;login: THE USENIX MAGAZINE

April 2004 • volume 29 • number 2

AN OPEN LETTER FROM THE USENIX ASSOCIATION REBUTTING SCO'S POSITION ON OPEN SOURCE SOFTWARE

see page 62

inside:

USENIX

The Advanced Computing Systems Association

USENIX Upcoming Events

Third Virtual Machine Research and Technology Symposium (VM '04)

Sponsored by USENIX in cooperation with ACM SIGPLAN

MAY 6–7, 2004, San Jose, CA, USA

http://www.usenix.org/events/vm04

The Second International Conference on Mobile Systems, Applications, and Services (MobiSys 2004)

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

JUNE 6–9, 2004, Boston, MA, USA

http://www.sigmobile.org/mobisys/2004/

2004 USENIX Annual Technical Conference

JUNE 27–JULY 2, 2004, Boston, MA, USA

http://www.usenix.org/events/usenix04

2004 Linux Kernel Developers Summit

JULY 19-20, 2004, Ottawa, Ontario, Canada

13th USENIX Security Symposium

AUGUST 9–13, 2004, SAN DIEGO, CA, USA

http://www.usenix.org/events/sec04

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

SEPT. 27–OCT. 1, 2004, Amsterdam, The Netherlands

http://www.sane.nl

Internet Measurement Conference 2004 (IMC 2004)

OCT. 25-27, 2004, Taormina, Sicily, Italy

http://www.icit.org/vern/imc

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

NOVEMBER 14–19, 2004, Atlanta, GA, USA

http://www.usenix.org/events/lisa04/

Paper submissions due: April 20, 2004

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

DECEMBER 6–8, 2004, San Francisco, CA, USA

http://www.usenix.org/events/osdi04

Paper submissions due: May 26, 2004

For a complete list of all USENIX & USENIX co-sponsored events, see

http://www.usenix.org/events

contents

;login: Vol. 29, #2, April 2004

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

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

$80 of each member's annual dues is for an annual subscription to *;login:*. Subscriptions for nonmembers are $110 per year.

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

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

motd

by Rob Kolstad

Dr. Rob Kolstad has long served as editor of *;login:*. He is SAGE's Executive Editor, and also head coach of the USENIX-sponsored USA Computing Olympiad.

<kolstad@usenix.org>

Why Can't They . . . ?

I really try to project a positive image and create positive slants on things. In fact, I usually hate sentences that begin with "Why can't they . . ." or "Why don't they . . .", because these phrases seem to introduce rhetorical questions – people generally don't really want to know why they {can't|don't} do {stuff}. Usually, there's a good reason.

But I can hardly stand it any longer. Maybe if enough people point out the obvious, things can change.

Let's start with Microsoft's security push. *Network World* reports on page 6 of its February 2, 2004, issue that "Microsoft sought to advance its Trustworthy Computing Initiative last week" with a US$6.8 billion budget and IE browser modifications.

That sounds pretty darn good to me. You'd think you could get a result with that much money (which comes to $25 for almost every person over the age of two in the whole of the United States). I wonder, though.

How hard is it to disable execution of incoming email? Wouldn't you think they would have started on that project by now? And seen some results? I can't imagine that it could cost an entire billion dollars. The most recent worm demonstrated yet again that people lack the impulse-control to click "no" in attachment warnings. In fact, I'll argue that, in general, the informational value of "Click YES to accept this potentially harmful {widget}" has degenerated almost to nil, rendering such warnings useless for the general user. There are simply too many of them, and the general user has little understanding or concern.

Are customers really demanding the "infect my computer with a single keyclick" feature? I can't imagine that's true. Why can't Microsoft address this? Imagine the time and money it would save just on the most recent cleverly socially engineered malware.

On another topic, why can't spam be stopped? Or at least slowed down? What entity is running around trumpeting, "Spam is OK! It's a sign of a healthy industry! We should all embrace this vibrant new way of learning about new products!"? I honestly think that advertising industry members believe they have a right to figure out any way possible to annoy me with a commercial message, and that mitigating their efforts is somehow unpatriotic. Don't even start to talk to me about the benefit to me of popups and the even more insidious popunders.

Returning to spam, the old "it's not illegal even if you don't like it" argument is gone, to a great extent. A quick perusal of my 400/day spambox shows that the number of people who even begin to label their spam properly approaches 0.5% (it doesn't exceed 0.5%, it barely approaches 0.5% on some days). Yes, that's 99.5% noncompliance with our shiny new federal CAN-SPAM law. When will lawmakers judge the law a failure? Why in the world are we creating another multi-billion-dollar industry (spam elimination) so that email can be usable again as it was before we spent the money? Just keep saying: Each $1B is more than $3 for every living person in the USA. And don't kid yourself, you'll end up paying for it one way or another in higher prices or lower functionality in every single purchase you make.

Consider how many person-hours and money-units are consumed:

- removing spam
- fixing filters
- administrating anti-spam software

- upgrading networks to handle the extra load
- "protecting" children from porn spam
- consoling users who are offended
- denying that it's {*x*}'s fault
- learning that an important email was missed since it was buried in spam.

Why does anyone think this is OK? It's not OK! It's not even close! People should be screaming, yet there seems to be more of a collective sigh of inevitability, along with a promise of a solution from Microsoft (make mailing lists expensive). The costs are pervasive, yet we continue to tolerate it. Please, stop tolerating spam. How to solve the spam problem? Remove one of:

- easy access to the Internet (not possible)
- people clicking through to purchase spam products (not possible)
- anonymity (possible – make PKI or some other technology work so that you know who's sending you email)
- ability to take money via credit cards

It's a bit easier to see why computer service is so challenging and frustrating, given the way software can mess up a system's installation and configuration. My laptop's LCD display cable is apparently broken. Of course, the phone service force couldn't even consider this as the problem until discussing new drivers, new window systems, etc. And, regrettably, they're doing a good job.

Why can't Microsoft make a desktop operating system that has at least a tiny bit of robustness? Why can random programs change my home page? Why can random programs write into the system startup so that my home page changes back to http://sexygirls.ru or whatever every time I reboot? Were customers really demanding this feature? I can well imagine it: "I'm too lazy to make Russian porn pages my home page, so please make sure any random Java program [or pick your favorite mechanism] can get so deep into my system that I have to reload Windows from scratch in order to get the system back to where it's supposed to be." Are we all really this stupid? I think not. By the way, I do my best never to click "Yes." I have no idea how that damned registry entry got into my system.

Some problems really are hard to solve. But I think we didn't scream loud enough when "executable email" came into being or Cantor and Siegel sent us down the road to spam. I distinctly remember screaming about executable email and being told "customers demand this." Maybe, just maybe, we should try to counter stupid marketing decisions with a bit more objection than a collective sigh of resignation.

Any ideas on how to do this are solicited and will be warmly received.

;login:

EDITORIAL STAFF

EDITOR
Rob Kolstad *kolstad@usenix.org*

CONTRIBUTING EDITOR
Tina Darmohray *tmd@usenix.org*

MANAGING EDITOR
Alain Hénon *ah@usenix.org*

COPY EDITOR
Steve Gilmartin

PROOFREADER
jel *jel@usenix.org*

TYPESETTER
Festina Lente

MEMBERSHIP, PUBLICATIONS, AND CONFERENCES

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

apropos

by Tina Darmohray

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

tmd@usenix.org

Increased Visibility

A friend recently called me from an airport bookstore and said he was thumbing through a book I might find interesting. After looking it up on the Web, I agreed and ordered *The Accidental Leader: What to Do When You're Suddenly in Charge*, by Michael Finley and Harvey Robbins. The book is split into two main areas: what to do if you unexpectedly are tapped to run the show, and how to do it, including specifics on identifying and leveraging your assets, managing teams, and even firing people.

As with most of the things I manage to read these days, I carried this book with me and squeezed in a couple of pages of reading in between errands, standing in lines, and, yes, during meetings. Interestingly, this book seems to have a provocative effect on people, since I received many comments from folks who observed me reading it. Predictably, people talked about climbing the corporate ladder or looking for new employment. Many wondered if it was a good book. Those who know me well asked why in the world I was reading that book.

Like so many techies, I've loathed the trappings of management and tended to give greater respect to technical achievement. I've now decided that becoming a leader may happen by accident, but becoming a good one is far from accidental. I think my delayed understanding of management's integral role in achieving overall success results from the fact that they suffer from their own workplace success, just as we system administrators do, i.e., "If we're doing our job well, no one notices."

The very best managers may be so smooth at their job we don't even notice what they're doing. In reality, they're working toward a key set of top-level tasks that are essential to our success. They set priorities, identify resources, match assets to goals, leverage team strengths, defend turf, provide feedback, and boost accomplishments.

Being the manager of system administrators is a double-whammy of the "suffering from your own success" syndrome: If you're doing a good job and the folks who are working for you are doing a good job, no one knows that you're doing a good job of managing what they're doing a good job of. Whew. It's clear, if you're in the position of managing system administrators, good old-fashioned PR has got to be one of your top priorities. Here are some ideas on what's newsworthy and what to do with it.

Toot Your Horn

Take advantage of opportunities to let your user community know that things are getting done. If you're having trouble thinking about what to talk about, take a mental accounting of the status reports you request in your team meetings. Chances are some of these same topics would be suitable for user updates. For example, when you make upgrades, let people know, and make sure to put it in "what's in it for me" terms they can understand. Short newsworthy sound bites can go a long way – for example, "Storage capacity on the central mail servers was doubled over the weekend, preventing the need to decrease users' mail spool allocation."

Create and Measure Metrics

At raise time, I used to complain that other managers could cite the number of lines of code their employees had written over the year and I could only say, "We kept the network and servers running." Of course it wasn't that clear-cut, but you get the idea. Keep track of your work. Create metrics and track performance against them, e.g., number of uninterrupted days servers are up, number of help-desk calls answered, number of patches applied, etc. We tend to view these kinds of things as "all in a day's work," but although individually they represent short time-slices, they are still deliverables, and when they are tracked and presented as a total deliverable, they have a pretty good thunk factor.

Document Milestones

System and network administrators are all about staying ahead of the curve. It takes sustained effort and know-how to keep systems current and poised to accommodate the future. If you upgrade the OS on your server pool, swap out all your old routers, or change to a new higher-capacity backup system, make sure you document these "unseen" milestones and get credit for staying "ready to meet future business needs." Make sure these milestones are shared with your higher-ups so that they know you're on board to ensure the success of the organization.

Manage Expectations

Communicate out, up, and down. Let your peers, your higher-ups, and your users know what to expect. Be proactive about giving status when things are good and when things are bad. Let people know what is going on and, as much as possible, avoid surprises. This applies from service outages to personnel changes.

Raise Awareness

Phones, heat, light, serviceable restrooms, and computers are the things we expect to "be there" for us when we get to the office each day. Depending on whether you're a glass is half full or half empty person, you may feel that means system administrators are essential or close to, ahem, well, we won't go there. Computers and networks are no longer dispensable in the workplace, and the folks who make them run aren't either. Use every opportunity to position your work as essential to accomplishing the goals of the organization.

Becoming a successful manager is no accident; it takes deliberate effort in the key areas that are essential to the success of your team. For managers of system administrators, raising the visibility of system administration accomplishments and their role in the overall success of an organization has got to be a top priority. Keep at it, and keep visible!

letters to the editor

To the Editor:

I very much enjoyed the focus-on-security issue of *;login:*. However, I wish to take exception to one statement in one article. In Abe Singer's article "Life Without Firewalls," the very first sentence ends ". . . four years without an intrusion on our managed machines."

I have no doubts at all that SDSC went four years without detecting an intrusion. But there are two possible explanations. The preferred explanation is that there were no successful intrusions. The ugly second explanation is that there were intrusion(s) that were so skillful and sophisticated that they could not be detected.

I know and respect Abe, and I know that he's aware (or at least, Marcus's cat has assured me, there's a 95% probability that he's aware) of this distinction, so I put this sentence down to compact prose. But I have met at least one senior government official who honestly believed that if they had not detected an intrusion, none had happened. He could not be persuaded that the second possibility even existed. Given the state of diplomatic relations between the country in question and the US at the time, I believe that he was almost certainly wrong.

Sorry to be pedantic, but that's one of the statements I simply won't let past without comment. (The other one is "Encryption ensures message integrity" . . . so don't say it.)

Greg Rose
ggr@qualcomm.com

The author responds:

Greg is correct. I intended an aside or footnote to point out that we had no compromises *that we knew of*, but it was omitted. Mea culpa.

Abe Singer
abe@sdsc.edu

wireless Java™ application development

by Qusay H. Mahmoud

Dr. Qusay H. Mahmoud is an assistant professor at the Department of Computing and Information Science, University of Guelph, and associate chair of the Distributed Computing and Communications Systems Technology program (University of Guelph-Humber).

qmahmoud@cis.uoguelph.ca

Developing wireless applications using the Wireless Application Protocol (WAP) is similar to developing Web pages with a markup language, because WAP is browser-based. While Java Servlets and Java Server Pages (JSPs) can be used to generate WAP's WML pages dynamically, all communications between the device and the application go over the wireless link, and this is expensive. In addition, WAP isn't really suitable for developing wireless interactive applications such as mobile games. The Sun Java 2 Micro Edition (J2ME) platform can be used to develop wireless interactive applications or MIDlets that can be downloaded over the air and installed on the device.

This article presents an overview of the genesis of the J2ME platform and walks you through a sample wireless application to give you a flavor of what's involved in developing wireless Java applications. It is worth noting that the J2ME is already deployed on millions of devices, such as cell phones, that are available from Motorola/Nextel, Nokia, and other vendors.

Introduction to J2ME

The Java 2 Micro Edition (J2ME) is aimed at the consumer and embedded-devices market. It specifically addresses the rapidly growing consumer space that contains commodities such as cellular telephones, pagers, Palm Pilots, set-top boxes, and other consumer devices. It is targeted at two product groups: *personal*, *mobile*, *connected information devices* (e.g., cellular phones, pagers, and organizers) and *shared*, *fixed*, *connected information devices* (e.g., set-top boxes, Internet TVs, and car entertainment and navigation systems). The groups are addressed using different configurations and profiles.

Configurations

Cellular telephones, pagers, organizers, etc., are diverse in form, functionality, and feature. For these reasons, the J2ME supports minimal configurations of the Java Virtual Machine (JVM) and APIs that capture the essential capabilities of each kind of device. At the implementation level, a J2ME configuration defines a JVM and a set of horizontal APIs for a family of products that have similar requirements on memory budget and processing power. In other words, a configuration specifies support for: (1) Java programming language features, (2) JVM features, and (3) Java libraries and APIs.

Currently, there are two standard configurations: the Connected Limited Device Configuration (CLDC) and the Connected Device Configuration (CDC). The CLDC is aimed at cellular phones, pagers, and organizers, while the CDC targets set-top boxes, Internet TVs, and car entertainment and navigation systems. In this article we are more concerned with the CLDC.

As you can see from Figure 1, a JVM (e.g., the K Virtual Machine or KVM) is at the heart of the CLDC. Note that CLDC 1.0 was the initial version, but today CLDC 1.1, the enhanced version, is the standard. A major difference between the two is that

CLDC 1.0 didn't include support for floating point numbers (so you could not declare variables of type float or double), but CLDC 1.1 does.

The K Virtual Machine

The K Virtual Machine (KVM) is a compact, complete, and portable Java virtual machine specifically designed from the ground up for small, resource-constrained devices. The design goal of the KVM was to create the smallest possible complete JVM that would maintain all the central aspects of the Java programming language but would run in a resource-constrained device with a few hundred kilobytes of total memory. The J2ME specification describes that the KVM was designed to be: (1) small, with a static memory footprint (40–80 KB), (2) clean and highly portable, (3) modular and customizable, and (4) as "complete" and "fast" as possible.

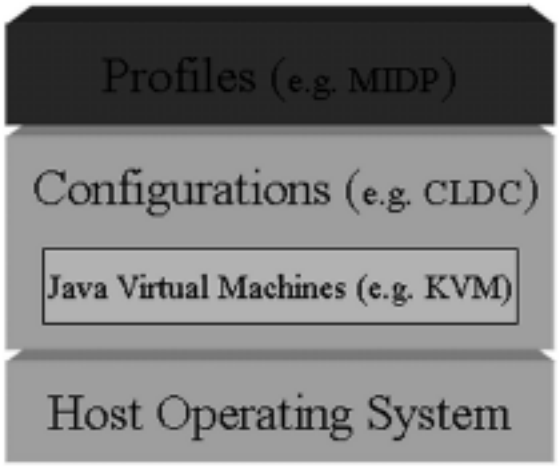


Figure 1: High-level architecture of J2ME

Profiles

The J2ME makes it possible to define Java platforms for vertical markets by introducing profiles. At the implementation level, a profile is a set of vertical APIs that reside on top of a configuration, as shown in Figure 1, to provide domain-specific capabilities such as GUI APIs.

Currently, there is one profile implemented on top of the CLDC, the Mobile Information Device Profile (MIDP), but other profiles are in the works. The MIDP 1.0 was the initial profile and has several constraints (e.g., no support for low-level sockets). MIDP 2.0 is the enhanced version of MIDP with several new features, including end-to-end security (support for HTTPS), as well as support for sockets.

JVM Supporting CLDC vs. J2SE JVM

There are several differences between a JVM supporting CLDC and the Java 2 Standard Edition (J2SE) JVM. A number of features have been eliminated from a JVM supporting CLDC, either because they are too expensive to implement or because their presence would have imposed security problems. Therefore, in a JVM-supporting CLDC such as the KVM, there is:

- *No floating point support:* CLDC 1.0 does not support floating point numbers and therefore no CLDC-based application can use any floating point numbers and types such as `float` or `double`. This is mainly because CLDC target devices do not have hardware floating point support. Note that CLDC 1.1 *does* include support for floating point numbers.
- *No finalization*: Finalization is not supported, meaning that the CLDC APIs do not include the method `Object.finalize()`, and therefore there is no finalization of class instances.
- *Limited error handling*: Runtime errors are handled in an implementation-specific manner. The CLDC defines only three error classes: `java.lang.Error`, `java.lang.OutOfMemoryError`, and `java.lang.VirtualMachineError`. Other types of errors are handled in a device-dependent manner that would involve terminating the application or resetting the device.
- *No Java Native Interface (JNI)*: A JVM-supporting CLDC does not implement the JNI, mainly for security reasons and because implementing JNI is considered expensive given the memory constraints of the CLDC target devices.
- *No user-defined class loader*: A JVM-supporting CLDC must have a built-in class loader that cannot be overridden or replaced by the user. This is mainly for security reasons.

- *No support for reflection*: No reflection features are supported, and therefore there is no support for RMI or object serialization.
- *No thread groups or daemon threads*: While a JVM-supporting CLDC implements multi-threading, it should not have support for thread groups or daemon threads. If you want to perform thread operations for groups of threads, collection objects should be used to store the thread objects at the application level.

The CLDC APIs

The J2SE APIs require several megabytes of memory and therefore they are not all suitable for small devices with limited resources. In designing the APIs for the CLDC the aim was to provide a minimum set of libraries that would be useful for application development and profile definition for a variety of small devices. The CLDC library APIs can be divided into two categories:

1. Classes that are a subset of the J2SE APIs: These classes are located in the java.lang, java.io, and java.util packages. They have been derived from the J2SE 1.3. However, note that not all classes from these packages have been inherited.
2. Classes that are specific to the CLDC: These classes are located in the javax.microedition package and its subpackages.

