds (either successful or unsuccessful attempts).

BLOG SPAM

Blog spam (also called link spam or comment spam) is defined by Wikipedia as "any web application that displays hyperlinks submitted by visitors or the referring URLs of web visitors" [1]. The target of the spammers can be any page that accepts comments from the general public, including wikis, blogs, and Web-based discussion boards. (For the purposes of this article, the term "blog spam" is used to denote any discussion-based spam mechanism.) The goal of the spammer is often to increase search engine rankings by increasing the number of link counts to the spammer's target site.

The solutions to the problem of blog spam are closely tied to the software packages used to implement the discussion boards themselves. Without a centralized exchange point (as exists with email spam), it is difficult to generalize a solution for blog spam.

INSTANT MESSAGE

Instant messaging spam (a.k.a. spim) is defined for the purposes of this article as commercial messages received via AOL Instant Message, ICQ, or any similar real-time messaging channel. The advent of protocols like Jabber [2] holds a lot of promise for controlling spim, but much work needs to be done, as the tools are still not very sophisticated.

Solutions

So what can be done about these new forms of spam? In general, the information sent in email spam is similar to the information sent by spammers in other forms. For example, spammers will send a URL or telephone number as part of their message. If the anti-spam solution uses content analysis, then the same information used to filter email spam can be used to filter other types of spam. However, content analysis in the form of header information is not possible, as user-identifiable headers don't exist for most non-email-type communication channels.

CONTROLLING SMS SPAM: PROVIDER SIDE

Regarding cell phone spam, the place to catch SMS spam is before (or at) the email-to-SMS gateway. Traditional content-based email anti-spam methods can be useful to the provider prior to the message entering the provider's SMS system. These methods are well documented elsewhere and are not covered here.

CONTROLLING SMS SPAM: SUBSCRIBER SIDE

Some cell phone providers (e.g., Verizon Wireless) give their subscribers the ability to change the external email address used for sending text messages to the subscriber's cell phone. This change can be made to obfuscate the subscriber's email address, making it harder for spammers to "guess" potential recipients. Other capabilities may be present in SMS provider networks, such as restricting senders (whitelisting/blacklisting).

CONTROLLING BLOG SPAM

Spam to blogs and similar discussion groups is most often handled by the software that implements the blog itself. This is because there is not much in the way of protocol or other clearinghouse mechanisms with blogs (as there is with email spam in the form of message servers like Sendmail). Some methods used by blog/wiki software to limit spam include:

- Periodically scanning blogs and removing messages associated with known spammer URLs
- Using a CAPTCHA (Turing test) to force the poster to prove that they are a human and not a spammer
- Using whitelists/blacklists of IP addresses posting allowed/disallowed

Blog spam is a difficult problem to solve, as the usual email spam issues such as false positives and what to do with posts identified as spam still apply. How do you allow the moderator to reinstate a blog posting incorrectly classified as spam?

CONTROLLING SPIM: SERVER SIDE

The ability to control instant message spam is arguably the most advanced of the three types of non-email spam handling covered here. One example of the maturity is the simple fact that there is a commercial product in this space, namely, Perimeter Manager for IM [3] from Postini. This service utilizes Postini's email spam processing network to identify IM messages that are potentially spam messages.

On the open source side, there isn't really a solution currently available. While AIM and similar protocols have proxy capability, this author is aware of no firewalls that enable end users to filter instant messages in any way.

Jabber is an open source instant messaging protocol which may improve open source IM spam filtering. There is currently an experimental Jabber standard titled "SPIM-Blocking Control" which enables some level of spim filtering [4]. This is targeted at the large "zombie networks" of machines that often send IM spam (as well as other undesirable messages). Some of the techniques used to control spim include:

- Whitelisting/blacklisting functionality
- Automatically exchanging lists of IDs that have sent spim to users or that don't answer specific challenges to prove the ID is a real person

Unfortunately, existing techniques are simple and don't include complex content analysis such as URL-checking and similar content-filtering capabilities.

CONTROLLING SPIM: CLIENT SIDE

On the IM client side, the controls available are directly dependent upon the IM client that is used. Most allow whitelists and blacklists, but beyond that not much is available.

Conclusion

SMS spam, blog spam, and spim are here to stay and are only going to get worse. The good news is that these forms of spamming are not too widely used, and basic whitelisting/blacklisting techniques can be utilized to filter most of this junk from your daily life. In the case of blog spam, methods exist and are used to help reduce the problem. However, there is a tight integration between the anti-spam blog software and the blog application itself, so stand-alone methods don't exist.

Although there is a commercial solution to the problem of spim and limited anti-spam standards are being developed for the Jabber IM protocol, no open source solution currently exists for spim.

I'd like to thank Todd Underwood of Renesys and Scott Petry of Postini for their input into this article.

REFERENCES

[1] http://en.wikipedia.org/wiki/Blog_spam

[2] http://www.jabber.org/

[3] http://www.postini.com/postini_solutions/im_security.php

[4] http://www.jabber.org/jeps/jep-0159.html

Wikipedia page for SMS spam: http://en.wikipedia.org/wiki /Mobile_phone_spam

MICHAEL RASH

single packet authorization with fwknop

Michael Rash holds a master's degree in Applied Mathematics and works as a security research engineer for Enterasys Networks, Inc. He is the lead developer of the cipherdyne.org suite of open source security tools, including PSAD and FWSnort, and is co-author of the book *Snort-2.1 Intrusion Detection,* published by Syngress.

mbr@cipherdyne.org

ONE YEAR AGO, IN THE DECEMBER 2004 issue of *;login:*, in the article entitled "Combining Port Knocking and Passive OS Fingerprinting with Fwknop," I described a technique for combining passive OS fingerprinting with a method of authorization called port knocking Since that time I have implemented a new method of securing IP-based communications called Single Packet Authorization (SPA) [1], which draws on some of the strengths of port knocking and fixes some of its weaknesses. Fwknop retains the ability to generate encrypted port knock sequences and incorporate additional criteria on the OS required to honor such sequences, but the default authorization method has been switched to SPA due to the benefits this strategy has over traditional port knocking.

This article discusses Single Packet Authorization as implemented by fwknop, suggests why you would want to use it, and provides an example of using fwknop to provide an additional layer of security for OpenSSH. Fwknop is free software released under the GNU Public License (GPL) and can be downloaded from http://www.cipherdyne.org/projects/fwknop/.

The Chief Innovations of Port Knocking

When the concept of port knocking [2] was announced in 2003, many competing implementations were rapidly developed. At last count, portknocking.org lists nearly 30 different software projects dedicated to the specific visions of port knocking promoted by their respective authors. Some of these projects are more complete than others, but in general they stay true to port knocking's chief innovation, the communication of information across sequences of connections to closed ports. The port numbers themselves, instead of the application payload portion of TCP segments or UDP datagrams, transmits the information as it is sent from the port knocking client to the server. Of course, the term "server" only applies to the portion of the port knocking scheme that is designed to passively receive packets; there is no traditional server that listens via the Berkeley sockets interface. The information typically sent in a port knock sequence communicates desired access through a packet filter that is protecting a particular service or set of services. The knock server gathers the knock sequence via a passive monitoring mechanism such as firewall-log monitoring or using libpcap to monitor packets as they fly by on the wire. This allows a kernel-level packet filter (such as Netfilter in the Linux kernel) to be configured in a default drop stance so that the only connections to a protected service that are allowed to be established are those that have first been associated with a valid port knock sequence. This is a powerful concept, because the end result is that code paths available to a would-be attacker are minimized. Even if an attacker possesses an exploit (0-day or otherwise) for a service that is actually deployed on a system, it is rendered useless, since a connection cannot even be established without first issuing a valid knock sequence. When an attacker uses the venerable Nmap with all of its sophisticated machinery to enumerate all instances of a vulnerable service accessible throughout a network, services protected in such a manner will not appear in the list.

Single Packet Authorization vs. Port Knocking

So far we have discussed the two most important ways that port knocking is used to enhance security: the passive communication of authentication information, and the server-side use of a packet filter to intercept all attempts to connect with a real server that are not associated with a knock sequence. These two features are also used in Single Packet Authorization to increase security, but this is where the similarities between port knocking and SPA abruptly end.

In port knocking schemes, the communication of information within packet headers, as opposed to the packet payload, severely limits the amount of data that can stilll be transferred effectively. The port fields of the TCP and UDP headers are 16 bits wide, so only two bytes of information can be transferred per packet in a traditional port knock sequence. This assumes that other fields within the packet header are not also made significant in terms of the knock sequence, but any conceivable implementation would be able to transmit much less information than a protocol that makes use of payload data. If two bytes of information were all that were required to communicate the desired access to a knock server, this would not be a significant issue, but it is not enough to simply create a mapping between a knock sequence (however short) and opening a port. We also want our messages to resist decoding by an attacker who may be in the enviable position of being able to monitor every packet emanating from the knock client. This requirement can be satisfied by using an encryption algorithm, but even a symmetric block cipher with a reasonable key size of, say, 128 bits forces at least eight packets to be sent at two bytes per packet.

As soon as multiple packets become involved, we need to try to ensure that the packets arrive in order. This implies that a time delay is added between each successive packet in the knock sequence. Simply blasting the packets onto the network as quickly as possible might result in out-of-order delivery by the time the packets reach their intended target. Because the knock server is strictly passively monitoring packets and consequently has no notion of a packet acknowledgment scheme, a reasonable time delay is on the order of about a half-second. Given a minimum of eight packets to send, we are up to four seconds just to communicate the knock sequence. In addition, if there were ever a need to send more information, say on the order of 100 bytes, the time to send such a message is longer than most people would be willing to wait. Single Packet Authorization has no such limitation, because the application payload portion of packets is used to send authentication data. The result is that up to the minimum MTU number of bytes of all networks between the client and server can be sent in a single message, and no cumbersome time delays need to be introduced. Fwknop uses this relatively large data size to communicate not only detailed access requirements in SPA messages, but also entire commands to be executed by the fwknop SPA server. Of course, all SPA messages are encrypted, and the algorithm currently supported by fwknop is the symmetric Rijndael cipher, but the upcoming 0.9.6 release will also support asymmetric encryption via GPG key rings and associated asymmetric cipher(s).

An additional consequence of sending multiple packets in a slow sequence is that it becomes trivial for an attacker to break any sequence as it is being sent by the port knocking client. All the attacker needs to do is spoof a duplicate packet from the source address of the client during a knock sequence. This duplicate packet would be interpreted by the knock server as part of the sequence, hence breaking the original sequence. Programs like hping (see http://www.hping.org) make it exceedingly easy to spoof IP packets from arbitrary IP addresses. Single Packet Authorization does not suffer from this type of easy injection attack.

In addition to making it difficult for an attacker to decode our messages, we also require that it not be possible for the attacker to replay captured messages against the knock server. A mechanism should be in place that makes it easy for the server to know which messages have been sent before and not to honor those that are duplicates of previous messages. It is not enough just to encrypt knock sequences even if the IP address to which the server grants access is buried within the encrypted sequence; consider the case where a knock client is behind a NAT device and the attacker is on the same subnet. If a knock sequence is sent to an external knock server, then the IP address that must be put within the encrypted sequence is the external NAT address. Because the attacker is on the same subnet, any connection originating from the attacker's system to the external knock server will come from the same IP as the legitimate connection. Hence the attacker need only replay a captured knock sequence from the client in order to be granted exactly the same access.

In the world of traditional port knocking there are ways to prevent replay attacks, such as altering knock sequences based upon time, iterating a hashing function as in the S/KEY system [3], or even manually changing the agreed upon encryption key for each successful knock sequence. However, each of these methods requires keeping state at both the client and the server and does not scale well once lots of users become involved. It turns out that Single Packet Authorization facilitates a more elegant solution to the replay problem. By having the SPA client include 16 bytes of random data in every message and then tracking the MD5 (or other hashing function) sum of every valid SPA message, it becomes trivial for the server to not take any action for duplicate messages. The ability to send more than just a few bytes of data within an SPA message is the essential innovation that really makes this possible. Fwknop implements exactly this strategy, which will be demonstrated in the example below.

Port knocking schemes generally use the port number within the TCP or UDP header to transmit information from the knock client to the knock server. However, there are lots of IP protocols, such as ICMP and GRE, that have space reserved for application-layer data but have no corresponding notion of a "port." Theoretically, SPA messages can be sent over any IP protocol, not just those that provide a port over which data is communicated. One such protocol currently supported by fwknop is ICMP.

