LINUX JOURNAL

Advanced search

Linux Journal Issue #129/January 2005

Archive Index

Advanced search

Staying Current with Your Distribution's Security Updates

Jeremy Turner

Issue #129, January 2005

Keeping software up to date is the first lesson for beginning Linux administrators. Jeremy covers how to do it with the most popular update tools, click by click.

One of the key elements of making and keeping Linux a prime-time player in your desktop or server environment is ensuring that it is current with security patches. You take measures to address security at the network and hardware levels, but it takes only one security hole to compromise your entire environment. All users, whether they are commercial, nonprofit or home users, must know how to update their systems and applications, and they must do so regularly.

Two steps are key to keeping your system clean: knowing when to update and actually performing the updates. The first can be solved by monitoring security bulletin mailing lists for your specific distribution. The second can be solved in numerous ways through graphical and command-line tools. Some distributions also include auto-upgrading software utilities that can help you monitor your system.

I admit that I use the terms update and upgrade interchangeably when referring to moving from one version of a software package to another. These essentially mean the same thing. You also want to be careful when updating software so you do not install a version of a package you did not intend to. Development versions of packages usually carry a different version series. If the version differs by too much, check for a different update.

This article investigates both command-line and GUI tools for keeping your Linux system up to date. We specifically look at Debian 3.0 (Woody), Mandrake 10.0, SuSE 9.1 and Fedora Core 2.

Knowing When to Update

So how do you know when you should update? One good method is to subscribe to the security bulletins that your distribution provides. The on-line Resources provide URLs for the distributions covered in this article here and their respective security mailing lists. These usually are low-traffic mailing lists to alert you of security-related patches or updates. They also usually provide direct links for downloading the updated packages and MD5 sums to ensure you have a clean package. You manually can install a package this way. You also might need to grab any dependencies, if necessary.

Another method for knowing when to update is to use a script or application that polls for any updates. SuSE 9.1 and Fedora Core 2 include easy methods for automatically updating your current software with GUI tools. Debian and Mandrake also both have easy GUI tools and can be scripted to download packages in the middle of the night, letting you upgrade later.

I must offer a word of caution on upgrading software when no one is present to monitor the process. For instance, I heavily configure the Apache Web server. When I upgrade, it always asks me if I want to replace my config files. I usually run `diff` to see what the changes will do, but I rarely let them overwrite my config file. Make sure you note any changes in the software versions that are upgrading if you have any critical applications. Always back up your critical application config files.

RPM-Based Distributions

The RPM command-line tool is a manual and dependable method for upgrading your RPM security update. The rpm command has a lot of switches for various options, but for upgrading packages, you should run:

```
# rpm -Uv package.rpm
```

For the RPM file, you can specify a local filename, or even an FTP or HTTP location. If your security mailing list includes direct URLs for package updates, command-line updating is very simple. For more information on the rpm command-line tool, check out the RPM Web site or the man page.

Debian-Based Distributions

Debian and other Debian-based distributions use dpkg as their package management system. It used to stand for Debian GNU/Linux package manager. The dpkg FAQ page states that it no longer stands for anything, because it is used in non-Debian and non-Linux environments. This package manager does the mid-level work for APT, the Advanced Packaging Tool, and GUI tools such as

Synaptic. Much like RPM, dpkg includes a plethora of command-line switches, but we focus on the simple upgrade switch:

```
# dpkg -i package.deb
```

The -i switch instructs dpkg to install the package. If a prior version of the package exists, dpkg removes the prior version and installs the newer version. Unlike rpm, dpkg requires wget or curl to download the package before installing.

Debian 3.0 (Woody)

Advanced Package Tool (APT) is where you probably will do most of your command-line package management in Debian. APT uses a list of repositories with available packages. If there is a newer package version in the repository's Package list, APT downloads the package and hands the process over to dpkg. First, make sure you have the security update source in your sources.conf file. It should read:

```
deb http://security.debian.org/ stable/updates main
```

Instead of the word stable, you might have woody instead, but either will do. After editing the sources.conf file, you also need to update your available package list. To update and then upgrade them, run the apt-get two-step:

```
# apt-get update
# apt-get upgrade
```

This upgrades only packages that do not require modifications to other packages. To upgrade packages that do require some sort of dependencies, run:

```
# apt-get update
# apt-get -u dist-upgrade
```

The -u switch shows exactly which packages will be upgraded, newly installed or removed. You can set these lines to run from the crontab and have your machine download, but not install, the latest packages you need. A command to put in your crontab file might look like:

```
(apt-get update && apt-get -dy upgrade) \
| mail -s "`hostname` update" root
```

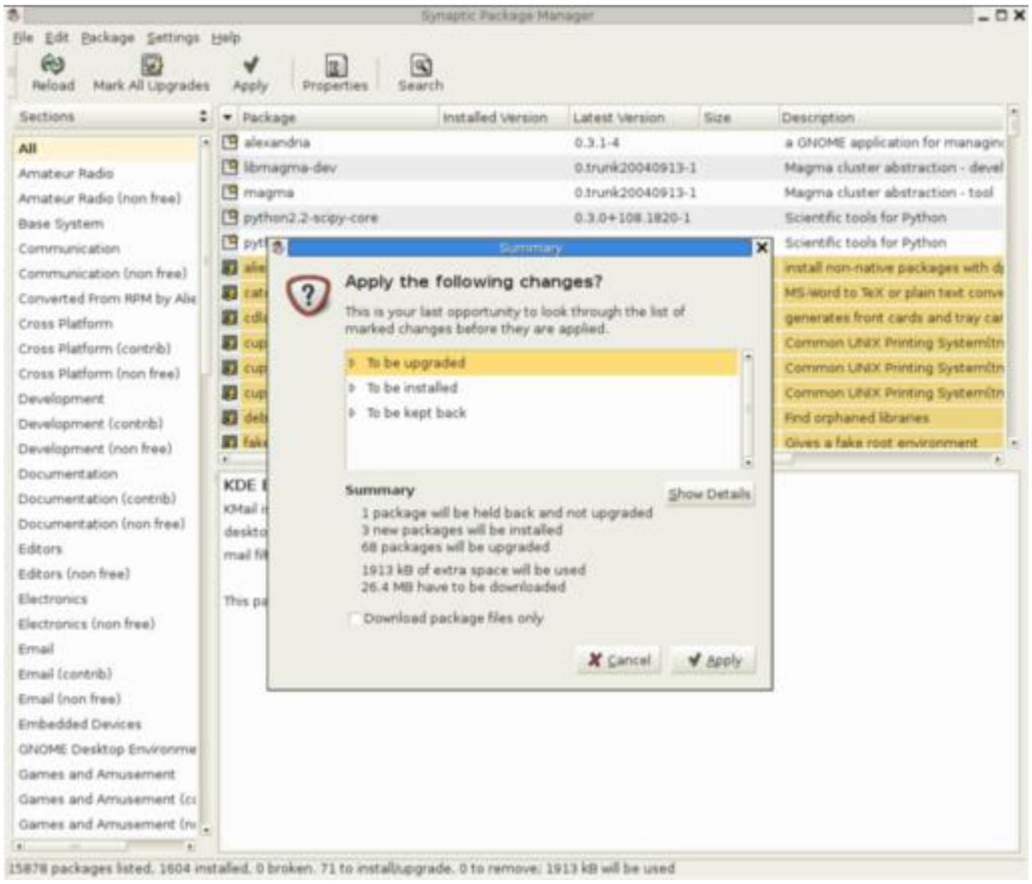
This command downloads the list of the latest packages and, if successful, downloads the packages that need to be updated. It sends the results by e-mail to the root user. Substitute your user name or e-mail address as necessary. When you receive e-mail notifying you that there are updates, you can run:

```
# apt-get upgrade
```

This installs the previously downloaded packages allowing you to be present at the console or terminal. Some package upgrades require additional user input, so it may not be wise to run a completely automated upgrade solution.

Available on the GUI side for Debian users, Synaptic is a complete front end to dpkg. To run Synaptic, go to the Debian menu in your desktop environment and select Apps→System→Synaptic Package Manager. Synaptic works much the same as APT. To update your list of available packages, click the Reload button at the top left of the window. A window list of mirror locations updates you on the status of the package list download. When Synaptic finishes downloading the package lists, you can view all available upgrades. Packages that need to be upgraded have a green box and an arrow pointing up. Newly available packages have a yellow star on the box. Installed packages have a green box, and not installed packages have a white box.

To download and install all package updates, click the Apply button. You then are prompted with a window detailing which packages will be upgraded, installed, kept back or removed (Figure 1). Kept back means that the package would require other dependencies that were not stated specifically. Clicking Apply begins downloading the updates. Following the download process, the updates will install in a terminal-like text box, allowing you to answer questions if needed. When finished, click the Close button (Figure 2).



Package	Installed Version	Latest Version	Size	Description
alexandria		0.3.1-4		a GNOME application for managin
libmagma-dev		0.trunk20040913-1		Magma cluster abstraction - devel
magma		0.trunk20040913-1		Magma cluster abstraction - tool
python2.2-scipy-core		0.3.0+108.1820-1		Scientific tools for Python

Apply the following changes?

This is your last opportunity to look through the list of marked changes before they are applied.

- To be upgraded
- To be installed
- To be kept back

Summary

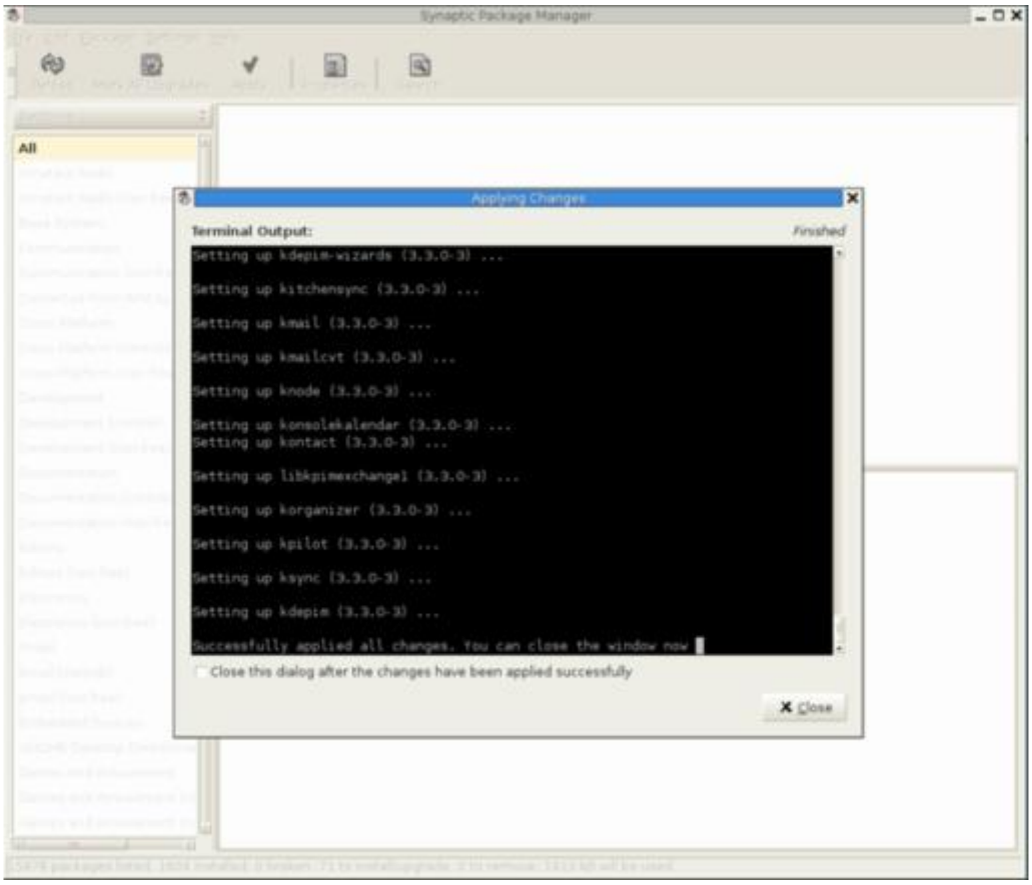
1 package will be held back and not upgraded
3 new packages will be installed
68 packages will be upgraded

1913 kB of extra space will be used
26.4 MB have to be downloaded

Download package files only

15878 packages listed, 1604 installed, 0 broken. 71 to install/upgrade, 0 to remove; 1913 kB will be used

Figure 1. Synaptic Showing the Applications to Be Modified



Terminal Output:

```
Setting up kdepim-wizards (3.3.0-3) ...

Setting up kitchensync (3.3.0-3) ...

Setting up kmail (3.3.0-3) ...

Setting up kmailcvt (3.3.0-3) ...

Setting up knode (3.3.0-3) ...

Setting up konsolekalendar (3.3.0-3) ...
Setting up kontact (3.3.0-3) ...

Setting up libkpimexchange1 (3.3.0-3) ...

Setting up korganizer (3.3.0-3) ...

Setting up kpilot (3.3.0-3) ...

Setting up ksync (3.3.0-3) ...

Setting up kdepim (3.3.0-3) ...

Successfully applied all changes. You can close the window now
```

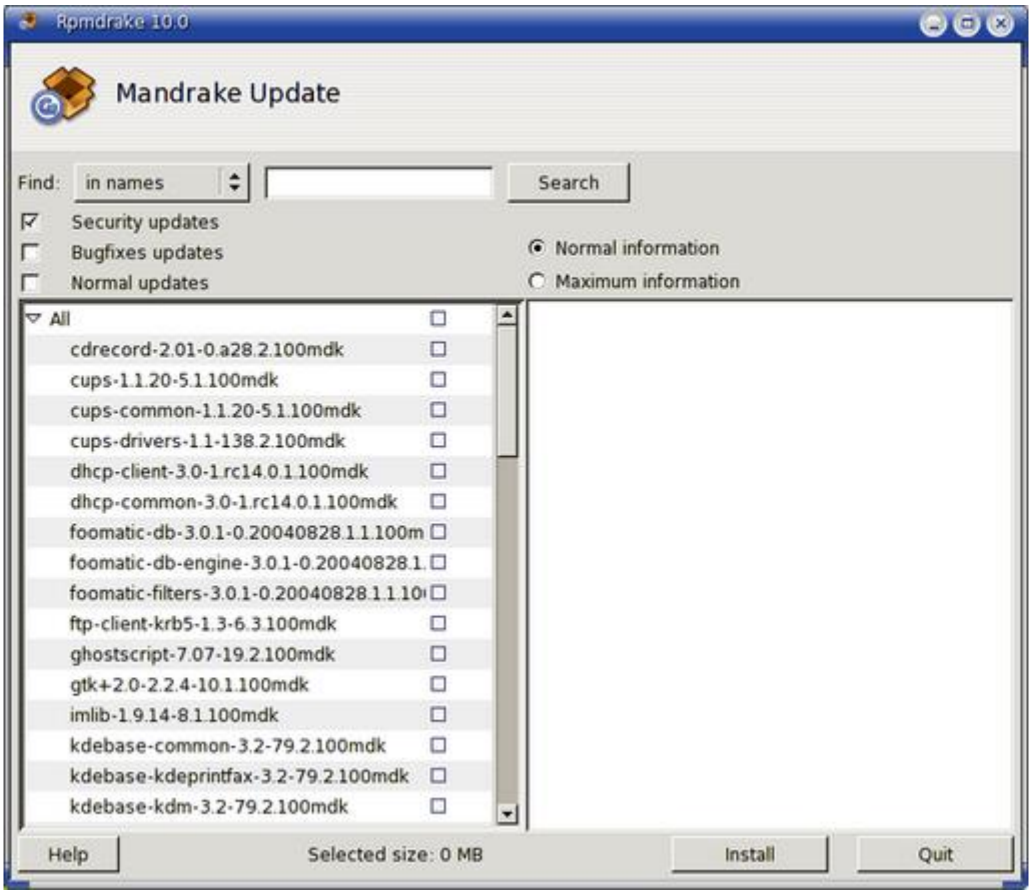
Close this dialog after the changes have been applied successfully

Figure 2. Synaptic after All Upgrades Have Been Performed

Mandrake 10.0

When installing Mandrake 10.0, one of the final steps before the first login is to check for any critical updates. If you are installing this distribution from scratch, this would be a great step. However, what do you do now that Mandrake is installed, and you need a patch for a security hole?