MIDP Programming

Anyone who has some hands-on programming with Java can start developing MIDP applications (or MIDlets) right after reading this article. MIDP programming is easier than J2SE programming because the MIDP API is simpler. You need to learn about a few classes before you start writing your own MIDlets. Your MIDlet must inherit from the MIDlet class of the javax.microedition.midlet package, then you simply override some methods; the MIDlet lifecycle methods are: startApp(), pauseApp(), and destroyApp(). To handle events, you must implement the CommandListener interface. Here is a simple MIDlet example:

Listing 1: LoginMIDlet.java

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.MIDlet;

/**
* This login MIDlet prompts the user for a username and a password. If the
* user enters the correct account information, a list of options is
* displayed, otherwise an error message is displayed.
*
* @author: Qusay H. Mahmoud
*/
public class LoginMIDlet extends MIDlet implements CommandListener {
    private Display display;
    private TextField userName;
    private TextField password;
    private Form form;
    private Command cancel;
    private Command login;

    // Constructor
    public LoginMIDlet() {
        userName = new TextField("LoginID:", "", 10, TextField.ANY);
        password = new TextField("Password", "", 10, TextField.PASSWORD);
```

```
    form = new Form("Sign in");
    cancel = new Command("Cancel", Command.CANCEL, 2);
    login = new Command("Login", Command.OK, 2);
}

// MIDlet lifecycle method: called when the MIDlet is started:
public void startApp() {
    display = Display.getDisplay(this);
    form.append(userName);
    form.append(password);
    form.addCommand(cancel);
    form.addCommand(login);
    form.setCommandListener(this);
    display.setCurrent(form);
}

// MIDlet lifecycle method: called when MIDlet is paused:
public void pauseApp() {
}

// MIDlet lifecycle method: called when the MIDlet is destroyed:
public void destroyApp(boolean unconditional) {
    notifyDestroyed();
    }

// Checks if the user enters the correct account information:
public void validateUser(String name, String password) {
    if (name.equals("qm") && password.equals("guessit")) {
        menu();
    } else {
        tryAgain();
    }
}

// Display a list of services:
public void menu() {
    List services = new List("Choose one", Choice.EXCLUSIVE);
    services.append("Check Email", null);
    services.append("New Message", null);
    services.append("Address Book", null);
    services.append("Customize", null);
    services.append("Sign Out", null);
    services.addCommand(new Command("Back", Command.CANCEL, 2));
    services.addCommand(new Command("Select", Command.OK, 2));
    display.setCurrent(services);
}

// Display an error message if the user enters the incorrect account info:
public void tryAgain() {
    Alert error = new Alert("Login Incorrect", "Please try again", null,
            AlertType.ERROR);
    error.setTimeout(Alert.FOREVER);
    userName.setString("");
    password.setString("");
    display.setCurrent(error, form);
}

// Handle events:
```

```
        public void commandAction(Command c, Displayable d) {
            String label = c.getLabel();
            if(label.equals("Cancel")) {
                destroyApp(true);
            } else if(label.equals("Login")) {
                validateUser(userName.getString(), password.getString());
            }
            // add code to handle user's selection from list of services
        }
    }
```

In Listing 1, the midlet and lcdui packages are imported. The midlet package defines the MIDP, and the lcdui package provides graphical user interface APIs for implementing user interfaces for MIDP applications. It is worth noting that the lcdui package is not a subset of Swing/AWT, simply because the Swing/AWT assumes certain user interaction and provides a rich feature set (such as resizing overlapping windows) not found on mobile devices. The basic unit of interaction on a mobile device is the screen – users' interaction with wireless applications by navigating through screens.

Each MIDlet must extend the MIDlet class, similar to applets, which allows for the orderly starting, stopping, and cleanup of the MIDlet. Therefore, a MIDlet must not have a public static void main() method.

In LoginMIDlet, the Command class is used to encapsulate the semantic information of an action. The command itself contains only information about a command, but not the actual action that happens when a command is activated. The action is defined in a CommandListener associated with the screen. Let's look at the following command statement:

```
Command infoCommand = new Command("Info", Command.SCREEN, 2);
```

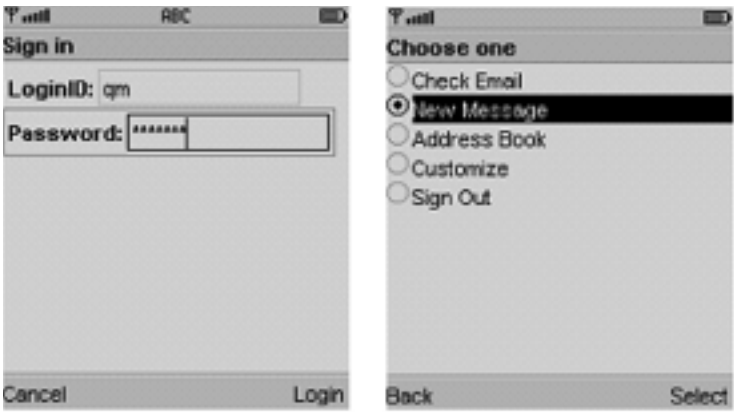


Figure 2: LoginMIDlet launched and activated

A command contains three pieces of information: a label, a type, and a priority. The label (which is a string) is used for the visual representation of the command. The type of the command specifies its intent. And the priority value describes the importance of this command relative to other commands on the screen. A priority value of 1 indicates the most important command, and higher priority values indicate commands of lesser importance. When the application is executed, the device chooses the placement of a command based on the type of the command, and places similar commands based on their priorities.

Figure 2 shows the screens when the application is first launched.

Development Tools

There are several commercial and freely available tools for developing wireless Java applications. My favorite tool is Sun's J2ME Wireless Toolkit (J2ME WTK), which is easy to use and freely available. The J2ME WTK provides a comprehensive tool set and emulators for developing and testing wireless applications, and it is available for the Windows, Linux, and Solaris platforms. It simplifies the development of wireless applications by automating several steps such as preverification and creating Java Archive

(JAR) and Java Application Descriptor (JAD) files (more on this later). Figure 3 shows the interface for the J2ME WTK.

The J2ME WTK can be downloaded from *http://java.sun.com/products/j2mewtoolkit*. To test the LoginMIDlet described above, create a new project, call it Login, and call the MIDlet LoginMIDlet. Then copy my LoginMIDlet.java (the above code) to the apps\Login\src directory of your J2ME WTK installation. The next step is to compile (click on the Build button) and run (click on the Run button) the application. You can choose a device to emulate the application on.

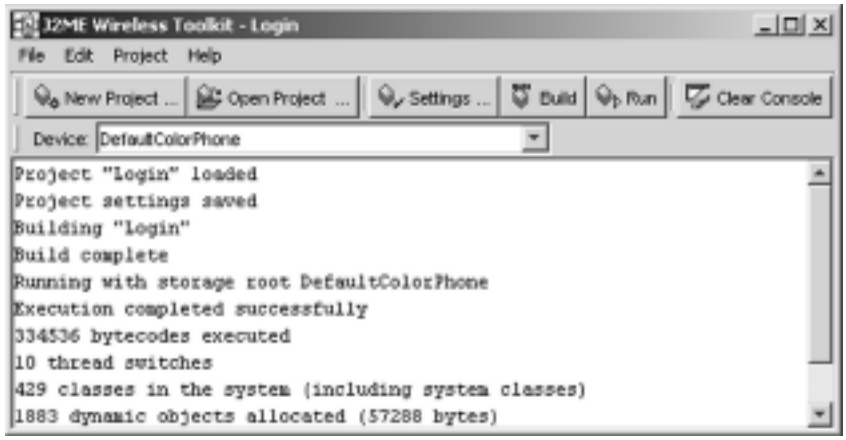


Figure 3: The J2ME Wireless Toolkit

Behind the Scenes

As I mentioned above, the J2ME WTK automates the processes of preverification and packaging of the application. This is done when you click on the Build and Run buttons. So what are preverification and packaging?

Class Verification

In the J2SE Java virtual machine, the class verifier is responsible for rejecting invalid class files. A JVM-supporting CLDC must be able to reject invalid class files as well, but the class verification process is expensive and time-consuming and, therefore, is not ideal for small, resource-constrained devices. The KVM designers decided to move most of the verification work off the device and onto the desktop, where the class files are compiled or onto a server machine from which applications are being downloaded. This step (off-device class verification) is referred to as preverification. The device is simply responsible for running a few checks on the preverified class file to ensure that it was verified and is still valid.

Therefore, after compiling the .java into .class, the .class file must be preverified using the preverify command (in J2ME WTK). This command preprocesses the program for use by the KVM without changing the name of a class. The preverify command takes a class or a directory of classes and preprocesses them. Luckily, this task is automated by the J2ME WTK.

Packaging the Application

If an application consists of multiple classes, a JAR file is used to group all the classes together so that the application is easy to distribute and deploy. In the above example, a JAR file (Login.jar) is created – the J2ME WTK automates this using the jar command.

The next step in packaging is creating a manifest file (or application descriptor), which provides information about the contents of the JAR file. The application descriptor is used by the application management software on the device to manage the MIDlet. It is also used by the MIDlet itself to configure specific attributes. The file extension of the application descriptor is jad, which stands for Java Application Descriptor. There is a predefined set of attributes to be used in every application descriptor. One of the attributes is the MIDlet-Jar-Size, which is used by the application management soft-

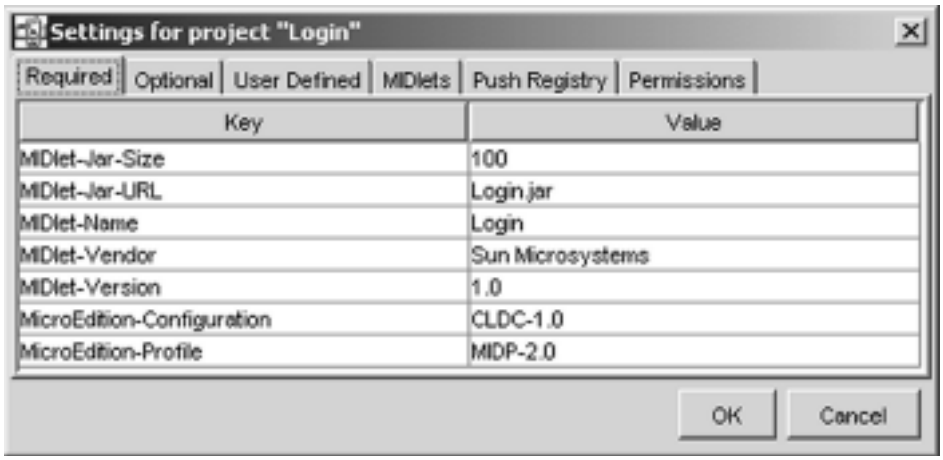


Figure 4: JAD file for the Login project

ware to determine whether the device is capable of running the MIDlet before it downloads it (over the air) to the device. Figure 4 shows the JAD file for the Login project (click on the Settings button of the J2EME WTK to see this).

Deploying Applications

If you have a Java-enabled cell phone from Motorola/Nextel, you can download applications on it either over the air (you'll get billed for the air time) or from the Internet through your PC using a data cable. For more information, visit *http://idenphones.motorola.com*.

Once you have tested the application and you are satisfied with what you see, you can deploy it on a Web server simply by uploading its JAR and JAD files to a Web server. Now your application is downloadable. However, you need to add the following new MIME type to your configuration file and restart the Web server:

```
text/vnd.sun.j2me.app-descriptor  jad
```

Integrating WAP and J2ME

MIDlets combine excellent online and offline capabilities that are useful for the wireless environment, which suffers from low bandwidth and network disconnection. Integrating WAP and MIDP opens up possibilities for new wireless applications and over-the-air distribution models. Therefore, WAP and MIDP should be viewed as complementary rather than competing technologies. In order to facilitate over-the-air provisioning, there is a need for some kind of an environment on the handset that allows the user to enter a URL for a MIDlet, for example. This environment could very well be a WAP browser. Similar to Java Applets that are integrated into HTML, MIDlets can be integrated into a WML page. The WML page can then be called from a WAP browser, and the embedded MIDlet gets installed on the device. In order to enable this, a WAP browser (with support for over-the-air provisioning) is needed on the device. An alternative approach for over-the-air provisioning is the use of Short Message Service (SMS), as has been done by Siemens, where the installation of MIDlets is accomplished by sending a corresponding SMS. If the SMS contains a URL to a JAD file specifying a MIDlet, then the recipient can install the application simply by confirming the SMS.

Conclusion

This article introduced the J2ME platform and described the development model for wireless Java applications. The MIDlet provided shows that programming with J2ME is easier than programming with J2SE, because the API is simpler and there are only a dozen classes you need to learn. I hope this article helps you get started with wireless Java application development. Stay tuned for more articles on wireless Java.

FURTHER READING

Java 2 Micro Edition (J2ME):
http://java.sun.com/j2me

The J2ME Wireless Toolkit:
http://java.sun.com/products/j2mewtoolkit

Learning Wireless Java, by Qusay H. Mahmoud, available from O'Reilly

Motorola iDEN phones:
http://idenphones.motorola.com

Sun's Source for Java Developers:
http://java.sun.com

musings

by Rik Farrow

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

rik@spirit.com

In one of those weird twists of fate, being popular meant that you got bombarded with many copies of the MyDoom virus in late January and early February 2004. The MyDoom virus (A version) used its own SMTP engine to mail copies of itself, as an attachment, to names culled from users' address books. The same email addresses were also used as sender addresses, so you would also get besieged with bounce messages. If you were deluged with copies of MyDoom, you just received a measure of your online popularity.

Adding insult to injury, most anti-virus email server products, when armed with a signature to detect MyDoom, would send a notification to the forged sender, so you were also bombarded with assertions that you had sent the MyDoom virus, and thus, were infected. By implication, you should have bought the AV vendor's product so you could be protected, and spam other people by sending messages to forged sender addresses.

You might think that I am being unfair, but a simple inspection of mail headers would show that the alleged sender of MyDoom had nothing to do with the source address of the system that sent the email. You would think that companies that deal with viruses, which get spread mainly by email, would at least be capable of running trivial checks on email headers before spamming people with useless notices.

MyDoom turns out to have another interesting tie into spam. Some security friends were discussing the virtues of blacklisting entire netblocks. Many people have already decided to reject all SMTP connections coming from Asian netblocks, as many of these countries have large broadband (mostly DSL) networks of home users, and these home systems have become a popular haven for the open relays used by spammers. Some of my friends were going further, deciding that they should also be blocking the netblocks used by the large US-based broadband providers, such as Comcast, ATTBI, and RoadRunner.

To me, this appeared to be a little extreme. So I decided that I should take a deeper look. I started by grepping the Received lines added to incoming email by my own email server. My Spam directory had, at that time, about 4000 emails in it. I also asked Brian Martin, a friend from attrition.org who not only saves spam emails but diligently sends messages off to abuse@whatever for his Received headers. Together, we had close to 10,000 Received lines that included both the sender's IP address and the domain name resolved by Sendmail or postfix in the form