Finally, to an observer of network traffic, a port knock sequence is indistinguishable from a port scan— that is, it is a series of connections to various port numbers from a single IP address. Many network intrusion detection systems have the capability of detecting port scans, and have no way to know that a port knock sequence is not an attempt to enumerate the set of services that are accessible from the IP address of the client system. Hence, any intermediate IDS that has its port scan thresholds set low enough (i.e., the number of packets associated with a port knock sequence exceeds the thresholds within a given period of time) will generate port scan alerts for each port knocking sequence. Although this by itself does not create a problem for port knocking implementations in terms of the port knocking protocol, it can draw undue attention to anyone actually using port knocking on a network that is monitored by an IDS. By contrast, Single Packet Authorization does not create a significant enough network footprint to generate an IDS port scan alert.

Fwknop Single Packet Authorization Message Format

In order for an fwknop SPA client to authenticate and allow application of the subsequent authorization criteria [4], several pieces of information must be securely communicated to the knock server. An fwknop client transmits the following within each SPA message:

- 16 bytes of random data
- local username
- local timestamp
- fwknop version
- mode (access or command)
- desired access (or command string)
- MD5 sum

The 16 bytes of random data ensures that each SPA message has an extremely high probability of being unique and hence allows the fwknop server to maintain a cache of previously seen messages in order to thwart replay attacks. The local username enables the fwknop server to distinguish between individual users so that different levels of access can be granted on a per-username basis. The version number allows the fwknop message format to be extended while preserving backwards compatibility for older versions of the software. The mode value instructs the server that the client either wishes to gain access to a service or run a command, each of which is specified in the next field. The MD5 sum is calculated over the entire message and is then used by the server to verify message integrity after a successful message decrypt. All the above values are concatenated with ":" characters (with base64 encoding applied where necessary so as not to break the field separation convention), and the entire message is then encrypted with the Rijndael symmetric block cipher. A symmetric key up to 128 bits long is shared between the fwknop SPA client and the SPA server.

Fwknop in Action

Now let us turn to a practical example: we will illustrate how fwknop is used in the default Single Packet Authorization mode to protect and gain access to the OpenSSH daemon. First, we configure the fwknop server to allow access to TCP port 22 by the "mbr" username once a valid SPA message is monitored. This is accomplished by adding the following lines to the file /etc/fwknop/access.conf:

```
SOURCE: ANY;
OPEN_PORTS: tcp/22;
KEY: <encrypt_key>;
FW_ACCESS_TIMEOUT: 10;
REQUIRE_USERNAME: mbr;
DATA_COLLECT_MODE: ULOG_PCAP;
```

In server mode, fwknop can acquire packet data by using libpcap to sniff packets directly off the wire or out of a file that is written to by a separate sniffer process, or by using the Netfilter ulogd pcap writer [5]. In this case the configuration keyword DATA_COLLECT_MODE instructs the server to respect SPA messages that are collected via the ulogd pcap writer. For this example, let us assume that the IP address on the server system is 192.168.10.1, that fwknop is running in server mode, and that Netfilter has been configured to drop all packets destined for TCP port 22 by default.

Now, on the client (which has IP 192.168.20.2), we first verify that we cannot establish a TCP connection with sshd:

```
[client]$  nc -v 192.168.10.1 22
```

So far, so good. The netcat process appears to hang because we fail to even receive a reset packet back from the TCP stack on the server; Netfilter has dropped our SYN packet on the floor before it can hit the TCP stack. Having a completely inaccessible server is not of much use, of course, so now we execute the following to gain access to sshd:

```
[client]$  fwknop -A tcp/22 -w -k 192.168.10.1
[+] Starting fwknop in client mode.
[+] Enter an encryption key. This key must match a key in the file
/etc/fwknop/access.conf on the remote system.

Encryption Key:

[+] Building encrypted single-packet authorization (SPA) message...
[+] Packet fields:

Random data: 5628557594764037
Username:    mbr
Timestamp:   1132121405
Version:     0.9.5
Action:      1 (access mode)
Access:      192.168.20.2,tcp/22
MD5 sum:     q8vIpYY6q3qEflaFtU3Jag

[+] Sending 128 byte message to 192.168.10.1 over udp/62201...
```

Sure enough, we are now able to establish a TCP connection with port 22:

```
[client]$  nc -v 192.168.10.1 22
192.168.10.1 22 (ssh) open
SSH-2.0-OpenSSH_3.9p1
```

Fwknop running on the server has reconfigured Netfilter to allow the client IP address to talk to sshd. Even though fwknop will expire under the access rule after 10 seconds, by using the Netfilter connection-tracking capability to accept packets that are part of established TCP connections before packets are dropped, the SSH session remains active for as long as we need it.

Finally, to illustrate the ability of fwknop to detect and stop replay attacks, suppose that an attacker were able to sniff the SPA message above as it was sent from the client to the server (by default, fwknop sends SPA messages over UDP port 62201, but this can be changed via the -p command line argument):

```
[attacker]#  tcpdump -i eth0 -c 1 -s 0 -l -nn -X udp port 62201
tcpdump: verbose output suppressed, use -v or -vv for full protocol
decode
listening on eth1, link-type EN10MB (Ethernet), capture size 65535
bytes
01:44:12.170787 IP 192.168.20.2.32781 > 192.168.10.1.62201: UDP,
length: 128
    0x0000:  4500 009c 246a 4000 4011 768f c0a8 1406  E...$j@.@.v.....
    0x0010:  c0a8 0a01 800d f2f9 0088 9fc3 6736 576a  ............g6Wj
    0x0020:  5234 7374 4941 4358 3935 4152 6541 4778
R4stIACX95AReAGx
    0x0030:  3342 7848 7569 7776 786e 557a 3531 5131  3BxHui-
wvxnUz51Q1
    0x0040:  5532 3976 4872 7144 6e69 3330 514f 4d72
U29vHrqDni30QOMr
    0x0050:  6661 5a48 4845 304c 3631 4767 636a 6e37
faZHHE0L61Ggcjn7
    0x0060:  6a64 7a6e 787a 726c 4f53 314c 5051 6877
jdznxzrlOS1LPQhw
    0x0070:  394b 424f 3963 6b61 5232 2b6f 5474 736c
9KBO9ckaR2+oTtsl
    0x0080:  574d 484c 574f 7736 7468 4161 7a58 3976
WMHLWOw6thAazX9v
    0x0090:  2b65 6746 6352 2f2f 6776 4352           +egFcR//gvCR
1 packets captured
2 packets received by filter
0 packets dropped by kernel
```

Now the attacker can replay the encrypted SPA message on the network as follows in an effort to gain the same access as the original message [6]:

```
[attacker]$ echo
"g6WjR4stIACX95AReAGx3BxHuiwvxnUz51Q1U29vHrqDni30QOMrfaZ
HHE0L61Ggcjn7jdznxzrlOS1LPQhw9KBO9ckaR2+oTtslWMHLWOw6th
AazX9v+egFcR//gvCR"
|nc -u 192.168.10.1 62201
```

On the server, this results in the following syslog message, indicating that fwknop monitored the message replay and took no further action:

```
Nov 16 01:50:11 server fwknop: attempted message replay from:
192.168.20.6
```

Conclusion

Single Packet Authorization has several characteristics that make it more powerful and flexible than port knocking for protecting network services. Its data transmission capabilities, coupled with its clean strategy for preventing replay attacks, make it an ideal candidate for expanding the configuration of packet filters to drop all connections to some critical services by default. This makes the exploitation of vulnerabilities within such services much more difficult, because an arbitrary IP address cannot enumerate or interact with these services until a valid SPA message is generated.

REFERENCES

[1] MadHat was the first person to coin the term "Single Packet Authorization" at the BlackHat Briefings in July of 2005. However, the first available implementation of SPA was in the 0.9.0 release of fwknop in May of 2005 (with SPA code available via the http://www.cipherdyne.org/ CVS repository dating back to March of 2005; see http://www.cipherdyne.org/cgi/viewcvs.cgi/fwknop/fwknop).

[2] M. Krzywinski, "Port Knocking: Network Authentication Across Closed Ports," *SysAdmin Magazine* 12 (2003): 12–17.

[3] RFC 1760: The S/KEY One-Time Password System.

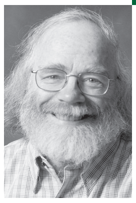
[4] The terms "authentication" and "authorization" in this context are commonly construed to mean the same thing. However, authentication refers to the verification that a communication from one party to another actually came from the first party, whereas authorization essentially refers to the process of verifying whether one party is allowed to communicate with a second party at all.

[5] See the Netfilter ulogd project: http://www.gnumonks.org/projects/ulogd/.

[6] Even if the replay were successful, access would only be granted for the IP address of the client, which is encrypted within the SPA message and hence not available to the attacker. If, however, the client is behind a NAT address, this may not matter because the external address would be the same, so it is important to stop replay attacks regardless of whether the client address is encrypted within the SPA message.

STEVEN M. BELLOVIN, BILL CHESWICK, AND ANGELOS D. KEROMYTIS

worm propagation strategies in an IPv6 Internet



Steve Bellovin, a member of the National Academy of Engineering, is a professor of Computer Science at Columbia University. He is one of the creators of Netnews, a long-time researcher on network security, and co-author of the first book on firewalls.

smb@cs.columbia.edu

Bill Cheswick is Chief Scientist at Lumeta, a company he co-founded to explore commercial intranets and the Internet. Ches did early work on firewalls and proxies and is co-author of the first book on firewalls.

ches@cheswick.com

Angelos Keromytis received his Ph.D. in computer science from the University of Pennsylvania in 2001. Currently he is an associate professor of computer science at Columbia University. His research interests include self-healing software, system reliability, design and analysis of network and cryptographic protocols, and denial-of-service protection.

angelos@cs.columbia.edu

IN RECENT YEARS, THE INTERNET HAS been plagued by a number of worms. One popular mechanism that worms use to detect vulnerable targets is random IP address-space probing. This is feasible in the current Internet due to the use of 32-bit addresses, which allow fast-operating worms to scan the entire address space in a matter of a few hours. The question has arisen whether or not their spread will be affected by the deployment of IPv6. In particular, it has been suggested that the 128-bit IPv6 address space (relative to the current 32-bit IPv4 address space) will make life harder for the worm writers: assuming that the total number of hosts on the Internet does not suddenly increase by a similar factor, the work factor for finding a target in an IPv6 Internet will increase by approximately 2^{96}, rendering random scanning seemingly prohibitively expensive.

Some worms, such as Melissa, spread by email. These worms will not be affected by the adoption of IPv6; though the space of possible email addresses is vast, these worms typically consult databases such as Microsoft Outlook's address book.

On the other hand, life will indeed be harder for address-space scanners, such as Code Red and Slammer. Clever heuristics can cut the search space dramatically. More specifically, multi-level searching and spreading techniques can negate the defender's advantage. However, the code size required for worms will increase, which may help prevent Slammer-like attacks. This has created the impression that an IPv6 Internet would be impervious to similar kinds of worms.

In the past, there have been two forms of address-space scans. Some worms use a uniformly distributed random number generator to select new target addresses. This strategy is indeed unlikely to succeed in an IPv6 world. Other worms preferentially spread locally, by biasing the search space toward addresses within the same network or subnet. This will be a more successful strategy, though at first glance the 80-bit local space (nearly twice Avogadro's number!) would seem to be a formidable obstacle. We observe that certain strategies can improve the attacker's odds. In particular, by taking advantage of local knowledge and patterns in address-space assignment, the attack program can cut the search space considerably.

We discuss a number of strategies worms could use in an IPv6-based Internet to find new targets. We separate these into two categories, wide-area and local-area searches, somewhat mirroring the IPv6 address architecture. We argue that worms will use different types of information sources to first determine existing networks and establish a presence there, and then spread locally inside an organization. We hope to illustrate that simple reliance on the IPv6 address space for protection against scanning worms is not a wise defensive strategy, and we suggest areas where research could assist in detecting and limiting future worm propagation.

The IPv6 Addressing Architecture

Addresses in the IPv6 addressing architecture, defined in RFC 3513, come in a number of different flavors. Those of interest to us are link-local addresses, unique local addresses, global addresses, and multicast addresses.

All forms of unicast address are conceptually divided into two pieces, a network section and a host section. Roughly speaking, the network section identifies the particular LAN; the host section identifies the particular node on the LAN. In fact, both sections have internal structure. Furthermore, the address is generally divided into two 64-bit halves. (There are subtleties that lie outside the scope of this article.)

In the network section of the address, the first 10 bits denote the scope of the address. The next 38 bits identify the site and (implicitly) the ISP, as explained in RFC 3177. In order to promote hierarchical addressing, only the largest ISPs have their own address allocations; smaller ISPs are assigned space by their upstream provider. Identifying the set of all ISPs considerably reduces the search space for the attacker.