Mandrake 10.0 users have a nice GUI package management application called rpmdrake. You can find it by clicking on the KDE star menu and selecting System→Configuration→Packaging→Mandrake Update. You also can run `rpmdrake` as root on the command line. Answer a couple of questions, and then you are presented with a list of packages that need updating due to security updates (Figure 3). To update all of them, click in the box on the All line, then press the Install button, and grab your favorite beverage!



[✓] Security updates
[ ] Bugfixes updates
[ ] Normal updates

[✓ Normal information]
Maximum information

All
cdrecord-2.01-0.a28.2.100mdk
cups-1.1.20-5.1.100mdk
cups-common-1.1.20-5.1.100mdk
cups-drivers-1.1-138.2.100mdk
dhcp-client-3.0-1.rc14.0.1.100mdk
dhcp-common-3.0-1.rc14.0.1.100mdk
foomatic-db-3.0.1-0.20040828.1.1.100m
foomatic-db-engine-3.0.1-0.20040828.1.
foomatic-filters-3.0.1-0.20040828.1.1.10
ftp-client-krb5-1.3-6.3.100mdk
ghostscript-7.07-19.2.100mdk
gtk+2.0-2.2.4-10.1.100mdk
imlib-1.9.14-8.1.100mdk
kdebase-common-3.2-79.2.100mdk
kdebase-kdeprintfax-3.2-79.2.100mdk
kdebase-kdm-3.2-79.2.100mdk

Selected size: 0 MB

Figure 3. rpmdrake's List of Available Package Upgrades

After downloading and installing all updates, you are presented with a dialog box letting you know everything has been installed. It's that easy.

The command-line urpmi package was installed with my stock installation of Mandrake 10.0. urpmi acts much like APT, allowing you to use multiple sources to update packages. These repositories can be accessed by CD-ROM, a local

RPM directory or an FTP or HTTP Internet source. For our purpose of installing security updates, we want to run something like the following command:

```
# urpmi.addmedia --update updates \
ftp://example.com/Mandrake10.0/RPMS \
with ../base/hdlist.cz
```

This adds security updates from an FTP mirror to your list of sources. You need to substitute the ftp:// URL with a real mirror. The Web site Easy urpmi gives you a nice Web interface to choose your nearest mirror, your architecture and from which source pools you'd like to download updates.

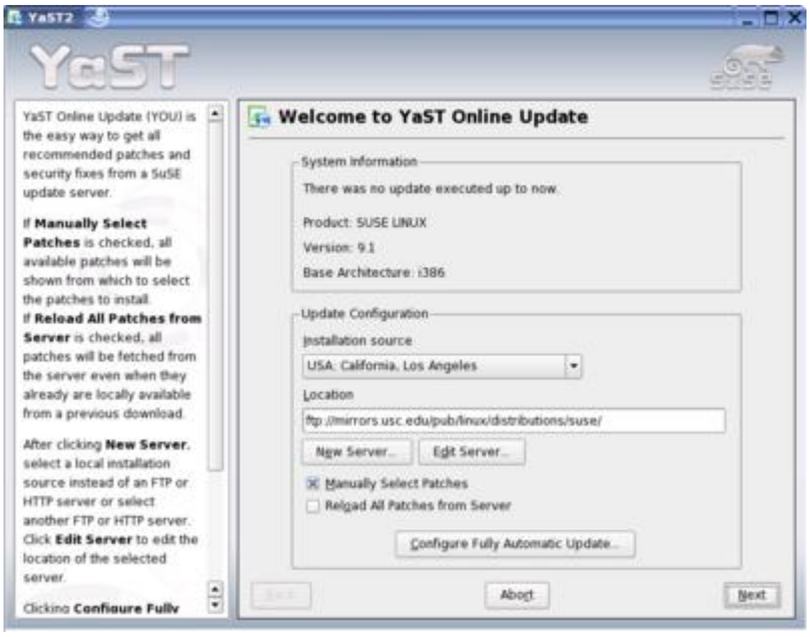
To update the list of available packages and then install all package updates, do the urpmi two-step:

```
# urpmi.update -a
# urpmi --auto-select
```

You then are prompted to install the updated packages and any dependencies, if necessary.

SuSE 9.1

SuSE 9.1 has a similar method for installing updates by using the YaST2 Online Update (YOU) GUI tool. You can find this by clicking on the SuSE icon, then System→YaST. After entering your root password, click on Software and then Online Update. You can choose your installation source or add a new server manually (Figure 4). Additionally, you can configure YOU to download and/or install updates automatically at a specified time each day. Clicking Next downloads information that tells you what packages need to be updated. After this list is updated, we are presented with the list of packages, a patch description and disk usage (Figure 5). In the list of patches, red lines denote security updates, blue lines denote recommended updates and black lines are optional updates. To perform the upgrade, click Accept. After the updates are completed, click Finish, which configures a few system services. In addition to the YOU system, you can use the rpm command from the command line.


YaST Online Update (YOU) is the easy way to get all recommended patches and security fixes from a SuSE update server.

If **Manually Select Patches** is checked, all available patches will be shown from which to select the patches to install.
If **Reload All Patches from Server** is checked, all patches will be fetched from the server even when they already are locally available from a previous download.

After clicking **New Server**, select a local installation source instead of an FTP or HTTP server or select another FTP or HTTP server. Click **Edit Server** to edit the location of the selected server.

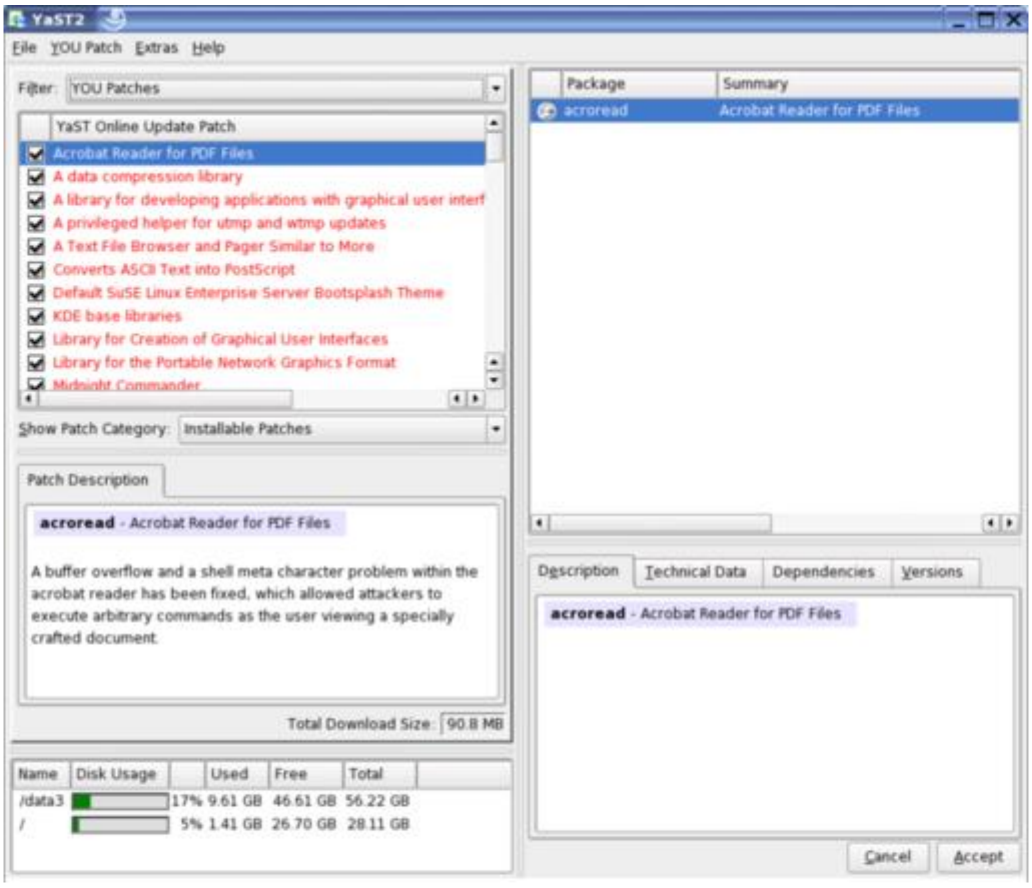
Clicking **Configure Fully**

Welcome to YaST Online Update

System Information
There was no update executed up to now.
Product: SUSE LINUX
Version: 9.1
Base Architecture: i386

Update Configuration
Installation source: USA: California, Los Angeles
Location: ftp://mirrors.usc.edu/pub/linux/distributions/suse/
[✓ Manually Select Patches]
Reload All Patches from Server

Figure 4. YaST2 Online Update's Mirror Selection Process


Filter: YOU Patches

YaST Online Update Patch
✓ Acrobat Reader for PDF Files
✓ A data compression library
✓ A library for developing applications with graphical user interf
✓ A privileged helper for utmp and wtmp updates
✓ A Text File Browser and Pager Similar to More
✓ Converts ASCII Text into PostScript
✓ Default SuSE Linux Enterprise Server Bootsplash Theme
✓ KDE base libraries
✓ Library for Creation of Graphical User Interfaces
✓ Library for the Portable Network Graphics Format
✓ Midnight Commander

Show Patch Category: Installable Patches

Patch Description
acroread - Acrobat Reader for PDF Files

A buffer overflow and a shell meta character problem within the acrobat reader has been fixed, which allowed attackers to execute arbitrary commands as the user viewing a specially crafted document.

Total Download Size: 90.8 MB

Name	Disk Usage	Used	Free	Total
/data3	17%	9.61 GB	46.61 GB	56.22 GB
/	5%	1.41 GB	26.70 GB	28.11 GB

Package	Summary
acroread	Acrobat Reader for PDF Files

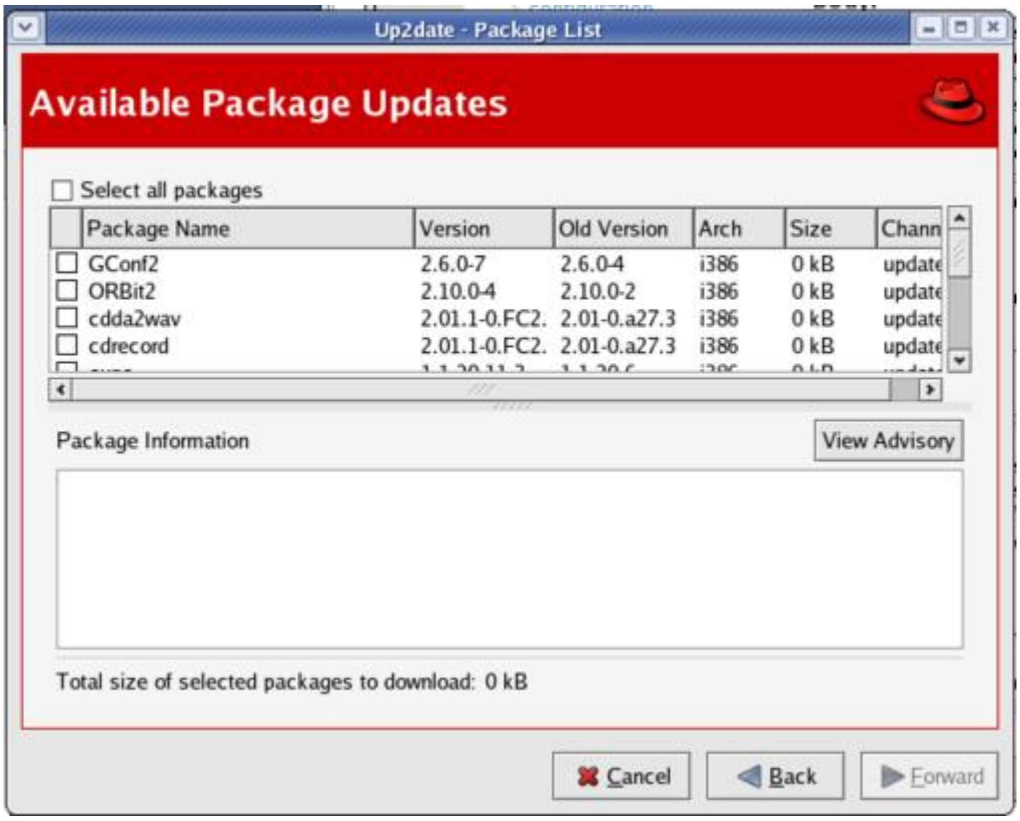
acroread - Acrobat Reader for PDF Files

Figure 5. YaST2 Online Update's List of Available Package Upgrades

Fedora Core 2

The Red Hat Update Agent, up2date, has been around for several Red Hat versions and is present in Fedora Core 2. To check for new software updates in Fedora Core 2, right-click on the red exclamation point in the system tray and choose Check for updates. To download and install the latest updates, right-click on the red exclamation point and choose Launch up2date. You can choose the defaults. The first time you run up2date, you are asked whether you want to install the Red Hat GPG key signature. I chose yes on my system.

In the Channels menu, you can subscribe to two channels or repositories where updates are kept, fedora-core-2 and updates-released-fc2. Channels in up2date are similar to the repository in APT or urpmi. You are asked to note any packages you want to skip. The package already listed for me was a kernel upgrade. Clicking Forward gives you a list of available software updates (Figure 6). To select all updates, click the check box next to Select all packages.

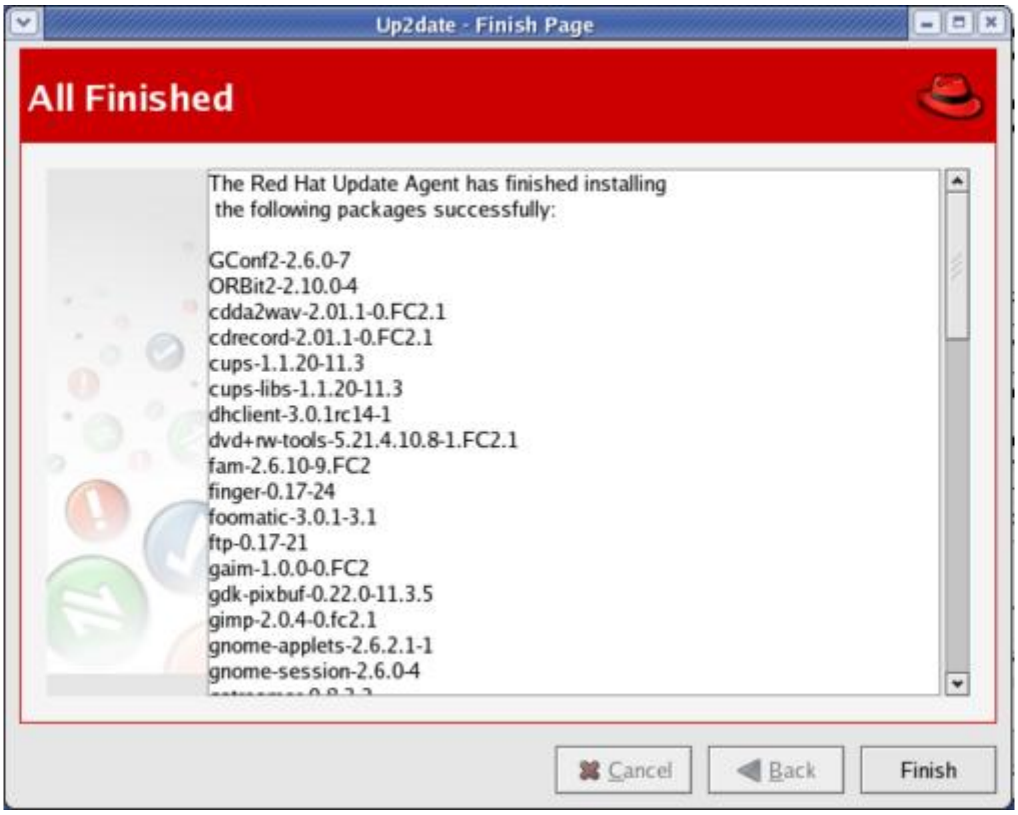


	Package Name	Version	Old Version	Arch	Size	Chann
☐	GConf2	2.6.0-7	2.6.0-4	i386	0 kB	update
☐	ORBit2	2.10.0-4	2.10.0-2	i386	0 kB	update
☐	cdda2wav	2.01.1-0.FC2.	2.01-0.a27.3	i386	0 kB	update
☐	cdrecord	2.01.1-0.FC2.	2.01-0.a27.3	i386	0 kB	update

Total size of selected packages to download: 0 kB

Figure 6. Up2date's List of Available Package Updates

Clicking Forward starts retrieving packages. Again, a break with your favorite beverage will do nicely at this point. When the download process is finished, click Forward to start the installation process. When the installation process is finished, you are given a nice summary of exactly what packages were installed and their versions (Figure 7).



```
The Red Hat Update Agent has finished installing
 the following packages successfully:

GConf2-2.6.0-7
ORBit2-2.10.0-4
cdda2wav-2.01.1-0.FC2.1
cdrecord-2.01.1-0.FC2.1
cups-1.1.20-11.3
cups-libs-1.1.20-11.3
dhclient-3.0.1rc14-1
dvd+rw-tools-5.21.4.10.8-1.FC2.1
fam-2.6.10-9.FC2
finger-0.17-24
foomatic-3.0.1-3.1
ftp-0.17-21
gaim-1.0.0-0.FC2
gdk-pixbuf-0.22.0-11.3.5
gimp-2.0.4-0.fc2.1
gnome-applets-2.6.2.1-1
gnome-session-2.6.0-4
```

Figure 7. Up2date All Finished Downloading and Installing

Fedora Core 2 also is based on the RPM system, which allows you to use the rpm command at a terminal.