```
Received: from mail11.cybertrails.com
    (mail.cybertrails.com [162.42.150.35])
```

The system that delivered this particular email to my system gave its name in the HELO message as mail11.cybertrails.com. The source IP address during the TCP transaction was 162.42.150.35, and the reverse lookup of that address was mail.cybertrails.com. Note that I am not beating up on Cybertrails, a past ISP that apparently continues to forward spam to me based on an email address that I never used and have asked them to kill. They are now on my REJECT list.

Next, I used a Perl one-liner to extract the resolved domain name and IP address from the Received lines, then used another one-liner to extract just the last two names in each domain (cybertrails.com in the above example). Finally, I ran that result through

a pipeline that sorted the names, used uniq -c to count them, and sorted the counts by uniq in reverse order. Now I had a sorted list of the sources of the 10,000 spam messages that Brian and I had received over about a month.

The results were revealing, but not too surprising. Comcast.net and attbi.com together accounted for 15% of all spam received, with RoadRunner coming in third with 5%. Many other well-known broadband providers fit into the top 20 offenders, for a total of over half of all the spam we had received.

Now, this is not a scientific survey, just a quick peek at a sample of spam. I make several assumptions, such as that I trust Brian's and my own SMTP MTA to correctly resolve addresses, and that spammers are not adding forged Received lines, complete with domain names and addresses in the format used by UNIX MTAs. But it appears to me that my friends who have been considering blocking all email from broadband providers as a way of stopping spam really do have a good point.

One person in the group, known as Hobbit, has already blocked many broadband providers, but with a very reasonable twist. He permits SMTP from the mail servers for each broadband domain, based on the MX records for that domain. I thought this was a great idea, as it means that only email relayed by the SMTP servers for a domain would be permitted. Most spammers will not use an ISP's mail relays, but send email directly, using their own SMTP engines installed via viruses or other attacks, or via open relays. Hobbit could still receive legitimate mail, relayed from "registered" servers – those that have MX records for the sender's domain.

When I started looking deeper, I discovered that what Hobbit had done was a lot of work. For example, there is a *fl.comcast.net*, but fortunately only two MX records for all of comcast.net, corresponding to four IP addresses. Other broadband providers actually support a set of SMTP servers, represented by MX records, for each subdomain, which are often organized by state. And this information is subject to change.

I decided I didn't want to maintain a comprehensive list of networks to block while permitting only registered SMTP servers within these blocks. Sure, blocking some netblocks looks like a good idea, something that would quickly free up my system for other tasks (using `client_acl` of Postfix or your favorite MTA to reject connections from clients that don't correspond to MX records). But I think that there is a better way.

What if ISPs blocked outgoing connections to port 25 that come from clients that don't have their own domains and MX records? Some ISPs do this already, although obviously not some of the larger ones. Doing so would require that the ISP actually maintain filter rules that would permit the SMTP servers that it knew about, while blocking all others. It appears to me that doing this would block the most commonly abused spam relays. It would also prevent the ISPs from being added to spammer blacklists.

Of course, getting ISPs to do any useful filtering has been wishful thinking so far. Back in 1998, RFC 2267 sought to prevent source address spoofing by encouraging ISPs to block packets that enter their networks from leaf networks that have spoofed source addresses. Source address spoofing on the Internet would have vanished if this had been accomplished. Obviously, it hasn't.

Now what do you think would have happened to MyDoom, and SoBig, and many other of the big viruses that spread using their built-in SMTP engines, if these filters were in place? The ISPs' filters would have blocked attempts to connect to any SMTP servers outside of their networks, slowing the spread of these viruses to a crawl.

ISP filtering certainly could be done. One of the reasons it is not done is that it means more work for the ISPs – work that many are loathe to do, as it costs money and affects the bottom line.

So we get deluged with viruses, virus bounces, AV warnings, and spam, largely because of ISPs not willing to take responsibility for managing their own networks properly. Perhaps it is time for a little regulation.

document it with a picture

A Case Study

Introduction

Shortly after joining the Storage Group at HP Labs as system administrator, I was asked to add some disks to the group's production XP512 disk array. At that time, I didn't have much experience with that array model, so one of the researchers helped me navigate the console menus. We documented the whole process using a digital camera, which was a method of documenting sysadmin tasks I had never seen before. This method proved to be quite useful: A few months later, I needed to add more disks to the array, but I had forgotten some details of the procedure. The pictures were there to help me recall the necessary steps, saving me hours of research.

Digital photography is a technique particularly well suited for documenting tasks that require interacting with hardware, as it shows the steps taken chronologically. Well-chosen pictures can be self-explanatory and are often clearer than documentation manuals, because they show precisely the task being performed. It is a time-efficient technique, too: It adds little overhead to the task being performed and can help eliminate future mistakes, thereby saving more time.

Our department has developed a set of simple Perl scripts [Laff03] that enables us to generate a Web page quickly, right after we download the pictures to a laptop. There is no need to order the pictures chronologically by hand either: The camera itself assigns reasonable (sequential) filenames to the images.

After the Web page is generated, we publish it on our intranet and may add some comments later by editing the HTML code (see Figure 1). We usually document interaction with the disk array console this way, too. The photos of the screens are easy to read if taken correctly, as Figure 2 shows.

With some discipline, this is an efficient way to document a variety of system administration tasks. Taking some notes during the process helps, but even without comments the images usually serve well enough. Experience has also shown us that when we don't take pictures, we find ourselves regretting it later.

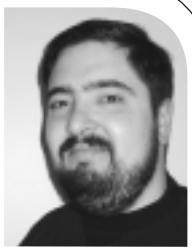
This approach is often useful for infrequent but complex tasks, especially ones that involve physical operations. In the next section, we will compare this approach to some alternatives, and will then summarize our experiences in the last section. Pictures and references are collected at the end.

Why This Is a Good Idea

Our main focus is documentation of system administration tasks as a training tool. Some of the ideas discussed in this article will apply to documentation of forensic procedures or Capability Maturity Model-related documentation of tasks, but that is not our primary objective.

Eric Anderson's thorough survey [Ande99] of the LISA papers published between 1987 and 1999 shows only a few papers discussing sysadmin training. Two papers ([Shar92] and [Hunt93]) describe the use of text-based trouble-ticket software as a

by Hernan Laffitte



Hernan Laffitte has worked in UNIX administration and as a consultant since 1993. He currently works as sysadmin for the Storage Systems Department at Hewlett-Packard.

hernan@hpl.hp.com

and Eric Anderson

Eric has been a researcher at HP Labs working on storage and information management since 1999. He wrote his dissertation on system administration at U.C. Berkeley.

anderse@hpl.hp.com

FIGURES ARE GROUPED TOGETHER STARTING ON PAGE 18

Physical procedures can only be documented by photos and video.

training aid for novice system administrators. The Indiana University USAIL Web-based education project is presented in Tomp96. Finally, Wendy Nather's seminal *;login:* article [Nath93] discusses the benefits of the school of hard knocks.

Suppose you are performing a complex system administration task and you want to document what you are doing. You want the documentation to explain the task to a new sysadmin, or maybe to yourself six months from now. You can use commands such as "script" and "tee" to keep a record of what happens on your terminal. Maybe take a screenshot or two to show how you use a graphical tool.

In fact, there are several ways to go about documenting a sysadmin procedure. A basic taxonomy of the different documentation techniques would be:

1. Text document: just writing the documentation manually.
2. Snapshots: Any time-frozen representation of the state of a system taken at different points in time: e.g., photographs, screenshots, cut-and-paste of the text in a Telnet session.
3. Time series: techniques to generate a continuous picture of a system as it changes through time: e.g., video recording, commands like "tee" and "script," a log of a Telnet session, or some screen recording utility such as Lotus ScreenCam.

We can of course create multimedia documents that combine the three techniques.

So, there are lots of ways of documenting a sysadmin task, of which taking a photo of the computer screen does not immediately appear to be the most efficient option. However, in some cases it can the best solution. Here are some reasons why:

- Physical procedures can only be documented by photos and video. Until we can buy a wearable Memex [Bush45] that records all our actions, a digital camera can be the next best thing. With a camera, you document what you are seeing now – and will probably see again the next time you have to perform the same task. See Figure 3 for examples of a "this part goes here" kind of operation that would be difficult to document in a non-pictorial manner.
- Pictures are more selective than video. It's easier to use them to show the important parts only. Browsing the pictures is faster, and less boring. Taking photos is also less intrusive than filming. As a downside, there is always the possibility of forgetting to photograph some important step.
- For some GUIs, taking a screenshot or running a screen-recording program is not feasible. Some reasons for this:
    - Hardware limitations: The consoles of some devices (disk arrays, printers) are simply too dumb.
    - Security: Even if the console of a dedicated device is a PC-like machine running a full-featured operating system, for security reasons, it may not be connected to the network. This makes it difficult to download a screenshot, even if one could record the shots to the local disk.
    - Self-discipline: In our case, we decided not to connect the console of our main disk array to the network. This way, we don't have the convenience of being able to wipe out all its contents from the comfort of our desks.
- A regular screenshot will not usually show where the cursor is or which button it's clicking on at the moment. A ScreenCam movie will show this information, but it's platform-dependent. Taking a photo of the screen is a reasonable compromise. See Figure 2 for an example.

- In a text-only environment, "script" and "tee" may be an alternative, but their output can sometimes be illegible. For example, CURSES-based tools often generate near-infinite sequences of control characters. Finding useful information in the resulting ASCII animation can be a chore (see Figure 5).
- Digital photos are cheap and convenient. As of August 2003, a 2-megapixel camera with 128MB of flash RAM can be bought for around $150 new or $100 used. This represents less than a few hours of the fully loaded cost of an administrator. So if the pictures save you a day of work, the camera has already paid for itself. These small cameras fit into a pocket, and the time cost to taking the pictures and uploading them to a Web site can be less than 10 minutes. Adding a few comments to the resulting Web page adds some time, but always less than writing a text-only document. For example, accurately describing in writing the steps to replace a printer's transfer kit would be more difficult than just having the pictures.
- Documents of a known "good" state before performing a complex operation can also be useful, especially when trying to debug a problem in some operation. In one case we failed to document a procedure because it was performed by a company service engineer. When something went wrong, we brought in more experienced administrators, but they could only see the current, broken state, rather than the steps taken to get there or information from before the operation. As a result, solving the problem took longer than necessary. Subsequent to this experience, we have set a new policy of documenting all changes regardless of who performs them.

Experience and Summary

Several months elapsed before I had to install a second disk group in the XP512 disk array. Of course, by that time I had already forgotten most of the procedure for adding the disks. Did I need to click on the button labeled "Maintenance" or the one labeled "Install"? Fortunately, the pictures were there to help.

The second run was also documented with the camera. This time it wasn't necessary to add as many comments as before, only to comment on the differences from the previous run. Currently, every time I have to perform some task on the disk array, I keep referring to these photos. I also take pictures of new procedures to help my memory the next time they need to be done. We now have more than 10 sets of documentation photographs.

The first set of pictures was taken before I joined the company, when the Storage Department acquired the XP512 disk array. Because we are a research group, the idea was to assemble the array ourselves, and we took pictures of the whole process for future reference. While installing the array, we realized that it was a natural step to also take pictures of what we were doing on the console and interleave them with the pictures of what we were doing at the hardware level (connecting cables, installing disks, etc.). The resulting set of pictures is a "story" that clearly reflects all the steps we took to install the disk array.

We believe that this technique will be even more useful at larger sites. A senior administrator could perform a task the first time and record the pictures, and then a junior administrator could use the pictures to perform the same steps later. If they also take pictures as they go along, then if a problem occurs, the senior administrator can easily review the steps that were taken.

Accurately describing in writing the steps to replace a printer's transfer kit would be more difficult than just having the pictures.

REFERENCES

[Ande99] Eric Anderson and Dave Patterson, "A Retrospective on Twelve Years of LISA Proceedings," *Proceedings of LISA '99 13th Systems Administration Conference*, November 1999.

[Bush45] Vannebar Bush, "As We May Think," *Atlantic Monthly*, July 1945.

[Hunt93] Tim Hunter and Scott Watanabe, "Guerrilla System Administration: Scaling Small Group Systems Administration to a Larger Installed Base," *Proceedings of LISA '93 Seventh Systems Administration Conference*, November 1993.

[Laff03] See *http://www.hpl.hp.com/personal/Hernan_Laffitte* for source code samples and full-color example pictures.

[Nath93] Wendy Nather, "Think or Thwim: The Cold Creek Approach to Systems Administration Training," *;login:*, vol. 18, no. 4, July/August 1993, p. 22.

[Shar92] James M. Sharp, "Request: A Tool for Training New Sys Admins and Managing Old Ones," *Proceedings of LISA '92 Sixth Systems Administration Conference*, October 1992.

[Tomp96] Raven Tompkins, "A New Twist on Teaching System Administration," *Proceedings of LISA '96 10th Systems Administration Conference*, September 1996.

In conclusion, digital photography is an efficient and useful way of documenting sysadmin operations, and is excellent for reminders on infrequently performed tasks. It complements other approaches to documenting procedures, and should therefore be considered in the arsenal of techniques administrators can use to make their lives easier.

Future work being considered in this area includes:

- Find an efficient way to perform character recognition on the pictures of the devices' consoles.
- Newer digital cameras support audio clips. Having one of these perform speech recognition on the sound bits could further ease and improve the documentation process.

Acknowledgments

The authors of this article would like to thank Alistair Veitch for his valuable comments, which improved this text tremendously.

Figures

Figure 1a: A Web page generated by our Perl script from the contents of the digital camera. The three links under each thumbnail take the user to versions of the image in different resolutions.

Figure 1b: A detail of the same page, after adding comments. The HTML code could be more polished.

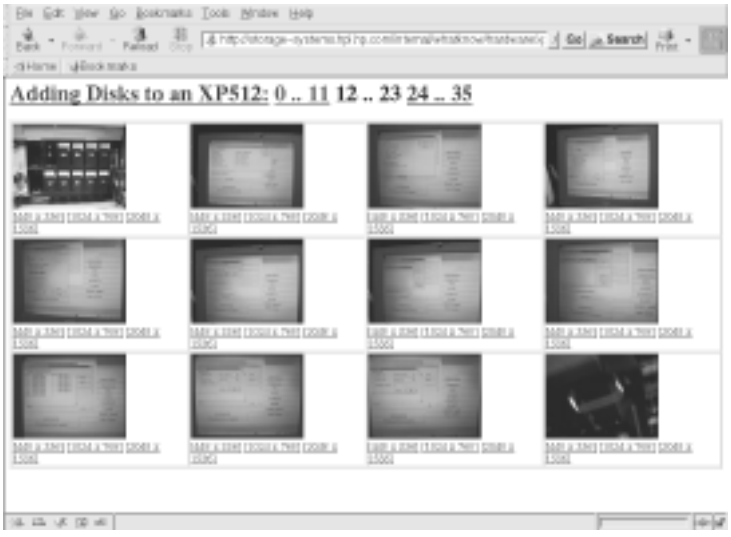


Figure 1a

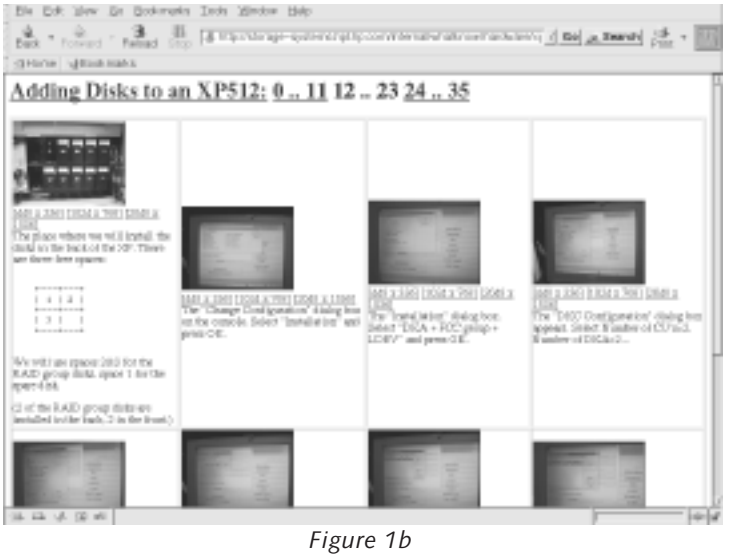


Figure 1b

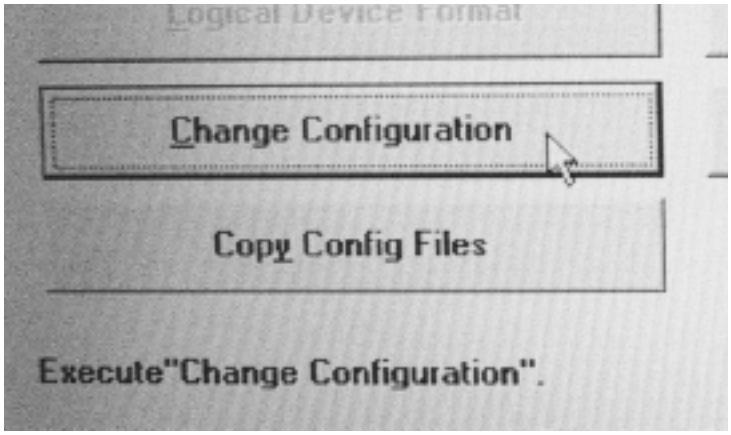


Figure 2

Figure 2: The text from a screenshot is usually legible. We generally use a Canon S30 digital camera, which provides 2048x1536 resolution (3-megapixel). The department also has two 2-megapixel HP digital cameras.

With our 3-megapixel camera, photographs of the screen are easy to read, provided they are taken with a steady hand.

This figure also shows how we can record the fact that the mouse pointer is here, clicking on this button, etc. This is a useful feature for recording the interaction with a GUI.

Figures 3a–3b: An example of "this piece goes here."

Figure 3a

Figure 3b

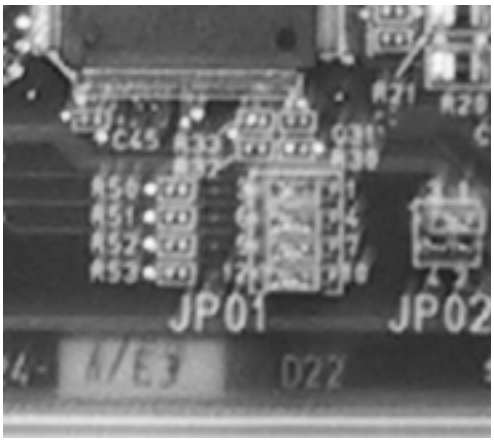


Figure 3c

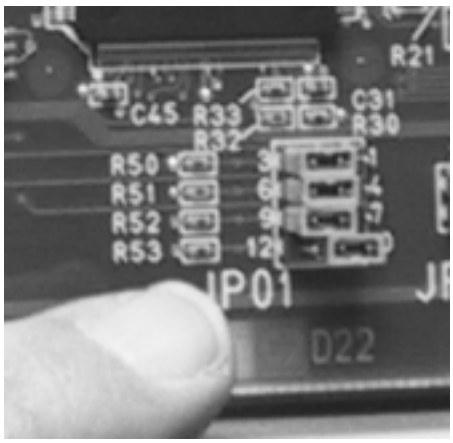


Figure 3d

Figures 3c/3d: Moving dip switches around is also best described graphically, as this "before and after" sequence illustrates.

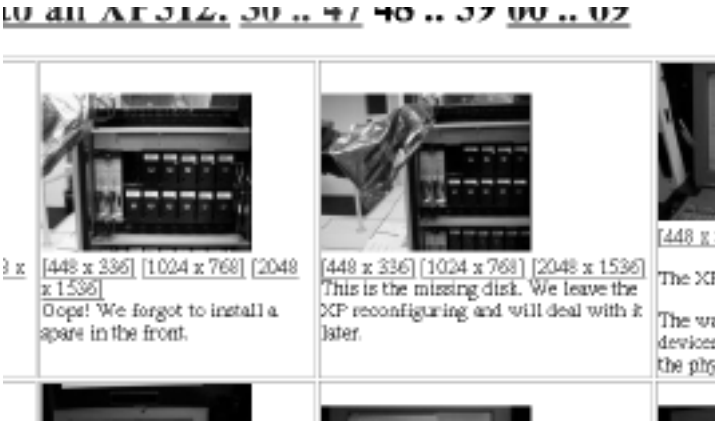


Figure 4: We also document the mistakes we make during the procedure. In this case, a disk we forgot to install. The second panel shows the empty slot; the third panel, the author's hand holding the missing disk.

Figure 4

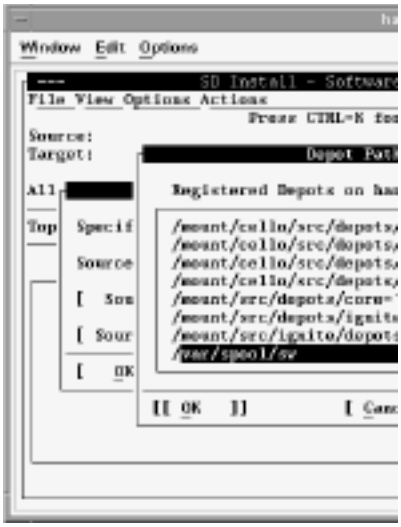


Figure 5a

Figure 5b

Figures 5a–5b: What the competition looks like. A screenshot of a CURSES-based GUI (left) and the output it generates when we record it using the "script" command (right). Tracking what the user did can be quite challenging.

keeping employees by keeping them happy

Part III — The Finer Points

by Christopher M. Russo

Chris is an information technology manager with extensive experience building and running high-performance enterprise, software development, and quality assurance teams.

cmrusso@3rdmoon.com

Employee retention is a difficult challenge that faces most managers today. This is the third in a series of articles designed to help managers better understand the unique needs of employees, the measures necessary for keeping them happy, and the justification and reasoning for doing so.

If you have missed one or more of the articles in this series, please feel free to read them at the author's Web site, located at *http://www.3rdmoon.com/crusso/articles*.

Once you have the basics down of ensuring that you are not personally a management problem for your employees, it is time to start working on the slightly more advanced principles of keeping your staff happy. I feel it is important to stress again that these are ideas most of which I use and all of which I have found to be successful. The important thing is to pay attention to what is going on in your team and be creative. Remember – every situation, every company, every group, every employee is different!

Bad Apples

Any employee can have a very significant effect – either positively or negatively – on the general morale and well-being of your team members. It is very important to understand that the impact of a single team member run amok can turn your good employees into bad ones.

For example, let's say that you have someone on your team who is consistently late, not covering their workload, and causing more problems than they resolve. Your other team members are ultimately going to have to pick up the slack for this person, which is going to be very frustrating for them. Over the long haul, they are likely to start wondering why they are bothering to work so hard, when clearly it doesn't matter all that much anyway – after all, if that one lackluster employee can shirk duties so egregiously with no concern for accountability, then why shouldn't they?

Or say that you have a great team that has been working together and doing a fantastic job for months. One day you bring in a new employee and it becomes evident within the course of a week or two that the person is not working out. Perhaps he is completely ignoring standards that your team agreed on, or maybe he's not bothering to attend meetings that are critical to the operation of the team. Maybe the person has key information that he's not inclined to share with others. Any of these things can be very disruptive and frustrating for your staff, which once again will promote dissatisfaction and unhappiness.

One way or another, a bad employee is likely to appear in your team eventually. It just happens. Sometimes you make a bad judgment call on a hire; other times, people have serious life problems that affect their work; and sometimes people have simply been on the job a little too long and need to get away – temporarily or permanently. You need to deal with such problems quickly and efficiently or your team productivity will come crashing down around you.

Hiring the Right People

The first and simplest step in avoiding a bad apple is to do your best to steer clear of hiring them in the first place. This may seem like an oversimplification – primarily because it actually is. Doing this properly is an enormous amount of work and will take great dedication from both you and your team members.

First, be certain to spend some time working with your team to clearly define what kinds of people you are looking for, and how many of each type you would like to have. It is also very important not only to focus on the skills that you are seeking but also to coordinate with the personality types and general qualities that would make an individual a success in your organization. Having this definition ready and available will help you and your team to make decisions during the hiring process about who is appropriate and who is not. Keep in mind that this definition is likely to change on a daily basis, depending on how the team grows, what needs come and go, and how much you learn about your own team's needs through the process. Be sure to consider this constantly and rethink it often.

Second, have your team members interview all prospective candidates when they come in for interviews. When the interviews are complete, make sure that you have a discussion with all of the members in the interview process before you make your final decision. In some cases, you may need or want to get the entire team into a room to discuss the issue for 15 minutes or so and try to come to a consensus on how you should proceed. This not only helps you ensure that you have found the right person, but also makes certain that the decision to, or not to, hire someone is a decision made by the team and not just by you. People brought in under these circumstances are usually far more successful, because the members of the team feel that they were genuinely involved in the decision, and have an even more deeply vested interest in the new person's success.

Dealing with the Wrong People

When you do find that someone on your team has a problem, you must deal with him or her quickly and effectively. There really is no short way to explain how to handle this – it is a long and arduous process that can take weeks or even months to resolve. The best I can do within the scope of this document is to give some basic tips.

First, be sure to speak to the employee immediately. Try to communicate in a constructive and positive way and gently articulate the reasons for your distress. Always approach the situation from a standpoint of "I'm certain this is not deliberate, so how can I help you address it?"