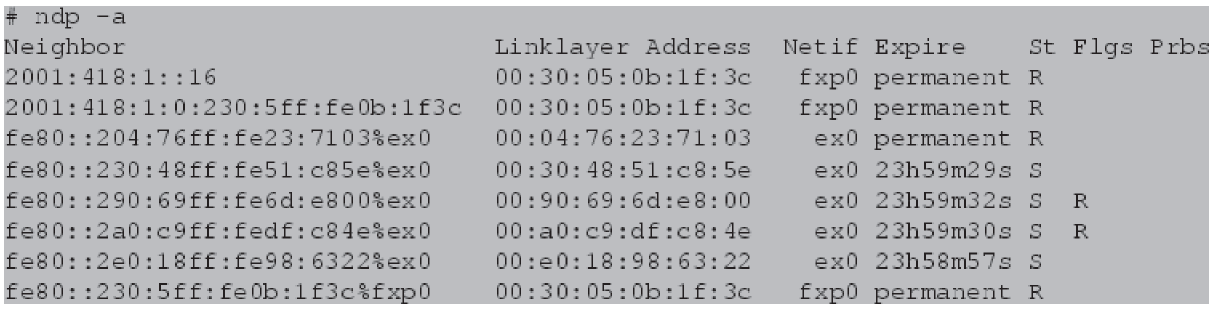
The next 16 bits in the network section of the address identify the subnet within each site. No site will have 2^{16} subnets, though identifying the allocated subnets could pose a challenge for the attacker.

There are several possible formats for the host identifier (the last 64 bits of the address). Clients will often generate their own addresses via stateless auto-configuration, as described in RFC 2460.

Sources of Information

As we already mentioned, IPv6 worms can spread by using a two-level strategy. Here we present several information sources, divided into local and wide-area sections. We do not claim that this list is exhaustive; however, the list we do present is probably sufficient and is undoubtedly indicative of a much larger class of information sources that could be exploited.

NEIGHBOR DISCOVERY



```
# ndp -a
Neighbor                             Linklayer Address   Netif Expire    St Flgs Prbs
2001:418:1::16                       00:30:05:0b:1f:3c    fxp0 permanent R
2001:418:1:0:230:5ff:fe0b:1f3c       00:30:05:0b:1f:3c    fxp0 permanent R
fe80::204:76ff:fe23:7103%ex0         00:04:76:23:71:03     ex0 permanent R
fe80::230:48ff:fe51:c85e%ex0         00:30:48:51:c8:5e     ex0 23h59m29s S
fe80::290:69ff:fe6d:e800%ex0         00:90:69:6d:e8:00     ex0 23h59m32s S  R
fe80::2a0:c9ff:fedf:c84e%ex0         00:a0:c9:df:c8:4e     ex0 23h59m30s S  R
fe80::2e0:18ff:fe98:6322%ex0         00:e0:18:98:63:22     ex0 23h58m57s S
fe80::230:5ff:fe0b:1f3c%fxp0         00:30:05:0b:1f:3c    fxp0 permanent R
```

FIGURE 1: NEIGHBOR DISCOVERY TABLES ON AN IPV6 HOST.

In IPv6, Neighbor Discovery as described in RFC 2461 is used to map IP addresses to local network addresses (e.g., Ethernet addresses), similar to the way ARP is used in today's IPv4 networks. As such, it can be a rich source of information about machines on the LAN. A sample listing of a Neighbor Discovery table is shown in Figure 1. A worm that has infected a node in the LAN can thus determine the addresses of other existing nodes in the same LAN.

ROUTING TABLES AND PROTOCOLS

Typically, host routing tables only contain entries for other local hosts plus a default route entry for all traffic outside the LAN. Many organizations, however, internally run routing protocols such as OSPF or RIP. The few IPv6 networks we are familiar with actually use RIPng (an adaptation of RIP for IPv6 networks, described in RFC 2080), and in the future may run OSPFv6 or IS-IS. In such an environment, a worm would be able to either directly consult the host routing tables (e.g., using the UNIX netstat command) or participate in a routing protocol, if only as a passive listener. In either case, the worm would be able to determine other valid subnets within the organization and subsequently target those [1].

INTERFACE IDENTIFIERS

The Neighbor Discovery tables provide another useful hint: a list of some locally used network cards. If stateless autoconfiguration is used, the high-order 32 bits of the low-order (host section) 64 bits of the IPv6 address identify the manufacturer of the card. In many organizations, common purchasing patterns mean that LAN cards in use will largely be from a small set of manufacturers. (We informally sampled two large, heterogeneous LANs, one educational and one corporate. In each case, there was a reduction to about 40 different card types, from 161 and 227 hosts, respectively.) For each such manufacturer identifier, there are at most 2^{24} possible addresses. This is a search space comparable to what is successfully exploited by today's IPv4 worms.

MULTICAST PING

Multicast is a fundamental part of IPv6 design, which unfortunately can be abused for target discovery by worms. RFC 3513 notes that FF0E:0:0:0:0:0:0:101 addresses "all NTP servers in the Internet." An NTP query to that address might locate many victims. While such a packet is unlikely to traverse the entire Internet, FF05:0:0:0:0:0:0:2 would find all routers at a site, and FF02:0:0:0:0:0:0:1 would send to all hosts on the local link. Fortunately, FF05:0:0:0:0:0:0:1, the larger-scope analog to send to all hosts at a site, is not defined.

A related IPv6 concept is that of "anycast" addresses, defined in RFC 2526, which can be used to locate the "closest" instance of a service. A worm can exploit this and other service-location mechanisms such as SLP, DHCP, DNS, and LDAP to locate local targets for attack. Service-location mechanisms are likely to be increasingly used in an IPv6 Internet, both because of increased host mobility and due to the difficult-to-memorize addresses.

HOST CONFIGURATION AND LOG FILES

Computers are generally configured with the addresses of other important local computers, such as email gateways, local file servers, Web proxy servers, local DNS servers, the /etc/hosts file in UNIX, SSH known_hosts files, *etc*. A sufficiently versatile worm could examine likely places for such configuration files—the registry on Windows machines is one such location—to discover other attack targets. Furthermore, although a worm may use a non-email infection vector (e.g., a buffer overflow for a popular service), it can still use archived user email to find new targets (hostnames).

DNS ZONE TRANSFERS

Typical DNS servers are configured such that they do not allow zone transfers from hosts other than the authorized secondary servers. However, some organizations have a mixed record on restricting zone transfers from hosts inside that organization. Thus, it may be possible for a worm to acquire a complete list of all hosts in a domain, once a host inside that domain has been infected; this list would include all hosts with static addresses as well as those using Dynamic DNS Updates.

PASSIVE EAVESDROPPING

Although most local area networks are switched, wireless networks offer the potential for discovering new targets on the local network simply by monitoring traffic. Furthermore, random-address flooding can be used in networks such as Ethernet to force a switch to effectively broadcast all local traffic and incoming external traffic.

WIDE-AREA INFORMATION SOURCES

Wide-area information sources can be used to determine valid IPv6 prefixes (networks) to target. Often, they also provide the addresses of valid hosts (typically servers) in that domain. Even when they do not, however, they can be used as a starting point for scanning. Although we do not have sufficient data, an informal poll of network operators suggested that servers would be assigned addresses statically, and that these addresses would be located in the low end of the subnet address range, significantly easing the task of a scanning worm.

ROUTING PROTOCOLS

Routing protocols provide information on address prefixes that are in use. These can be used both locally and across the Internet.

Local use is easy: the attack program just listens to local routing traffic. This may require joining the “all routers” multicast group, but there are no access controls that would prevent that from happening.

Remote use is more interesting, but perhaps more problematic. There are no inherent IPv6 features that would permit easy capture of BGP routing information by an ordinary host. On the other hand, there are public archives of routing data, such as the one available at www.routeviews.org. If this data is available for IPv6—and it is a valuable operational and research resource—a worm could use it for propagation purposes.

SERVER LOGS

Web, DNS, and incoming email servers are typically contacted by client machines from many different places. The log files of such contacts offer a good mechanism for wide-area spread. A more ambitious worm could kill off the legitimate server and grab its port number, thus collecting new addresses in real time.

SERVER ADDRESSES

Anecdotal evidence suggests that IPv6 servers tend to have low-numbered addresses. The prefix alone is hard to remember; administrators tend to select easily memorizable values for the low-order bits. This human tendency can be exploited by worms.

SUBVERTING NEIGHBOR DISCOVERY

A worm-infected host could impersonate the LAN router using Neighbor Discovery and divert all traffic to/from external hosts to itself. Such attacks are known and exploited in the current IPv4 Internet (e.g., the dsniff toolkit); while they are more difficult in an IPv6 environment, they are still possible. Using this attack, a worm would be able to find valid IPv6 addresses outside the local area network (whether in remote organizations or other LANs within the same organization). Passive eavesdropping can be equally fruitful in determining remote IP addresses (by capturing incoming packets), as discussed previously.

SEARCH ENGINES AND DNS

Web search engines are a particularly attractive source of information on potential targets, especially if the worm is targeting Web servers, as was the case with Code Red. Although such engines typically only point to Web servers, they can be used to identify valid prefixes by determining the hostname of a Web server and resolving its IPv6 address through DNS. Likewise, DNS itself can be used as a search engine for valid hostnames, by exhaustively searching for all words (and combinations of words) from a dictionary. In [2], we showed that a DNS worm in IPv6 could spread as fast as an IPv4 address-scanning worm.

PEER-TO-PEER PROTOCOLS

The most intriguing form of wide-area data is peer-to-peer networks. By participating in topology maintenance, watching queries and responses, and sending out occasional queries of its own, a worm could learn the addresses of many different hosts. File-swapping networks such as Morpheus, Kazaa, and Gnutella offer particularly attractive targets, as do more “traditional” presence protocols such as IRC, Jabber, and others.

Strategies for Spreading

Based on our discussion of information sources in the previous section, we believe that scanning worms in an IPv6 Internet will use a two-phase approach for discovering and infecting targets:

- Discover valid IPv6 prefixes using search engines, server logs, routing table information, etc. These sources may indicate specific targets within those prefixes (e.g., a Web server listed on a search result, or a host participating in a peer-to-peer network), or simply the valid prefix (as may be the case with getting a copy of a BGP table). In the second case, targeted address scanning may be needed, but by starting at the low end of the range a worm will maximize its chances of finding a server.
- Once inside an organization's network, use local information sources to determine the identity (address or hostname) of other nodes to infect. The repertoire of the worm is significantly richer here, and we believe that vulnerable nodes will be infected fairly quickly once the worm has established a presence.

This two-phase approach can also be extended to propagation. Intuitively, propagation across organizations calls for an approach distinct from that used for spreading within organizations. We believe that multi-partite worms such as Nimda will appear more frequently: email or Web-downloadable executable content (e.g., Java or Javascript embedded in every page served by a Web server) is particularly useful in propagating across administrative domains, as it appears to be difficult to intercept at the firewall [3]. A worm that manages to infect a popular Web server will be able to propagate widely and quickly to many different networks, potentially without raising suspicion for some time (pull model); email worms (push model) can exploit the social and professional interactions between individuals and organizations to spread.

More generally, client/server worms can operate efficiently in two different modes. In client mode, they search for and infect servers of some type. Once they've penetrated a server, they use a different technique to attack clients that connect to it.

Once a worm has managed to penetrate a new environment, it can switch to something more akin to traditional address scanning, using the information gathered using the techniques described in "Local Information Sources," above, as hints to direct the scanning process.

Discussion

The problem of locating hosts is not limited to the authors of malware. Network administrators and security officers responsible for intranets have a keen interest in the population of hosts found on their networks. They generally have extensive tools for auditing and updating such hosts to keep them up to date. Network management companies are often paid according to the number of hosts they manage. And, of course, unknown and unregistered hosts that appear on an intranet can be a concern, possibly violating perimeter security or network connection policy.

On traditional IPv4 intranets, the various techniques described above, along with simple or multi-protocol network probes, are used for host discovery. A computer inventory, especially including MAC address information, can be quite useful for tracking hosts. On IPv4 networks, MAC information is obtained, via SNMP, of ARP caches in routers. This incomplete information is about the best we can do.

In principle, the Ethernet addresses can be monitored as part of the IPv6 addresses as traffic travels through company checkpoints: the bottom 48 bits of an IPv6 address are supposed to be the MAC address. Whether this is the MAC address or a small integer, as we have seen, or even a cryptographically derived address as proposed in RFC 3972, well-placed flow monitors can collect census information. Similar information is already available from routers on a read-only basis using SNMP version 1 or 2, which has a sniffable community string. SNMPv3 is not widely used, but should be—as we have seen, network population information is going to become more sensitive.