Another package management front end that has received notoriety is the Yellow dog Updater Modified, or Yum. Yum is much like APT, but it has several differences that the author explains on the Yum Web site. In essence, Yum acts like urpmi or APT in dealing with package repositories, and then it hands the actual package installation off to RPM. The anaconda installer uses Python bindings for RPM access, so you can count on the Python support staying around.

Conclusion

There's a saying in baseball: “You're only as good as your last at-bat.” The computer application of this principle is that your system is only as secure as your last update. A fancy network firewall and a magnetic-stripe server-room door key are good security steps, but running an outdated version of Apache or OpenSSH can bring your systems to a halt if you don't keep your Linux systems up to date.

Resources for this article: /article/7862.

Jeremy Turner has been a Linux user for more than five years and has a passion for helping users learn open-source software. He hacks PHP, sings first tenor, watches too much baseball and checks his e-mail regularly (jeremy@linuxwebguy.com).

Archive Index Issue Table of Contents

Advanced search

Point-and-Click E-Mail Crypto

Roy Hoobler

Issue #129, January 2005

Make secure e-mail a habit with easy GUI tools to send, receive and authenticate encrypted messages.

I use a laptop with Linux, and I don't want people reading my mail if the laptop falls into the wrong hands. I've also had my e-mail monitored, and I didn't want the network administrator to view anything personal. GnuPG offers good encryption and is available for all. With KDE's KGPG and KMail, things are even easier. This article explains how to use KGPG for e-mail and file encryption. It may get a little complicated, but by following this article, you should have everything up and running within an hour or so. If you have any questions, please write me—try out your new secure e-mail if you like. My address is in this article.

What Is GnuPG?

Gnu Privacy Guard (GnuPG) is an implementation of the OpenPGP standard. These standards grew out of the work done by Philip Zimmerman and his PGP (Pretty Good Privacy) software. PGP has been around since 1991 and is now proprietary software. However, OpenPGP standards were established in 1997, and version 1.0 of GnuPG appeared in 1999.

GnuPG is all done through the command line and is quite complex. Tools to simplify it for you are available; this article covers KDE, KGPG and KMail.

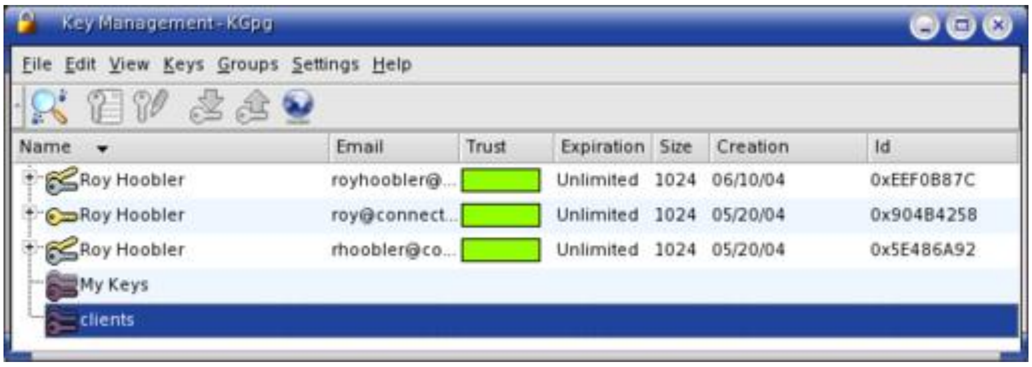
GnuPG and PGP are compatible. For those already using PGP, if you use the IDEA algorithm, there is some more work involved with switching to GnuPG; otherwise, there shouldn't be a problem. If you need to communicate with or replace PGP 2.x, see the on-line Resources for this article.

It takes discipline to implement and enforce privacy and security policies for an organization. I was in a Military Intelligence unit back in my Army days, and security was taken very seriously. If policies were not followed, there were serious consequences. For any organization, someone should be appointed as a security manager and given the proper authority to make sure guidelines are followed. Just like any good practice (such as using CVS for code), once people are trained to encrypt sensitive information, it will become standard. Depending on the size of the organization, it could take anywhere from a week to a month of checking and training to get the policy in place.

Creating Your Own Key

As an example, I'm going to encrypt a message and send it from my work account to my rhoobler@comcast.net mail account. Both accounts are using KMail. Next, I send an encrypted reply back to my work account. First, however, I set up keys for the comcast.net account, so I have somewhere to send it.

Once you install KGPG, you should have an icon on the system tray. If not, you can launch it from a terminal by typing `KGPG -k`. The -k option is important to bring up the user interface—the Key Management tool (Figure 1). Without the -k option, KGPG runs as a service in the background, and the system tray icon appears. Clicking on the system tray icon brings up the the user interface.



Name	Email	Trust	Expiration	Size	Creation	Id
Roy Hoobler	royhoobler@...	🟩	Unlimited	1024	06/10/04	0xEEF0B87C
Roy Hoobler	roy@connect...	🟩	Unlimited	1024	05/20/04	0x904B4258
Roy Hoobler	rhoobler@co...	🟩	Unlimited	1024	05/20/04	0x5E486A92
My Keys						
clients						

Figure 1. Use the Key Management tool to browse GnuPG keys by name or e-mail address.

First, I make a private and public key pair for my comcast.net e-mail account. Inside the Key Management program, select the Keys→Generate Key Pair menu item, which brings up a rather simple dialog box. Selecting Expert Mode launches GnuPG in a terminal. For now, enter your name, e-mail address and a comment using the dialog window. Depending on your security policies, you can set when keys expire as well.

The next step is entering a passphrase for GnuPG. This is very important. If you forget this, you won't be able to read any messages, and KGPG prompts you for a passphrase whenever you want to read anything. Then wait a few seconds for GnuPG to make your keys for you. As a safety measure, I also suggest making a revocation certificate (name the file something like rhooblerrev.asc). If your system is ever stolen or compromised, you can send out this certificate to let people know your public key is void.

That's it. Now we have a public and private key for rhoobler@comcast.net. Two things are left: exporting the public key and optionally exporting the private key. These keys are exported separately; for the public key, I name the file rhoobler.asc and for the private key, rhooblerprivate.asc. Beware: if someone gets hold of your private key and guesses your passphrase, the attacker can read encrypted mail for you and cryptographically sign mail as you.

After exporting the private key and the revocation keys, burn them to a CD-ROM and put it in a safe or a safe-deposit box, then delete the revocation (rhooblerrev.asc) and private key (rhooblerprivate.asc) files from your hard drive.

GnuPG keeps the following files in each user's .gnupg directory. All are read/writable only by the user:

- gpg.conf: general GPG configuration.
- pubring.gpg: list of public keys.
- secring.gpg: list of secure (private keys).
- trustdb.gpg: database file that records who trusts whom.

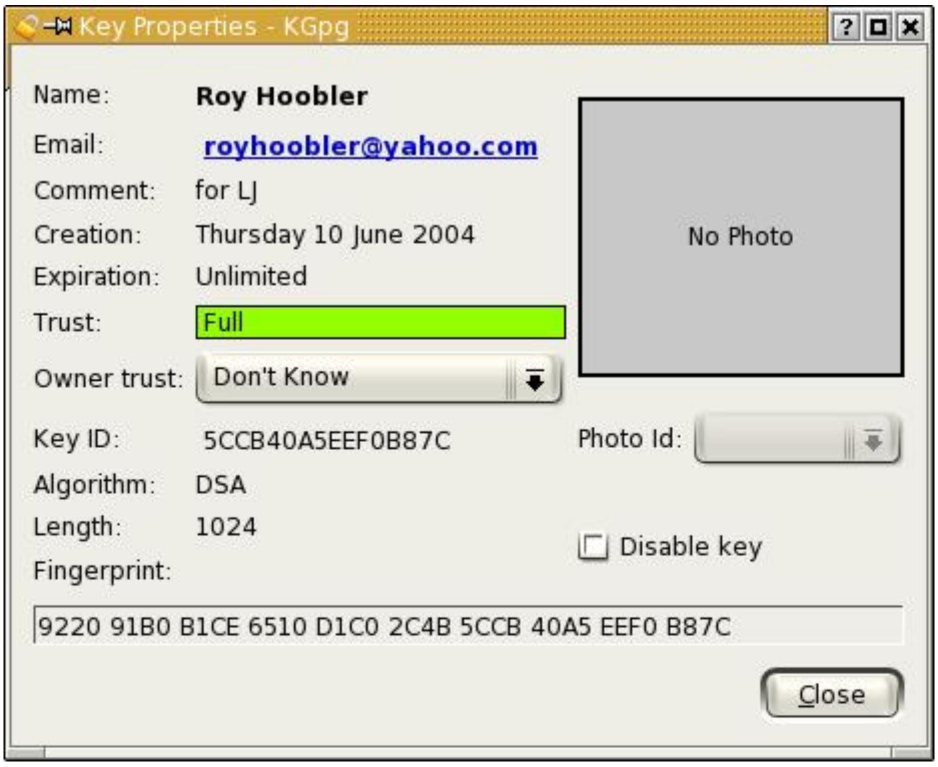
For convenience, now I export the public key for rhoobler@comcast.net to a default keyserver. Using the Key Management tool, select the key and right-click, and then choose Export Public Key(s). Another simple dialog box comes up with three options. I choose Default keyserver and then OK. The keyserver can be configured in the Settings menu, but by default it is subkeys.pgp.net, which always has worked for me. You also can export it to a file then e-mail it, or upload the key to your Web site. There is nothing wrong with everyone knowing your public key, but they should verify that it is yours (more about this later). Having your public key means they can encrypt files that only you will be able to open.

So, now we can encrypt files and send encrypted e-mails. But it works only if you have someone to share this information with. For this article, I went to my other workstation and set up another set of keys for my business e-mail account, using the same set of steps.

The next step is to import the rhoobler@comcast.net public key by using the Key Manager from the keyserver and selecting the globe icon or File→Key Server Dialog from the menu. You can type in the e-mail address and import the key. Before fully using the key, select it from the main dialog window then select Keys→Sign Keys from the menu.

If I wanted to set up a large group, I could create my own keyserver, sign the keys and then distribute them. Another approach is for people to e-mail their keys to others, then meet in a physical location and have them verify and sign each other's keys. This builds the web of trust. For example, I've signed Bill's key and Bill has signed Kate's key. If I get mail from Kate, her key can be added to my trust database.

Keys have a fingerprint, and if I am not sure whether a key is authentic, I can look at the fingerprint and call the person to verify it. The fingerprint can be found in the Key Manager by selecting the key and then selecting Edit Key from the menu (Figure 2).


Key Properties - KGpg

Name: Roy Hoobler
Email: royhoobler@yahoo.com
Comment: for LJ
Creation: Thursday 10 June 2004
Expiration: Unlimited
Trust: Full
Owner trust: Don't Know
Key ID: 5CCB40A5EEF0B87C
Algorithm: DSA
Length: 1024
Fingerprint:
9220 91B0 B1CE 6510 D1C0 2C4B 5CCB 40A5 EEF0 B87C

No Photo
Photo Id:
Disable key
Close

Figure 2. To check the authenticity of mail, simply call the sender and check the key fingerprint.

Encrypting Files

KGPG is integrated nicely into KDE and KDE applications. The most useful is the Konqueror browser. Once KGPG is installed, you can right-click on a document, and under the Actions menu, create an encrypted version of the document. One of the options is to shred the original, which makes a lot of sense if keeping the unencrypted version is a problem. When encrypting files, you can add multiple keys for different people who can read the document. If you shred the original file, make sure you include your own key whenever you encrypt it. If I use only my rhoobler@comcast.net key, I am the only person that can decrypt the file.

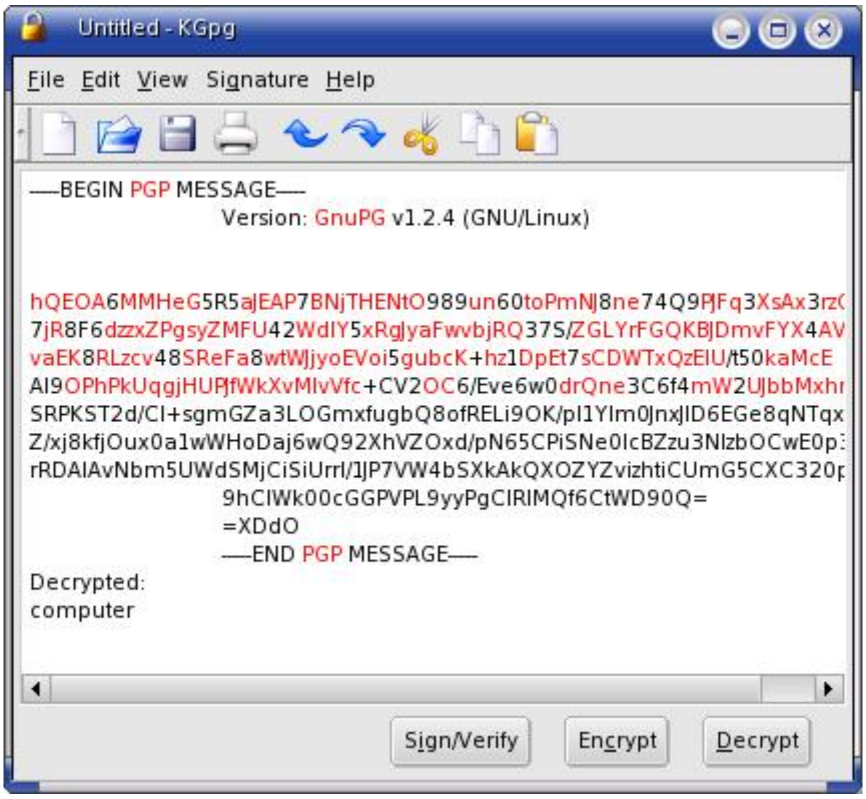
Sending E-Mail

Finally, we are ready to send an encrypted e-mail. Using KMail (or Kontact), type a message—I'll use rhoobler@comcast.net as the To: address. Select the Lock icon (or Options→Encrypt from the menu). When you click Send, a dialog comes up. If you do not see the recipient's key, press the refresh button. Also, if you didn't sign the key, it won't show up; go back and sign it with the Key Management tool and then press the refresh button. Finish typing the message and click Send.