Second, document everything. This is painful but critical, because if your dealings with this person wind up resulting in a dismissal, you will need the documentation to work with HR, and possibly even the legal department, to back up your decision.

Third, if it looks even slightly worse than a "quick fix," be sure to involve HR right away. Call your business representative and explain the situation and see what is the best approach to handle such issues within your particular organization.

Fourth, bolster and protect your team. This is a tough one. It is important for your team to be aware that you know of the problem and are attending to it, as it will help alleviate their feelings of distress that you may be ignoring the issue that is causing them so much pain. However, it is inappropriate and potentially damaging to the rela-

tionship between the troubled member and your team if you handle this poorly. Again, talk with HR and see how they feel you should best handle it.

Last, remember to keep your cool. This is often very difficult, as sometimes employees like these are causing you as much pain as they are causing your staff – in many cases, they may even be causing you more. If you lose it and start yelling or handling things in an unprofessional manner, it's going to hurt your chances of turning it around, and may even wind up landing you in court.

The Wrong Way to Use Contractors

Contractors are an excellent way to supplement your team's skill set and ensure you have a little more manpower behind your projects. However, if handled improperly, contractors can be a horrible detriment to your team's productivity and morale.

The absolutely worst approach is for a manager to view a contractor as a "hired gun" – someone to come in and "clean up this mess," "really bring this team into line," or anything of that sort. In other words, if you plan on bringing in some contractor to act as some sort of renegade, working in their own way, ignoring the team – or, worse, deliberately slamming the team and the way they work – then the results are likely to be disastrous.

This misguided approach is often the result of a manager trying to "catch up" or "get ahead" by "bringing someone in who really knows [some skill]." To many such managers this seems like a great idea, but the result is usually the same. It may very well get the project done and out in time, but there's a very good chance that the project may not even be right because the contractor has worked alone, and it's almost certain that the contractor and their final result will upset everyone in the team. What's more, your team is going to hold you accountable, not the contractor. Clearly this will be very damaging to your own relationship with your staff.

The trick is to treat contractors almost exactly like everyone else. Obviously, there is not usually justification for sending contractors to training (and your permanent staff may resent you for doing so), but put your consultants through exactly the same interview process as anyone else. In fact, you should consider being more strict on the requirements than you would with a permanent candidate, because the contractor will have less time to ramp up and no justification for training.

Once the contractor is in your organization, be sure to mentor and guide her of him through the ramp-up period as you would anyone else, and be certain to involve the contractor in every aspect of the team relationship that you can – from meetings to game nights. He or she need to be a member of the team through and through. Personally, I call my contractors "temporary permanent staff." It's a distinction that makes a big difference in my team, and even seems to make the contractors happy, since I treat them with more respect and consideration than many of their other clients do.

Mentorship

One of the most frustrating and distressing things for a new employee is to have no idea what they should be doing with their time. Most people feel pretty bad when they have to "bother the manager" with questions about what they should be doing, and they feel even worse when they spend a whole day sitting in the office staring at the wall.

Treat contractors almost exactly like everyone else.

Granted, if they are staring at the wall you may have other issues, but the easiest way to avoid this problem is to assign the new person a mentor the first day they come in the door.

Mentors are great, because they can guide the new employee through the rigors of getting started up in your organization, as well as assigning them some initial tasks that they should be doing to come up to speed. Additionally, they can be the first point of contact for basic questions such as "Where do I fill out time sheets?" "Where can I get CDs to install Visual Basic?" or even "Where is the lunchroom?"

Assigning a mentor will also enable you, the manager, to go about your business without being concerned that you are not unfailingly available for your new employee. I spend about half of my time in meetings – this can sometimes make me fairly inaccessible, and the last thing I want is for my new employee to feel like he or she cannot get help on his or her first days of work. Most of my staff members, however, are usually around, so if one of them is acting as a mentor to the new employee, there's almost always someone available to handle such issues. Plus, if something comes up that does need my immediate attention, the mentor knows how to find me and will be less nervous about disturbing me when it is appropriate.

Be sure to define the parameters of the mentor role with the chosen staff member. For example, it's perfectly appropriate for my new guys to talk to their mentor about technical issues such as where our documentation is kept and how to fill out a timesheet, but it is inappropriate for them to discuss their compensation or other personnel-related issues.

Promote an Environment of Communication

There is absolutely no question that if there is not ample communication within your team, you will have many problems, and most people will find themselves frequently frustrated with you and other team members.

One of the best ways I have found to ensure that communication flows within my group is to promote a very casual atmosphere. The simple fact is that I tend to hire people who are interesting, bright, and capable. Because of this, I often find that I enjoy talking with them and attempt to engage them on non-work-related topics all the time. Since I happen to come from the same general background as most of my employees, this is usually pretty easy. (We're a bunch of geeks with some really odd hobbies.)

In time, I find that the employees frequently stop by my office to say hello and talk about whatever happens to be on their mind. More often than not it's just something funny or a mention of a new way to tweak up my 3-D card, but a lot of the time the conversations are work-related, or possibly just taper into a work-related discussion within a few minutes.

It seems very non-productive on the surface, and people do spend an incredible amount of time in my office talking about movies, X10, Team Fortress Classic, and all manner of things. It seems, however, to make it very easy for my staff to approach me with pretty much anything, and since people come to talk to me frequently, I also get a steady flow of information about what is going on in my team.

The other wonderful side-effect is that other team members usually jump in on conversations, which means that not only are my staff more comfortable with me, but also

with each other. This helps everyone communicate on issues that are more serious with less stress and consternation – after all, how can anyone be so bad if they have their house X10ed to the rafters and are completely addicted to capping people with a sniper rifle in a 3-D computer world, right?

Resources

Your employees cannot do their jobs unless you provide the resources that they need to do them. It seems like a very simple statement, yet for some reason traditional management seems more concerned about the more obvious budget and time issues that are right in front of them, and are unable to see the long-term effects of "cutting a few dollars here and there."

If I told you that I wanted you to build a house, but only made available one box of nails and a rock to bang them in with, you'd think I was crazy. You would think I was even crazier if you had almost no idea how to build a house at all, and that I refused to give you any sort of training on how to do it or any money to hire someone else to do it for you.

How is that any different from asking someone to administer a new technology and not allowing them to attend training on it? Or even to go to the store to buy a book? Would you want to be in a situation where your job depended on your ability to support a production service or product that you didn't know from a hole in the wall? I don't think so.

Books are ridiculously cheap. Consider how much it costs to pay your employees by the hour (roughly their salary divided by 2000), and try to figure out how many hours it will take of your employee banging his head against the wall on a difficult problem with the service, and then compare it to the price of the book. In most cases, you can fairly well assume that you are costing your company money by not allowing them to purchase it.

Now consider that training – it's a lot more expensive, right? Most technical training seminars run on the order of $1200–2500. Certainly, even if you figure several tough problems appearing and figure in the hourly rate of your employee, you're not likely to quite make up that figure.

However, consider the fact that the time your employee is wasting is actually compounded by the fact that they cost a heck of a lot more than just their salary. In most cases, if you include all the support staff, facilities and equipment costs, health insurance and other benefits, you can almost double the cost associated with your employee spinning his or her wheels for hours on end.

Now factor in the aspect that your employee is likely to be working on something that is holding up your company's productivity – and this is most especially true in the technology field. What if your mail server or file server goes down? Perhaps your source control is damaged and you are unable to complete that big project, or lose hundreds of hours of development time for a staff of 10. What if your e-commerce Web site is losing out on hundreds of orders every hour . . . every minute . . . while your bewildered staff member fumbles through online support pages for six hours? Ouch. That's getting very expensive, very quickly.

Consider, too, that your employee is most likely miserable during this time period. He is stressed more than you can possibly imagine, feeling helpless, lost, and totally unable

Your employees cannot do their jobs unless you provide the resources that they need to do them.

Very few companies train their employees properly, if at all.

to deal with a problem that he knows full well is crushing the company's productivity. Not to mention that you are probably going to be yelled at by your management for the same reason. Needless to say, this will make neither you nor your employee feel very good about his life in his present role, and if it keeps up he is very likely to quit.

Now what if your employee could have fixed the entire thing in under an hour had you had the forethought to send your employee to training and had bought a couple of books?

There's also a very interesting twist on this theme – as I mentioned, most management does not seem to get this, so very few companies train their employees properly, if at all. If you do train your staff properly, they will be very happy, reluctant to leave, and word on the street will be that you take good care of your staff, which will actually attract more talent to your organization.

DeMotivational Posters

"Aspire!" "Success!" "Achievement!" . . . "Rubbish!"

Good lord. What pointy-haired, disconnected, completely unaware individual decided that those posters were a "great way to motivate the staff"? For the most part, they do the exact opposite.

Most people are very critical of things like this, and it usually boils down to a simple formula: If you have to say it, then there is a very good chance that either you're not doing it or it isn't working.

My company recently posted an enormous banner on the outside of the building touting the company name and "A great place to work!" Now, truth be told, I actually agree with this statement, but every single one of my staff looked at that and groaned painfully, because it is indicative of the kind of management that just doesn't get it. I'm happy to report that the banner only stayed up for a couple of weeks, and I'm pretty sure it was taken down because they got lambasted for it.

Believe me, rather than investing $5000 to have a big banner printed with some goofy slogan, to be plastered on the side of the building for all to see and despise, have free ice cream day in the cafeteria every Friday for a few months. Then rather than claiming that it is a great place to work, it actually will be. I'm certainly not attempting to claim that the availability of frozen dairy products makes a workplace, but keep going with this general approach and you'll get there eventually.

Buzzword Bingo

Along the same line as DeMotivational Posters is the excessive use of industry buzzwords and catch phrases. To understand what I am talking about, it is important to understand what I mean by each. When I say "buzzword," I am referring to the continual dropping of words that relate to hot topics in the industry. "ASP," "Information Super Highway," "intranet," and "e"-anything are good examples. When I say catch phrases, I'm referring to words and phrases that are used by people to quickly and conveniently summarize a commonly used business or industry concept. For example, "I don't think my team has the bandwidth to support that" or "Let's put a stake in the ground."

The reason that they are similar to the DeMotivational Posters is that those who use them repeatedly are usually demonstrating only that they are too wrapped up in the "secret handshake and code word" of management and aren't really saying anything

with any substance. Think about it. What does "I think we have the bandwidth to support this plug-and-play operation, assuming that everyone is motivated and we really put a stake in the ground" mean? It sounds like a whole lot of nothing.

This sort of behavior makes the manager exhibiting it look really foolish, both in the eyes of his or her staff and in those of other people in the organization. And the manager looking bad can reflect directly upon the manager's team as well. Of course, this makes the team feel badly about their situation, and generally makes them very unhappy.

I worked in one environment where the people on my team wrote a program that would generate random bingo cards where the values in the squares were buzzwords and catch phrases. They would bring these to meetings conducted by a certain manager, and would play a game of buzzword bingo during the meeting. Many times during such meetings, someone would suddenly yell out "BINGO!" Appropriately enough, the manager usually laughed and had absolutely no idea that he was openly being made fun of. All hail Dilbert.

Work Should Be Fun

I don't know about anyone else, but I don't personally want to do anything that isn't fun. In fact, pretty much my only goal in life is to ensure that each and every day is as fun-filled as humanly possible.

Therefore, I work very hard to promote a work environment that is all about doing what makes my staff happy, as long as it can fit within the needs of the business. If I cannot find one of my staff a role that makes them happy within the needs of the business, I will try my hardest to get them into a department where they will be happy. My goal is to make it so that every morning my staff members jump out of bed, excited to go to work. When the day is coming to a close, I want them to be so excited about what they are doing that they literally have to tear themselves away from their project – not because I want them to work long hours, just because I want them to be enjoying themselves.

If you can promote such an attitude within your organization, you are likely to have a lot of happy people — after all, if they're having fun, they will be happy.

Game Night

Believe it or not, I go out and buy games like Quake III Arena and Half Life and expense them to the company. Then I give each of my employees a copy and have them install them on their machines at work. Then, every Tuesday night I buy everyone dinner and we sit around the office for several hours playing games.

It's very funny.

To make it even more entertaining, I hook up a conference call and we all get on and yell at each other while we're playing.

I've been doing this kind of thing at work for several years now, and truth be told I actually do it primarily because it's just a heck of a lot of fun. However, I was recently taken aback when one of my staff pulled me aside and said, "Chris, I feel very tight with this team, and I think the thing that really brought us closest together were these silly game nights."

What does "I think we have the bandwidth to support this plug-and-play operation, assuming that everyone is motivated and we really put a stake in the ground" mean?

Little freebies always seem to make people happy.

I was floored. Certainly, I figured having fun together would promote some level of team friendship, but I never thought it would be so significant. Needless to say, we still play every Tuesday night, and we even try to involve members from other groups. It's been so successful that my boss has specifically asked me to set up a game server accessible to the Internet so that people in remote locations can join in the fun.

It's usually a little difficult to get everyone to play the first night, as people tend to be squeamish about it – either because they've never played or think it's "weird." Encourage them at least to give it a shot, because it's a great time and well worth it.

Thanks for Coming to Work: Have a Shirt!

It's really kind of strange, but little freebies always seem to make people happy. I'm no different. I couldn't be more pleased than when someone gives me a shirt, or takes me out for lunch, or gives me a free Porsche. OK, I'm dreaming a bit here.

The point is that little things like this really make a big difference to people, especially when given at strategic times. If my team works incredibly hard for months on end to roll out a new and important product, I try to commemorate the occasion by having a shirt or other article of clothing made with the company logo and possibly a small bit of tasteful text with the team and/or product name embroidered on it.

It doesn't necessarily have to be for big team wins, either. In some cases, I have given out a shirt or some other small token because one individual really worked far beyond what was reasonable on a critical issue, or when someone did another team a favor that was particularly noteworthy.

It doesn't really even have to be something that one gives for a win. Despite my continual attempts to discourage it, many of my staff members feel compelled to work late nights and over weekends. For these people, I try very hard to run out and get them some food, or at least encourage them to order something and expense it. I have even driven 45 minutes back to work on a Sunday to deliver lunch to a couple of guys who were hard at work trying to get a product out the door before noon on Monday.

Keep in mind that no one said it had to be food or clothing! It could be a silly toy or possibly even something grandiose like a bonus or a weekend getaway for the employee and his or her partner. Maybe a week of cleaning service or pet care for a staff member who is having a hard time pulling away from work. The point is to be creative.

Now, many people may start to wonder about the costs associated with doing things of this nature – after all, some of these suggestions can really add up!

The truth is that most of them are very economical. Shirts cost on the order of US$10–40 each – the former being a nice t-shirt, and the latter being a very nice button-down or rugby shirt. Dinner or lunch usually costs at most $25–30 per person, and if it's from a takeout place, it usually only costs about $10. Silly toys are usually $5–15 each. (I highly recommend Nerf weaponry – a great toy as well as excellent stress relief.) Even a cleaning service or pet care is usually no more than $20–50 a day. Of course a weekend getaway can be several hundred dollars, but it's up to you to determine what is appropriate and within your budget.

If you're still concerned about the costs, consider what you are getting in return. In most cases, we are talking about situations where people are doing more than they really need to in order to get their jobs done. It's a cold-steel way of thinking about

this, but the truth is that the cost associated with you giving them a little something in appreciation is nothing compared to the costs in labor and delayed completion of tasks that would be the result if the employees were not essentially giving you their extra time free of charge.

Even when there is no obvious "justification" for such gifts, they go a long way in making employees feel welcome and appreciated, which of course makes them happy and contented in their jobs.

Conclusion

Clearly there is a lot to do when trying to keep a team happy and together. I have only addressed some of the main points. There are a great many more that could be addressed, and a significant amount of detail that could be put into each. I hope, however, that these will help you get started on the road to significant employee happiness and retention.

Now take the list of principles that you created after reading my last article and attempt to update it with some of the things you would like to work on from this one. Take a moment to review your progress so far and think about how much you have or have not improved in the time that has passed.

Here's the tough part. What I really would like you to do is schedule some time to sit down with your human resources partner to review the list together. Discuss your feelings with the person and see what they think of your ideas and how you might better proceed in achieving your goals. Don't worry, though – they are there to help you and will likely be very pleased to know that you are even thinking about this.

Lastly, if you happen to come up with anything that you think works better than some of the things I have suggested, or perhaps you feel I have missed a principle that you feel is critical, please send me email and let me know. If I get enough of these, I will write a follow-up article and may include a synopsis of your findings in it.

If you happen to be in the tech industry, let's have a contest. I'm trying to see if I can keep no fewer than one-half of my staff for no less than three years. That's quite a challenge – do you think you're up to it? Drop me a line and let me know how you fare. Good luck!

feng shui for computer geeks

by Christine Bagwell

Christine Bagwell heads University of California San Diego's Instructional WWW Development Center.

cbagwell@ucsd.edu

Have you ever wondered whether you have something beyond a technical problem – whether you might have disturbed the natural order of the universe? Are you not only unable to keep your machines going, but you are the first to contract each new Trojan that comes out, and when you drop toast it lands butter-side down? Is your horoscope failing you? Do you get grumpy when it's not posted promptly near the coffee pot, or do you scour multiple astrology sites for the best outlook? Maybe it's time to try Feng Shui, the ancient "Chinese system of maximizing the accumulation of ch'i (vital energy of the universe) to improve the quality of life" (Skinner 1997). Some of the advice is common sense, while other aspects should be reserved for the very desperate.

Clutter

The first item of business is to get rid of clutter, as it creates "confusion, shame and guilt" (CAMEX '02). Clutter creates a pattern of struggle in your life and symbolizes your stagnant energy. It's time to let your local computer parts recycler take your 80MB hard drives, 4MB SIMMs, and VGA monitors. Are you the way I used to be and claim to use "pile management"? Who are you kidding? Do you really need the stack of handouts from USENIX 1989 or the $3 totebag from COMDEX 1995? Worse yet, have you refused to part with shells of previously hot computers – their guts strewn about your office? "If in doubt, throw it out" is a guideline I was told by a Feng Shui consultant. She also advised that if you are really having trouble deciding to discard an item, journal about it.