A census of local IPv6 addresses could be kept in each router, up to a point. These could be collected and consolidated by authorized network administrators, but should be protected better than current router contents. Network discovery would proceed in two stages: first, discover the routers, perhaps with traceroute-style Internet mapping techniques, then gain administrative access to the router and dump the flow history information.

In any case, network administrators will be in the same game as the virus and worm writers, but with the home-field advantage. They need new tools for IPv6 networks to collect this data, with better protection of the acquired data from access by malware.

Conclusion

We have outlined a number of techniques that scanning worms can use in an IPv6 Internet to locate potential targets. These techniques are equally applicable to the current IPv4 Internet, albeit not as efficient as random scanning. Although "conventional" address-space scanning is prohibitively expensive in that environment, we believe that the diversity of sources we discussed (which is by no means exhaustive) guarantees a rich target set for worms.

The implication is that we cannot afford to rest on the assumption of inherent protection in the IPv6 addressing scheme; further research in worm detection and containment is needed. For our future work, we plan to investigate how much "coverage" our techniques can give us in the current Internet (as a measure of the effectiveness of the approach), as well as determine ways of monitoring requests to these information sources that could reveal worm-scanning activity.

REFERENCES

[1] C.C. Zou, D. Towsley, W. Gong, and S. Cai, "Routing Worm: A Fast Selective Attack Worm Based on IP Address," in *Proceedings of the Workshop on Principles of Advanced and Distributed Simulation (PADS)*, June 2005.

[2] A. Kamra, H. Feng, V. Misra, and A.D. Keromytis, "The Effect of DNS Delays on Worm Propagation in an IPv6 Internet," in *Proceedings of IEEE INFOCOM*, March 2005.

[3] D.M. Martin, S. Rajagopalan, and A.D. Rubin, "Blocking Java Applets at the Firewall," in *Proceedings of the Symposium on Network and Distributed System Security*, February 1997.

DAVE BROWN

MythTV: some assembly required

. . . BUT THE RESULTS CAN BE SPECTACULAR

Dave Brown holds a bachelor's degree in Programming and Operating Systems from University of Phoenix. He works for PalmSource, Inc., and has worked for Be and Apple.

dbrown@smurfless.com

WHEN I FIRST HEARD ABOUT MYTHTV I was impressed, but I figured it was pretty "out there." I guessed someone had set up a super-customized Linux build with custom hardware and then posted the code. A Linux-based personal video recorder (PVR) that played music and most normal computer video clips, and that was free, was too good to believe. TiVo, still going strong at the time, wanted some hundreds of dollars to get the box and a promise of $15 a month to keep it going, and ReplayTV was similar; there were no other options worth considering. You couldn't take video off either TiVo or ReplayTV, lawyers were circling the companies like vultures, and I only paid $15 a month for cable. So I couldn't justify getting MythTV, but I kept reading. Before long I realized that this was a pretty standard bit of hardware, and the software didn't look all that scary. I was fascinated.

This is where I should probably mention my nervous twitch. I almost always have to be doing something. Often two or more things. I can't stand to wait on a build. I have to have something sitting open on the side to switch back to. You might say I have a preemptive nervous system. You might also say I'm slightly neurotic.

Some of the features I was looking for I simply could not find in commercial packages. This is because when money changes hands, lawyers know they can get a slice. Faceless projects have the advantage of being allowed to do many things that companies can't, at least not in the long term. Things like commercial detection and skipping, semi-automatic conversions to DVD or MPEG-4, and all sorts of things that lawyers and their owners really don't want you to do unless you're paying them each time you do it.

But as I realized how expensive commercial DVRs were over time and how ultimately nerfed they were, I started looking harder at the other options. When I looked at the mounting pile of unused hardware in the corners of my house, I decided it was worth a shot. I was newish to Linux, having come recently from a BeOS background, and was looking for a reason to get out of Windows-land. Ultimately, I was tired of finding my movies and CDs lying around the house because someone, and I'm not saying who, couldn't be bothered to find the jacket. If I could get something able to play back videos and CDs without damaging the originals, so much the better.

About the Project

The project itself allows a pretty flexible combination of hardware. The suggested hardware list is much beefier than what you can actually get away with, but I have to admit that using something in the suggested range would make most of the setup considerably easier and faster. I won't repost their suggested hardware list, but as generalities you need:

- A video and audio input
- A tuner. I separate this because you can use an external tuner instead of an in-computer tuner
- A window manager
- A way to see: VGA-enabled TVs seem to be favorites, but I use a plain '80s-era TV
- A way to interact with your system: Most people will want to use a remote control, but you can use a keyboard and mouse
- A network connection
- Linux flavor of the month, preferably with a 2.6 kernel

The suggested range of hardware at the time I was working was a 1.2GHz processor, an nVidia-based video card and a 2.6 kernel; the rest was flexible. Some of these were "suggested hardware" as described by documentation, and some of it was required-because-nobody-is-going-to-help-you-with-anything-else. Honestly, this is still a really good base setup. If you want to do hardware-compressed encoding, most people are using ivtv driver–compatible cards from Hauppauge.

If you want only—and I mean only—hardware-assisted MPEG-2 encoding and playback, you could go with a slower processor. I actually do this at home right now.

If you want to use the DVD ripper/converter/player, you will need a DVD-ROM and, I strongly suggest, a processor over 1.5GHz.

If you plan on doing commercial detection, second the processor upgrade.

If you want to do general video playback (e.g., MPEG-4), second the processor upgrade again.

Pseudo-Log

I think the most useful thing to others would be my view of building my various MythTV boxes and the problems and impressions I had of each try. I'll try to keep things factual, although opinion and impression have a lot to do with the experience. Please keep in mind that while communities as a whole may be good, I'm recording impressions of a specific implementation as experienced by a new-to-Linux user.

First Try: RedHat Linux 9

My first attempt to install MythTV was on RedHat Linux 9.1. I was not looking to subscribe to anything, but I had used RedHat 7.x before and was familiar with the conventions. I had to compile MythTV by hand, which was not difficult. However, what *was* difficult was gathering the various prerequisites, requiring a couple of days full of bad links, out-of-date mirrors, etc. Once the prerequisites were installed, things compiled cleanly.

I found a very good walk-through of someone else's install to help me. Base MythTV worked on a monitor. I remember clearly thinking, "This is what I have wanted a computer to do for years." Others online had most of the subpackages working as well (MythWeather, MythDVD, MythMusic, etc.), but I was stuck on getting TV-Out to work on the Hauppauge 350 video card. Display on a SVGA monitor was quite good, but in my house it was hardly a usable solution. I also could not get past a compiler error in one of the codecs required for the transcode daemon (to convert DVDs to MPEG-4). On the MythTV forums I met a stunned silence. Eventually I found there were version conflicts between the base RedHat libraries and what was expected by the codecs, so I had to chase down the source and compile the modules (with lots of optional flags—must always have lots of optional flags).

By the end of this attempt I was turned off by constantly having to update random unconnected bits of the RedHat system to suit the current bug fixes. I would not be surprised if someone has had better luck than me. I know at the time I was a Linux newbie. But I was willing to try the hype of Gentoo, whose instructions on the MythTV site, "emerge mythtv," seemed alluring.

Second Try: Gentoo 2004.1

The second attempt was based on numerous reports of the Gentoo Linux distro, 2004.1 at the time. For the out-of-the-box experience, this went far better. First of all, Gentoo for the layman is just wonderful. No more wandering from search to bad link to bad link looking for where to get package xyzzy. You just tell it what item you really want, and it does it. Having an BeOS background, I'd rather just tell it "put this on and don't be dumb about it." My first trip through Gentoo was a learning experience in the nuts and bolts of the Linux world—the first time really building my own kernel and truly understanding some of the lower services of Linux.

I was still using my dual P3-500 system with my MPEG-2 encoder card. I blanked the hard drives with a cheer, followed the very friendly documentation on the gentoo.org site, and was on my way. I can't praise this documentation enough. It probably taught me more about the basics of Linux as a kernel than anything else I've done. I won't bore you with the details of how I got from hardware and kernel to command line, but I did, and that's where I started working on the MythTV install.

I was able to get MythTV's base features working with the magic incantation "emerge mythtv" and about 20 minutes of configuration. But it took 28 hours to compile on my relic. Had I but known, there was a binary package CD I could have used to get everything up to the X11 install prebuilt, but I didn't notice it at the time. At the end of the compile, I followed the Gentoo X11 configuration guide for my monitor and the basic setup guide for MythTV. The drivers for the Hauppauge card were available with another simple command ("emerge ivtv"), and in no time I could capture precompressed video. I finished the MythTV setup, and, bang, I was up and running on my monitor, pausing live TV, saving videos to my hard drive. I was extremely happy with this. And, even on this old hardware, it was very, very usable.

It was also very rough. I had to start X manually each time, which I later fixed. I had to unmute ALSA every time, which I never quite solved. There was a lot of configuration left to do. I also wanted MythVideo and MythDVD to work. So I started on these extra functions, and this is where

I got hung up in the end. MythWeather worked in one shot. MythDVD was particular, but far easier to fix than in RedHat. Some minor searches on the Internet provided me with the required configuration settings, and I was able to get it working, for the most part, in a few hours. But now I was hitting hardware limits. After all, I was using hardware that was current in 1999 or 2000, not 2004. MPEG-4 playback was impossible, and background processes like ripping or converting video took up all the processor time far too easily. I was also unhappy with the unbelievable amount of noise generated by the six different fans in the case.

A Sidebar on Hardware

This also marked my first real attempt at TV-Out. I was using a low-end nVidia AGP card at the time that included a TV-Out. Before I get jumped over this, let me say that I have good friends at nVidia who really, truly know their stuff. But I'm also telling you, if I ever try to use an on-card TV-Out again, I will poke my eyes out. It's awful. The same is true of my ATI cards. I don't know the hardware specifics of why, but it is. But as I was having trouble with the configuration that would pipe the video correctly to the TV-Out of the Hauppauge PVR-350 card, I stuck with the nVidia TV-Out. Please note that MythTV's support for the Hauppauge card has improved greatly since this time.

At this point I'm going to digress into the wild and wooly topic of hardware in the living room. This is a matter that people vary on greatly, from the silence zealous to the power zealous. I'm going to try to give you an abbreviated version based on my experience, simply so that you can see how it might affect you. First of all, my system was far too loud. Vacuum cleaner decibels. Unwatchably loud. The images were on the screen, but you couldn't hear over the system until your stereo was Far Too Loud. After about three days, I just yanked the plug and the house as a whole gave a cry of relief for the lack of noise. In two weeks I had substituted a Zalman ultra-quiet power supply and (from the junk bin) a quieter hard drive, and was far happier. But it was still a four-year-old system doing things that were over its head, and the video quality was really starting to wear on us. To compensate, I got the remote control working. This is when my wife first started showing signs of interest.

Concerning hardware as a whole, though, I was able to find many near-silent PCs designed for the living room, ranging from $100 "quiet kits" to $1500 complete silent systems. I was still being a cheapskate, so I watched and bought nothing. What killed this system was the unmute-by-hand problem between ALSA and the 2.4 kernel. By this time a 2.6 kernel version of Gentoo was out, and the audio problem and video quality were the system's deathblow.

Attempt 2.5: SageTV

Honestly, I'm glad I tried SageTV. SageTV is a commercial product that uses the same hardware, only it runs Windows. I converted my Gentoo box to Windows 2000 and installed SageTV. To their credit, in about an hour the thing ran. But it never ran for more than a day. Blame it on whatever you want: hardware, OS issues, driver issues, or anything else, but the stability problems were formidable. I eventually set it up to reboot every day at 3 a.m. to try to keep it going. It would still occasionally lock up and destroy a show or two. I have no idea what the real causes were, but the instability of the hardware and software in combination killed this configuration.

What I did love about this solution, however, was that the shows were stored as MPEG-2 and were easily shared to a PC with a DVD burner, and the files had nice obvious names. With some basic DVD authoring software, I was able to burn DVDs of my kids' favorite shows at broadcast quality.

I must also mention that this was the first time the PVR-350 TV-Out worked. And let me tell you, it is a beautiful thing. It's not double-interlaced, it's not desaturated, it's . . . right. It looks like it comes out of my VCR instead of an overhead projector. This alone is what kept me plugging away at SageTV for almost two months.

Then MythTV added the ability to detect and flag commercials, and automatically skip them. Between SageTV not having all the things I wanted and the stability problem, I stopped trying. SageTV is a base PVR that worked with little configuration and used very little CPU time, and I'm glad I saw that it is possible. But I wanted the rest of the checkpoints on my list. Also note that SageTV now has a DVD and video player built in.