With KMail, decrypting messages is built in. When you receive an encrypted message, you are asked for your passphrase and the message opens. If you are sending an encrypted message, if the e-mail address is in your keyring, it is encrypted and sent automatically. You also can send the message encrypted to several people at once, as long as you have their public keys.

Another method is to encrypt the file with KGPG and send the encrypted file as an attachment. KMail automatically decrypts the attachment for viewing (select view not open). For Web-based e-mail clients, you can download the file and decrypt or view it with Konqueror.

If you are using a Web-based e-mail client such as Yahoo mail, you can cut and paste the encrypted messages from the clipboard to the KGPG editor by right-clicking on the Tasks Tray icon, and then select decrypt clipboard. The same holds true for encrypting messages.



```
—–BEGIN PGP MESSAGE—–
                    Version: GnuPG v1.2.4 (GNU/Linux)


hQEOA6MMHeG5R5aJEAP7BNjTHENtO989un60toPmNJ8ne74Q9PJFq3XsAx3rz
7jR8F6dzzxZPgsyZMFU42WdIY5xRgJyaFwvbjRQ37S/ZGLYrFGQKBJDmvFYX4AV
vaEK8RLzcv48SReFa8wtWJjyoEVoi5gubcK+hz1DpEt7sCDWTxQzEIU/t50kaMcE
AI9OPhPkUqgjHUPJfWkXvMIvVfc+CV2OC6/Eve6w0drQne3C6f4mW2UJbbMxh
SRPKST2d/CI+sgmGZa3LOGmxfugbQ8ofRELi9OK/pl1YIm0JnxJID6EGe8qNTqx
Z/xj8kfjOux0a1wWHoDaj6wQ92XhVZOxd/pN65CPiSNe0IcBZzu3NlzbOCwE0p
rRDAIAvNbm5UWdSMjCiSiUrrl/1JP7VW4bSXkAkQXOZYZvizhtiCUmG5CXC320
                    9hCIWk00cGGPVPL9yyPgCIRIMQf6CtWD90Q=
                    =XDdO
                    —–END PGP MESSAGE—–
Decrypted:
computer
```

Figure 3. Decrypt Clipboard

Signing E-Mail

More popular than encryption is signing e-mails. Of course, this doesn't encrypt the text, but signing a message proves that it is from you. If I sign all my e-mails, because it is policy, and you get an e-mail from me that is not signed (or the signature doesn't match), you can assume it is a fake and alert whoever needs to be notified.

With KMail, the e-mail message is color-coded to let you know if it is a signed message and if the message came from a trusted source—yellow means signed, and green means signed and trusted.

Creating Groups/Other Options

Another handy tool in KGPG is the ability to create groups of keys. I could have an Administrative group that contains three or four keys. When sending a message, I can select that group and send it out. Later, if a recipient forwards the message to another person in the group, it already will be ready to read.

One other thing, under Configure KGPG, use the ASCII Armor option. It should be on by default. This makes signatures and encryption in plain text, so it is easy to mail, print and cut and paste. Without ASCII Armor, some files will be binary and may cause problems.

Summary

Time permitting, I'll try to decrypt and answer any encrypted messages that may come in. Because KGPG is included with KDE, it is included with most Linux distributions. Setting up a few keys and testing it yourself is only an hour or two worth of work.

With GnuPG and KGPG, using keys and encryption is a viable solution if you need to tighten up security. In my career, a lot of attention has been given to security for connections and transactions over SSL, but little attention has been given to files and e-mail. With KDE, and some effort, having secure e-mail is easy to set up. One idea to start with is encrypting any e-mail you send to your manager or the owner of the company. Another idea is to set up a private folder on the network that stores only encrypted documents. Following these types of security policies makes encryption easier to implement.

Resources for this article: /article/7863.

Roy Hoobler is owner of Connect Computing, Inc. (www.connectcomputing.com). As an independent consultant with ten years of proprietary software experience, his firm now focuses on helping small businesses to use Linux, as well as implement open-source business applications.

Archive Index Issue Table of Contents

Advanced search

Networking in NSA Security-Enhanced Linux

James Morris

Issue #129, January 2005

Break through the complexity of SELinux with a working example that shows how to add SELinux protection to a simple network server.

In this article, we take a look at how SELinux can help increase the security of networked systems, as well as the design and implementation of its network-specific security controls. We then walk through an example of using SELinux policy to lock down a simple network application.

Overview: SELinux Roles, Types and Domains

SELinux provides strong general security for networked systems. It allows systems to be locked down tightly so that services have only the minimum set of rights required to operate. This implementation of the principle of least privilege helps contain security breaches arising from buggy code, malicious code, user error and malicious users.

For example, an externally facing Web server normally might be hardened in a variety of ways, including:

- Disabling unnecessary services.
- Running server software in chroot jails.
- Local packet filtering with iptables.
- Privilege management with sudo.
- Locking down configuration files.

This is a good, multilayered approach to security, implementing the principle of defense in depth.

SELinux adds another security layer, mandatory access control (MAC). Standard SELinux implements MAC via type enforcement (TE) combined with role-based access control (RBAC), under the control of a centrally managed security policy, enforced by the kernel. Unlike traditional UNIX security, normal users do not have any control over SELinux security policy (hence the term mandatory), while the superuser, root, can be split into isolated administrative roles for authorized users, called separation of duty.

Traditional discretionary access control (DAC) is further restricted by the TE model, which assigns types to operating system objects such as processes, files and network resources, then defines rules for interactions between them. (The type of a process usually is referred to as a domain.) This allows for fine-grained access control, extending the principle of least privilege well beyond the scope of typical OS hardening.

SELinux Network Access-Control Architecture

SELinux is built upon the LSM (Linux Security Modules) and Netfilter APIs in the 2.6 kernel. LSM and Netfilter are both access-control frameworks consisting of strategically located hook points within the kernel. Kernel flow is redirected from these hooks to security modules such as SELinux, which perform access-control calculations and return a verdict to the hook. A hook uses the verdict returned from the security module either to allow normal kernel flow to continue or prevent it.

One of the core design principles of SELinux is that it mediates access at the OS object level. Rather than a naive approach where a security monitor decides whether a program can execute a particular system call with certain arguments, SELinux looks at the full security context of the program during execution, the security label attached to the object being accessed and the action being taken. For example, `ls` run by the system administrator is different from `ls` run by a normal user.

The general form of an SELinux permission is:

```
action (source context)
(target context):(target object classes)
permissions
```

Here's an example from SELinux policy:

```
allow bluetooth_t self:socket listen;
```

This provides the bluetooth_t domain with the listen permission for sockets labeled with its own security context. So, a process running in the bluetooth_t domain is allowed to invoke listen() on a socket that it owns.

The self designator is simply a shorthand way of making the target context the same as the source context. This commonly is used in policy relating to sockets as they typically are labeled with the same security context as the creating process.

Network Object Labeling

Under SELinux, objects are labeled with a security context of the following form:

```
user:role:type
```

For example:

```
root:staff_r:staff_t
```

is the context of a process being run by root via the staff_r role in the staff_t domain. The label associated with port 80 is:

```
system_u:object_r:http_port_t
```

The system_u user and object_r role are default values for system objects. It wouldn't make any sense for a port to have a real user or role; it's not owned by anyone and it doesn't initiate any actions that require a verdict from SELinux.

Sockets

A socket is labeled by its associated inode and is categorized as either a generic socket or one of the following socket subclasses:

- UNIX stream.
- UNIX datagram.
- TCP.
- UDP.
- Raw (includes ICMP and other non-TCP/UDP).
- Netlink families.
- Packet.
- Key (pfkeyv2).

Subclasses of sockets can be distinguished in security policy, providing for fine-grained control and flexibility over different network protocols:

```
allow lpd_t printer_port_t:tcp_socket name_bind;
```

This rule allows only a TCP socket created in the lpd_t domain to bind to a port of type printer_port_t.

Ports

IPv4 and IPv6 ports are labeled implicitly within the kernel, as specified by policy. The format for labeling port types is:

```
portcon protocol { port number | port range }
        context
```

The following defines a security context for labeling the standard printer port:

```
portcon tcp 515 system_u:object_r:printer_port_t
```

Network Interfaces

Each network interface (netif) is labeled with a security context, as specified in policy. Network interfaces are labeled as follows:

```
netifcon interface context default_msg_context
```

The default_msg_context parameter is intended to be used for labeling messages arriving on the interface, but is not currently used.

Here are some examples of netif labeling in policy:

```
netifcon eth0 system_u:object_r:netif_intranet_t [...]
netifcon eth1 system_u:object_r:netif_extranet_t [...]
```

Nodes

Under SELinux, a node refers to an IPv4 or IPv6 address and netmask. It allows host and network addresses to be labeled with security contexts via policy.

The format for labeling nodes is:

```
nodecon address mask context
```

Here are examples of labeling localhost addresses:

```
nodecon 127.0.0.1 255.255.255.255
        system_u:object_r:node_lo_t
nodecon ::1   ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff
        system_u:object_r:node_lo_t
```

Network Hooks and Permissions

Access-control hooks are implemented for every socket system call, allowing all socket-based network protocols to be mediated by SELinux policy. A few hooks are used only for housekeeping, but otherwise, hooks generally are used to check one or more access-control permissions.

As there are a large number of generic socket controls, see Table 1 for the relationships between the hooks, socket system calls and permissions.

Table 1. The Relationships between Hooks, Socket System Calls and Permissions

Hook	System Call	SELinux Permission
selinux_socket_create	socket	create
selinux_socket_post_create	socket	n/a
selinux_socket_bind	bind	bind
selinux_socket_connect	connect	connect
selinux_socket_listen	listen	listen
selinux_socket_accept	accept	accept
selinux_socket_sendmsg	sendmsg, send, sendto	write
selinux_socket_recvmsg	recvmsg, recv, recvfrom	read
selinux_socket_getsockname	getsockname	getattr
selinux_socket_getpeername	getpeername	getattr
selinux_socket_setsockopt	setsockopt	setopt
selinux_socket_getsockopt	getsockopt	getopt
selinux_socket_shutdown	shutdown	shutdown

Internally, the socket() system call is decomposed into two hooks. The selinux_socket_create hook is used to check whether the process can create a socket of the type requested. The selinux_socket_post_create management hook is used to assign a security label and socket class to the newly allocated inode associated with the socket.

The SELinux permissions also abstract the way system calls and other operations are viewed from a security point of view. Note, for example, that the getattr permission is used for the getsockname() and getpeername() system calls. They are seen to be equivalent security-wise by SELinux. Similarly, all of the sendmsg()- and recvmsg()-based system calls are reduced for security management purposes into simply read and write. For the curious, the code behind these hooks can be found in the 2.6 kernel, in security/selinux/hooks.c.

As sockets are also files, they inherit some of the access controls associated with files. Table 2 lists the file-specific hooks and permissions inherited by sockets.

Table 2. File-Specific Hooks and Permissions

Hook	System Call	SELinux Permission
selinux_file_ioctl	ioctl	ioctl
selinux_inode_getattr	fstat	getattr
selinux_inode_setattr	fchmod, fchown	setattr
selinux_file_fcntl	fcntl	lock
selinux_file_lock	fcntl, flock	lock
selinux_file_permission	write, write, read	append, write, read

UNIX Domain Controls

Under Linux, UNIX domain sockets can be created in an abstract namespace independent of the filesystem. Additional hooks have been implemented to allow mediation of communication between UNIX domain sockets in the abstract namespace, as well as to provide control over the directionality of UNIX domain communications. The selinux_socket_unix_stream_connect hook checks the connectto permission when one UNIX domain socket attempts to establish a stream connection to another. The selinux_socket_unix_may_send hook checks the sendto permission when one UNIX domain socket transmits a datagram to another.

Another feature of UNIX domain sockets under Linux is the ability to authenticate a peer with the SO_PEERCRED socket option. This obtains the user ID, group ID and process ID of the peer. Under SELinux, we also can obtain the security context of a peer via a new socket option SO_PEERSEC. Calling getsockopt(2) with this option invokes the selinux_socket_getpeersec hook, which copies the security context to a buffer passed in by the user. This is used for local IPC, such as Security-Enhanced DBUS.

Netlink Controls

Netlink sockets provide message-based user/kernel communication. They are used, for example, to configure the kernel routing tables and IPSec machinery.

Netlink communication is asynchronous; messages can be sent in one context and received in another. When a Netlink packet is transmitted, the sender's security credentials in the form of a capability set are stored with the packet and checked on reception. This allows, for example, the kernel routing code to determine whether the user who sent a routing table update is really permitted to do so.

As part of the LSM Project, capabilities logic was moved out of the core kernel code and into a security module, so that LSMs could implement different security models if needed.

The SELinux module uses the selinux_netlink_send hook to copy only the NET_ADMIN capability to a Netlink packet being sent to the kernel.

The selinux_netlink_recv hook is invoked when security-critical messages are received. SELinux uses this hook to verify that the NET_ADMIN capability was copied to the packet during transmission and, thus, whether the sending process had the capability.

An increasing number of Netlink families are being implemented, and SELinux defines subclasses of Netlink sockets for those that are security-critical. This allows the socket controls to be configured on a per-Netlink family basis (for example, to differentiate routing messages from kernel audit messages).

SELinux also is able to determine, by using the selinux_netlink_send hook, whether messages on certain types of Netlink sockets are read or write operations and then apply the nlmsg_read or nlmsg_write permissions, respectively. This allows fine-grained policy to specify, for example, that a domain can read the routing table but not update it.

IPv4 and IPv6 Controls

SELinux adds several controls for TCP, UDP and Raw socket subclasses. The node_bind permission determines whether a socket can be bound to a specific type of node. This obviously is useful only for local IP addresses and can be used to restrict a dæmon to binding to a specific IP address.

The name_bind permission controls whether a socket can bind to a specific type of port. This permission is invoked only when the port number falls outside of the local port range. The local port range is where the kernel automatically allocates port numbers from (for example, when choosing the source port for an outgoing TCP connection) and can be configured through the sysctl net.ipv4.ip_local_port_range. On a typical system, this range is:

```
$ sysctl net.ipv4.ip_local_port_range
net.ipv4.ip_local_port_range = 32768    61000
```

Thus, name_bind is invoked only when a socket binds to a port outside this range. SELinux always invokes the permission for ports below 1024, regardless of the sysctl setting. Both of these bind-related controls are called from the selinux_socket_bind hook, which is invoked through the bind(2) system call.

The send_msg and recv_msg permissions are used to control whether a socket can send or receive messages through a specific type or port.

A set of permissions is implemented that controls whether packets can be received or sent over TCP, UDP or Raw sockets for specific types of netif and node objects. These are tcp_send, tcp_recv, udp_send, udp_recv, rawip_send and rawip_recv.

These message-based controls are invoked for incoming packets at the selinux_sock_rcv_skb hook, the first point in the networking stack where we reliably can associate a packet with a recipient socket. For outgoing packets, SELinux registers a Netfilter hook and catches them at the IP layer; outgoing packets still have socket ownership information attached at this stage.

All of the above controls are protocol-independent in that they operate on both IPv4 and IPv6 protocols.

Network Policy

We've covered enough theory now to look at a real example of SELinux policy for a simple network application. Due to space limitations and the complexity of real-world networking, we develop a policy for a simple TCP echo client.

The source code for the client is available at the Web site listed in the on-line Resources for this article. Briefly, it creates a TCP socket, connects to a remote host's echo port, writes some text and then reads it back.

My workstation has two Ethernet interfaces, and in this example, eth0 is on an intranet, and the server I am connecting to has the IP address 10.3.1.2.