```
vi journal

> I bought a second processor for my old streaming server in 1999 and never
> got it working right. It's been exposed to sun (I happened to set it in front
> of a window) and every visitor to my office in the past three years has
> fondled the contacts. I'd throw it away, but it reminds me of a kinder,
> gentler era when my customers and I had no interest in Windows Media
> Server. What to do? The Feng Shui master says to ask, "Is it useful? Does it
> lift my energy? Do I love it"?
```

If you are completely unable to throw things out, involve a metal person (one of the five elements: see sidebar). Metal people love to clean.

```
vi journal

> After much anguish, Kevin pried the processor from my hands. It is now on
> the pallet for Surplus Sales. The stagnant air is circulating again.
```

If you are a metal person, be careful about helping others. Practicing Feng Shui for others causes you to take on a great karmic responsibility. You should be careful with others' luck. If I forgot to make disclaimers at the beginning of the article, I just want to say I don't take any responsibility for anyone who takes this advice. I have my own karma to deal with already!

Circulate That Energy

Another item that can help stagnant energy spaces is a bright or reflective object. If hanging crystals will raise too many eyebrows around the department, see whether your office manager will spring for a few mirrors. Keep that ch'i moving! Try hanging wind chimes in the entryway to your computer room. They not only help your energy, but also alert you to prying "super users" who are trying to fiddle with your servers! If possible, a fountain or fish pond should be placed in the southeast sector of the office. For your company, this might mean the shareholders would greatly benefit if you can convince your CEO to place a fish pond in her or his office. Feng Shui practitioners recommend plants in the workplace. Aside from representing a life force, they purportedly help with office relations. Rows of plants should be used to block workers from being seen as soon as a user walks in the door. This might have seemed intuitive. Who wants their boss to catch them checking sports stats or see they are *still* reading Slashdot? Now you have another excuse to barricade yourself. Gurus recommend buying three identical plants. For example, if you have been arguing with a particular user you would place one plant in his/her office, one in your office, and one in a common place, such as a break room or mailroom. Don't let them get too wilted or they will have the opposite effect. Coworkers will get the message that your place of work does not care for people – "If they can't take care of this plant, how will my needs be met?"

Last Christmas one of my programmers bought me a lucky bamboo. He surprised me by placing it near my phone, knowing how much I enjoy using thate technology. It has indeed helped. After hoping it would cause less calls, I accidentally knocked it over, spilling its water and shorting out my phone. Feng Shui works!

Colors

Colors can take on a meaning of their own in Feng Shui. Described as the Basic Bagua, nine colors represent different aspects of life.

Wealth (Purple)	Fame & Reputation (Orange)	Partnership (Pink)
Family (Green)	Health (Yellow)	Children & Creativity (White)
Knowledge (Blue)	Career (Black)	Helpful People (Gray)

You can apply these in a variety of ways. You might try wearing certain colors to an important meeting. Some dignity should be preserved no matter how good it is for your ch'i. I don't care if you are proposing to a supermodel, orange and pink are contraindicated. You should also not paint your house purple and green (though I have seen it done, unfortunately). Installing blue lights in your office or computer room might make people think you are more Nutty Professor than Einstein.

Watch Your Back!

Though you might not gain funding for crystals, mirrors, ficus trees, or repainting, you may be able to change your luck with some rearranging of your office or computer room. If you come away with one thing, let it be that you do not want your back to the door – ever! You need to protect it with a wall or other desk. Do not point your desk directly at colleagues. This can create unconscious hostility. That guy who keeps clicking on viruses in Outlook may be doing it unconsciously intentionally. Try rearranging his office. If nothing else, he will be (a) really confused and (b) too scared of you to

THE FIVE ELEMENTS

Elements are associated with your year (most important), month, day, and hour of birth. These elements together make up the Chinese horoscope (Skinner 1997).

1. Subtract 10 from the last two digits of the year you were born.
2. Repeat until you have a number less than or equal to 10.

Number	Element
0	Metal
1	Metal
2	Water
3	Water
4	Wood
5	Wood
6	Fire
7	Fire
8	Earth
9	Earth
10	Metal

test your patience further. Remind him that you have a master key in addition to control over his mail and thus his whole life!

Using your personal magic number, you can determine the best direction for you to face when seated at your desk. I was pleased to find that I was facing west. I went overboard, also changing my seat at a standing meeting. When coworkers asked why I had moved from the seat I'd held for the last three years, I giddily replied, "It wasn't my auspicious direction!" (BTW, that was #2 on my 10-point action plan to ruin my career.) Weeks later, a colleague threw my life into a tailspin by bringing me a compass. I had assumed our building was built squarely pointing north. I had been facing southwest, not west, the whole time! There are also auspicious directions for toilets. The one saving grace was that I no longer needed to sit sideways. Nonetheless, I would have rather found I had been facing "great prosperity & success" more than "relationships & family." I can work on my ch'i for family at home.

Another easy item is to avoid criss-crossing wires. You thought interference was the worst evil to come of that sloppy practice. You probably didn't know you were introducing conflicting energies. Depending on how bad your situation is, you may need a few tie wraps or a whole rewire of your network closet. The door may be closed, but your ch'i is getting messed up right behind it!

You also don't want sharp edges. They can create feng sha, "a noxious ch'i destroying wind" (Skinner 1997). And you thought that unpleasant smell was the desktop support guy. Stay away from desk lamps with sharp edges, opting for round models. If your bookshelf has blunt edges, attach doors. In my office, the earthquake retrofitting did just this. The 2" Plexiglas affixed to each shelf softens the sharp edge of the shelf. This could be the perfect reason to go from telco racks to enclosed racks in your computer room. Give your ch'i a break!

How have you been inspired? Send me anecdotes or jpegs of your most recent Feng Shui change to cbagwell@ucsd.edu.

REFERENCES

CAMEX 2002. "Feng Shui and Your Retail Space." Los Angeles, CA. 2/21/2002

Skinner, Stephen. *Feng Shui*. New York, NY: Paragon, 1997

STUG 20 Years Ago

by Peter H. Salus

peter@netpedant.com

The Software Tools User Group published its 12th newsletter ("Software Tools Communications") in July 1984. Among other things, it announced the availability of two implementations of the "Basic Tape" for IBM CMS and for IBM VMS. (I still have my copies of "CMS Notes" by Marc Donner: 1981, revised 1984.)

Also announced was a "potpourri" tape, dubbed the "Toys Tape." Nancy Travis wrote, "Despite its title, the tools included are far from trivial, including the long-awaited YACC, LEX, LISP, and TCS as well as improved versions of format and the archiver."

There was also a new version of Ratfor along with ratfix programs.

(For those unfamiliar with Software Tools, the software tools, and Ratfor (a) buy and read a copy of Kernighan and Plauger and (b) take a look at the NetBSD packages collection, devel/ratfor.)

Vern Paxson "revealed" that RTSG (Real-Time Systems Group) had developed a Lex tool. Ken Poulton wrote about his shell enhancements ("I have implemented much of csh in Ratfor").

There were contributions from Neil Groundwater and Van Jacobson, as well as a report from the EUUG by Teus Hagen.

Thanks, guys.

When I was working on my UNIX history, a decade ago, Debbie Scherrer let me borrow her STUG archive; Lou Katz gave me several issues of the Communications. I'd like to thank Chris Kantarjiev for giving me his collection, completing my set.

Finally, it's 15 years since I left USENIX. This means that the Boston meeting will see Ellie Young's 15th year as executive director. Congratulations!

using C# abstract classes

by Glen McCluskey

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

glenm@glenmccl.com

In our last column we discussed C# interfaces, a mechanism for specifying a contract, a particular set of methods, that an implementing class must define.

In this column we'll consider another somewhat similar feature known as abstract classes. Such classes are a basic design and structuring tool for C# applications and allow you to provide partial class implementations that can be customized.

An Example

Imagine that you're doing some work with benchmarking and performance analysis, and you'd like to develop some C# utility classes to aid in this effort. You need one utility that executes a particular routine or task repeatedly and keeps track of the elapsed time.

Here's some C# code that captures this idea:

```
using System;
using System.Threading;

abstract public class PerfUtils {
    abstract public void DoRun();

    public long TimeRun(int repcount) {
        long currtick = DateTime.Now.Ticks;
        for (int i = 0; i < repcount; i++)
            DoRun();
        return DateTime.Now.Ticks - currtick;
    }
}

public class BenchMark1 : PerfUtils {
    public override void DoRun() {
        Thread.Sleep(500);
    }
}

public class PerfUtilsDemo {
    public static void Main() {
        PerfUtils pu = new BenchMark1();
        long elapsed = pu.TimeRun(10);
        Console.WriteLine("elapsed time in milliseconds = " +
            elapsed / 10 / 1000);
    }
}
```

PerfUtils is an abstract class, meaning that it declares but does not define all its methods. An abstract class, like an interface, specifies a contract that must be fulfilled or implemented by another class (BenchMark1). In the case at hand, the TimeRun method is implemented, but the DoRun method is not – it's an application-specific method that a subclass must supply.

Since an abstract class does not have definitions for all its methods, it's not possible to create instances of such classes, and the following code will evoke a compiler error:

```
abstract public class A {
    abstract public void f();
}

public class AbstractNew {
    public static void Main() {
        A aref = new A();

        aref.f();    }

}
```

If such code was legal, then at runtime there might be calls to unimplemented methods.

We can say that an abstract class must be derived from or extended in order to be of any value; that is, there must be a further class that uses the abstract class. By contrast, the other extreme is a sealed class that cannot be derived from at all. For example, this code is illegal:

```
sealed public class A {
    void f() {}
}

public class B : A {}
```

Sealed classes are useful in a case where a derived class would alter the semantics of the class in some way, causing it to break.

Factoring Common Functionality

One of the key differences between an interface and an abstract class is that an abstract class can provide partial implementations of some methods, as a base, and thus factor out common functionality. By contrast, interface methods cannot be defined, and so the following code is invalid:

```
public interface IA {
    void f() {}
}

public class B : IA {
```

```
    public void f() {}

    public static void Main() {}
}
```

In the earlier example, the common functionality is TimeRun, a routine that's useful across a range of applications. It works with an application-specific method DoRun, supplied in a derived class.

Another example of factoring is the code below which illustrates a bit of a framework for putting together some collection classes (lists, hashtables, etc.):

```
public interface ICollection {
    int Size();

    bool IsEmpty();

    // ... other methods ...
}

abstract public class Collection : ICollection {
    abstract public int Size();

    public bool IsEmpty() {
        return Size() == 0;
    }

    // ... other methods ...
}

public class ListCollection : Collection {
    public override int Size() {
        // ... logic for computing size of ListCollection ...

        return 0; // dummy return
    }
}

public class Test {
    public static void Main() {
        ICollection ic = new ListCollection();
    }
}
```

ICollection is an interface that declares some common methods all collections will have. These methods include both Size, used to obtain the number of elements currently in the collection, and IsEmpty, which determines whether a collection is empty. Since IsEmpty can be implemented in terms of Size, it makes sense to provide a definition in the abstract class. By contrast, the appropriate logic to compute the size of a collection will vary, depending, for example, on whether the collection is a list or a hashtable.

Using interfaces and abstract classes together in this way is a very powerful technique. The abstract class serves as a means of factoring common functionality and providing an implementation for it. But because an interface is also defined, it's possible to sidestep the abstract class and start over with your own custom implementation that implements the interface. If you program in terms of interfaces, as we described in the last column, then it's easier to substitute your own implementation for that provided to you in a standard library.

Other Differences Between Interfaces and Abstract Classes

An abstract class can provide a partial implementation, as we mentioned above, whereas interfaces are used to specify but not implement a contract.

Another difference involves multiple inheritance. It's possible to implement more than one interface at a time, like this:

```
public interface IA {
    void f();
}

public interface IB {
    void g();
}

public class C : IA, IB {
    public void f() {}
    public void g() {}

    public static void Main() {}
}
```

whereas the corresponding code with abstract classes is not permitted:

```
abstract public class A {
    abstract public void f();
}

abstract public class B {
    abstract public void g();
}

public class C : A, B {
    public override void f() {}
    public override void g() {}

    public static void Main() {}
}
```

Implementing multiple unrelated interfaces can be quite useful, and sometimes the term "mixin" is used to describe this technique. For example, in the previous column we declared an interface:

```
public interface IDistance {
    double GetDistance(object obj);
}
```

A class implements this interface to provide functionality to compute the distance between two objects, for example the Euclidean distance between X,Y points or the number of days between two calendar dates. The class could implement several of these interfaces, each one adding a bit of functionality.

Polymorphic Programming

Our final example illustrates another aspect of programming with abstract classes. Modern object-oriented languages make use of what is called polymorphic programming, with virtual functions as another term for the same idea. This idea centers on programming with a common interface across a hierarchy of classes and their associated objects, and runtime binding for method calls.

We can tie down this concept by considering an example:

```
using System;

abstract public class A {
    abstract public void f1();

    public virtual void f2() {
        Console.WriteLine("A.f2");
    }

    public virtual void f3() {
        Console.WriteLine("A.f3");
    }
}

public class B : A {
    public override void f1() {
        Console.WriteLine("B.f1");
    }

    public new void f2() {
        Console.WriteLine("B.f2");
    }

    public override void f3() {
        Console.WriteLine("B.f3");
    }
}

public class Polymorphic {
    public static void Main() {
        A aref = new B();

        aref.f1();
        aref.f2();
        aref.f3();
    }
}
```

In this code, we create a new B object and assign it to a base class reference (A is the base of B). We then call methods f1, f2, and f3 through the base reference.

What happens when the methods are called? For f1, B.f1 is called, because the object pointed at by the A reference is really a B, and we specified that B.f1 overrides A.f1, and A.f1 is abstract anyway.

The same consideration applies to B.f3. The method is virtual (bound at runtime), and we're operating on a B object.

What about f2? It's marked as virtual in A, but B declares f2 to be "new," that is, the virtual dispatch hierarchy is broken. So A.f2 is called.

Virtual method dispatch is extremely powerful. For example, suppose that you have an abstract class Graphics that represents a graphics object and declares a Draw method. Then you have a variety of classes that extend the abstract class and that represent graphical objects like Circle and Line and Point and Rectangle. Instances of these classes can be assigned to a Graphics reference (pointer), and then the Draw method can be called on each instance, without worrying about the exact type of the object being referenced.

Abstract classes are fundamental building blocks that you can use to structure your C# programs. They are a good choice if you'd like to provide a partial class implementation that can be extended and customized.

practical perl

by Adam Turoff

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

ziggy@panix.com

An Introduction to Web Services

Web services are easily the most hyped topic in software development since the big "Push Technology" craze of the mid-1990s. Proponents believe that Web services represent the future of software development. Detractors believe that Web services are nothing more than an ever-expanding orb of complexity.

In fact, the concept of Web services is a very broad topic with ill-defined edges. There's a grain of truth in each of these perspectives.

At the low end, Web services are just a new form of remote procedure calls (RPC). They generally use HTTP as a transport layer, coupled with some form of XML to encode messages and data. This enables simple CGI programs, mod_perl handlers, and stand-alone Web servers to implement a service. By using HTTP and XML, Web services make it easier to focus on writing an application other than on the low-level details of handling sockets, implementing a protocol, or debugging client-server transactions.

At the high end, Web services are a loosely federated series of specifications that describe business processes. These specifications are expressed in XML-based languages that describe how to execute a business process, such as processing a purchase order or provisioning new hardware. Other activities that can be described under the Web services rubric include transactional reliability, process coordination, and security.

With these two different perspectives on Web services, it is easy to get confused. On the one hand, Web services are just a new way to build software using ubiquitous standards such as HTTP and XML. On the other hand, Web services are a very high-level description of how to automate business, where the implementation details are mostly irrelevant. In this article, I focus on the first meaning of "Web service" – the low-level reinvention of RPC using HTTP and XML.

Building Services

Let's consider a practical example. Suppose you wanted to work with your friends to create a dictionary. You need to collect definitions and look up terms you have defined already. You could start with a server that supports the dict protocol, except that that protocol does not allow you to create new definitions.

Where do you begin? You could start by creating a new protocol, "newdict," that supports the features you need. Then you would need to create (and test!) new software for your client and server programs. Frankly, that path leads to a lot of work and not a lot of value. Your mission here is to create a new dictionary, not rediscover the arcana of dealing with sockets and writing protocols.

Another alternative is to use a preexisting RPC library. Instead of developing a protocol, using RPC allows you to create one service that responds to multiple procedure calls, such as add_definition and get_definition.

This is where Web services become interesting. Low-level RPC libraries require packing and unpacking data (bytes) to send messages over the network. On the other hand, Web services transfer data between clients and servers using XML. This helps, because Perl's support for low-level RPC libraries has always been weak, but its support for XML and text has always been strong. Additionally, Web services are a language-neutral RPC mechanism. It is easy to write a connect for Perl clients to Java and .Net Web services, or vice versa.

There are three main ways to write a Web service: REST, XML-RPC, and SOAP. REST is not a Web service technology, but a description of how the Web works, and a set of conventions for how to write well-behaved programs that respect those principles. Chances are good that if you've ever written a CGI script, you've written a REST service.

XML-RPC is a very simple protocol for creating Web services. It is an XML description of how to invoke a remote procedure, and an XML description of the corresponding result. XML-RPC is a simplification of SOAP, a richer Web services protocol that can be used for simple RPC or for more complex interactions involving message-based communications.

Perl supports all three kinds of Web services. To use REST Web services, the only thing that is absolutely required is a Web client library, like LWP. The RPC::XML and XMLRPC::Lite modules both support XML-RPC, and the SOAP::Lite module supports the SOAP protocol. (XMLRPC::Lite can be found in the SOAP::Lite distribution.)

Publishing a Dictionary Authoring Service

One of the easiest ways to start with Web services is to add an XML-RPC interface onto an existing program. Here are the guts of a newdict program to look up definitions and define new terms. It has two main actions, add_definition and get_definition, which insert data into a database and query it:

```
#!/usr/bin/perl

use strict;
use warnings;
use DBI;

my $dbh = DBI->connect("dbi:SQLite:dbname=dictionary.db");

my $insert_stmt = $dbh->prepare("INSERT INTO dictionary (term, definition)  VALUES (?, ?)");

my $select_stmt = $dbh->prepare("SELECT definition FROM dictionary WHERE term=?");

sub add_definition {
    my $term = shift;
    my $definition = shift;

    return $insert_stmt->execute($term, $definition);
}

sub get_definition {
    my $term = shift;

    $select_stmt->execute($term);

    my $rows = $select_stmt->fetchall_arrayref();

    ## Transform a list of lists of strings into a list of strings
    my @definitions = map {$_->[0]} @$rows;

    ## Return a structure containing the term, and all definitions found
    return {term => $term,
        definitions => \@definitions};
}
```

That's it! Like many services (DNS, dict, whois), this newdict service is merely an interface to a data store. In this case, the data store is a simple SQLite database. The newdict service could grow into something more complex, but this is enough to get started.

The rest of the server side of this application – responding to requests, and running as a daemon – is handled through a Web service library. To publish this small program as an XML-RPC Web service, all that's necessary is to add some glue code using the RPC::XML module from CPAN:

```
use RPC::XML::Server;

my $server = new RPC::XML::Server(port => 10000);

## Add a method to add new definitions
$server->add_method({
        name => "newdict.add_definition",
        code => \&add_definition,
        signature=>['int string string'],
    });

## Add a method to add search for definitions
$server->add_method({
        name => "newdict.get_definition",
            code => \&get_definition,
            signature=>['struct string'],
    });
```

```
## Launch the service
## (This statement never returns)
$server->server_loop();
```

The first step is to create a RPC::XML::Server object and specify which server port to use (10000 in this example). Next, each operation that this XML-RPC server will support must be added into the server object with a call to add_method. Each method is defined with a name (like newdict.add_definition), a reference to the subroutine to call, and a description of the inputs and outputs to that method.

The third part of the XML-RPC method definition, the "signature," is a textual description of data types for the result and input values for a method. XML-RPC defines eight basic datatypes that can be used in XML-RPC messages: integers, floating point numbers, Boolean values, date/time values, strings, and also binary structures, arrays, and hashes (called struct in XML-RPC).

With XML-RPC, methods return a single value, and the first datatype in a method signature is the result. The other values in the signature describe the types of the input parameters. In the example above, the add_definition method returns an integer value and takes two strings, a term to be defined and its definition. The get_definition takes a single string as input (the term to be defined) and returns a hash containing both the term and its definitions.

Using RPC::XML::Server, the final bit of glue is the call to $server->server_loop(). This starts a stand-alone Web server (using HTTP::Daemon) and waits for requests to arrive. Each XML-RPC request will be an HTTP request containing a bit of XML that describes what method to call and the parameters to pass to that method. For each message received, the RPC::XML::Server object will parse the XML message, identify what method is requested, pass the arguments, and receive a result. The server object will then encode the result in XML and send back to the client program, where its XML-RPC library will perform the same deserialization.

We're almost done. The RPC::XML library calls RPC methods with an initial parameter containing information about the XML-RPC request received from the client. This parameter needs to be captured in each of the published methods:

```
sub add_definition {
    my $xmlrpc = shift;
    my $term = shift;
    my $definition = shift;

    ##...
}

sub get_definition {
    my $xmlrpc = shift;
    my $term = shift;

    ##....
}
```

And now we're done. By adding a few lines of glue code and making two small changes to the original newdict program, we now have a newdict Web service.

Using the Dictionary Authoring Service

Now that we have a newdict service, anyone with an XML-RPC library can communicate with it. All that is necessary is a URL to find this XML-RPC server (*http://localhost:10000/* in this example), the names of the methods to call, and an understanding of the signatures for those methods.

To connect to this service, we can use the RPC::XML::Client module that also comes with RPC::XML distribution:

```
#!/usr/bin/perl

use strict;
use warnings;

use RPC::XML::Client;
```

```
my $client = new RPC::XML::Client("http://localhost:10000");

$client->simple_request('newdict.add_definition',
    'XML-RPC',
    'An RPC Protocol');