Third Try: KnoppMyth 5.1

A number of people were talking about how easy it was to slap a system together with KnoppMyth. So I downloaded and burned KnoppMyth 5.1, which included MythTV 0.18 (iirc), which was just out and included all the features on my list.

By now, I realized I was going to have to break down and get some dedicated hardware for this adventure if I wanted everything. I settled on a VIA Epia M-10000 system and a quiet case with an integrated low-noise power supply. As far as decibels go, this combination is great. The hard drive is still the loudest part of the system, but the total cash investment was $400 including RAM.

KnoppMyth is a dream to install when it works. It does a very competent job of detecting your hardware and bringing up the default configuration. But it took three tries to get the KnoppMyth installer to finish correctly. This may have been an anomaly, but it made me skeptical. I was pleasantly surprised to find that the KnoppMyth distribution included a Nehemiah kernel. It even included the ivtv capture driver. If you look hard enough, there are instructions that show how to put video out the PVR-350 card as well.

After the base install and 20 minutes of configuration, I was again able to watch live TV on the monitor, as well as use most of the other base functions. As long as I was satisfied with just MythTV, MythVideo, MythWeather, and had a monitor or TV with VGA to work with, the system had almost zero bring-up time and was truly a delight.

But it was fraught with other bugs that made it harder and harder to finish and polish. I could not watch or rip DVDs (later fixed). Configuration of the TV-Out of the PVR-350 took me *much* too far from the "click and go" configuration I was after. I rapidly ended up in the same swarm of one-off versioning problems I had with RedHat, plus some bugs peculiar to KnoppMyth, having to retrieve and hand-install everything. The mire deepened, and I moved back to . . .

Fourth Try: Gentoo 2005.1

This one was by far the most successful install. I decided to stage the system on a full AMD Athlon 1GHz system with 1GB of RAM—an otherwise quiet system that I had used as my Windows machine for two years. The base install of Gentoo had changed while I was away, but not by much. This time through I used the binary install for most basic packages, probably saving me a full day of compiling.

After the magic “emerge mythtv,” I left for about an hour and came back to find everything compiled and waiting. Then I unzipped my old configuration files and copied the relevant parts onto the new system. Before long I was able to get to a full KDE 3.4, MythTV 0.18, MythDVD, MythWeather, MythVideo, TV-Out on the Hauppauge PVR-350, and all associated hardware acceleration. Everything I wanted to work was working; I couldn’t believe the day had come. The sound was set correctly at boot time, the video hardware acceleration was fine, and I was able to watch MPEG-4s flawlessly. I was a happy camper. So I packed it into the trunk and hauled it off to show a friend. On the drive home, the extra-super-quiet-all-copper heatsync cracked the stinking processor. Needless to say, I was miffed. I just didn’t feel like dealing with trying to find a three-year-old processor for my generic brand motherboard. I took the hard drive out, placed it in the Nehemiah system, and started reconfiguring.

Fifth Try: Gentoo 2005.1 Again (Success!)

After recompiling the kernel, changing a couple of drivers, and recompiling mplayer, almost everything was working again. Setting X to use the Hauppauge PVR-350 meant that I didn’t have to reconfigure xorg or video drivers. The basic features are currently working, and it is far more stable. It goes a week or more before needing to relaunch MythFrontend. I suspect this is because the front end crashes and I don’t have a way to kill it with the remote. Because of the difference in true processor speed, I’m not able to get full speed out of most MPEG-4s.

Now that the basics are working, I’ve started working on other attached projects. I have become OS-agnostic over time, and have a pair of Mac OS X systems in the house. Someone has a precompiled MythFrontend for OS X that was able to connect to my master system, read the program guide, and play video (once I ran 100bT to the front room). I was very, very impressed. There is also the opportunity to set up secondary Linux front ends so you can have the master system do recording (headless if you like) in one room and have other systems watch throughout the house, networks and bus bandwidth permitting.

One thing happened very recently that brought me to a standing halt: Digital Cable. My wife accepted some special or another from the cable company, and they came out and installed one of those vile digital channel changer boxes. MythTV does have support for external channel changers, but I don’t have the external IR broadcaster cable. A friend of mine has built one, but says that they are wily beasts to set up. We immediately switched back to “Standard Standard Cable,” and things have been fine since. Some people have great success in using such changers, but if I can get away without one, I will.

Results and Overall Reaction

If I weren’t so stubborn about the other requirements on my checklist (skipping commercials, DVD caching, etc.), I would probably be better off with a commercial player and their service. But as TV is not the prevailing entertainment in my life, I am happy to set up my own recorder, play with the internals, get it working, and often end up with shows and videos I am interested in when I want them and with fewer ads per hour.

What I see as the biggest threat to this type of system is the difficulty in getting show listings. Since I started working with MythTV, there have been three separate systems for getting show listings. The first required nothing but was slow. The second required a signup and then a survey every three months to keep it going. The third is a cheap subscription with proceeds going back into the MythTV development community. There is no permanent answer here that I can see.

While I'm glad I set it up, I realize this system is not for everyone. But I have been thoroughly impressed with the results. I don't want to sound preachy or prophetic, but I feel that this is one of the things that has excited people about computers for years—the possibility that all these functions could be combined into one box. I hold with the conclusion of one of our local reporters: "The time of Appointment TV is fading." But as long as I have some control over what features I get to use, I'll be glad to go out of my way to get them.

INSTRUCTIONAL AND OTHER RESOURCES

MythTV and subprojects: http://www.mythtv.org/

Wiki: http://www.mythtv.info/

Mailing list archive: http://www.gossamer-threads.com/lists/mythtv/users/

ivtv (hardware driver for MPEG encoders): http://ivtvdriver.org/index.php/Main_Page

nuv2disc (burning directly from MythTV): http://extras.mythtvtalk.com/install_htm.html

One extremely complete install guide: http://wilsonet.com/mythtv/

book reviews

ELIZABETH ZWICKY

zwicky@greatcircle.com

with Richard Johnson, Sam Stover, Steve Manzuik, Ben Rockwood, and Ming Y. Chow

THE UNOFFICIAL LEGO BUILDER'S GUIDE

Allan Bedford

No Starch Press, 2005. 319 pp. 1-59327-054-2

The process of selecting books to review is, to put it politely, organic; it involves complex variables such as my level of interest in the topic, my level of knowledge about the topic, my estimation of readers' levels of interest, the other books in the stack, and whether or not I think a book is cool. Which is all by way of saying, no, Lego does not have much to do with advanced computing systems, but I think it's cool, and I'm betting a fair number of you do, too.

This book is cool. It's not rocket science, although there is a nice walk-through of how to design a space shuttle model. It would be a great Christmas gift for the person on your list with the big Lego collection and no very focused idea of what to do with it. You might be more reluctant to give it to anybody in your own household, as the storage suggestions may result in the reader developing entirely new ideas of the scale of a "big" Lego collection, and wanting closets-full. If you already have closets full of Lego, this book will give you the graph paper and the ideas to turn it into Lego cities, or whatever. It's suitable for older kids and young-at-heart adults. And you can feel good about giving it to kids, because it teaches some nice mathematics about ratios, making it genuinely educational.

I learned some neat stuff (the thin Legos are exactly 1/3 the size of normal-height ones), and it's my 18-month-old's second-favorite of the books I've reviewed, because it led me to build things she likes out of her Duplo. (Her favorite is a hardback with a penguin on it. She likes the penguin and finds it an especially intriguing size, for some unknown baby reason.)

THE LINUX ENTERPRISE CLUSTER: BUILD A HIGHLY AVAILABLE CLUSTER WITH COMMODITY HARDWARE AND FREE SOFTWARE

Karl Kopper

No Starch Press, 2005. 430 pp. 1-59327-036-4

Suppose you know not very much about Linux, and less about clusters, and somebody comes to you and says, "Hey, here's a pile of computers; build a cluster out of them, and, oh, by the way, we want to run business-critical software on it." If you sit down with this book and follow it through, at the end, I am convinced, you will have a reasonable solution to that problem. I don't know that it will be the best possible solution; this book walks through one particular set of tools, which undoubtedly won't be the best for every situation. I'm sure that serious Linux cluster aficionados will argue passionately about the author's choices. But there's no avoiding that problem if you want to explain the nuts and bolts of using a particular solution, which the author does very nicely.

The authors take an unusual but effective approach: they walk you through detailed recipes for setting up, not the production environment, but a test environment where you learn how all the parts work and how you can customize them for your purposes. This makes a nice balance between detailed, hand-holding exposition and getting the concepts you need to be able to extend the recipes into your environment.

If you already know something about clusters and Linux (or general UNIX system administration), go straight to chapter 5, bypassing the very general discussion of what a cluster is and a lot of background on kernel builds, SSH installation, rsync, and the like.

WEB MAPPING ILLUSTRATED

Tyler Mitchell

O'Reilly, 2005. 349 pp. 0-596-00865-1

Here's another one I think is cool. (Though it's about maps *on* the Web, not maps *of* the Web, which might have been even cooler.) I like maps, and this book made me want to run right out and add gratuitous maps to my Web site. Better yet, it made me think that the next time I'm on a project where the right thing is to put up an interactive map on a Web server, I will have an answer that doesn't involve all the Web programmers saying glumly, "Gee, that sounds really hard." (That's what happened the last two times, and I didn't get my interactive maps.) True, it wouldn't take 349 pages to explain it if it were really easy, but *Web Mapping Illustrated* tells you how to get and use open source tools to do powerful things with maps, with some basic information on getting and generating the data to go along with the tools. It's enough information to get people past the fear of the unknown.

One caution: it's meant for people who understand maps and want to put them on the Web. It gives some basic background for people who understand the Web but don't know much about maps, but it's probably only enough to make somebody like me able to make real mapmakers writhe in pain. If you want respectable maps, you're going to need either to be very conservative or to get somebody who knows a lot about maps. This book will take you past the edge of your mapmaking competence and induce the map equivalent of the ransom-note typography that was so popular when word processors first let amateurs play around with fonts. But hey, it's fun to do, even if it's not always fun to watch.

HP-UX 11I VERSION 2 SYSTEM ADMINISTRATION: HP INTEGRITY AND HP 9000 SERVERS

Marty Poniatowski

Prentice Hall, 2005. 643 pp.
0-13-192759-0

If you are an experienced administrator looking for information about HP-UX commands, particularly those specific to HP hardware, you may find some information of interest here. However, the book does not go into enough depth for my taste (it talks about how to use HP's remote install process but not about its underpinnings) and doesn't have enough detail for an inexperienced administrator (it says the author usually modifies the default partition layouts, but doesn't talk about how or why). It is also security-naive; while the author does make some gestures toward security, suggesting that hosts.equiv and .rhosts be used cautiously, he doesn't warn administrators that 6 characters is not a reasonable current minimum password length, that scanning your own network is liable to annoy not just the network administrators but also the security people, that remote SNMP system management has security implications, or that giving nonprivileged users backup and restore privileges has security implications. On the whole, I can't recommend this book. In most situations, you'd be better off with a good, general system administration book and HP's documentation.

PRACTICAL DEVELOPMENT ENVIRONMENTS

Matthew B. Doar

O'Reilly, 2005. 297 pages.
ISBN 0-596-00796-5

Most books that I review will eventually find their way to more appropriate homes. A few I keep a good tight grasp on. This is one of those few. I've put up with a wide range of development environments, and I understand varying parts of them to varying extents. But I don't understand them in the same way that I understand the ins and outs of a data center, for instance. This is a structured overview of all the parts that go into a development environment, with specific examples, comparisons of the good and bad points of common tools, and questions to apply to your own environment. In other words, it's exactly what I need to help me get to the point where I understand development environments as well as I do the system administration environments I've built from the ground up.

It covers software configuration management, build tools, bug tracking, testing, documentation, release, and maintenance, and it gives equal weight to commercial and open source solutions. Its advice is consistent with my experience; yeah, those common problems really are common.

This book will be most useful if you are building a development environment, or if you want to be a toolsmith (somebody who supports programmers directly, working on the tools that let them do development). But if you're just entering the wild and woolly world of programming and you want a scorecard so you can tell the players apart, it'll help you too. And I strongly recommend it for system administrators who support programming teams.