Here are the goals of the security policy:

- Grant the client only the OS access it absolutely needs.
- Allow the client to communicate only with inetd servers on the 10.3.1.0/24 subnet via eth0.

Policy

The following is an annotated security policy that meets these goals. To use it, install the SELinux policy sources package for your distribution, and cd to the top-level directory (/etc/selinux/strict/src/policy on my workstation).

Create a file called domains/program/echoclient.te, and add these policy entries as shown in Listing 1.

Listing 1. echoclient.te

```
# Simple echoclient policy for Linux Journal article
# File: domains/program/echoclient.te

# Define the echoclient_t type as a domain.
type echoclient_t, domain;

# Define echoclient_exec_t as a type of executable
# file.
type echoclient_exec_t, file_type, exec_type;

# This is a macro which will allow a correctly
# labeled executable to transition into the
# echoclient_t domain from the staff_t domain.
domain_auto_trans(staff_t, echoclient_exec_t,
                  echoclient_t)

# Designate which roles may enter the echoclient_t
# domain.
role staff_r types echoclient_t;

# This is a macro which allows the domain to use
# shared libraries.
uses_shlib(echoclient_t);

# Provide the permissions required to run the
# program when logged in via SSH as staff_t,
# allowing diagnostic and error messages to be
# written to the user's tty.
allow echoclient_t sshd_t:fd use;
allow echoclient_t staff_devpts_t:chr_file {
                               getattr read write };

# Network configuration

# These are the socket permissions required by the
# domain. Note that they are locked down to TCP
# sockets.
allow echoclient_t echoclient_t:tcp_socket {
               connect create read shutdown write };

# Allow the program to send and receive TCP messages
# to the echo port. In standard policy, the port is
# labeled as an inetd_port_t as it is one of a group
# of ports managed by inetd. You could modify the
# policy in net_contexts to lock this down to one
# port if needed.
allow echoclient_t inetd_port_t:tcp_socket {
                                recv_msg send_msg };

# Allow only TCP traffic over the intranet interface.
allow echoclient_t netif_intranet_t:netif {
                                tcp_recv tcp_send };

# Allow only TCP communication with internal IP
# addresses.
allow echoclient_t node_internal_t:node {
                                 tcp_recvtcp_send };
```

Add the following labeling definitions to the net_contexts file:

```
# Label eth0
netifcon eth0 system_u:object_r:netif_intranet_t
         system_u:object_r:unlabeled_t

  # Label the internal network.
nodecon 10.3.1.0 255.255.255.0
        system_u:object_r:node_internal_t
```

Update the types/network.te file:

```
# Define netif_intranet_t as a type of network
# interface.
type netif_intranet_t, netif_type;
```

Define a file context for the executable in a new file called file_contexts/program/echoclient.fc:

```
# Default file context for labeling
/tmp/echoclient -- system_u:object_r:echoclient_exec_t
```

Compile and load the policy:

```
$ make load
```

That's all—the policy is done. It seems like a lot to do, but it gets easier once you're familiar with the various policy files and types of policy entries needed. It also helps to use tools like audit2allow, which takes audit log denial messages and turns them into allow rules. It would be better to use a high-level GUI policy tool for day-to-day policy development; we've taken it step by step here to show how things work.

Testing

Now, build and label the client executable:

```
$ make echoclient
cc     echoclient.c   -o echoclient

$ restorecon /tmp/echoclient
```

Verify that it is labeled correctly:

```
$ getfilecon /tmp/echoclient
/tmp/echoclient system_u:object_r:echoclient_exec_t
```

You could have used `ls -Z` instead.

Let's see if it works—logged in as root in the staff_r role, using SSH:

```
$ id -Z
root:staff_r:staff_t

$ /tmp/echoclient 10.3.1.2
Sending message: 'Hello, cliche'
Received message: 'Hello, cliche'
```

It worked!

You can add auditallow rules to the policy to watch each permission being granted, if you want.

Let's verify that some of the policy rules are actually working.

1) Try to communicate with an IP address outside the intranet. Route the address locally, so you don't accidentally send a packet onto the Internet:

```
$ ip ro add 196.40.74.92 via 10.3.1.2 dev eth0

$ /tmp/echoclient 196.40.74.92
```

The program gets a TCP timeout, and the following audit denial message is generated when a packet is sent:

```
avc:  denied  { tcp_send } for  pid=10831
  exe=/tmp/echoclient saddr=10.3.1.1 src=32822
  daddr=196.40.74.92 dest=7 netif=eth0
  scontext=root:staff_r:echoclient_t
  tcontext=system_u:object_r:node_t
  tclass=node
```

As expected, the echoclient_t domain was denied access to transmit a TCP packet to a /node_t/ node, the default generic node context.

2) Try to communicate over the wrong interface. Route the echo server IP via the loopback interface, so packets will be sent there:

```
  $ ip ro add 10.3.1.2 via 127.0.0.2 dev lo

  $ /tmp/echoclient 10.3.1.2

avc:  denied  { tcp_send } for pid=10828
  exe=/tmp/echoclient saddr=10.3.1.1 src=32821
```

```
    daddr=10.3.1.2 dest=7 netif=lo
    scontext=root:staff_r:echoclient_t
    tcontext=system_u:object_r:netif_lo_t
    tclass=netif
```

This also is working correctly. The echoclient_t domain was denied access to transmit a packet over a netif_lo_t netif.

The echoclient program runs with a very minimal set of rights as defined in the policy. Anything not explicitly allowed is denied. The potential damage arising from a flaw in the program, user error or malicious user would be greatly confined by this policy.

This is a simple demonstration of how to meet network security goals with SELinux policy. A real-world policy would require several extra features, omitted for space and clarity, such as the ability to use ICMP messaging and DNS lookups. See the policy sources package of your distribution for some detailed examples, and also try some of the GUI policy tools.

Future Developments

It is likely that some form of labeled networking will be implemented for SELinux. This is where network traffic itself is labeled and typically is used in military and government environments dealing with classified information. An earlier version of SELinux used IP options to label packets, although it was dropped before merging with the upstream kernel as the hooks it needed were too invasive. A possible alternative is to integrate SELinux with IPSec and label the Security Associations (SAs) instead of the packets. A packet arriving on a specific SA would be labeled implicitly with the context of the SA. A prototype of this scheme was implemented for the preceding Flask Project, and it should be useful as a guideline.

More general integration of SELinux with network security components, such as cryptography and firewalling, also are areas for future exploration.

Acknowledgement

Thanks to Russell Coker for reviewing this article and providing valuable feedback.

Resources for this article: /article/7864.

James Morris (jmorris@redhat.com) is a kernel hacker from Sydney, Australia, currently working for Red Hat in Boston. He is a kernel maintainer of SELinux, Networking and the Crypto API; an LSM developer, and an Emeritus Netfilter Core Team member.

Encrypt Your Root Filesystem

Mike Petullo

Issue #129, January 2005

When you can't depend on physical security to keep your files safe, it's time to take the extra step of encrypting the filesystem. Although this article covers converting a PowerPC-based system, the principles are applicable to other architectures too.

In the *Linux Journal* article “Implementing Encrypted Home Directories” (August 2003), I described how to encrypt home directories transparently. This article describes how to implement another technique, an encrypted root filesystem. I discuss the GNU/Linux boot process and software requirements, present some instructions, introduce Open Firmware and discuss other relevant considerations. The system I use to teach these concepts is a New World PowerPC-based Apple iBook running a pre-release of Fedora Core 3. Despite these specifics, the concepts and procedures in this article can be applied to any device, architecture or operating system. My instructions assume you have a spare USB Flash disk and your system's firmware has the ability to boot off of it.

I also assume the reader is comfortable applying source patches and compiling programs. As of Fedora Core 3 Test 3, the mkinitrd and initscripts packages require patching to support an encrypted root filesystem. A basic understanding of how to manage partitions and create filesystems also is required. Performing a basic install of a Linux distribution is beyond the scope of this article.

Before presenting the technical steps involved, a higher-level concept must be discussed, trust. Trust is intertwined with cryptography and authentication. An implicit assumption of trustworthiness is given to any device that has an electronic key. For example, when I share my bank account PIN with an automatic teller machine, I trust that the ATM will not share my PIN with an inappropriate third party. In the same way, when I provide an encryption key to my computer, I assume the key will not be shared with anyone else. I trust the computer to keep the secret between us.

So, can you trust your computer? Unless you carry it with you everywhere, you really can't. This is true even if the disks have been encrypted. Consider this scenario: someone steals your computer as you sleep. The thief makes a copy of the encrypted contents of the computer, even though they are useless to him without their encryption key. He then replaces the encrypted laptop contents with something a little more diabolical and puts the computer back. When you wake up the next day, the computer prompts for an encryption password as it does every morning. But this time when you provide the key it electronically transmits the key to the thief. Because he now has a copy of your data and key, he can read your files.

This scenario may be a bit far-fetched, but it does illustrate a point. You can't trust your laptop. It's too big to keep your eyes on all the time. Therefore, no matter how well implemented your encryption system is, it is built without the prerequisite foundation of trust.

To ensure that we can trust the computer's boot process, we need to separate it from the computer. Consider this: you carry the keys to your car with you instead of carrying your car. Your encryption key is a natural conceptual leap from your car key. You can protect your encryption key more easily, so you don't have to carry your computer everywhere. To take things a little further and to address the above scenario, we also will place the software required to boot the computer on this key. The Flash disk will serve as this key. By protecting the software that boots the system initially, in addition to the encryption key, we can mitigate the risk of the boot process being hijacked.

An understanding of how your computer boots is required, because unlocking an encrypted root filesystem is integral to the bootstrap process. The current, stable kernel series, 2.6, optionally uses initramfs to help boot, as documented in LWN.net's “Initramfs Arrives”. Initramfs is a cpio archive that the kernel now knows how to unpack into a RAM-based disk. This unpacked filesystem contains a script that traditionally loads kernel modules needed to mount the root filesystem. In our case, this script also unlocks the encrypted root filesystem. More information on this subject can be found in the files buffer-format.txt and initrd.txt that are distributed with the Linux kernel sources.

Several filesystem encryption interfaces are available for Linux. Jari Ruusu's Loop-AES is one such project. Several cryptoloop variations that provide an encrypted loopback device also exist. This article focuses on the dm-crypt interface provided by recent 2.6 Linux kernels. This interface currently is preferred by the Fedora Project, and dm-crypt modules are provided by

Fedora's kernel packages. Also required is a statically linked cryptsetup. This utility simplifies the management of dm-crypt devices. Finally, parted and hfsutils are used to manage the boot filesystem.

Unfortunately, Fedora Core's anaconda installer does not yet support installing to an encrypted filesystem out of the box. To bypass this limitation, you must leave a partition free, install Fedora, format the free partition as an encrypted filesystem and copy the originally installed data onto the new encrypted filesystem. For the purpose of simplicity, I assume Fedora is to be installed onto two partitions: /dev/hda4, mounted at /home, and /dev/hda5, mounted at /. Because /home is not populated until after Fedora is installed, we can use /dev/hda4 as our spare partition and /dev/hda3 as the swap partition.

Install Fedora Core 3, mounting /dev/hda4 at /home and /dev/hda5 at /. Do not add any nonroot users yet, as /home will be wiped clean later. At this point, you should have a fully functioning Linux system.

Before an encrypted filesystem is set up, you should randomize the partition it will occupy. This eliminates a potential leak of information about the disk's contents. Figure 1 demonstrates an abstract disk that is half-full and not randomized properly. Figure 2 demonstrates a disk that was randomized properly before being formatted to contain an encrypted filesystem. Notice that, given Figure 1, one can gain some knowledge about its contents (such as that they span one-half of the disk). Figure 2 affords an adversary no such luxury. In this case, the disk could as easily be empty as full. A partition is randomized by overwriting its contents with random data: `dd if=/dev/urandom of=/dev/hda4`. This process can take a long time, because creating random data is somewhat difficult.

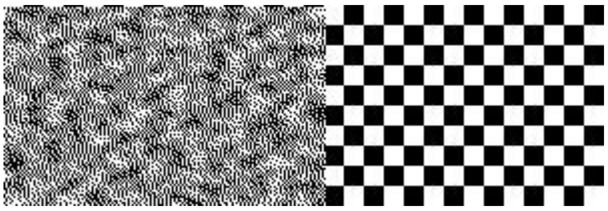


Figure 1. When you don't randomize the disk partition before creating the filesystem, an attacker can see how full it is.

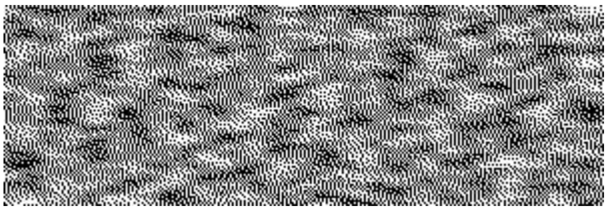


Figure 2. Randomizing the partition hides how much is used.

To create an encrypted ext3 filesystem on /dev/hda4, use the following steps:

1) Ensure that the aes, dm-mod and dm-crypt modules have been loaded into the kernel.

2) Unmount the partition that will host the encrypted root filesystem, /dev/hda4, from /home:

```
# umount /dev/hda4
```

3) Create a random 256-bit encryption key and store it at /etc/root-key:

```
# dd if=/dev/urandom of=/etc/root-key bs=1c count=32
```

This key will be copied to the Flash disk later.

4) Create a dm-crypt device, encrypted using the key you just generated:

```
# cryptsetup -d /etc/root-key create root /dev/hda4
```

Accessing /dev/mapper/root now provides an encrypted layer on top of /dev/hda4. By default, cryptsetup creates an AES-encrypted dm-crypt device and assumes a keyspace of 256 bits.

5) Create an ext3 filesystem on /dev/mapper/root:

```
# mkfs.ext3 /dev/mapper/root
```

6) Mount the new filesystem:

```
# mkdir /mnt/encroot
# mount /dev/mapper/root /mnt/encroot
```

7) Now that you have an encrypted filesystem, you must populate it with the contents of /dev/hda5 (the original root filesystem):

```
# cp -ax / /mnt/encroot
```

8) Finally, create an entry in /mnt/encroot/etc/crypttab so that various utilities know how the filesystem was configured:

```
root   /dev/hda4      /etc/root-key   cipher=aes
```

Now that we have our encrypted filesystem ready, it is necessary to understand a little more about the target architecture's boot process. Generally, computers have firmware that hands off execution to the software that will complete the system boot. Protecting firmware is beyond the scope of this article, so we assume that the system's firmware can be trusted. Most readers probably are familiar with the BIOS, the boot firmware used by the PC platform. I focus on Open Firmware, a boot system used by computer manufacturers such as Apple, Sun and IBM.

The installation instructions for NetBSD/macppc provide a good introduction to Open Firmware. We are interested in using Open Firmware's command-line interface to configure the computer to boot from a removable Flash disk. Open Firmware allows you to view the devices connected to a computer and view and set the value of firmware variables.

The Open Firmware prompt can be accessed by holding down option-command-o-f on a New World (G3 and later) Apple computer during the initial boot process.

The variable boot-device is used to determine what device the system should use to boot. The printenv command allows one to inspect its current value:

```
> printenv
[...]
boot-device     hd:,\\:txbi     hd:,\\:txbi
```

This essentially means "boot by executing the file of HFS type txbi on the first IDE disk." The second : character (before txbi) causes the token to be interpreted as an HFS file type. Otherwise, txbi would be interpreted as the path to a file. In my case, the token hd is actually an alias to the more complicated `/pci@f4000000/ata-6@d/disk@0`. This string represents the path through various subsystems to the first IDE disk. You can see what device an alias resolves to using Open Firmware's devalias command.