$client->simple_request('newdict.add_definition',
    'XML-RPC',
    'A Web Services Protocol');

my $result = $client->simple_request('newdict.get_definition', 'XML-RPC');

my $definitions = $result->{definitions};
print join("\n\t", "$result->{term}:", @$definitions), "\n";
```

This new dict-client program starts by creating an RPC::XML::Client object that will act as a proxy for the XML-RPC server found at *http://localhost:10000*. Each call to $client->simple_request() encodes an XML message to send to the XML-RPC server, where the request is processed and a result returned. The client object then translates the XML result into native Perl data structures, which are then returned from the simple_request method.

And that's it! All of the necessary glue code to talk to the server is handled in the RPC::XML::Client module. Communicating with an XML-RPC server is as simple as writing straightforward Perl code.

Conclusion

Web services are many things to many people. At the heart of it all are simple RPC mechanisms that use HTTP and XML. While Web services may not provide the highest-performance solutions, they greatly simplify the work required to create client and server programs.

Using XML-RPC is an easy way to get started using Web services with Perl. The RPC::XML module makes it easy to glue an XML-RPC server interface to an existing piece of code, and easy to write clients to talk to XML-RPC servers.

the tclsh spot

by Clif Flynt

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

clif@cflynt.com

Mobile Agents in Tcl

The previous "Tclsh Spot" article described how to build a client/server pair using the Secure Socket Layer extension (TLS) and a safe slave interpreter. This article extends that client/server pair to support transferring complete Tcl programs to the server for remote evaluation.

A mobile software agent is a set of code that can be sent from one trusted system to another to be evaluated on the remote system. A few years ago, this was considered a rather exotic style of program architecture, but it's become commonplace now. For example, when you download a Web page with a script component, your browser evaluates that code in a safe sandbox on your system. You are trusting that the code will not escape that sandbox and damage your system. Whether or not this trust is misplaced is left as an exercise for the reader. (If you hear a rant about IE and lack of proper sandboxes, you aren't mistaken.)

The protocol for the Client/Server pair described in this article is simple. The Server sends a Ready prompt when it's ready to accept input. The client sends Tcl commands to the server and waits for a Ready.

The simple client/server pair described in the previous article was able to implement most of this protocol, but had some shortcomings.

One of the least obvious problems is that using Tcl's basic interp create -safe command provides a bit more security than we need.

The safe interpreter created with the interp create -safe command is very limited in what it will do. Among the restrictions is that a safe interpreter has absolutely no access to the file system. This makes it impossible to load Tcl extensions or pure Tcl packages into the slave interpreter. We can extend a safe interpreter with the interp alias command that was described in the previous article to add support for loading packages, safe access to the file system, and such. In fact, there are enough requirements for extended functionality that Tcl provides a package with the common aliases already built.

The Tcl distribution comes with the ::safe:: package which extends the base safe interpreter. The interpreters created with the ::safe::interp create command still run in a safe sandbox, but also have hooks to allow a few more actions, including loading pure Tcl packages and extensions with a SafeInit entry point to be loaded into the safe slave.

There are several commands in the ::safe:: package (documented under man safe), but the three important ones are:

Syntax:::safe::interp create ?name? ?key value?

::safe::interp create	create a new safe interpreter. Returns the name of the new interpreter
?name?	An optional name for the new interpreter. The default name will be something boring like interp0.
?key value?	Optional keyword/value pairs to fine-tune the safe interpreter's environment. Some include:

-statics boolean
: True allows the slave interpreter to load statically linked packages (load {} Tk). False disables this ability. The default is True.

-nested boolean
: True allows the slave interpreter to load packages into sub-interpreters. False disables this ability. The default is True.

-deleteHook script
: A script to be evaluated before deleting an interpreter. This hook gives the slave interpreter a chance to do cleanup (perhaps log an exit message) before being destroyed.

Syntax:::safe::interpAddToAccessPath path

::safe::interpAddToAccessPath
: Adds directories to the list of directories the slave interpreter can search to find packages to load. This list is maintained in the parent interpreter. A safe slave cannot access the list, thus preventing a slave from leaking information about a filesystem to the outside world.

path
: The directory path to add.

Syntax:::safe::interpDelete interpName

::safe::interpDelete
: Delete the slave interpreter. All state for that interpreter will be destroyed.

interpName
: The name of the interpreter to be destroyed.

The following code uses the ::safe:: commands to create the interpreters. For a more restrictive environment, you can substitute the base interp create etc. for these.

The previous example server had only one set of state information and only one slave interpreter. For a single-purpose, stateless server, this is appropriate. However, if two agents tried to use such a server at the same time and each sent a script with the same name but a different body to be evaluated, one agent would be running the wrong code.

The agent server must maintain separate interpreters and sets of state information for each active connection, to keep the agents from interfering with each other. Fortunately, all the state for an active agent can be kept in that agent's interpreter. All we need to track in the server is the channel associated with an agent, the name of the interpreter for that agent, and incomplete input waiting to be evaluated.

In C, you might have an array of state structures to hold the necessary information. It might look like this:

```
struct agent {
    IO_channel *channel;
    Tcl_Interp *interp;
    char *input;
} activeAgents[10];
```

When data is read from a client, the server steps through the structures in the array activeAgents until it finds the appropriate element, identified by the IO_channel field.

Tcl does not support an array of structures the way you would define the data in C or Java. However, the associative array provides the same functionality for this requirement. In Tcl, we can initialize the equivalent data structure keyed by the channel identifier, like this:

```
set State(interp.$channel) [interp create -safe]
set State(input.$channel) ""
```

The procedure to establish a connection with a new client described in the previous "Tclsh Spot" article just waited for the handshake to be complete and established a fileevent to handle the input.

```
proc openConnection {channel clientaddr clientport } {
    global tlssignal

    # Wait until the handshake is complete
    fileevent $channel readable \
        [list handshakeHandler $channel $clientaddr]
    vwait tlssignal($channel)

    fileevent $channel readable\
        [list processMessage $channel]

    # fconfigure for line buffering.
    fconfigure $channel -buffering line
}
```

In the agent-based server, the procedure starts the same, but after the handshke is complete, it creates and initializes the interpreter for this agent. The interp alias for writeLog is a procedure that was defined in the previous article; it allows a safe slave interpreter to write to a predefined log file.

```
proc openConnection {channel clientaddr clientport } {
    global State
    global tlssignal

    # Wait until the handshake is complete
    fileevent $channel readable \
        [list handshakeHandler $channel $clientaddr]
    vwait tlssignal($channel)

    fileevent $channel readable [list readLine $channel]

    # fconfigure for line buffering.
    fconfigure $channel -buffering line

    # Create a safe interpreter
    set State(interp.$channel) [::safe::interp create]

    # Link the 'writeLog' procedure in this environment
    # to the 'log' procedure in the safe child interpreter.

    $State(interp.$channel) alias log writeLog
    puts $channel "Ready"
}
```

The next trick is to send the agent server a procedure to run. Since the server is running in a secure sandbox, we don't need a special protocol to support this: we can send normal Tcl com-

mands to the server. This even includes complex commands such as procedure and namespace definitions.

However, in order to download scripts, the agent server must be able to accept multiple line inputs. The previous server read a single line and evaluated it. If the line were something like "procedure foo {args} {," it would be incomplete, and the slave interpreter would throw an error. The server needs to know when a complete command has been received before it tries to evaluate that input.

This problem has been solved in many ways ranging from complex parsers to requiring a special pattern like "\n.\n" to mark the end of input.

Each of these techniques has some limitations:

1. An input parser must return the same results as the actual evaluation parser. Keeping these in sync and verifying that neither has unexpected exception conditions can cause headaches.
2. A special pattern must not be something that could exist in a real message.

Tcl has an elegant solution to this problem: use the actual parse engine to test input.

One of the powerful aspects of Tcl is how many of the interpreter's internal functions are exposed to the script writer. The script writer can use the complex parser that's already built into Tcl to test input data. Since the same parser is used both to test the input and then to evaluate it, this guarantees that the test parser and evaluation parser accept the same information,

The info command gives a Tcl script a sneak-peek into the state of the interpreter. You can get a list of known commands, global and local variables, the argument and body of a procedure, and a fair amount more.

The info complete command gives the script writer access to the Tcl interpreter's parsing logic. It returns TRUE (1) if the string it's presented is a complete Tcl command, and FALSE (0) if there are mismatched parentheses, quotes, braces, etc.

Syntax: info complete string

string	The string to check for matching braces, brackets, quotes, etc.

A server that only accepts single line commands from a client could look like this:

```
proc readLine {channel} {
    global State

    # Read a line of data
    set len [gets $channel line]

    # Close if we've recevied an EOF from the client
    if {($len <= 0) & [eof $channel]} {
        close $channel
        return
    }

    processMessage $channel $line
}
```

The agent server is just a little more complex. Once the server has received a complete command, it can eval it within the proper safe interp, but until then, it needs to save the data and check each time there's a new line.

The code below saves each line as it's read in an associative array indexed with the field name input and the channel identifier. It checks the saved text to see if it's a complete Tcl command, and if it is, processes it. If not, it appends a newline (the gets command strips off newlines), and continues.

One trick with the info complete command is that an empty string has no mismatched quotes, braces, etc., and is thus complete, even if it's meaningless. The script checks for empty lines just to avoid wasting time processing nothing.

```
proc readLine {channel} {
    global State

    # Read a line of data
    set len [gets $channel line]

    # Close if we've received an EOF from the client
    if {($len <= 0) & [eof $channel]} {
        close $channel
        ::safe::interpDelete $State(interp.$channel)
        return
    }

    # Put the data in a safe place
    append State(input.$channel) "$line"

    # And check to see if we've got a complete command

    if {(![string equal "" $State(input.$channel)]) &&
        ([info complete $State(input.$channel)])} {
        processMessage $channel
        set State(input.$channel) ""
    }   else {
        if {(![string equal "" $State(input.$channel)])} {
            append State(input.$channel) "\n"
        }
    }
}
```

The processMessage procedure is quite simple. All it needs to know is the channel identifier, which it can use to index into the State associative array to find the appropriate input and inter-

preter. It evaluates the script within the proper interpreter and sends the results to the client, along with a new Ready prompt.

```
proc processMessage {channel} {
    global State

    if {[string match "" [string trim $State(interp.$channel)]]} {
        return
    }
    set rply [$State(interp.$channel) eval $State(input.$channel)]
    puts $channel "$rply"
    puts $channel "Ready"
}
```

With the addition of a dozen lines of code (including some comments), we've changed the previous simple server into one that handles receiving scripts to be evaluated remotely, while maintaining the security of the system the server is running on.

The previous client was very simple. It sent a command to the server and waited for the reply. The reply was always a single line, and each command had a reply.

The protocol for communicating with this agent server is a little more complex. Input is requested with a Ready prompt, and other information is the response to the previous client command. We can add a procedure to watch for the Ready prompt. This procedure grabs a line of data and checks to see if it's the expected prompt. If it's not the prompt, the line of input is saved; when the prompt is received, the input is returned to the calling script.

```
proc wait4Prompt {channel prompt} {
    set rtn ""
    while {[string first $prompt [set line [gets $channel]]] != 0} {
        if {[eof $channel]} {error "CLOSED" "Channel Closed"}
        append rtn "$line\n"
    }
    return $rtn
}
```

With the addition of that procedure, a simple agent that requests a base64 encoding of some data might resemble this:

```
set setup {
package require base64
}

set serverSocket [tls::socket -password getPassword \
    -keyfile $certDir/clif@noucorp.com.key \
    -certfile $certDir/../1000.pem \
    -cafile $certDir/../rootca.pem \
    -request true -require true $host $port]

fconfigure $serverSocket -buffering line

wait4Prompt $serverSocket Ready

puts $serverSocket $setup
wait4Prompt $serverSocket Ready

puts $serverSocket [list base64::encode "This is a test
                                    message."]
set rply [wait4Prompt $serverSocket Ready]