BEHIND CLOSED DOORS: SECRETS OF GREAT MANAGEMENT

Johanna Rothman and Esther Derby

The Pragmatic Bookshelf, 2005.
167 pages. ISBN 0-9766940-2-6

This is a nice, small book on how to be a good manager, aimed at people working in large development environments. Its advice is entirely sensible (that is to say, I agree with it). There is nothing earth-shattering here, but there shouldn't be; good management books agree with each other and say mostly commonsense things that are easy to read and hard to implement. Its most radical move is a good, easy-to-swallow presentation of communication issues: how and, most important, why to say nice things about other people. I think this is an important issue for technical people, who tend to think "communication skills" is a management buzz phrase for "talks nonsense and wears a nice tie," whereas it's actually a management buzz phrase for "not torture to be around."

One of this book's strengths is that it gives nice, concrete examples. This is going to be most useful for people who're working in the sort of corporate product development environment that their examples are drawn from. The concepts are useful anywhere, but if you need the examples, you may find that these don't work as well for you if you're in a different environment.

If you are moving into management and want a short introduction to important management skills that respects technical people and explains things understandably without condescension, this is a nice place to start. You'll probably find yourself consulting some of its many references as you go forward, but just following its advice will go a long way toward making you a productive, useful manager.

AMBIENT FINDABILITY

Peter Morville

O'Reilly, 2005. 188 pages.
ISBN 0-596-00765-5

If the title didn't suggest to you that something was odd about this one, the cover would; the animal on it is in color, but it's otherwise a traditional O'Reilly cover. It's not a traditional O'Reilly book. No, you have not missed the release of a new programming tool called "Ambient" or "Findability." This is a book about the ways in which the ability to find things is changing the world.

It's an amusing and interesting book, and it convinced me that, yes, findability is really important. There's lots of inspiration here for designers of all kinds. At the same time, I found it ultimately frustrating. It feels like there ought to be a deep structure and some fundamental insights, but all I got was a bunch of neat stuff.

This book is definitely a good time, and it should be particularly enlightening to people just outside the world of the Web, or just entering it. It's full of pretty pictures and clever ideas; if you know somebody intelligent who doesn't understand why the Web really, really matters, this book should get the point across.

DATA PROTECTION AND INFORMATION LIFECYCLE MANAGEMENT

Tom Petrocelli

Prentice Hall, 2005. 256 pages.
ISBN 0-13-192757-4

This book covers data protection, starting from the types of data storage (everything from good ol' disks flung in a server through SAN and NAS, with explanations of SCSI and ATA and all of their cousins), through backup and restore, data replication, security, policies, and, finally, a brief flourish about managing not just data but information. That's a lot of stuff to try to get into 256 pages, and, in the end, I don't feel that it all fit well enough.

Most topics are covered only at a high, abstract level, which makes them hard to understand and apply. Furthermore, there are some odd omissions. For instance, in the chapter on backups, a number of failure modes are mentioned, but there's no mention of backup verification or testing, which is an obvious and important part of data protection. In the chapter on storage systems, there's no mention of RAID 4. In other places, you could be led to dangerously wrong conclusions: on-disk data encryption is not a panacea, and being able to back up open files is useful only if you have some reason to believe they're consistent enough to be usable when you restore them.

HOST INTEGRITY MONITORING WITH OSIRIS AND SAMHAIN

Brian Wotring and Bruce Potter (technical editor)

Syngress, 2005. 420 pages.
ISBN 1-597490-18-0

REVIEWED BY RICHARD JOHNSON

rjohnson@ucar.edu

The title of this book might lead you to expect a how-to manual for building and operating Osiris and Samhain. It does indeed contain such, but the book is far more useful than that. It's also intended as a "why" manual which starts by helping you answer the very basic question of whether you, on your particular machines, even need or want to use host integrity monitoring. Beginning with the why portion, the first four chapters concisely but not too choppily define integrity monitoring, describe what typically needs to be watched, cover typical attacks and the changes they'll produce (with examples of automated worms), and delve into the planning crucial for setting and meeting your specific monitoring goals. The next three chapters get into the how-to of installing and operating host integrity monitoring software, with a chapter each dedicated to Osiris and Samhain. Finally, the book covers stepping beyond the simple change notification facilities built into each of the systems, responding to incidents detected by the system, and more advanced countermeasures or pitfalls. The book flows well from chapter to chapter, particularly with the summaries at the end of each, which, somewhat amusingly, turned out to mirror my personal notes. I found it easy to read with comprehension from cover to cover.

Although the book's target audience consists of experienced system and security administrators (call it SAGE II+), the first half is also useful for technically inclined managers. It's a nice design, as it gives those of us in charge of the implementation a solid hook for bringing our superiors up to speed. Also in this portion, the discussion of what to monitor was particularly valuable. Even with 19 years of experience as a sysadmin, I gained new insight from the discussion of where (possibly malicious) changes can hide in various OSes. More important, the foundation here makes extrapolation to OS features that weren't in use before the book was written (e.g., Mac OS 10.4's new use of arbitrary file metadata in HFS+, which can be used to hide data) almost inevitable for a technical reader.

The build chapters don't rehash the man pages or release documentation for Osiris or Samhain. Instead, they evenly cover the strengths and weaknesses of each package, followed by build and installation tutorials with a clear eye to avoiding later management problems and mitigating security risks. Even picking a system that I don't normally deal with—building and installing an Osiris client on MS Windows—I found it easy to follow the instructions. Furthermore, the Samhain build chapter was met with, “Nice, I didn’t know that before, I’ll have to add that,” from a fairly experienced Samhain user.

A minor downside was that PGP signature verification instructions for the source code distributions in each of the build chapters were redundant. Particular advice for setting ultimate trust on a PGP key in Chapter 6, which left me feeling uneasy, was later qualified with a caveat in Chapter 7, but otherwise the sections might better have been consolidated. Somewhat more annoyingly (even though I understand why it is this way; it’s another book’s worth of material in itself), it would have been nice to see more advanced material on log monitoring, focusing on additional tools that can help us intelligently aggregate and process the tagged syslog output or database entries, specifically from Osiris and Samhain.

In the end, the book carries readers along, educating them and leaving them wanting more (with an idea of where to go to get that more). If you’re thinking of trying host integrity monitoring, though without the noise and maintainability problems common to such systems in the 1990s, this book will serve you well. More important, this book will help you figure out why you want to monitor host integrity in the first place, and then tune what you monitor to meet your goals.

FILE SYSTEM FORENSIC ANALYSIS

Brian Carrier

Addison Wesley, 2005. 569 pages.
ISBN 0-321-26817-2

REVIEWED BY SAM STOVER

sam.stover@gmail.com

I think this book is hands-down the best resource for file systems (FAT, NTFS, EXT2/3, and UFS1/2) and partition types (DOS, Apple, BSD, GPT, and Solaris Slices) I’ve read. It is not, however, designed as an introductory guide for a novice forensic analyst. The author does not focus on walking the reader through evidence handling, chain of custody, etc., nor does he focus on tutoring the reader in the use of common forensics tools such as EnCase. The goal of this book is to provide a foundation for a forensics investigator to work from, and I think it achieves that goal. And, as mentioned, it serves as a great reference for anyone doing any kind of file system and/or partition work.

A lot of the information in this book is available in other forms such as RFCs, vendor standards, etc., but this book brings everything together in one place. The book starts by giving a brief overview of the principles involved in digital forensics investigation, but then moves quickly into a low-level discussion of the different types of partitions used. From there, the aforementioned file systems are examined in intimate detail.

Prior to reading this book, I felt that I had a pretty good grasp of the different file systems and how they are put together. After going through the NTFS chapters, I soon realized how much I didn’t know, and I suspect that a lot of forensic investigators fall into the same trap. Current forensic tools do a lot of the heavy lifting with respect to file system analysis, and thus they make it too easy to conduct an investigation without completely understanding everything from the ground up.

In the way that W. Richard Stevens has provided us with an invaluable reference for the TCP/IP protocol stack, Carrier has given us an analogous reference for partitions and file systems. To further the analogy, some of the material in this book is very complex and could require a fair bit of effort from the reader to fully grok the file system or partition in question.

One last comment is that the author’s toolkit, called The Sleuth Kit (TSK), is used throughout the book to demonstrate the examples. I stated earlier that it was not a goal of this book to tutor a user on a particular forensic tool, and I stand by that. This book does not teach you how to use TSK, but there is a seven-page appendix that gives you the basics so that you can use the tool yourself to emulate what’s happening in the book.

Overall, this book is definitely a “must have” for anyone who wants to learn how file systems work—whether that is to be applied to forensic analysis or otherwise. It will occupy a space on my bookshelf right next to *TCP/IP Illustrated*, and will probably be referenced just as frequently.

ROOTKITS: SUBVERTING THE WINDOWS KERNEL

Jamie Butler and Greg Hoglund

Addison-Wesley, 2006. 324 pages.
ISBN 0-321-29431-9

REVIEWED BY STEVE MANZUIK

hellnbak@gmail.com

As someone who has been called a Windows security expert, I always find it a pleasure to come across a technical book that covers subject matter that is perhaps a little less known than your standard Windows security concepts. It is even a greater pleasure to find that same book is able to teach even an experienced security geek such as myself a few new tricks.

Most people, and when I say most people I am referring to myself, use technical books as a way to cure insomnia, while occasionally learning something along the way. Although *Rootkits: Subverting the Windows Kernel* is high on technical content, I found myself doing more learning than sleeping. When I first sat down to read

Rootkits, it was 2 a.m. and I was ready for something to put me to sleep. Instead, I found myself hopping out of bed to grab my laptop and make note of some of the techniques taught in the book.

The concept of a rootkit has been around for a very long time, especially in the *NIX world. Over the years we have seen them evolve from lame hacker tricks to more in-depth, harder-to-detect subversion methods. I have had many different roles in my career, but before landing in the eEye Digital Security Research Department I was an independent security consultant. I performed many incident response engagements for clients, which usually involved some sort of Linux rootkit installed on compromised systems. Butler and Hoglund have taken the concepts of the "old school" rootkits and applied them to the "new school"—Microsoft Windows. So if you are a Windows person, or interested in the Windows kernel, this is a book for you. Be sure to also check out the accompanying Web site (http://www.rootkit.com): you will find all kinds of samples used in the books and a great discussion forum where you can exchange ideas with the authors as well as with other security geeks.

Butler and Hoglund take the reader through the technical details of Windows rootkits, sparing nothing. This book is filled with useful information that will help reader understand exactly what a rootkit is, how they are used, and how to create your own rootkits that can subvert various detection routines. Of course, this book would not be complete without information on how to build a good host-based intrusion prevention system to resist such attacks.

Whether you are a junior security person or one of the old-timers in the industry, I highly recommend this book to you. I have spent a little over a decade in the IT and IT security industry, and I found this book complete enough to leave some new knowledge in my brain.

Oh, and if you are having trouble sleeping, fire me off an email and I can give you a list of the books in my library that actually do help put one to sleep.

ORACLE PL/SQL FOR DBAS

Arup Nanda and Steven Feuerstein

O'Reilly, 2005. 429 pages.
ISBN 0-596-00587-3

REVIEWED BY BEN ROCKWOOD

benr@cuddletech.com

Oracle PL/SQL for DBAs is O'Reilly's latest addition to its ever growing series of Oracle PL/SQL books, most of which are written by or co-authored with guru Steven Feuerstein. Unlike previous titles, such as *Learning Oracle PL/SQL* or *Oracle PL/SQL Programming*, this book is squarely aimed at experienced DBAs looking to better leverage capabilities of the database through PL/SQL interfaces. For the sake of completeness, the first chapter contains a whirlwind tour of PL/SQL from the ground up, but readers new to PL/SQL will find themselves lost in the dust.

The book focuses on three main topics: security, performance, and scheduling. In the performance category is in-depth discussion of cursors and table functions. Both chapters are chockfull of solutions and ways to better craft your queries and write your procedures, but little in the way of theory is offered. Security is covered by discussion of encryption within the database, auditing, row-level security, and a great chapter on generating random values. Scheduling is handled in a single chapter but supplies a great deal of insight on the topic.

Again, this is a book for experienced DBAs. It answers the question "How?" but not the question "Why?" Almost no background on cursors is given, for instance, making you reach for Google or your favorite DBAs reference. And the introduction to encryption is good for a laugh but little more. Clearly the authors have a talent for demonstrating functionality but have left background explanations for other, more suitable guides.

Perhaps the book's most redeeming value is the sense it provides of just how much can be done from within Oracle itself. The scheduling chapter, for instance, would be of great use to DBAs who have become too reliant on cron. While DBAs and sysadmins won't need every feature outlined in this book, there is clearly value in realizing what's available and using that information to better leverage your existing deployments.