To set the boot-device correctly we need to discover by what name Open Firmware knows our Flash disk. Examining the device tree printed by the ls command reveals the path to the Flash disk:

```
> dev / ls
[...]
        /pci@f2000000
                [...]
                /usb@1b,1
                        [...]
                        /disk@1
[...]
```

Now that we know a little bit about the firmware's view of the computer, we must spend some time investigating the software the firmware initially executes: the bootloader. Generally, Linux systems that run on Apple's PowerPC architecture employ a program called yaboot to boot the system. yaboot is similar to LILO or GRUB and contains two key programs, ofboot.b and yaboot. ofboot.b provides the first stage of the bootstrap process. Essentially, it is ofboot.b's job to determine what operating system to boot. For example, if a system has both Mac OS X and Linux installed, ofboot.b executes Mac OS X or Linux's bootloader. If the user chooses to load Linux, ofboot.b executes yaboot, the second stage of the bootstrap process. yaboot then loads the Linux kernel and, in our case, an initrd. Figure 3 provides a illustration of how Linux boots using an encrypted root filesystem on the PowerPC architecture.

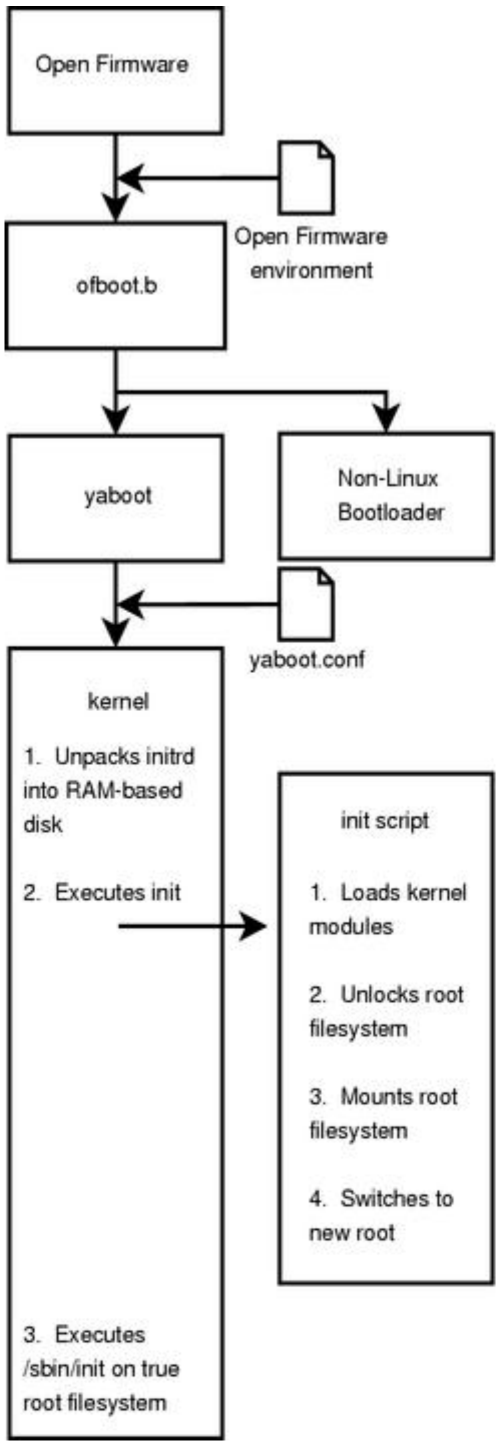

Open Firmware
Open Firmware environment
ofboot.b
yaboot
Non-Linux Bootloader
yaboot.conf
kernel
1. Unpacks initrd into RAM-based disk
2. Executes init
3. Executes /sbin/init on true root filesystem
init script
1. Loads kernel modules
2. Unlocks root filesystem
3. Mounts root filesystem
4. Switches to new root

Figure 3. The Booting Process on a PowerPC-Based System with Open Firmware

Our removable boot device requires the ofboot.b and yaboot programs, a Linux kernel and an initrd that contains the encryption key. Apple's current PowerPC-based architecture expects its boot media formatted using HFS.

1) Use the parted program to create the proper bootable partition on the Flash disk (mine is 64MB and is accessed using the device node /dev/sda):

```
# parted /dev/sda
(parted) mklabel mac
(parted) print
Disk geometry for /dev/sda: 0.000-62.500 megabytes
Disk label type: mac
Minor    Start       End     Filesystem  Name                  Flags
1          0.000      0.031              Apple
(parted) mkpart primary hfs 0.031 62.500
(parted) print
Disk geometry for /dev/sda: 0.000-62.500 megabytes
Disk label type: mac
Minor    Start       End     Filesystem  Name                  Flags
1          0.000      0.031              Apple
2          0.031     62.500              untitled
(parted) set 2 boot on
(parted) name 2 Apple_Boot
(parted) quit
```

2) Create an HFS on the boot partition:

```
# hformat /dev/sda2
```

3) Configure yaboot to boot off the appropriate device by modifying /mnt/encroot/etc/yaboot.conf. The following is a minimum configuration:

```
boot=/dev/sda2
ofboot=/pci@f2000000/usb@1b,1/disk@1:2
partition=2
install=/usr/lib/yaboot/yaboot
magicboot=/usr/lib/yaboot/ofboot
default=linux
image=/vmlinux
        label=linux
        root=/dev/hda4
        initrd=/initrd.gz
        read-only
```

The value `/pci@f2000000/usb@1b,1/disk@1:2` comes from our earlier inspection of the Open Firmware device tree, and `/pci@f2000000/usb@1b,1/disk@1` is the first disk on the USB bus on the PCI bus at f2000000. The device we are interested in is a disk, and `:2` means partition 2.

4) Install the bootstrap programs and kernel to /dev/sda2:

```
# ybin --config /mnt/encroot/etc/yaboot.conf -v
# mount /dev/sda2 /media/usbstick
# cp /boot/vmlinux /media/usbstick
```

At this point, the crypto-aware initrd must be installed onto the Flash disk. Fedora provides a tool named mkinitrd that can create an initrd. However, at the time this article was written, mkinitrd did not know how to mount an encrypted root. The patch at https://bugzilla.redhat.com/bugzilla/show_bug.cgi?id=124789 provides this functionality. Once the patch is applied, mkinitrd reads /etc/crypttab and creates an appropriate initrd:

```
mkinitrd --authtype=paranoid -f /media/usbdisk/initrd.gz <kernel version>
umount /media/usbstick
```

The file /mnt/encroot/etc/fstab should be updated to reflect the changes made:

```
/dev/mapper/root /      ext3    defaults        1 1
```

Encrypted swap or the absence of swap space entirely is a prerequisite for an encrypted filesystem. Reasons for this can be found in “Implementing Encrypted Home Directories” and in a BugTraq mailing-list thread titled “Mac OS X stores login/Keychain/FileVault passwords on disk”. When the patch at https://bugzilla.redhat.com/bugzilla/show_bug.cgi?id=127378 is applied to the initscripts package, Fedora allows users to encrypt their swap partitions using a randomly generated session key. Because swap space isn't generally required to be consistent across reboots, the session key is not saved when the system is powered down. To enable encrypted swap, complete the following steps:

1) Add the following line to /mnt/encroot/etc/fstab, replacing any previous swap record:

```
/dev/mapper/swap swap   swap    defaults        0 0
```

2) Add the following line to /mnt/encroot/etc/crypttab to tell the system how to perform the encryption:

```
swap    /dev/hda3       /dev/urandom    swap
```

At this point we should be able to reboot the system and use our encrypted filesystem. Again, we need to hold down option-command-o-f to enter the Open Firmware prompt.

As demonstrated above, the path to the Flash drive's second partition is /pci@f2000000/usb@1b,1/disk@1:2. Knowing this, we can build the path `/pci@f2000000/usb@1b,1/disk@1:2,\ofboot.b`. The , deliminates between the partition number and the filesystem path; \ofboot.b is the filesystem path, and \ is like UNIX's / with the filesystem root at the device's root:

```
> dir /pci@f2000000/usb@1b,1/disk@1:2,\
Untitled            GMT                     File/Dir
   Size/      date        time  TYPE        Name
   bytes   9/ 3/ 4   21:44:41  ???? ????  initrd.gz
 2212815   8/28/ 4   12:24:21  tbxi UNIX  ofboot.b
    3060   9/ 3/ 4    2:21:20  ???? ????  vmlinux
  141868   9/28/ 4   12:24:22  boot UNIX  yaboot
     914   9/28/ 4   12:24:22  conf UNIX  yaboot.conf
```

This confirms that Open Firmware can read the files required to boot the system. Setting the value of the boot-device variable to `/pci@f2000000/usb@1b,1/disk@1:2,\ofboot.b` causes the system to boot from the Flash disk: `setenv boot-device /pci@f2000000/usb@1b,1/disk@1:2,\ofboot.b`.

Once the system successfully boots from the encrypted root, it is necessary to destroy all of the data on /dev/hda5. This can be done with the same procedure used to randomize the root filesystem's partition: `dd if=/dev/urandom of=/dev/hda5`. You may want to perform this overwrite several times. For one standard on sanitizing disks, see Chapter 8 of the US Department of Defense's “National Industrial Security Program Operating Manual”.

Following a safe sanitization, /dev/hda5 may be used as /home. The /home filesystem also should be encrypted. Luckily, this is a much simpler process, because the system need not boot off of /home. Creating the filesystem itself is similar to the steps taken to create the root filesystem.

1) Ensure that the aes, dm-mod and dm-crypt modules have been loaded into the kernel.

2) Unmount the partition that will host the encrypted home filesystem, /dev/hda5, from /home:

```
# umount /dev/hda5
```

3) Create a random 256-bit encryption key, and store it at /etc/home-key. One way to do this is:

```
# dd if=/dev/urandom of=/etc/home-key bs=1c count=32
```

4) Create a dm-crypt device, encrypted using the key you just generated:

```
# cryptsetup -d /etc/home-key create home /dev/hda5
```

5) Create an ext3 filesystem on /dev/mapper/home:

```
# mkfs.ext3 /dev/mapper/home
```

6) Mount the new filesystem:

```
# mount /dev/mapper/home /home
```

7) Create an entry in /etc/crypttab, so that various utilities know how the filesystem was configured:

```
root     /dev/hda5       /etc/home-key    cipher=aes
```

8) Finally, update /etc/fstab to contain an entry for /home:

```
/dev/mapper/home /home   ext3    defaults         1 2
```

At this point, it is appropriate to begin adding nonroot local user accounts to the system. Setting up the encrypted root filesystem is now complete.

Having all of your data encrypted can be dangerous. If the encryption key is lost, your data is lost. Because of this, it is important to make backup copies of the Flash disk containing your key. It also is crucial to perform plain-text backups of the encrypted data. If you maintain a bootable rescue disk, it may make sense to rethink the system components that should be on it. A copy of your root and home filesystem keys, parted, hfsutils, the cryptography-related kernel modules and cryptsetup are excellent candidates.

How effective is this technique in protecting your data? In his book, *Secrets and Lies*, Bruce Schneier presents a technique that is useful in evaluating this. An attack tree can be used to model threats. Figure 4 presents the beginning of an attack tree for our encrypted filesystem. It is important to note that this attack tree is not complete and probably never will be.

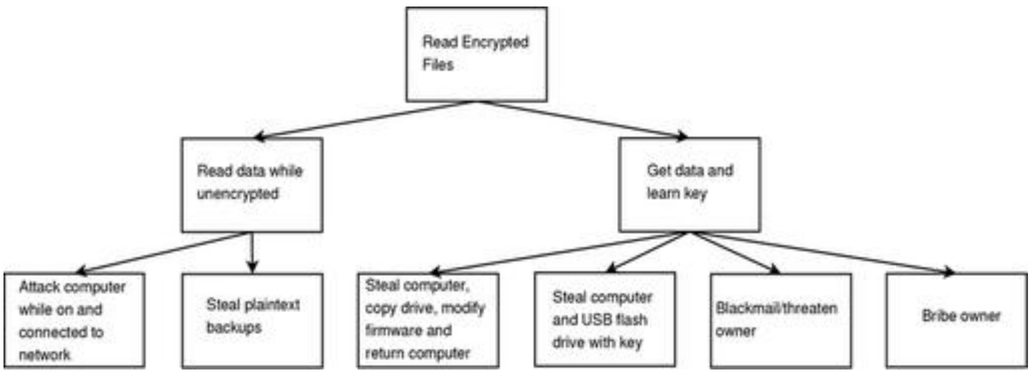

Read Encrypted Files
Read data while unencrypted
Get data and learn key
Attack computer while on and connected to network
Steal plaintext backups
Steal computer, copy drive, modify firmware and return computer
Steal computer and USB flash drive with key
Blackmail/threaten owner
Bribe owner

Figure 4. How can an attacker read the encrypted filesystem?

By using the techniques in this article and a little creative thinking, it is possible to make the data on your hard disk more resistant to certain types of theft. It is important to keep in mind the types of attacks that circumvent these defensive techniques. Though other techniques must be used to protect against network-based and other attacks, those described here are a powerful tool toward the goal of overall system security.

Resources for this article: /article/7865.

Mike Petullo currently is working at WMS Gaming as a test engineer. He has been tinkering with Linux since 1997 and welcomes your comments at lj@flyn.org.

How I Feed My Cats with Linux

Chris McAvoy

Issue #129, January 2005

Give your Linux box the power to control real-world events with an inexpensive microcontroller from Parallax, a Python program and some serial port magic.

Cats love toys. Our cats, Cotton and Tulip, slowly have taken over our house with their little plastic doo-dads—ping-pong balls, furry mice, bells, springs and things to scratch. The cats are rarely bored. On the weekends, my wife and I oblige the kittens by tossing their toys around the house, flinging strings and jingling bells. We scratch their backs and feed them treats. They're both in love with these little stinky fish treats; all we need to do is shake the can, and they stop whatever they're doing and dash to the kitchen. Their English lexicon now includes their names and the words good and treats.

Monday through Friday, nine to five, however, the cats are responsible for their own entertainment. While we're away, we're sure the cats have a good time with their toys. Our rugs almost always are moved around, ping-pong balls end up in water dishes and fur covers our chairs. The only real difference between the weekday and the weekend is our presence and the lack of treats.

We have to work, but that doesn't mean our cats should have to go without stinky little fish, right? Why should our economic necessities have a negative effect on their treat times? Isn't it our responsibility to build them an Internet-enabled, Linux-based, cat-feeding device?

Where do we start? Three ingredients are key to our Linux-based Internet cat feeder: logic on the system, a way to talk to a device and a device to talk to. I chose Python for the logic piece, talking over a serial port to a microcontrolled cat feeder of my own design. Let's start at the bottom, the device, and work our way up to the top, the logic.

The BASIC Stamp Microcontroller

I first heard about the BASIC Stamp microcontroller from an article on Slashdot in which three guys were using a BASIC Stamp to control a bolt gun. They had taken some great pictures of bolts destroying fruit. Microcontrollers, I soon learned, are everywhere. They're the bits of logic in our microwaves and our remote controls. They are tiny and often difficult to use.

Parallax, Inc., specializes in making microcontrollers for non-engineers, specifically for students and hobbyists. Parallax products are well documented, easy to use and relatively inexpensive. I bought the Homework Board, the most inexpensive starter kit, from Radio Shack for around $75 US. It came with a book, a bag of electronic components for the experiments in the book and the board and chip.

The Stamp itself actually is a PIC microcontroller with some memory. Typically, you need to program microcontrollers with a low-level language, such as Assembly. What sets the BASIC Stamp apart from a typical microcontroller is the programming language you use to make it do stuff. Parallax developed a superset of BASIC, called PBASIC, that makes it easy to build expressive, useful programs quickly. In addition, the Homework Board has an integrated solderless breadboard, which makes for quick rewiring of projects.

The BASIC Stamp has 16 I/O pins. Each pin is set to high, +5V, or low, 0V, based on programs you create. Say you want to make an LED blink. You attach one end to an I/O pin and the other to a ground pin. You write a program that says, every second, turn the I/O pin to high (on), wait for a second, then turn it to low (off). Now replace the LED with a servo, and we've got the start of the cat feeder.