puts "The reply is: $rply"
```

And that's a basic mobile agent client/server pair. In this implementation, only valid agents are allowed to use the server. The validation is done with the OpenSSL extension (TSL) which only allows connections from an agent that knows the passwords and keys to the SSL configuration files that exist on the server.

Including comments, this is 161 lines of code. Of course, interesting agents will have more application-specific instructions. As usual, this code will be available at *http://www.noucorp.com*.

CAN-SPAM

New Federal Anti-Spam Law Sets Nationwide Standards for Commercial Email Messages

The new federal law regulating spam went into effect on January 1, 2004. It creates a single nationwide set of rules governing commercial email that overrides the inconsistent standards in state anti-spam law. It also criminalizes certain conduct and creates new civil penalties with substantial monetary fines.

Unfortunately, CAN-SPAM is a toothless tiger that nullifies most aspects of every state's anti-spam laws and leaves spam victims without meaningful legal recourse. The single exception is that it doesn't preempt state laws which prohibit false or deceptive content in any portion of a commercial email message or its attachments. Thus, the suits that Serge Egelman is so joyfully bringing against spammers under state law as described in the accompanying article may soon be a thing of the past.

This article outlines CAN-SPAM's most salient features and presents recommendations for compliance.

The Need for Federal Legislation

Unsolicited commercial email ("spam") has dramatically increased during the past three years.[1]

In the absence of federal legislation, many states passed their own laws to regulate spam. Those laws imposed standards and requirements that differed significantly from state to state. Since email addresses don't specify geographic locations, it is almost impossible for commercial email senders to know with which of the disparate state statutes they were required to comply.

California passed a law in October that would have flatly prohibited sending unsolicited commercial email messages to recipients unless they opted in to receiving them. Other states passed different anti-spam laws with different compliance standards. Also, many of these state laws were opposed by the direct marketing sector. As a result, the pressure on Congress to establish nationwide (and more forgiving) rules became intense. CAN-SPAM is the result.

Summary of the New Law

The new law:

- Prohibits senders from using false "header" information, false return addresses, and deceptive subject lines
- Requires senders to include valid physical postal addresses in their messages
- Requires senders to provide an opt-out mechanism and to comply with all opt-out requests
- Mandates labeling in the subject line of messages containing adult-related content
- Makes advertisers legally responsible for compliance by their email service vendors

by Dan Appelman

Dan Appelman is a partner in the international law firm of Heller Ehrman, White & McAuliffe, LLP. He practices intellectual property and commercial law, primarily with technology clients, and is president-elect of the California Bar Association's Standing Committee on Cyberspace Law.

dan@hewm.com

1. In 2001, spam accounted for approximately seven percent of all email traffic, whereas it now accounts for over 50 percent. Some estimates are as high as 70 percent.

- Authorizes the Justice Department, the Federal Trade Commission, and other federal agencies to enforce the new law and establishes fines and jail terms for violators
- Permits state attorneys general and Internet service providers to bring civil suits and to be granted injunctive relief, money damages, and attorney fees
- Authorizes (but does not require) the Federal Trade Commission to create a national "do not email" registry
- Pre-empts all state laws that regulate commercial email, except to the extent that state law prohibits falsity or deception in messages or their attachments.

Principal provisions of the new law

No false or misleading transmission information

It is now illegal to send email messages that contain materially false or misleading "header" information (i.e., information that identifies the source, destination, or routing of an email message). Header information that does not identify the true email address of the sender because of false source information, because the sender has registered for multiple email accounts, or because the message has been routed through other computers for the purpose of disguising its origin is considered materially misleading.

CAN-SPAM makes it a crime to promote a trade or business, or the products or services of a trade or business, using commercial email messages that contain false or misleading transmission information. Businesses that hire advertising companies or email service vendors to help them promote their goods or services by email may themselves be deemed to have violated the Act if the email contains false or misleading transmission information, such as disguising the true identity of the sender.

No deceptive subject headings

It is now illegal to send commercial email that includes subject headings that are likely to mislead the recipient about the content or subject matter of the message.

Commercial email must include an identifier, an opt-out mechanism, and a physical address

All commercial email messages must include (i) a clear and conspicuous identification that the message is an advertisement or solicitation; (ii) a clear and conspicuous notice of the opportunity to opt out of receiving further commercial email messages; and (iii) the sender's postal address.

Commercial email must include a functioning, accurate, conspicuous return email address

The new law requires that all commercial email messages contain return email addresses that recipients can use to opt out of receiving future solicitations. These return addresses must appear conspicuously in the email message, they must be accurate, and they must function for at least thirty days after the transmission of any outgoing message.

No continued sending after opt-out

The new law gives senders a ten-day grace period after receiving any opt-out request in which to cease sending further email messages to that recipient. Continuing to send emails after that date will be deemed a violation of CAN-SPAM. It will also be illegal for the sender or any other person who knows of the recipient's opt-out request to give

the recipient's email address to anyone else. This provision is Congress's attempt to restrict the common practice of selling or exchanging recipient email mailing lists.

Activities that will make the violations described above more serious

Address harvesting and "dictionary" attacks

Under CAN-SPAM, it is illegal to transmit commercial email messages to email addresses that are obtained using automated means from certain Internet Web sites or online services. This prohibition includes collecting email addresses from those Web sites and online services that have privacy policies stating that their email addresses will not be shared with those who would use them for commercial purposes. The new law also prohibits sending commercial email messages to email addresses that are generated by combining names, letters, or numbers into random permutations.

Creating multiple electronic accounts

It is illegal under the Act to use automated means to register for multiple email accounts with an Internet registrar or online service provider.

Relaying messages through unauthorized access to other computers or networks

It is illegal under CAN-SPAM for senders to disguise the origin of their commercial email messages by sending them through computers or networks to which they do not have authorized access.

Special requirements pertaining to sexually oriented material

CAN-SPAM requires commercial email messages that contain sexually oriented material to include marks or notices in their subject headings to inform recipients of their content and to facilitate filtering. It also limits the content that can be initially viewable by the recipient of such messages to (i) the aforementioned marks and notices; (ii) an opt-out mechanism; and (iii) instructions on how the recipient may access the sexually oriented material. The Act orders the Federal Trade Commission to prescribe the content of the marks and notices.

The "do-not-email" registry

CAN-SPAM requires the Federal Trade Commission to send Congress a plan and timetable for establishing a nationwide marketing "do not email" registry similar to the new "do not call" registry. But implementation of the plan is discretionary, not mandatory, and comments by several FTC commissioners indicate that the Commission is unlikely to establish the registry.

How Will CAN-SPAM Be Enforced?

Most violations of CAN-SPAM will be enforced by the Federal Trade Commission under its authority to prosecute unfair or deceptive trade practices. The Commission will investigate consumer complaints, and those found to be engaging in unlawful practices will be subject to fines and possible prison sentences in actions brought in federal courts. Other agencies have authority to enforce the Act against certain defendants, such as banks and savings and loan associations, credit unions, broker-dealers, regulated investment companies, investment advisers, insurance providers, air carriers, and telecommunications service providers. In addition, the prohibitions against predatory and abusive email practices can be enforced by the federal Department of Justice.

CAN-SPAM can also be enforced by state attorneys general in civil actions in federal court. State officials can seek injunctive relief to prevent senders from continued viola-

tions as well as monetary damages and attorney fees. In addition, Internet access service providers who have been adversely affected by a violation of the Act can bring a civil action to enjoin further violations and to recover monetary damages and attorney fees.

Unlike many state laws that will now be preempted, the new federal law does not give private individuals the right to sue.

Against Whom Can CAN-SPAM Be Enforced?

In general, the new law will be enforced against those who "initiate" the transmission of unlawful email messages. The Act defines "initiate" very broadly. It not only includes those who actually send the messages, but also those who create or pay for them. Many companies hire email service vendors to help them with their solicitations; and the new law applies to both. In addition, the Act makes it unlawful for any person to promote goods or services by means of commercial email messages they know or should know contain false or misleading transmission information. Customers of commercial email service vendors must therefore have a reasonable level of confidence that the service they use is in compliance with CAN-SPAM's requirements or risk being held in violation of the Act themselves.

What is the Status of State Anti-Spam Laws After CAN-SPAM?

CAN-SPAM explicitly supersedes all state laws that regulate the use of email to send commercial messages except to the extent that a state law prohibits falsity or deception in any portion of a commercial email message or its attachments. Therefore, most state anti-spam statutes will be preempted, including most of the provisions of the new California law that would have prohibited sending commercial email messages absent some preexisting relationship between the sender and the recipient. The new federal law imposes a nationwide "opt-out" standard that nullifies the much tougher "opt-in" approach adopted by California and several other states.

Issues Created by CAN-SPAM

Problems for advertisers

Many commercial email solicitations involve three or more parties: the advertiser, the service provider that actually sends the email messages under contract with the advertiser, and the recipient. The new law makes the advertiser equally guilty or liable for the transgressions of the service provider. Advertisers will be required to closely supervise the practices of their service providers, and will be well advised to renegotiate their contracts to provide for indemnification and other protection in the event they are prosecuted or sued for the actions of their service providers.

Opt-out logistics

The new law gives senders ten days to comply with opt-out requests. In order to comply, the advertiser must purge the recipient's email address from all of its service providers' lists, which may be difficult within that short time frame. Advertisers may be compelled to require service providers to share opt-out information with one another, and service providers may resist those requests. If the opt-out requests are sent to the advertiser rather than directly to its service providers, the service providers may not find out about them in time to comply with the ten-day deadline.

Labeling requirements

The new law requires all commercial email messages to be clearly and conspicuously labeled as advertisements or solicitations. But the law includes no guidelines for satisfying this requirement.

Recommendations

Everyone who sends commercial email messages must comply with CAN-SPAM. Here are a few suggestions for complying with the new law:

- Know the requirements of the new law and adopt a compliance plan.
- Designate someone within your organization to be in charge of implementing the plan.
- Keep records of opt-out and opt-in requests. Document how your company complies with those requests.
- Include a clearly explained opt-out mechanism in every commercial email message.
- Label all commercial email messages as advertisements or solicitations.
- Understand the compliance procedures of your service providers and make sure those procedures satisfy CAN-SPAM's requirements.
- Renegotiate your contracts with service providers to provide that they will indemnify you from any liability resulting from their failure to fully comply with CAN-SPAM.
- Review your insurance coverage to determine whether and to what extent it protects you in the event of suit.

suing spammers for fun and profit

by Serge Egelman

Serge Egelman is an undergraduate in Computer Engineering at the University of Virginia. He expects to graduate next month barring any unforseen difficulties. He plans to continue battling spammers on the side as a graduate student next year

serge@guanotronic.com

How would you like to expand a body part by up to 30%? What would you say if you could get your degree from a leading non-accredited university for just three easy payments of . . . . Okay, we all see it every day, spam advertising everything from low-interest mortgages to an Extreme Colon Cleanser. A few recent studies have shown that spam, or unsolicited commercial email, accounts for nearly 50% of all email this year. That number is rapidly increasing.

Personally, I receive about 80 messages a day, though I have friends who receive 500–1000 messages. I run my own mail server with a fairly common Sendmail configuration; I subscribe to the black-hole lists and have all the default spam prevention measures enabled. I also use SpamAssassin (*http://www.spamassassin.org*) to sort most of my spam (procmail puts all the tagged spam into a different mailbox). However, almost a dozen messages still get through to my main mail spool each day. It is rather annoying, and so I have started taking action.

Last year I happened to notice that my state, Virginia, has an anti-spam law. Since then I have been archiving all my spam, tracking down those responsible, and taking them to court. The following is a summary of what I have done, what I have learned, and what you can legally do to decrease your spam volume.

DISCLAIMER: The following is just a personal account of my experiences and should in no way be interpreted as legal advice. I am not a lawyer and do not claim to be one. Additionally, all quotes contained herein are from memory and are therefore not verbatim.

The Laws

In 1997, Nevada became the first state to pass legislation regulating spam, and currently 35 states have passed spam laws. The state laws share many similarities. They make it illegal to send messages with forged headers, deceptive subjects, no opt-out mechanism, or no clear indication in the subject line that the message is indeed an advertisement. States that have laws use some subset of these requirements (e.g., Virginia makes it illegal to falsify header information, whereas New Mexico only makes it illegal to omit "ADV" from the subject line). The legislation also outlines civil remedies, and a few states go so far as to make sending unsolicited email a criminal offense.

What this means is that if you are in one of the 29 states that provide for a civil action, you can start taking spammers to court and getting awarded statutory damages for all the spam that you receive. The majority of these laws provide $10 per message, but a few (California and West Virginia) go as far as allowing $1000 per message, as well as all associated legal fees. Some of the states that do not allow civil actions on behalf of the recipient or the ISP allow complaints to be filed by the state attorney general (in the case of Pennsylvania, the attorney general shall remit 10% of the damages collected to the person who filed the complaint).

The main problem with current legislation is that it is not unified; though similar to each other, laws vary from state to state. Some state laws grant personal jurisdiction to the court over out-of-state spammers, some require the spammer and the recipient to be in the same state, and some require the spammer to have knowledge that the recipient is in a particular state. What is needed is a federal law to regulate all spam sent within this country.

In fact, the 108th Congress saw nine such bills, one of which, the 2003 CAN-SPAM Act, was signed into law at the end of last year and went into effect on January 1. The law outlines many of the aforementioned requirements for sending unsolicited mail.

First, it is illegal to send a message with forged header information. The law defines header information as:

> "the source, destination, and routing information attached to an electronic mail message, including the originating domain name and originating electronic mail address, and any other information that appears in the line identifying, or purporting to identify, a person initiating the message."

Essentially, it is illegal to put any false information in any part of the header. Next, the law makes it illegal to use deceptive subject lines. This is clarified by saying that the subject must not mislead a recipient "acting reasonably under the circumstances."

Finally, the law regulates the opt-out mechanisms that each message must contain and what must happen after a recipient has notified the spammer. Each message sent must contain either a URL or an email address to which the recipient can respond in order to opt out. Furthermore, this address must be functioning for at least 30 days after the original message was sent. The sender is also required to include his or her physical postal address. Upon being notified by the recipient, the sender has up to 10 days to cease sending further spam. Finally, the law mandates that the FTC outline a plan for implementing a nationwide Do-Not-Email registry within the next six months. This, of course, is similar to the Do-Not-Call registry. While this law has many provisions for regulating commercial email, the main question is, how are they to be enforced?

The CAN-SPAM Act provides a few different measures for enforcing the new restrictions. The law provides both criminal and civil penalties. A defendant deemed guilty can be fined and/or imprisoned for up to three years for the first offense. Repeated offenses can carry prison terms of up to five years. In terms of enforcement, the FTC is granted the most power; however, state attorneys general can also bring cases to court. Since this is a federal law, cases can be brought in any US district court. In terms of statutory damages, the FTC or a state attorney general can seek up to $250 for each message sent in violation of the law (up to a maximum of $2 million for any offense other than falsifying header information).

Unfortunately, the law is sparse on private civil remedies. The only private right of action that is outlined is for ISPs. ISPs that receive spam may take the spammer to court and claim statutory damages of $100 for every forged message and $25 for every message that violates any other part of the statute. Additionally, legal fees can be claimed. This, though, is a two-way street: The defendant may also claim legal fees if they get a favorable ruling. While this is the only private civil action that may be pursued, it might be available to more people than one might think. The law defines an ISP as any entity that provides email or any other Internet service to others. Thus, anyone who provides email or shell accounts is technically an ISP and can therefore pursue spammers under this law. However, in addition to the risk of losing and having to pay the spammer's legal fees, there is another downside: this law can only be used in a federal court, so in addition to the increased filing fees, you will most certainly need a lawyer.

While it is clear that this law has potential, there is much criticism surrounding it. Last year, the Direct Marketing Association (DMA) spent millions of dollars lobbying against the newly implemented Do-Not-Call registry. This legislation was a huge blow to the telemarketing industry (many telemarketing companies belong to the DMA). It might come as a surprise to hear that the DMA has been heavily lobbying in favor of the CAN-SPAM Act (affectionately called the I-CAN-SPAM Act by many anti-spam-

mers). Just recently, California amended their existing anti-spam laws to make the sending of any spam illegal (instead of just imposing restrictions, like most other states). This new law was set to go into effect on January 1. Unfortunately for Californians, the *federal* law contains a preemption clause:

> This Act supersedes any statute, regulation, or rule of a State or political subdivision of a State that expressly regulates the use of electronic mail to send commercial messages, except to the extent that any such statute, regulation, or rule prohibits falsity or deception in any portion of a commercial electronic mail message or information attached thereto.

This means that any state law that outright bans spam is now null and void. However, laws that only regulate deceptive spam (e.g., outlawing forged headers or fraudulent subject lines) will remain. While the new CAN-SPAM law outlines certain restrictions on how messages can be sent, it takes an opt-out approach: It is perfectly legal to send spam to recipients until they say to stop (or, more accurately, 10 days after that time). This, of course, leaves open the possibility for a spammer to sell the email address to someone else (and thus the recipient has to tell all subsequent spammers to stop). Additionally, the law legalizes spam sent from someone with whom you have an existing business relationship.

This is why it's no surprise that the nation's largest ISPs have heavily backed this legislation. Most media outlets have two sources of revenue: the subscribers who pay for the service, and the advertisers who pay to get their message seen by the subscribers (e.g., you pay for cable television, yet still have to endure commercials). This business model is now moving to ISPs. You pay them for an email account and Internet access; now they can legally send advertisements for additional revenue. Furthermore, it protects this business model in that they can forward you spam from companies who are paying and in turn pursue those who spam and have not paid for the privilege.

With all the more serious crimes occupying the time of the FTC and others charged with enforcing this act, it's laughable to believe that they will be devoting large efforts to enforcing it. In fact, California has had an anti-spam statute for a few years, yet the first criminal conviction occurred only a few months ago. The DMA and others who lobbied for the CAN-SPAM Act also understand this. In fact, when the bill was signed into law, Bill Gates wrote an article hailing this as a tremendous victory. Clearly, in the coming months we can expect a whole line of ludicrously priced Microsoft anti-spam products. So while this law probably won't reduce the flow of spam by itself, individuals . . . errr . . . "small ISPs" can still take action by saving spam and tracking down those responsible.

Tracking Down the Spammer

Spammers' use of open relays and forged addresses discourages most people from bothering to hunt down the sources of their spam. While there is still a lot of spam that comes from legitimate return addresses, most spammers go out of their way to obfuscate the origin. The fact of the matter remains, however, that they are trying to do business with you and therefore need to leave a way for you to contact them. Naturally, this means that they will include a URL for their Web site.

There are many tools online that can aid in locating the origin of your spam. The site that I use the most is Spamhaus (*http://www.spamhaus.org*). They have a running database of known spammers and known IP blocks that host spammers. Most spammers

will use fake whois information, so this site is an invaluable tool. Simply enter the IP address corresponding to the URL and, hopefully, it will return the lucky winner. If there are no hits, there is a good chance that the IP address of the corresponding DNS server (taken from the whois information for the advertised domain name) might be in the Spamhaus database. Spammers purchase large IP blocks and will often use them for spamming once or twice before moving on to a different IP block, but changing the IP of a DNS server so frequently is a lot more difficult.

If you have gotten lucky and have found a name and address for a spammer, the next step is to locate the address for their "registered agent." This is their retained attorney, whom you will serve with the court summons. This information is available online in most states and, by law, is required to be on file with the Office of the Secretary of State (or equivalent) in the state where the company is incorporated. Finding the address of the registered agent is often as simple as going to the secretary of state's Web site and using an online query tool. Some states charge for this information, but it is still publicly available. If all else fails, you are probably safe just using the regular business address of the company.

So You Are Going to Court

The first time I did this was in March of last year; I had received 80 messages sent by Etracks.com, Inc., a California-based marketing company. After the first message I received from them, I sent a response explaining that they had violated state law, and if they continued to do so I would not hesitate to settle this matter in court. Naturally, I never received a response, and the spam continued. So I went to the local courthouse to file a complaint in small claims court.

A friend who had been to small claims court a few times gave me some invaluable advice: Know exactly what you want before you go to the clerk's office, since they are not allowed to give any sort of legal advice. Their job is merely to give you the forms requested. With this in mind, I showed up at the court clerk's office with all the information pertinent to my case, and explained that I needed a Warrant in Debt for small claims court (usually called a court summons). I explained that under the statute this company owed me $800 plus my court fees. The clerk was a bit confused, since she did not realize an individual could take a corporation to small claims court. I was a bit baffled, since the court's Web site said that in fact I could. After arguing for a few minutes, I eventually went home to print out the Web site.

I returned with a hard copy of the page and also went one step further and printed out the section of state code that corroborated it. I then proceeded to fill out the proper forms and pay the court fee. Next, I wrote up a settlement letter and attached it to a copy of the summons (so that they would know I was serious). Again, I never received a response from the defendant.

On the date of the trial, I printed out all my communications with Etracks.com (my cease-and-desist letter as well as the settlement letter), the abbreviated headers for all messages received (rather than the full messages, I printed out the To, From, Subject, Date, and Received headers), and, finally, since it was my first time in court, I decided I should write up my testimony. When I got there, I ended up having to wait for about an hour before my case was heard. When it was, the judge was a bit surprised; this was apparently the first spam case he had ever heard. I explained a little bit about the problem, he asked a few questions, and since I had more than enough evidence and the

defendant failed to show up, I was awarded judgment for $841 ($800 for the original complaint, $41 for the court costs).

Send Lawyers, Guns, and Money

Since then I have been filing suits every few months. This is always preceded by a letter sent to the defendant. Last August I received a call from a man who identified himself as Brian Benanhaley, COO and in-house counsel for SubscriberBase, Inc. I had just filed suit against them for 47 messages and they had apparently just received their court summons. This was nothing new; I had been through this routine before – they call, threaten, and then don't follow through. I was expecting the same thing, to receive another default judgment. After just finishing the routine "we're not going to settle, so you better drop your suit since we're not at fault," Mr. Benanhaley went on to threaten to countersue. Having not wanted to really continue this conversation, I told him that he should consult my attorney.

My roommate's parents have a law firm in town, Snook & Haughey P.C., and while his parents did not seem very interested in my spam endeavors, one of the other attorneys in their firm did. He mainly practices business and consumer protection law; he had never done anything like this before and so was very interested in trying some of these cases. Being a poor college student, I had no intention of retaining anyone, but since he was interested in trying to make some case law (our research showed that no cases under the Virginia anti-spam statute had ever been appealed, so we were both hoping for an appeal so that we could set a precedent), he was willing to take the case on contingency.

My attorney, Jim Garrett, seemed to think that SubscriberBase was very serious; they had apparently retained a local firm. My only concern now was that the case was not as straightforward as my others. In Virginia, it is illegal to "falsify or forge header or routing information in any matter." The messages I received all came from seemingly random usernames, though the domains were registered to SubscriberBase. Replies to the messages would bounce. However, the defendants alleged that since they owned the domain names, this does not violate the law. I wanted a judge to make that decision.