PRIVACY: WHAT DEVELOPERS AND IT PROFESSIONALS SHOULD KNOW

J.C. Cannon

Addison-Wesley, 2005. 347 pages.
ISBN 0-321-22409-4

REVIEWED BY MING Y. CHOW

mchow@eecs.tufts.edu

Cannon delves into all facets of privacy, low-level and high-level. As indicated by the subtitle, the book targets developers and IT professionals, but I would recommend it for end users and managers as well. The first half provides a very comprehensive overview of privacy. Privacy-Enhancing Technologies (PETs) and Privacy-Aware Technologies (PATs) are presented, with numerous examples and features. Cannon also discusses privacy frameworks and legislation, including the Health Insurance Portability and Accountability Act (HIPPA), the Gramm-Leach-Bliley Act (GLBA), and even the Digital Millennium Copyright Act (DMCA).

The privacy issues with spam and emerging technologies (e.g., RFID tags), including solutions to mitigate privacy risks, are discussed in perceptive detail.

For developers, Cannon provides rich insights and effective techniques for incorporating privacy into both the development process and the products themselves, including discussions of privacy analysis, privacy specification, dataflow diagramming, and database protection. He presents a large-scale example of development from top to bottom. There are even checklists and templates that managers and developers can use immediately.

The public's growing concern about privacy is well founded, and they are demanding that government, business, and developers step up their efforts to preserve privacy. Cannon does a tremendous job of stressing the importance and value of integrating privacy into products and in business and, more important, in explaining how to do so. Throughout the book, he emphasizes trust, incorporating privacy into the development process early, and enabling users to control their own privacy. It is a huge and important challenge today to give end users security they can understand and privacy they can control. I recommend this book without reservation to those who want a competitive edge in this dire field.

USENIX Membership Updates

Membership renewal information, notices, and receipts are now being sent to you electronically.

Remember to print your electronic receipt, if you need one, when you receive the confirmation email.

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

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

You are welcome to print your membership card online as well.

The online cards have a new design with updated logos—all you have to do is print!

USENIX notes

USENIX MEMBER BENEFITS

Members of the USENIX Association receive the following benefits:

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

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

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

ONLINE JOBS BOARD for those who are looking for work or are looking to hire: http://www.usenix.org/jobs/

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

DISCOUNTS on registration fees for all USENIX conferences.

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

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

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

USENIX BOARD OF DIRECTORS

Communicate directly with the USENIX Board of Directors by writing to *board@usenix.org*.

2006 USENIX NOMINATING COMMITTEE REPORT

MARSHALL KIRK MCKUSICK

Chair, Nominating Committee

The USENIX Association is governed by its Bylaws and by its Board of Directors. Elections are held every two years, and all eight Board members are elected at the same time. Four of them serve as at large and four also serve as statutory officers—President, Vice-President, Secretary, and Treasurer.

Per Article 7.1 of the Bylaws of the USENIX Association, a Nominating Committee proposes a slate of Board members for the membership's consideration. As a practical matter, the purpose of a Nominating Committee is to balance continuity and capability so as to ensure that the incoming Board is composed of persons shown by their actions to be both dedicated to the Association and prepared to lead it forward.

The USENIX Nominating Committee is pleased to announce the list of candidates for the upcoming USENIX Board of Directors election:

President: Michael B. Jones, Microsoft

Vice-President: Clem Cole

Secretary: Alva Couch, Tufts University

Treasurer: Theodore Ts'o, IBM

At Large:

Matt Blaze, University of Pennsylvania

Gerald Carter, Samba.org/Centeris

Rémy Evard, Argonne National Laboratory

Niels Provos, Google

Margo Seltzer, Harvard University

The committee is very pleased that all of these individuals have agreed to commit their time to serving the USENIX Association.

"SAY HELLO, GRACIE." "HELLO, GRACIE."

Memo to sage-members mailing list from SAGE's new leader, sent on December 30, 2005

STRATA R. CHALUP

strata@sage.org

Hi folks,

As you've probably heard, I've now been the SAGE Programs Manager for USENIX for an entire week. . . .

I've got two working mantras in this job:

1) It's a community, not a contest.

2) What have we done for you *lately*?

I squirreled away a lot of ideas from the old SAGE Exec brainstorming sessions we had way back when I was a SAGE Exec. I'm in the process of going over those so I can ask y'all, the [SAGE] members, what are priorities for you. I've heard some opinions on that already, but more doesn't hurt. Some things people want seem to be competently underway by LOPSA [the League of Professional System Administrators]. Some still make sense for SAGE to implement as a convenience for its members, and others might not. There are lots of opportunities for cooperation, and I'll be talking to Tom Perrine and other LOPSA Board folks about this in 2006. . . .

cheers,
Strata
BayLISA member since 1996
USENIX and SAGE member since 1997
LOPSA member since Dec. 2005

[Please let us know what you'd like to see USENIX provide for system administrators. Send your comments and suggestions concerning benefits/programs/direction to ideas@sage.org.]

SUMMARY OF USENIX BOARD OF DIRECTORS MEETINGS AND ACTIONS

TARA MULLIGAN AND ELLIE YOUNG

The following is a summary of the actions taken by the USENIX Board of Directors from October 21, 2005, to December 20, 2005.

NEW CONFERENCES

First USENIX Security-Verified Electronic Voting Workshop: This event will be held in conjunction with the USENIX Security Symposium in August 2006. The workshop will focus on issues surrounding functionality and security of electronic voting. Ron Rivest will serve as program chair.

HotAC: USENIX will co-sponsor, with IEEE, the First Workshop on Hot Topics in Autonomic Computing (HotAC '06), which will focus on the complexity of large-scale systems (http://www.aqualab.cs.northwestern.edu/HotACI/). Alva Couch will serve as board liaison.

FINANCES

The Board approved the first draft budget for 2006. It does not include any raises in member dues or conference registration fees. The following requests for funds were approved:

- Sponsorship of the Computing Research Association's Snowbird Conference at the $3,500 level.
- John Lions Endowed Chair in Computer Science: This endowment has been established at the University of New South Wales, Australia, to honor a pioneer in UNIX and computer science. The endowment is presently funded for twenty years. In order to assist with funding that will make it permanent, USENIX will match funds donated by members up to a total of $250,000 during 2006. Members and the community at large will be asked to donate.
- Standards Activities: $54,000 will be allocated in 2006 to support the development of international standards in computing. Specific projects planned in the upcoming year include a C++ POSIX binding project, new library interfaces, and continuing Linux Standards Base work.
- The USA Computing Olympiad (USACO) supports pre-college students who have shown an interest in and promise for careers in computer science. USENIX will continue its support with a $15,000 grant.

SAGE

In 2005, despite a lengthy negotiation that consumed hundreds of volunteer and staff hours from representatives of both USENIX and the SAGE Inc. organization, now called The League of Professional System Administrators (LOPSA), about possible terms for having the new group become SAGE, no workable agreement emerged. Therefore the USENIX Board feels that it is in the best interests of USENIX and LOPSA to focus our respective efforts on building our own organizations for the good of our members. We do not believe that any continued negotiations on funding of LOPSA by USENIX under the present framework would be worthwhile, and any new proposal must ensure that it addresses the issues regarding asset transfer and funding previously raised by the Board. It is generally felt that any new proposal would have to be substantially different in order to satisfactorily address these issues.

The USENIX Board welcomes proposals for cooperation between LOPSA and USENIX where it makes sense to both organizations.

USENIX continues to be committed to serving its SAGE members and the system administration community both now and in the future. The USENIX Board will be actively looking at ways to refine our services for all our members, and your input to us is an important part of this process.

Members wanting to give the Board guidance about shaping future programs for system administrators should send their suggestions to suggestions@sage.org.

The USENIX Board formally thanked Rob Kolstad for his dedicated service to USENIX and SAGE over the years. (He resigned from his part-time role in SAGE in December 2005.) Strata R. Chalup has accepted the position of SAGE Programs Manager. See her introductory message on p. 91.

COMMITTEES

The SAGE Committee—Couch (chair), Jones, McKusick, Young—will oversee new services and benefits for SAGE members. The Audit Committee—McKusick, Ts'o, Geer, Young—will oversee the activities of the CPA firm that has been engaged to conduct the annual audit of USENIX financial statements in the spring of 2006, mandated by new legislation by the state of California.

NEXT MEETINGS

The USENIX Board agreed to hold several strategic meetings in 2006 to seek new topics for conferences and ideas for expanding member services. It will meet on April 1 in Berkeley, CA, on May 8 in San Jose, CA, and on May 31 in Boston, MA.

NEW!

;login: Surveys

To Help Us Meet Your Needs

;login: is the benefit you, the members of USENIX, have rated most highly. Please help us make this magazine even better.

Every issue of *;login:* online now offers a brief survey, for you to provide feedback on the articles in *;login:* . Have ideas about authors we should—or shouldn't—include, or topics you'd like to see covered? Let us know. See

http://www.usenix.org/publications/login/2006-02/

or go directly to the survey at

https://db.usenix.org/cgi-bin/loginpolls/feb06login/survey.cgi

Announcement and Call for Papers **USENIX**

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

Sponsored by the USENIX Association

http://www.usenix.org/sruti06/cfp

July 6–7, 2006 **San Jose, CA, USA**

Important Dates

Submissions due: *Thursday, April 20, 2006, 0400 UTC*
Notification of acceptance: *Thursday, May 18, 2006*
Final papers due: *Tuesday, June 6, 2006*

Conference Organizers

Program Chair

Steven M. Bellovin, *Columbia University*

Program Committee

Harald Alvestrand, *Cisco*
Dan Boneh, *Stanford University*
Jon Crowcroft, *Cambridge University*
Anja Feldmann, *Technische Universität München*
John Ioannidis, *Columbia University*
Balachander Krishnamurthy, *AT&T Labs—Research*
Chris Morrow, *UUnet*
Vern Paxson, *ICIR/ICSI*
Niels Provos, *Google*
Eric Rescorla, *Network Resonance*
Tara Whalen, *Dalhousie University*

Steering Committee

Clem Cole, *Ammasso, USENIX liaison*
Dina Katabi, *Massachusetts Institute of Technology*
Balachander Krishnamurthy, *AT&T Labs—Research*
Ellie Young, *USENIX*

Overview

The Internet is under increasing attack, with unwanted traffic in the form of spam, distributed denial of service, viruses, worms, etc. Unwanted traffic on the Internet has manifested itself as attacks via many protocols (IP, TCP, DNS, BGP, and HTTP) and popular applications (e.g., email, Web). Often these attacks have a direct economic motivation. SRUTI seeks research on the unwanted traffic problem that looks across the protocol stack, examines attack commonalities, and investigates how various solutions interact and whether they can be combined to increase security. Original research, promising ideas, and steps toward practical solutions at all levels are sought. We look for ideas in networking and systems, and insights from other areas such as databases, data mining, and economics. SRUTI aims to bring academic and industrial research communities together with those who face the problems at the operational level. SRUTI is a one-and-a-half-day event. Each session chair will play the role of a discussant, presenting a summary of the papers in the session and a state-of-the-art synopsis of the topic. The workshop will be highly interactive, with substantial time devoted to questions and answers. Submissions must contribute to improving the current understanding of unwanted traffic and/or suggestions for reducing it. All submissions to SRUTI '06 will be via the Web submission form, coming soon to http://www.usenix.org/sruti06/cfp. The Proceedings of the workshop will be published. To ensure a productive workshop environment, attendance will be by invitation and/or acceptance of paper submission.

Topics

Relevant topics include:

- Architectural solutions to the unwanted traffic problem
- Scientific assessment of the spread and danger of the attacks
- Practical countermeasures to various aspects of unwanted traffic (spam, DoS, worms, etc.)
- Cross-layer solutions and solutions to combination attacks
- Attacks on emerging technologies (e.g., sensors, VOIP, PDAs) and their countermeasures
- Privacy and anonymity
- Intrusion avoidance, detection, and response
- Viruses, worms, and other malicious code
- Analysis of protocols and systems vulnerabilities
- Handling errors/misconfigurations that might lead to unwanted traffic
- Attacks on specific distributed systems (e.g., P2P) or network technologies (e.g., wireless networks)
- Data mining with application to unwanted traffic
- New types of solutions: incentive-based, economic, statistical, collaborative, etc.

Paper Submissions

All submissions must be in English and must include a title and the authors' names and affiliations. Submissions should be no more than six (6) 8.5" x 11" pages long and must be formatted in 2 columns, using 10 point Times Roman type on 12 point leading, in a text block of 6.5" by 9". Papers should be submitted in PDF or Postscript only.

PDF users should use "Type 1" fonts instead of "Type 3," and should embed and subset all fonts. You can find instructions on how to do this at https://www.fastlane.nsf.gov/documents/pdf_create/pdfcreate_01.jsp and http://ismir2005.ismir.net/pdf.html.