The I/O pins also listen for +5V or 0V. PBASIC even has a built-in function that allows an I/O pin to read serial data, the basis of which are high/low charges that make up binary words. Don't worry too much about serial connections yet; we cover them more in the next section. For now, understand that the BASIC Stamp can receive a command easily from a Linux system over a serial cable and turn on a servo that drives our cat feeder.

Linux and the STAMP

Parallax has done a great job of creating a fun community of hobbyists. Two mailing lists are devoted to its products, and dozens of sites have ideas for projects. Although the best integrated development environment for the BASIC Stamp is available only for Microsoft Windows, a tool called bstamp has been created, with Parallax's help, to program a BASIC Stamp with Linux. An example of tokenizing a program and running it, follows:

```
# bstamp_tokenize catcode.bs2 catcode.tok
PBASIC Tokenizer Library version 1.16

# bstamp_run catcode.tok
Model: Basic Stamp 2
Firmware version BCD = 16
Ack = 0
Ack = 0
Ack = 0
Ack = 0
Ack = 0
Ack = 0
Ack = 0
Ack = 0
Ack = 0
Ack = 0
Ack = 0
Ack = 0
DEBUG OUTPUT:  (Press [Control]-[C] to complete sequence)
_________________________________________________________
Waiting for Command
Received Command: B
Feed the kitty!
Waiting for Command
Received Command: B
Feed the kitty!
Waiting for Command

__________________________________________________________
Received [Control]-[C]!
Shutting down communication!
```

The Much-Maligned Serial Cable

Everything I know about serial, I learned from the excellent “Serial Howto” by David S. Lawyer and Greg Hankins. It's a thick document, with a lot of interesting, low-level information about the RS-232 standard.

Although the BASIC Stamp communicates with bstamp over a serial connection, the serial port provided with the Homework Board is not a good candidate for true serial communication. Parallax wired the port in a nontraditional way. For one thing, all commands sent to the port are echoed back to the host, which makes two-way communication difficult.

The RS-232 standard dictates that the electrical signals that travel along our cable be +/- 12V. Because of this, if we hook up a serial connection directly to our Stamp I/O pins, we most likely will burn it out, as it expects +5V. The solution is to use an intermediary to lower the 12V signal coming from the PC to 5V and boost the 5V signal coming from the Stamp to 12V. Such a chip does exist, and it is called a MAX232. Luckily, you can get a MAX232-based RS-232-compliant adapter specifically built for solderless breadboards from a Texan named Al Williams. The device is called the RS-1, and a link to his Web site is included in the on-line Resources for this article.

Starting with the 2.4 kernel, Linux names serial ports as /dev/ttyS0, 1, 2, 3 and so on. These device files act like any other file. You open them, read or write to them and close them. The OS buffers reads and writes with a 16k buffer, which is more than adequate for most serial communication. This is good; you don't have to worry about losing bits simply because you weren't reading at the exact moment your device sent them across the wire. It also means you need to flush the buffers explicitly on the OS side when you're ready to send.

Because the port is treated as a file, you need to set the permissions accordingly. In my case, because I ultimately want a CGI program to drive the feeder, I made apache the owner. If you're in a secure environment, you always could `chmod 777 /dev/ttyS0`, but this obviously is insecure. It's best to decide up front what you want to do with your port and set the permissions in as secure a way as possible.

Python Takes Control

Because Linux treats our serial port as a file, it's easy to use Python to talk to the Stamp. Python's file object makes it simple to read and write files:

```
>>> f = open("/tmp/cotton.txt",'w')
>>> f.write("Cotton loves treats!")
>>> f.close()
>>> f = open("/tmp/cotton.txt",'r')
>>> f.read()
'Cotton loves treats!'
>>> f.close()
```

As you can see, however, although it's easy to open and close a file, doing so could get tricky if that file actually is a serial port. Fortunately for us, our Python script needs to write only a letter at a time to tell the feeder to dispense a treat. That said, I wanted to use as robust a method of communication as possible, and all this opening and closing worried me, as I see this project as something that always will be a work in progress. Maybe the cats will want to hit a button that sends us a message at work, who knows? The point is, I wanted something that was more serial-aware than a straight file handle. Luckily, someone else wanted the exact same thing. Chris Liechti has been nice enough to create PySerial for exactly this sort of situation. Here's an example of PySerial in action:

```
>>> import serial
>>> sp = serial.Serial(0)
>>> sp.portstr
'/dev/ttyS0'
>>> sp.write("F")
>>> sp.readline()
```

We don't actually open /dev/ttyS0, we open 0. PySerial is smart enough to know we mean the first serial port and opens it accordingly. This also means that PySerial is cross-platform, so you don't have to know that your port is /dev/ttyS0 on one machine and /dev/ttya on another. It's all handled by PySerial.

Now that Python is talking over the serial port, we need to get it on-line. I admit, I'm not terribly fond of Python in a CGI environment. Don't let that stop you though; there's a working group whose mission it is to see the CGI libraries improved, and several Python Web frameworks make CGI unnecessary. In addition, mod_python, in its latest release, has included Python Server Pages (PSP), a PHP-like syntax for mixing Python directly into an HTML page. In short, you have a lot of options when it comes to using Python on-line. For our purposes, however, the Python CGI library is more than enough to keep our kittens well fed.

Here's a brief CGI example for a bare-bones cat feeder:

```
#!/usr/bin/env python
import serial
import cgi

class Feeder:
  def __init__(self):
    self.port = serial.Serial(0)
  def feed(self):
    self.port.write("B")

print 'Content-Type: text/html'
print # Blank line marking end of HTTP headers

cgiParameters = cgi.FieldStorage()
control = Feeder()
control.feed()

print "<p>Thanks for feeding the kittens!"
```

First of all, I import the PySerial and CGI modules and then I declare a class feeder. The constructor for the class opens the serial port. The class has one method, feed, which sends an arbitrary character, in this case B, down the wire to the feeder. On the other end, PBASIC is listening for the character B and dispenses a treat when it sees it.

Let's Feed the Kittens!

I built the cat feeder with a carousel design, where the treats would be put into cells, divided by a rotating paddle, driven by a servo. When the paddles rotate, a load of treats drops through a cutout in one of the cells to a food bowl. I used a container meant for storing frozen waffles for the carousel, with a custom cut rotating paddle and a Parallax servo to drive it all. The whole assembly, including the BASIC Stamp circuit, is housed in a plastic storage box in my home office. The box is connected to my Web server on /dev/ttyS0 for the feeder and /dev/ttyS1 for the debug port. Figure 1 is a picture of the cat feeder on my shelf. I'm using Fedora Core 1, with Apache 2.0.48.

Figure 1. Cotton Getting a Treat from the Feeder

The initial problem was how do I determine that the paddles have rotated enough to drop the treats and stop them from rotating? The easiest solution was to put a small sensor on the side of the carousel that would detect a paddle passing in front of it. I chose a sensor from Parallax used primarily to find a black line on the ground. I put a flat black piece of posterboard on the edge of each paddle and the sensor on the bottom of the carousel right after the edge of the hole. When the feeder feeds, the sensor detects when the first paddle moves past; when the second paddle passes the sensor, the servo stops.

I wired up the Stamp relatively quickly over a few days of experimenting. The attached breadboard is a great feature of the Homework Board. I was able to rewire and test circuits quickly without having to solder and desolder. Figure 2 shows the completed schematic. Figure 3 is a picture of the wired-up Homework Board.

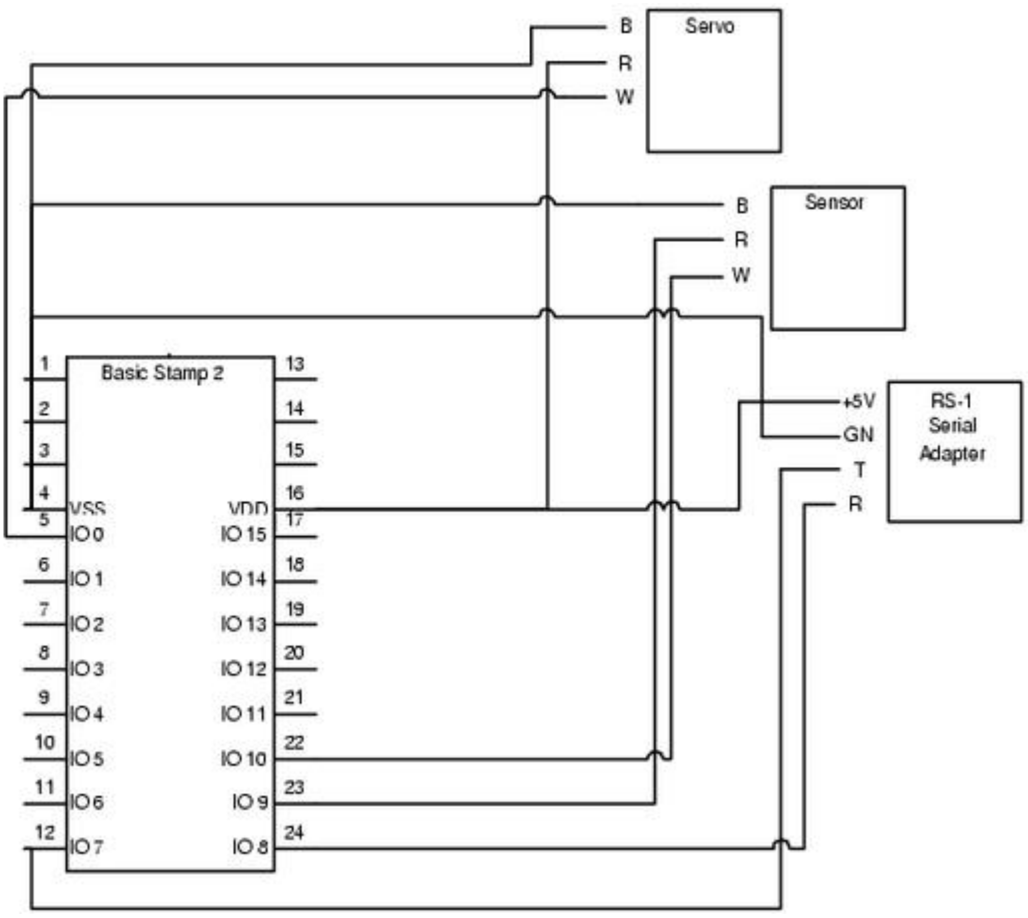


Figure 2. The Feeder Circuit

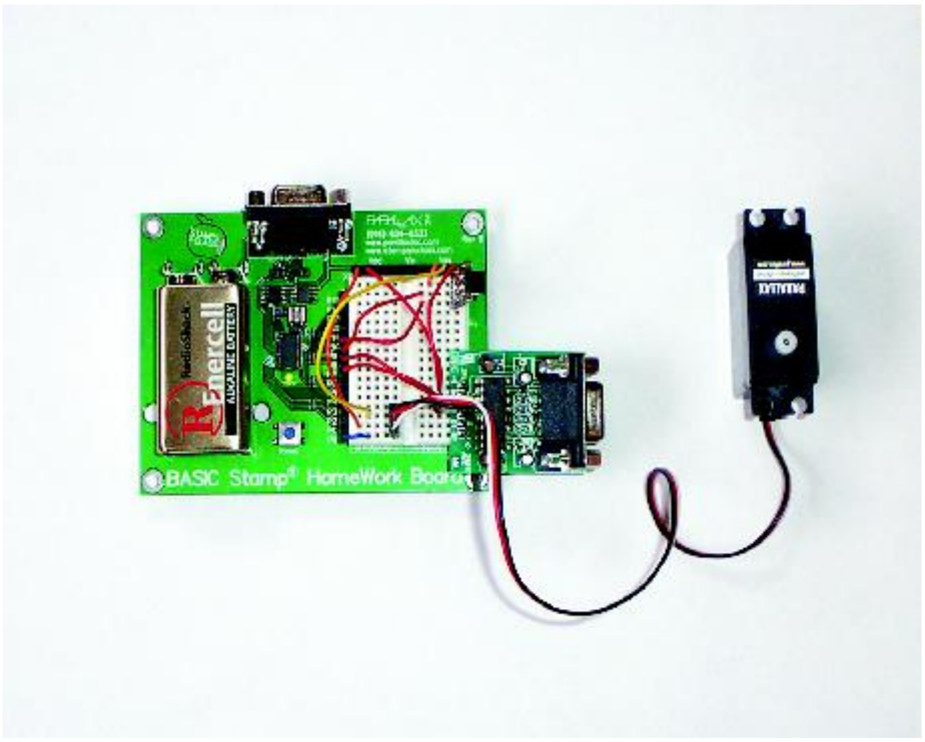


Figure 3. The Completed Circuit

Writing the code for the BASIC Stamp mostly was a matter of cutting and pasting code examples from the Parallax Web site. Listing 1 is the final code that I used on the Stamp. PBASIC relies heavily on GOTO statements for flow control, which took some getting used to on my part.

Listing 1. The Feeder PBASIC Code

```
'{$STAMP BS2}
cmd     VAR     Byte
temp    VAR     Word
LineSnsrPwr     CON 10
LineSnsrIn      CON 9
Sense   VAR     Word
SStart     VAR Word

main:
  DEBUG "Waiting for Command", CR
  SERIN 7, 84, [cmd]
  DEBUG "Received Command: ", cmd, CR
  IF cmd = "B" THEN feed
  GOTO main

feed:
  DEBUG "Feed the kitty!", CR
  HIGH LineSnsrPwr ' activate sensor
  HIGH LineSnsrIn ' discharge QTI cap
  PAUSE 1
  RCTIME LineSnsrIn, 1, SStart
  DEBUG "First Reading: ", DEC SStart, CR
  GOTO sensor

feed2:
  IF Sense < (SStart - 200) THEN pastfirst
  IF Sense > (SStart + 200) THEN stopfeed
  FOR temp = 1 TO 100
    PULSOUT 0,600
  GOTO sensor

sensor:
  HIGH LineSnsrPwr ' activate sensor
  HIGH LineSnsrIn ' discharge QTI cap
  PAUSE 1
  RCTIME LineSnsrIn, 1, Sense
  GOTO feed2

pastfirst:
  DEBUG "Past First!", CR
  SStart = Sense
  GOTO sensor

stopfeed:
  DEBUG DEC Sense, CR
  GOTO main
```

Listing 2 shows the complete Python code we use to drive the feeder. The CGI script uses the built-in shelf module for Python; shelf allows you to store live objects in a DBM database. In addition to shelf, I'm also using the Cheetah Template engine. The line `t = Template(open('feeder.tmpl.py').read())` opens an HTML template called feeder.tmpl.py, reads the contents and uses it as the Cheetah template. The template format looks something like `<p>The cats have been fed $fed times</p>`. When we set the template variable, t.fed, to some number (say 5), the line then becomes `<p>The cats have been` `fed 5 times</p>`. My wife, a graphic designer by trade, whipped up some graphics for the page.

Listing 2. The Final Python CGI Code

```
#!/usr/bin/python
import serial
import cgi
from Cheetah.Template import Template
import shelve

t = Template(open('feeder.tmpl.py').read())
port = serial.Serial(0)

class Feeder:
  def __init__(self):
    self.total_fed = 0
  def feed(self):
    self.total_fed = self.total_fed + 1
    port.write("B")
  def getTotalFed(self):
    return self.total_fed

print 'Content-Type: text/html'
print # Blank line marking end of HTTP headers

form = cgi.FieldStorage()

d = shelve.open("feeder.dbm")
if d.has_key("control"):
  """ if shelf file exists, open it, otherwise create
  it and a new instance of the Feeder class """
  control = d['control']
else:
  control = Feeder()
  d['control'] = control

if form.has_key("command") and \
form['command'].value == 'feed':
  """ if we received the feed command,
  feed, otherwise, show the index page"""
  control.feed()
  contents = """
    <p class="header">
    Thanks for the Treat!</p>
    <p class="body">Meow!</p>
    <p valign="bottom">
    <a href="index.py">Back</a></p>"""

else:
  """The index welcome page"""
  contents = """
    <p class="header">Cotton & Tulip Love Treats!</p>
    <p class="body">
    Click the Fish Below to Give
    Cotton and Tulip a Treat</p>
    <p><a href="index.py?command=feed">
    <img border="0" src="images/dance_fish.gif">
    </a></p>
    <p><br><p>
    <p class="footer" valign="bottom">
    The kitten feeder is an honest to goodness device
    attached to a Linux Server in Chris and Camri's
    apartment.
    """
"""Set the variables that Cheetah will use"""
t.contents = contents
t.fed = control.getTotalFed()
"""Print the complete Page"""
print t

"""Save the control to our shelf"""
```

```
    d['control'] = control
    d.close()
```

The cat feeder is open for business. Occasionally, a treat jams up the works, but 95% of the time, the cats get a stinky little fish. We already have plans for cat feeder v.2.0. We'd love to add a Webcam to see the kittens during the day, as well as move the device to a wireless laptop in the kitchen. The feeder is, as with most of our projects, a work in progress. Feel free to go to the Web site and give Cotton and Tulip a treat.