In the weeks approaching the trial, the saga became odder and odder. When SubscriberBase first contacted me, they kept insisting that they would be filing a countersuit in their state of South Carolina. Jim insisted that there would absolutely be no merit to any suit that they would bring; I hadn't done anything wrong. They might allege that this was a frivolous suit, but he said that would also be without merit since they continued to spam me even after I had sent them a clear cease-and-desist letter. Each week they would call with a new empty threat; they were still making a counterclaim, this time in Virginia. Next they said that they wouldn't make a counterclaim but would ask the judge for their fees. A week or so before the trial, they called Jim asking if I would be willing to pay their legal fees. He said that he asked them to repeat that, since he was sure he misunderstood. In fact they were indeed asking if I would voluntarily pay for their expenses. You can guess our response.

The fact that I had a lawyer was a fairly big surprise for them. They were in for a few more surprises. I had recruited two volunteers from the Computer Science Department to testify as expert witnesses: a professor and the system administrator. In addition to going through the department's mail logs and finding spam sent by SubscriberBase, we made another huge discovery the day before the trial. I had not noticed this earlier, but the X-Mailer header was clearly forged on all of the messages.

The 47 messages claimed to have been sent from 29 different mail clients running on 11 different platforms (including Amiga). This clearly qualified as "falsified in any manner." We made a printout of all the X-Mailer headers to be entered into evidence during the trial.

On the day of the trial, we met at the law firm to make our final plans. It turned out that my roommate's father, Lloyd Snook, a criminal defense lawyer, would be doing the cross-examinations and the closing arguments. The five of us arrived at the courthouse, and met the opposing four in the hallway (their COO/counsel, CEO, a technical person who didn't look any older than 15, and their local retained counsel).

We decided to call their CEO, Jeff French, first. He answered a few questions about what his company does, his role, their business model, etc. Then he explained in detail what systems they use: a cluster of Linux servers sending out messages via RoboMail (a commercial mass-mailing package). Finally, he explained how they obtain addresses: They purchase lists from other companies of addresses that are "confirmed opt-in."

It was then my turn to testify. I explained how I am a student and run my own server which provides email and Web hosting and that I receive an inordinate volume of spam. I explained that when I register on various sites, I try to do a fairly good job of reading privacy policies and never register for sites that outright say they will sell my personal information. I also create aliases to help in determining where spam is coming from (and which sites are violating their privacy policies).

During my cross-examination, Mr. Benanhaley listed a few sites, asking if I had been to any of them. I had heard of one and explained that I have tracked a lot of spam to that site. I also explained that the defendants had sent spam to three different addresses of mine. It was clear to the court that they make no effort to confirm that each address has really opted in. Finally, I was asked what I was discussing with a friend in the hallway prior to the trial.

Recently I have been working with a group of friends on an idea for a cryptography-related startup and was discussing this outside the courtroom; I guess they must have overheard me. Because we are currently working with another lawyer on a related patent, I really did not want to divulge too many details. So I glanced over at my lawyer, and responded, "I'm sorry, but I really don't see how this is relevant."

"Please just answer the question," said the judge. "If there are objections, your legal team will raise them."

"Objection, your honor. Relevance?"

"Sustained."

The defense then approached the bench and began explaining to the judge how I was committing barratry and that this case was entirely without merit. He went so far as to pull out printouts of slides that were posted on our UNIX User's Group Web site; I had given a talk a few months earlier on current trends in anti-spam tools as well as anti-spam legislation. Most of the people in the audience (in addition to my legal team) were now giggling, since we all knew what I was really talking about and that the defense was really grasping here.

The judge sounded very annoyed; they argued back and forth for what seemed like 10 minutes. I changed my mind and interjected, "If you want, I can just answer." My lawyers shut me up; they were having fun watching the judge lecture the opposing

The 47 messages claimed to have been sent from 29 different mail clients running on 11 different platforms (including Amiga).

counsel. The judge ended the argument by saying, "Regardless of the plaintiff's motivations, we are here to determine if you have violated the law."

After another hour or so of testimony (both my witnesses testified that these messages contained falsified information, though we were still only arguing about the addresses in the From line), we took a recess.

The defense called their CEO and asked that he be an expert witness. My legal team objected after it became clear that his experience was limited to his role as CEO of SubscriberBase. The judge agreed; he could only testify based on his own experiences. During the cross-examination, we asked him why they use the randomly generated From addresses. He explained that "anti-commerce" individuals use various filters to stop spam, and thus the company's "legitimate" advertisements often get filtered out by accident; therefore they take measures to get past such filters.

With this statement, he had admitted that they intentionally format messages to evade filters in order to increase their profit; it was time for the coup de grace. "You testified earlier that you exclusively use RoboMail under Linux to send messages. Why is it that there's no identifier corroborating that in these messages?"

"The same reasons I just mentioned."

"Then maybe you can explain why 29 different mail programs are listed," my lawyer said and handed the list to the CEO, "and why there are also 11 different operating systems mentioned here. Was it not your earlier testimony that you used Linux exclusively?"

Everyone on the defense team suddenly turned bright red. They knew it was over. Half the courtroom was giggling. Mr. French finally responded, "I'm not a technical person, so I'm not entirely positive what we use, come to think of it."

"Well let's just go through the list then." My lawyer began naming off everything on the list, without stopping to wait for responses. When he came to the Amiga, he waited. We all wanted the defense to say that they use an Amiga for their spamming.

"It's entirely possible. We might use one. I would have to double check."

Mr. French was finished testifying. We assumed that given the recent humiliation and disqualification as an expert witness, they would not even bother trying to get their technical person to testify. The defense rested. Closing arguments were made. They argued that "clearly" I opted in and that what they are sending was not even spam.

We asserted that, hypothetically, even if I had opted in, the majority of the messages were received after I sent them a cease-and-desist letter (a copy of which was in evidence). Additionally, the court was reminded that the defense had testified that they willfully altered the headers to evade filters.

After four hours of testimony and arguments, the judge spoke. He started by saying that he was "confident that whatever decision is reached, it will only determine which side files the appeal." He, of course, was right. Unfortunately, this was a different judge from my previous cases, and he was not very familiar with the law. He said that the case would be taken under advisement for a week. Outside I commented, "Hopefully, the first thing he does is go home and check his email."

A week passed before Jim called: we won. I was awarded $470 for the spam, $50 for the filing fee, and $5000 in legal fees (*http://www.guanotronic.com/~serge/opinion.jpg*). The

defendants had 10 days to file an appeal; we were confident that they would, and we welcomed it. And they did, the day after the initial ruling was entered into the court record. In Virginia, when a defendant appeals a civil case, they must also post a bond for the amount awarded to the plaintiff. They had 30 days to do this, but failed. Thus the appeal did not occur and the saga came to an end.

So You've Won a Judgment

I have since been to court about half a dozen times, and have yet to lose a case, though I have only had one spammer show up. Since my first time in court, the judge seems to have gotten friendlier and more interested in what I am doing. No one likes spam; judges receive it too. I am currently owed a little over $5000, but being owed money is quite different from actually receiving it. However, there are legal methods that can be used to aid in the collection process.

The first thing that should be done is to put the judgment on file in a court of record; in most cases, this will be a court above the small claims court (in Virginia, for instance, small claims is part of the general district court, and the court of record is the state circuit court above that). This means that creditors and anyone else with an interest in your debtor will be able to readily see that you are owed money, and this can adversely affect their credit rating.

Once the appeals period is over, various legal means can be used to enforce your judgment. Most of these, however, will only work if the defendant resides or owns property in the state of the judgment. If they do, the first thing that you must do is locate their assets and property; this is done with a Summons to Answer Interrogatories (again, this might be called something else in other states), which means they must show up in court to answer your questions.

If they own real estate, you can place a lien on it (so that your judgment must be satisfied before they can sell the property). If they own other personal property (or property belonging to the business), you can get a Writ of Fieri Facias (sometimes called a Writ of Execution) to have their property sold at public auction. Finally, you can get a Garnishment Summons to garnish their wages or any bank accounts that they have. In any case, you will probably end up the winner (also, all costs incurred during collection can be added to the judgment in most states).

If your debtor resides in another state, there is still hope. You can have the judgment domesticated to the debtor's home state. This means the judgment goes on file with a court there and becomes legally binding in that state. You can then use the measures mentioned above, assuming they are applicable in the new state. To domesticate a judgment; usually, all that is involved is sending a certified copy of the original judgment along with a court fee.

I personally have been going a step further and using a collection agency. Their job is to keep contacting the debtor until they are willing to pay. Collection agencies usually work on contingency and will take anywhere from 30 to 50% of the total amount collected. I use Dun and Bradstreet Small Business Solutions (*http://sbs.dnb.com*), which has another advantage: They are the largest business credit reporting company. This means that when a company is not cooperating with the collection process, their credit report will be adversely affected.

If you do not have the time to try collecting the judgment on your own, using a collection agency is probably the easiest solution. The only disadvantage is that it can take a

long time before you see any money. I just received my first check from the agency for around $400 after three months, care of Mr. Joshua Baer of Skylist.net (I was owed $631 before the commission). For those short on time or energy, there is still one other option: selling the judgment. There exists a market for court judgments: an individual or corporation will give you a fraction of the judgment's value in exchange for the right to collect on it. It won't be much, but it's something and you will receive it quickly.

Conclusion

While I do not think I have personally made an impact on the spam problem, I have certainly helped to decrease the amount I have been receiving. Most have stopped sending me messages after being served with their court summons, though with one company it took three judgments before they got the message. The majority of states have laws that can be exercised by any resident, but such laws are useless unless they are used. The same applies to the newly enacted federal law. Spammers are not going to be deterred by the laws until more individuals begin to take action.

the bookworm

by Peter H. Salus

Peter H. Salus is a member of the ACM, the Early English Text Society, and the Trollope Society, and is a life member of the American Oriental Society. He owns neither a dog nor a cat.

peter@netpedant.com

BOOKS REVIEWED IN THIS COLUMN

SENDMAIL COOKBOOK
CRAIG HUNT
Sebastopol, CA: O'Reilly, 2004. Pp. 388.
ISBN 0-596-00471-0.

POSTFIX
KYLE D. DENT
Sebastopol, CA: O'Reilly, 2004. Pp. 264.
ISBN 0-596-00212-2.

DESIGN RESEARCH
BRENDA LAUREL, ED.
Cambridge, MA: MIT Press, 2004. Pp. 334.
ISBN 0-262-12263-4.

XML HANDBOOK, 5TH ED.
CHARLES F. GOLDFARB AND PAUL PRESCOD
Upper Saddle River, NJ: Prentice Hall, 2003. Pp. 1200 + 2 CD-ROMs.
ISBN 0-13-049765-7.

I've got to begin with a confession this month: I was going to write about *Malware*. But Chuvakin's review came in, and it's one with which I concur 100%. So I've not bothered. Just read the review following this column and go with what Anton says. It's a fine book.

I also need to apologize: In the December column, I mentioned Waldrop. Knowing just how many computer folk read SF, I made an unwarranted assumption. Howard Waldrop is an SF author. I think he's up there with the best, but he's neither as famous nor as popular as Gibson or Sterling. So I apologize to those who wrote me and to others – I didn't intend to be a snob. (But if you get a chance, read some of Waldrop's stuff.)

We've All Got Mail

OK. We've all got copies of Costales, Allman, and Rickert (*Sendmail*), yet we still may not have completely mastered the sendmail.cf file (sometimes I wonder whether anyone has, but that merely exposes one of my shortcomings).

Craig Hunt's *Sendmail Cookbook* really helps. I was especially impressed by his chapter on AUTH (pp. 242–273); it is extraordinary. The chapters on masquerading (pp. 103–150) and securing mail transport (pp. 274–317) are very fine, too. While the chapter on spam is good, I fear that it just isn't enough. Unfortunately, I don't have a panacea. I get a lot of trash every day, and I can't adjust my filters as rapidly as header variants and bogus subject lines originate.

Craig, this is a fine job.

Another fine job is Dent's *Postfix*. This is a very good guide to a splendid MTA. Written by Wietse Venema while he was at IBM Research, Postfix was released as open source in 1998 and has become fairly widely employed since. This book is, to the best of my knowledge, only the second book on Postfix, and the less said about Blum's padded book of several years ago, the better.

Dent has done a really neat job. In fact, his chapter on blocking spam is a great supplement to Hunt's.

Designed to . . .

Design Research is a brilliant anthology, full of interesting articles and thought-provoking assertions. I found it extraordinarily difficult to read, however. This is because the book designers ("The Offices of Anne Burdick, Los Angeles") have run amok. Each section is demarcated by color (orange for Section 1; sage for 2; pale yellow-green for 3; etc.). The yellow-green was near impossible for me to read. I have no idea what the diagram labels on p. 143 say, or most of the headings in the following section; I cannot discriminate the letters from the background.

So, in some manner, Section 3 ("Process") was opaque to me. Many of the texts were interesting, even though they were periodically interrupted by areas of color in which some things were intelligible, but the whole was not.

The volume is "over-designed" to death, resembling an early issue of *Wired*. Surely, this is not the result of any sensible "design research."

I have long admired the work of Brenda Laurel. What a pity to have the content marred and obscured by out-of-control designers.

XML

There's a new (fifth) edition of Goldfarb's *XML Handbook*. Previous editions were quite useful. The new one is quite enormous, and could use a bit of editing and some reorganization. This is an excellent updating, nonetheless.

book reviews

MALWARE: FIGHTING MALICIOUS CODE

ED SKOUDIS (WITH LARRY ZELTSER)

Upper Saddle River, NJ: Prentice Hall, 2004.
Pp. 647.
ISBN 0-13-101405-6.

Reviewed by Anton Chuvakin

I rarely label something a "masterpiece," but Ed Skoudis' *Malware: Fighting Malicious Code* is nothing short of that. The book is an amazing combination of depth and breadth, which I always love in a security book. Moreover, it combines these with lively and easy to follow presentation style as well as Ed's trademark humor (featuring the traditional overuse of the word "evil"). In many regards, the book is more fun to read and more packed with material than his previous work, *Counterhack*. The book also strongly conveys the excitement that the author obviously feels about this field.

The book covers the wide scope of malicious code (viruses, worms, mobile code, rootkits, Trojans, backdoors) in a logical and well-structured fashion. This is not your grandmother's "virus book," as it covers all sorts of malicious programming and scripting. Chapter summaries, reasons "why you need to know," examples, clear diagrams, accurate analogies (often abused in other security books) are all there to educate and entertain. Early on, I thought that some of the examples were a bit simplistic, but later I noticed that they worked extremely well, especially for some of the technologies I was not intimately familiar with (such as the Windows kernel).

The book starts with a nice, clear definition of "malicious code," which helps to set the frame for the rest of the book. It goes on to cover all the types of malware outlined above. Highlights included the exciting material on future worms and possible trends in worm activity; coverage of various browser-based attacks, including evil plugins, ActiveX, and XSS (as utilized by malware); the presentation on sniffing backdoors and hacks using VNC; and the coverage of source-Trojans (with detailed analysis of recent attacks against common open source software) and some neat data-hiding tricks.

The section on rootkits (two chapters for application and kernel-level), however, was my favorite, presenting this malicious technology in a logical, very well-written fashion. Starting with brief but useful overviews of Linux and Windows kernels, coverage continues by noting "five ways to manipulate a kernel" for malicious purpose. The material on Windows rootkits and kernel tricks is fascinating. Several examples of fairly recent kernel rootkits are analyzed for both platforms.

If the rest of the book is exciting, the author's discussion in Chapter 9 of the possibility for BIOS and CPU microcode malware is simply awesome. The book follows this up with coverage of some end-to-end malware-related attacks scenarios, which are lots of fun to read.

The book is topped with a chapter on analyzing malware, complete with suggestions for a lab setup and a structured presentation of various analysis approaches (static and dynamic). An analysis template is there as well.

Overall, the book is a great read for any security professional, system admin, or aspiring hacker. Its focus includes both attacks and defenses, with a slight bias toward attack (it also often touches on "defenses against defense" tricks, utilized by malicious software). UNIX and Windows platforms are both covered at almost equal levels of detail.

SECURITY WARRIOR

CYRUS PEIKARI AND ANTON CHUVAKIN

Sebastopol: O'Reilly, 2004. Pp. 525.
ISBN 0-596-00545-8.

Reviewed by Rik Farrow

Security Warrior is touted as an advanced book, and some parts of it actually are. I obtained a copy of the book because I was interested in learning more about reverse engineering of hostile code. The book does start out with four chapters on reverse engineering, with the chapter on working with Linux the most extensive in terms of material and explanation.

The second chapter covers Windows code disassembly tools and says, quite correctly, that since access to source code is rare in the Windows world, the tools have considerably greater maturity than in the Linux world. The authors mention several of the tools, but do not discuss the structure of Windows programs, which really disappointed me since I wanted to learn more about Windows disassembly. Chapter 3 goes into much greater detail about the structure of Linux programs and how the C compiler works. My impression was that the authors assume that their audience already understands Windows programming in intimate detail, and they themselves are exploring how to disassemble Linux with the primitive tools available.

The authors do provide working code and scripts that help with disassembling Linux programs, and that is a real plus. These code examples can be found on one of the authors' Web site. But the real focus of the disassembly techniques does not appear to be exploring hostile code, but discovering how to bypass checks on serial numbers and other copy protection or access control schemes. That was not what I wanted, as I expect to see more hostile Linux code to appear in the future. I expect code that, like the viruses and worms familiar from its Windows'

counterparts, will not come with source code.

The remainder of the book provides a beginner to intermediate text on general computer-security topics, with some glaring errors. For example, on page 186, at the end of the TCP/IP handshake, the "command is received and resets the sequence number to zero." Huh? Haven't these guys spent any time with their eyes glued to sniffers?

The chapter on UNIX security touches on some interesting topics, but provides little useful advice. The suggestion that some accounts in the passwd file should simply be deleted appears seriously misadvised, and the authors completely miss the significance of ownership and permissions on system directories – pretty basic stuff.

This is not a bad book – I simply wish that it had been better tech edited and more focused on reverse engineering of hostile code. There are other books that cover incident response, honeypots, UNIX security, and other topics in much greater detail.

QUICKSILVER

NEAL STEPHENSON

San Francisco: Harper Collins, 2003. Pp. 926.
ISBN 0-380-97742-7.

Reviewed by Rik Farrow

While not a technical book, *Quicksilver* seems to me worth mentioning since it was written by Neal Stephenson, the keynote speaker at USENIX '03 and author of *Cryptonomicon*, a very popular book among the computing community. When I learned that Stephenson planned on venturing into historical fiction, I was at first disappointed. But *Quicksilver* did not disappoint me. It is a book to savor.

Quicksilver deals with the people and events at the end of the 16th century, primarily in England and Europe. Throughout the book, Stephenson brings to life characters such as Newton, Leibnitz, Hooke, and other famous natural philosophers in a way that makes the era in which they lived exciting and real. Stephenson examines not only the birth of science, but the political, religious, technological, and social structure on which it depended. Stephenson writes superb dialogue that reveals his characters' thoughts and feelings while educating the reader at the same time. I loved his exploration of the evolution of the financial world of banks and markets.

The book's pace is somewhat uneven in that there are slow sections – for example, exchanges of letters. But these are more than compensated by the brilliant episodes of action salted throughout the book. I found myself reading the book for its prose and dialogue, learning about this period of history, and then getting caught up in a chase scene that just could not be postponed. I highly recommend this book, both for its entertainment value and for what you can learn not only about history but also about how humans operate.

USENIX notes

USENIX MEMBER BENEFITS

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

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

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

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

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

DISCOUNTS on registration fees for all USENIX conferences.

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

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

FOR MORE INFORMATION REGARDING MEMBERSHIP OR BENEFITS, PLEASE SEE

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

OR CONTACT

office@usenix.org

Phone: 510 528 8649

An Open Letter from the USENIX Association Rebutting SCO's Position on Open Source Software

February 27, 2004

The SCO Group, Inc. (SCO), has recently sued IBM and Novell and launched broad attacks on the legality of and the economic justification for so-called open source licensing, including the free licensing of Linux. As an organization dedicated to advancing the skills and contributions of computer researchers and developers, the USENIX Association is compelled to address and refute the position SCO has taken regarding open source software.

Since 1975, USENIX has brought together the community of engineers, system administrators, scientists, and technicians working on the cutting edge of the computing world. USENIX was here before SCO. USENIX was here before Linux. USENIX and its members serve as an unparalleled demonstration that the best way to support advances in computer programming and to create better computer programs (and to help the American economy) is by sharing innovations, rather than keeping them secret or charging large amounts of money for access to them, as SCO advocates.

SCO argues that open source software, and in particular the General Public License (GPL), by means of which Linux and many other open source programs are licensed without charging fees, are "a threat to the U.S. information technology industry." SCO's own programmers themselves use open source computer software tools, so it is difficult to explain SCO's position except by noting its hypocrisy. Many of the most popular computer development tools are available to programmers worldwide for free through the contributions of the open source development community. If their developers were to charge substantial fees for their use or to withdraw them from distribution entirely, commercial programmers such as SCO and non-commercial programmers alike would be the worse for it.

SCO specifically argues that open source (free) licensing "undermines our basic system of intellectual property rights." This assertion lacks any legal justification and therefore appears to be merely self-serving. Nothing in our intellectual

property laws requires inventors to charge substantial fees for access or use of their inventions. In fact, the laws of copyright and patents, which underlie the intellectual property rights that most often protect computer software programs, give their owners complete discretion in deciding how large their licensing fees should be, or, indeed, whether to impose fees at all.

SCO specifically argues that open source software "has the potential to provide our nation's enemies or potential enemies with computing capabilities that are restricted by U.S. law." Intellectual property law is not the right place to impose restrictions on the use of computer programs abroad. That's what our export control laws do. This confusion between intellectual property licensing and export policy shows how bankrupt SCO's arguments are. Furthermore, the U.S. export control authorities have acknowledged the impossibility of restricting the geographical distribution of most computer software programs. In any event, neither area of law hinges on whether software programs are licensed for fees or for free, or whether the innovations are kept secret or are shared.

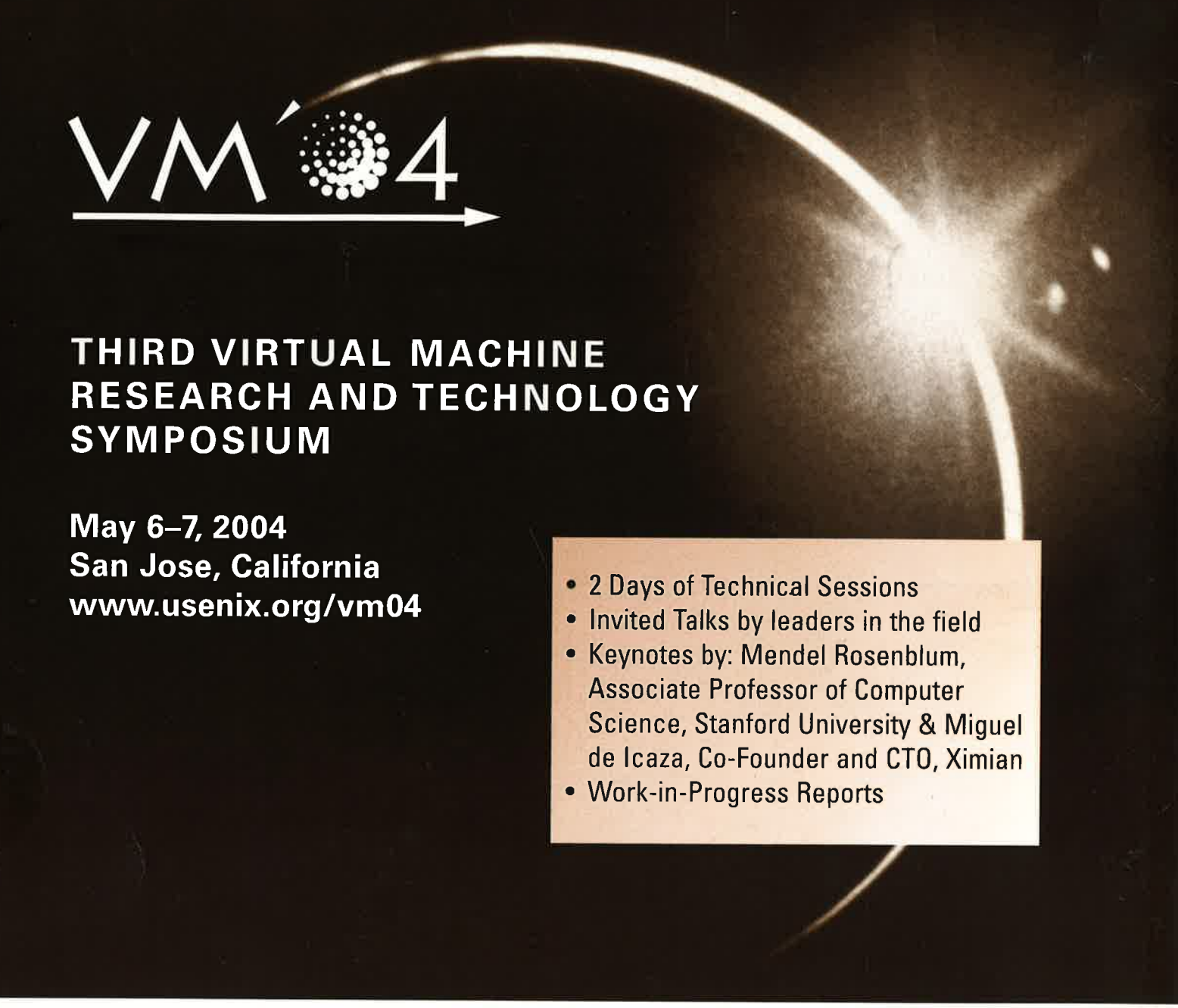
SCO specifically argues, "Each Open Source installation displaces or preempts a sale of proprietary, licensable and copyright-protected software." This would only be true if the open source applications were superior or at least equal to their proprietary counterparts. America has always asserted that the marketplace is the best regulator. Expensive products stimulate the introduction of less expensive and better substitutes. Intellectual property laws do not change that basic principle of capitalism. SCO's desire to be protected against competition is understandable, particularly if its products are inferior to those of its open source competitors. But it is unreasonable to expect that intellectual property laws will shield SCO from the normal operation of the marketplace.

Intellectual property law has always balanced the need to give inventors protection from competitors with the need to give society the benefit of their innovations and to let the marketplace regulate fees through the mechanisms of supply and demand. Intellectual property laws have never given inventors absolute protection against the competition of lower-cost substitutes. Copyright laws, for example, only protect against copying. If substitute programs are not copies, then they do not infringe, and they are free to compete with the original programs in the marketplace. Inventors who find they can't compete against lower-cost or free substitutes are compelled to find other things to sell. SCO's claims that open source developers are damaging our system of intellectual property rights and are threatening the viability of our technology industry are intellectually dishonest. Indeed, the open source community's practice of sharing innovations and of making them available for free clearly stimulates development and invigorates the technology sector. From the software that controls the majority of the world's Web servers to the software that makes tasks easier on your desktops, open source development has enhanced the American economy.

Society is better off when consumers have choices and when products compete with one another on the basis of functionality and price, and inventing is facilitated when inventors share their ideas. USENIX supports the right of programmers to choose whether to charge for their programs or to make them available for free, and we oppose any attempt to change the balance inherent in our intellectual property laws.

Sincerely,

Marshall Kirk McKusick, President
USENIX Board of Directors



THIRD VIRTUAL MACHINE RESEARCH AND TECHNOLOGY SYMPOSIUM

May 6–7, 2004
San Jose, California
www.usenix.org/vm04

- 2 Days of Technical Sessions
- Invited Talks by leaders in the field
- Keynotes by: Mendel Rosenblum, Associate Professor of Computer Science, Stanford University & Miguel de Icaza, Co-Founder and CTO, Ximian
- Work-in-Progress Reports

;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
RIDE ALONG ENCLOSED