Each submission should have a contact author who should provide full contact information (email, phone, fax, mailing address). One author of each accepted paper will be required to present the work at the workshop.

Authors must submit their papers by 0400 UTC, Thursday, April 20, 2006. This is a hard deadline—no extensions will be given. Final papers are due on Tuesday, June 6, 2006, to be included in the workshop Proceedings.

Simultaneous submission of the same work to multiple venues, submission of previously published work, and plagiarism constitute dishonesty or fraud. USENIX, like other scientific and technical conferences and journals, prohibits these practices and may, on the recommendation of a program chair, take action against authors who have committed them. In some cases, program committees may share information about submitted papers with other conference chairs and journal editors to ensure the integrity of papers under consideration. If a violation of these principles is found, sanctions may include, but are not limited to, barring the authors from submitting to or participating in USENIX conferences for a set period, contacting the authors' institutions, and publicizing the details of the case.

Authors uncertain whether their submission meets USENIX's guidelines should contact the program chair at sruti06chair@usenix.org or the USENIX office, submissionspolicy@usenix.org.

Accepted material may not be published in other conferences or journals for one year from the date of acceptance by USENIX. Papers accompanied by nondisclosure agreement forms will not be read or reviewed. All submissions will be held in confidence prior to publication of the technical program, both as a matter of policy and in accordance with the U.S. Copyright Act of 1976.

How to Submit

Authors are required to submit papers by 0400 UTC, Thursday, April 20, 2006. This is a hard deadline—no extensions will be given. All submissions to SRUTI '06 must be electronic, in PDF or PostScript, via a Web form, which will be available at http://www.usenix.org/sruti06/cfp/.

Authors will be notified of acceptance decisions via email by Thursday, May 18, 2006. If you do not receive notification by that date, contact the program chair at sruti06chair@usenix.org.

Announcement and Call for Papers **USENIX**

7th Symposium on Operating Systems Design and Implementation (OSDI '06)

Sponsored by USENIX, in cooperation with ACM SIGOPS

http://www.usenix.org/osdi06

November 6–8, 2006 **Seattle, WA, USA**

Important Dates

Paper submissions due: *April 24, 2006*
Submissions acknowledged: *April 28, 2006*
Notification of acceptance: *June 30, 2006*
Papers due for shepherding: *Mid-August 2006*
Final papers due: *September 5, 2006*

Conference Organizers

Program Co-Chairs

Brian Bershad, *University of Washington*
Jeff Mogul, *Hewlett-Packard Labs*

Program Committee

Martín Abadi, *University of California, Santa Cruz*
Brad Calder, *University of California, San Diego*
Brad Chen, *Intel Corporation*
Peter Druschel, *Max Planck Institute for Software Systems*
Garth Gibson, *Carnegie Mellon University and Panasas*
Derek McAuley, *Intel Corporation*
Rob Pike, *Google Inc.*
Mema Roussopoulos, *Harvard University*
Dawn Song, *Carnegie Mellon University*
Chandu Thekkath, *Microsoft Research*
Robbert van Renesse, *Cornell University*
Jim Waldo, *Sun Microsystems, Inc.*
Bill Weihl

Steering Committee

Eric Brewer, *University of California, Berkeley*
Peter Chen, *University of Michigan, Ann Arbor*
Mike Jones, *Microsoft*
Ellie Young, *USENIX*

Overview

The seventh OSDI seeks to present innovative, exciting work in the systems area. OSDI brings together professionals from academic and industrial backgrounds in what has become a premier forum for discussing the design, implementation, and implications of systems software.

The OSDI Symposium emphasizes both innovative research and quantified or illuminating experience. OSDI takes a broad view of the systems area and solicits contributions from many fields of systems practice, including, but not limited to, operating systems, file and storage systems, distributed systems, mobile systems, secure systems, embedded systems, networking as it relates to operating systems, and the interaction of hardware and software development. We particularly encourage contributions containing highly original ideas, new approaches, and/or groundbreaking results.

Submissions that are deemed too far from these topics may be rejected without a full review.

Submitting a Paper

A good paper will demonstrate that the authors:

- are attacking a significant problem,
- have devised an interesting, compelling solution,
- have demonstrated the practicality and benefits of the solution,
- have drawn appropriate conclusions,
- have clearly described what they have done, and
- have clearly articulated the advances beyond previous work.

Submissions will be judged on originality, significance, interest, clarity, relevance, and correctness. Accepted papers will be shepherded through an editorial review process by a member of the program committee.

Papers accompanied by nondisclosure agreement forms are not acceptable and will be returned to the author(s) unread. All submissions are held in the highest confidentiality prior to publication in the Proceedings, both as a matter of policy and in accord with the U.S. Copyright Act of 1976.

In addition to citing relevant, published work, authors should relate their OSDI submissions to relevant submissions of their own that are simultaneously under review for other venues.

Simultaneous submission of the same work to multiple venues, submission of previously published work, and plagiarism constitute dishonesty or fraud. USENIX, like other scientific and technical conferences and journals, prohibits these practices and may, on the recommendation of a program chair, take action against authors who have committed them. In some cases, program committees may share information about submitted papers with other conference chairs and journal editors to ensure the integrity of papers under consideration. If a violation of these principles is found, sanctions may include, but are not limited to, barring the authors from submitting to or participating in USENIX conferences for a set period, contacting the authors' institutions, and publicizing the details of the case.

Authors uncertain whether their submission meets USENIX's guidelines should contact the program chairs, osdi06chairs@usenix.org, or the USENIX office, submissionspolicy@usenix.org.

Authors of accepted papers will be expected to provide both PDF and HTML versions of their paper, for inclusion in the Web and CD-ROM versions of the Proceedings. Authors of accepted papers will also be expected to sign a Consent Form, agreeing not to publish their papers elsewhere within 12 months of acceptance, except for electronic access as permitted in the Consent Form. One author per paper will receive a registration discount of $200. USENIX will offer a complimentary registration upon request.

Deadline and Submission Instructions

Authors are required to submit full papers by 9:00 p.m. PDT on April 24, 2006. **This is a hard deadline—no extensions will be given.**

Submitted papers must be no longer than 14 single-spaced 8.5" x 11" pages, including figures, tables, and references, using 10 point type on 12 point (single-spaced) leading, within a text block 6.5" wide x 9" deep. Papers not meeting these criteria will be rejected without review, and no deadline extensions will be granted for reformatting. Pages should be numbered, and figures and tables should be legible in black and white, without requiring magnification. Papers so short as to be considered "extended abstracts" will not receive full consideration.

Papers must be in PDF format and must be submitted via the Web submission form, which will be available on the Call for Papers Web site, http://www.usenix.org/osdi06/cfp.

The title and author name(s) should appear on the first page of the submitted paper. (Reviewing is not blind.)

For more details on the submission process, authors should consult the detailed online submission guidelines at http://www.usenix.org/events/osdi06/cfp/guidelines.html.

All submissions will be acknowledged by April 28, 2006. If your submission is not acknowledged by this date, please contact the program chairs promptly at osdi06chairs@usenix.org.

Outstanding Paper Awards

The program committee will, at its discretion, give out awards for outstanding papers. Papers of particular merit will be forwarded to *ACM Transactions on Computer Systems* for possible publication in a special issue.

Work-in-Progress Reports

Are you doing new, interesting work that has not been previously presented and that is still in too early a phase for publication? The OSDI attendees could provide valuable feedback to you. We are particularly interested in the presentation of student work. Details on submitting Work-in-Progress session proposals will be made available on the Symposium Web site by July 2006.

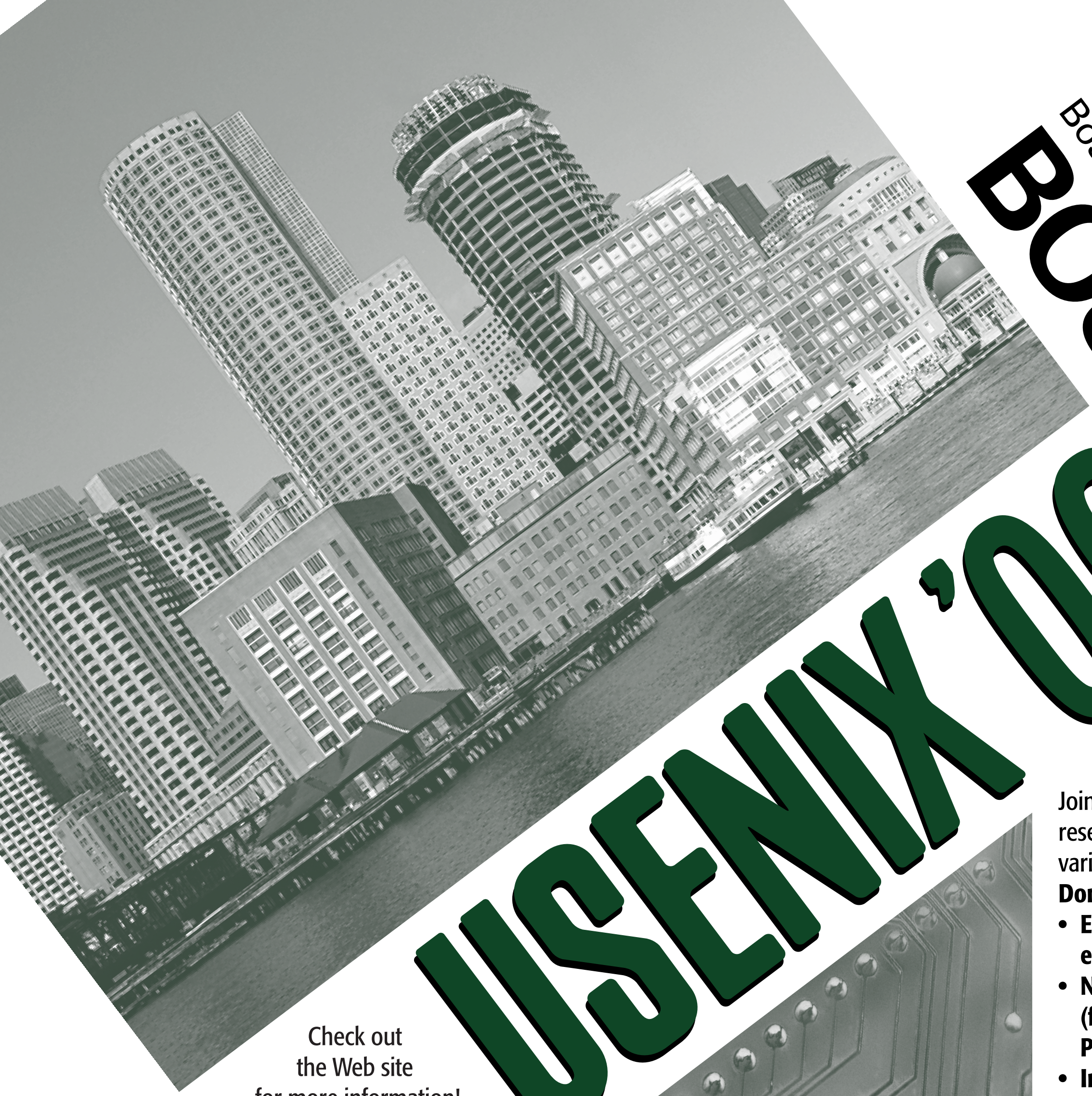
Poster Session

We plan to hold a poster session in conjunction with a social event at the Symposium. Details on submitting posters for review will be made available on the Symposium Web site by July 2006.

Registration Materials

Complete program and registration information will be available in August 2006 on the conference Web site. The information will be in both HTML and a printable PDF file. If you would like to receive the latest USENIX conference information, please join our mailing list: http://www.usenix.org/about/mailing.html.

Rev. 9/19/05



May 30 – June 3, 2006

Boston Marriott Copley Place

BOSTON

USENIX '06

Annual Technical Conference

Join us in Boston for 5 days of groundbreaking research and cutting-edge practices in a wide variety of technologies and environments.

Don't miss out on:

- **Extensive Training Program featuring expert-led tutorials**
- **New! Systems Practice & Experience Track (formerly the General Session Refereed Papers Track)**
- **Invited Talks by industry leaders**
- **And more**

Please note: USENIX '06 runs Tuesday–Saturday.

Check out the Web site for more information!
www.usenix.org/usenix06

www.usenix.org/usenix06

;login:

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710

POSTMASTER
Send Address Changes to *;login:*
2560 Ninth Street, Suite 215
Berkeley, CA 94710

PERIODICALS POSTAGE
PAID
AT BERKELEY, CALIFORNIA
AND ADDITIONAL OFFICES