Resources for this article: /article/7741.

Chris McAvoy is a UNIX Administrator in Chicago, Illinois. He lives with his wife, Camri, and their two cats, Cotton and Tulip. Chris and Camri have a lot of hobbies. Their Web site is www.lonelylion.com.

Archive Index Issue Table of Contents

Advanced search

Application Defined Processors

Dan Poznanovic

Issue #129, January 2005

By rebuilding a system's logic on the fly, this project can make one FPGA do the work of tens or hundreds of ordinary processors.

Application defined processors are based on the concept of reconfigurable computing (RC). RC is a computing technology that blurs the line between software and hardware and provides the basis for the next big steps forward in delivering high performance with reduced power and space requirements. RC is implemented using hardware devices that can be reconfigured. Processors in an RC system are created as hardware that is optimized for the application that executes in it.

This article explains RC, examines SRC systems that implement RC and shows the performance advantage RC provides over traditional microprocessors. We also explore the programming model for RC and discuss the potential RC provides for supporting Open Hardware.

What Is Reconfigurable Computing and Why Do I Care?

RC is a form of computing based on hardware that can be created dynamically for each application that will run in it. RC hardware is comprised of chips whose logic is defined dynamically rather than at the time the chips are fabricated. RC has been around for many years and implemented in a number of different hardware components, such as field programmable gate arrays (FPGAs), field programmable object arrays (FPOAs) and complex programmable logic devices (CPLDs). What is important to application developers is that today's reconfigurable chips have a clock rate and capacity that make it practical to do large-scale computing with RC hardware.

The most familiar chip type used to implement RC is the FPGA. An FPGA is a chip composed of SRAM memory cells used to define a configuration for the chip. FPGAs contain logic gates, flip-flops, RAMs, arithmetic cores, clocks and configurable wires to provide interconnection. FPGAs can be configured to implement any arbitrary logic function and, therefore, can be used to create custom processors that can be optimized to an application.

So, a collection of FPGAs could be configured to be a MIPS, SPARC, PowerPC or Xeon processor, or a processor of your own design. In fact, the processor need not even be an instruction processor. It could be a direct execution logic (DEL) processor that contains only computational logic requiring no instructions to define the algorithm.

DEL processors hold great potential for high performance. A DEL processor can be created with exactly the resources required to perform a specific algorithm. Traditional instruction processors have fixed resources, adders, multipliers, registers and cache memory and require significant chip real estate and processing power to implement overhead operations, such as instruction decode and sequencing and cache management.

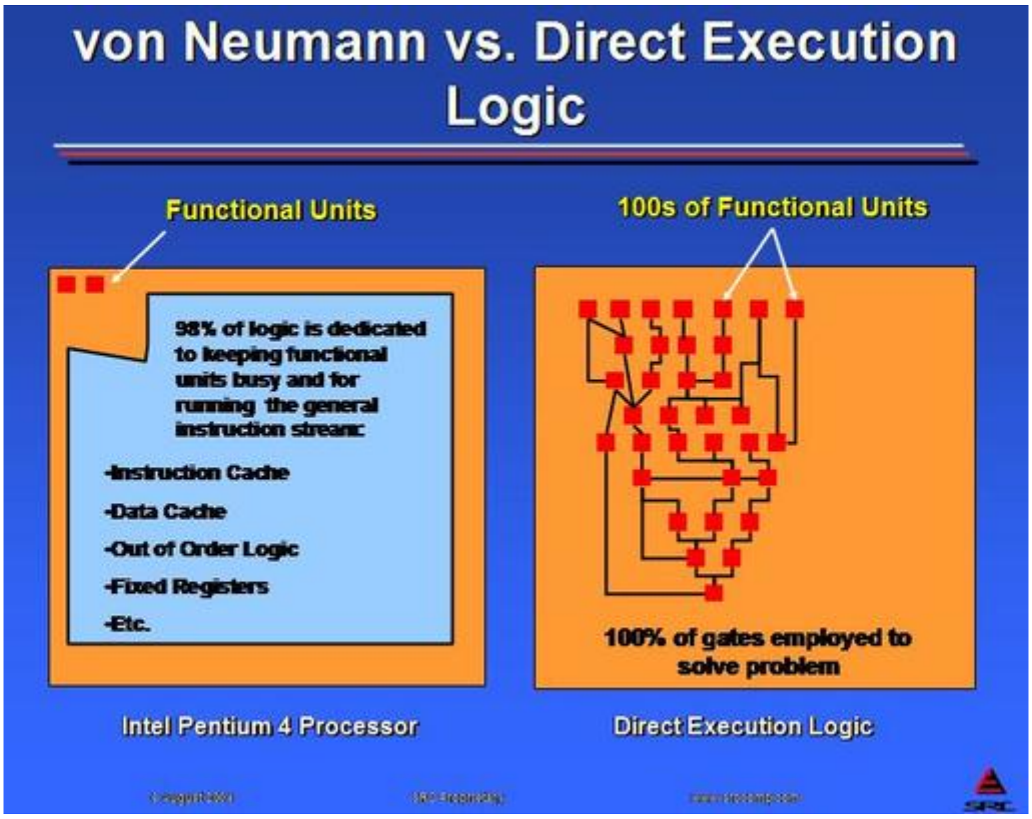
DEL processors are reconfigurable computers created for each application in contrast to a fixed architecture microprocessor where one size fits all. A DEL processor delivers the most efficient circuitry for any particular application in terms of the precision of the functional units and parallelism that can be found in the algorithm. Being reconfigurable, a unique DEL processor can be created for each application in a fraction of a second.

But why do you care that a DEL processor can be created dynamically for an application, and that it uses its chips more effectively than a microprocessor? The answer is simple: performance and power efficiency. A DEL RC processor can be created with all of the parallelism that exists within an algorithm without the overhead present in a microprocessor. For the remainder of this article, RC processors are assumed to be implemented using FPGAs in order to be more specific in the discussion.

How Is that High Performance Achieved?

Performance in RC processors comes from parallel execution of logic. RC processors are completely parallel. In fact, the task of constructing the logic for a given algorithm is to coordinate the parallel execution such that intermediate results are created, communicated and retained at the proper instants in time.

A DEL processor is constructed as a network of functional units connected with data paths and control signals. Each computational element in the network becomes active with each clock pulse. Figure 1 shows a fragment of logic for computing an expression and contrasts the utilization of the chip versus a von Neumann instruction processor, like the Intel Pentium 4 microprocessor.



von Neumann vs. Direct Execution Logic

Functional Units

98% of logic is dedicated to keeping functional units busy and for running the general instruction stream:

- Instruction Cache
- Data Cache
- Out of Order Logic
- Fixed Registers
- Etc.

Intel Pentium 4 Processor

100s of Functional Units

100% of gates employed to solve problem

Direct Execution Logic

Figure 1. Direct execution logic can put all logic gates to work on the real problem.

Even though a microprocessor can operate at a clock frequency of 3GHz and the FPGA chips operate in the 100–300MHz frequency range, the parallelism and internal bandwidth on a DEL processor can outperform the microprocessor by orders of magnitude better delivered performance. Figure 2 presents some benchmark comparisons between SRC's DEL processor, MAP, and a typical von Neumann instruction processor, the Intel Xeon 2.8GHz microprocessor. Parallel execution of exactly the required number of functional units, high internal bandwidth, elimination of instruction processing overhead and load/store elimination all contribute to overcoming the 30× difference in clock frequency between the MAP and the Intel microprocessor.

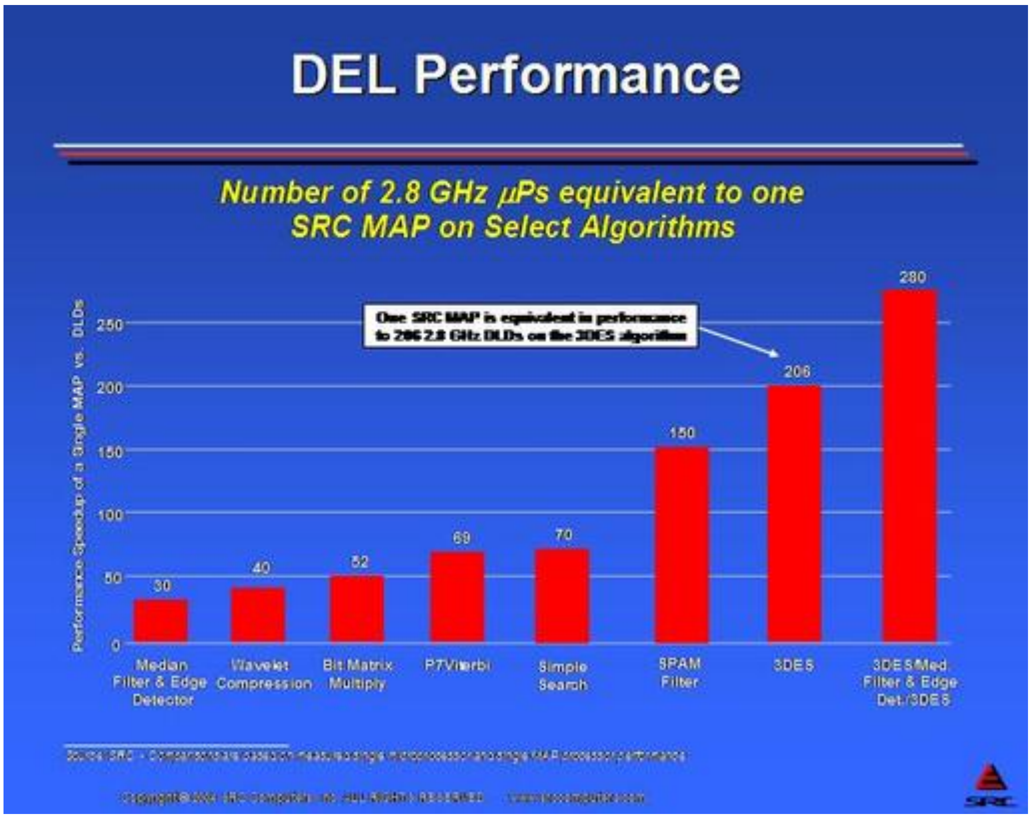


Figure 2. Number of 2.8GHz microprocessors required for the same performance as a MAP direct execution logic processor.

But Can a DEL Processor Run Linux?

DEL-based processors could run Linux, but do they need to? Code segments within the Linux kernel certainly might benefit in performance from running on a DEL processor, and applications within the Linux distributions also could achieve higher performance. However, the role of an operating system, and the kernel in particular, is to manage the hardware such that applications achieve their required performance levels. In other words, the OS is supposed to stay out of the way and let applications consume the hardware.

Applications do a lot more than intense computation. They interact with users, read and write files, display results and communicate with the world through Internet connections. Thus, applications require both computational resources and the services of an operating system. Heavy computation with high parallelism benefits from DEL processors. Although serial code could run as DEL, it is best serviced in a traditional microprocessor.

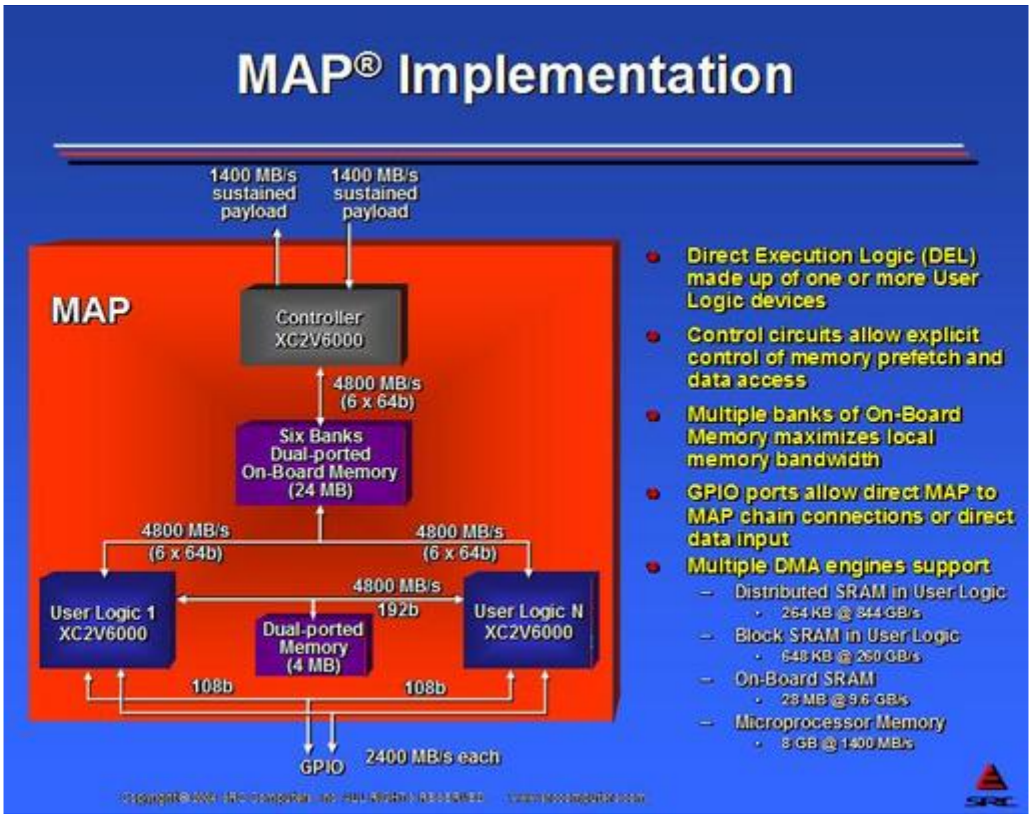
The best combination of hardware for running most applications is a mix of microprocessor and DEL processors. This combination allows applications to achieve orders of magnitude performance gains while still running in a standard Linux environment with all of the OS services and familiar support tools. The portion of an application that is predominantly sequential or that requires OS services can run in a traditional microprocessor portion of a system, while applications and even portions of the OS that benefit from the DEL parallelism run on a closely coupled DEL processor.

SRC Computers, Inc.'s RC System

SRC has created systems that are composed of DEL processors and microprocessors. SRC systems run Linux as the OS, provide a programming environment called Carte for creating applications composed of both microprocessor instructions and DEL, and support microprocessor and DEL processor hardware in a single system.

The DEL Processor—MAP

The patented MAP processor is SRC's high-performance DEL processor. MAP uses reconfigurable components to accomplish control and user-defined compute, data prefetch and data access functions. This compute capability is teamed with very high on- and off-board interconnect bandwidth. MAP's multiple banks of dual-ported On-Board Memory provide 11.2GB/sec of local memory bandwidth. MAP is equipped with separate input and output ports with each port sustaining a data payload bandwidth of 1.4GB/sec. Each MAP also has two general-purpose I/O (GPIO) ports, sustaining an additional data payload of 4.8GB/sec for direct MAP-to-MAP connections or data source input. Figure 3 presents the block diagram of the MAP processor.



- Direct Execution Logic (DEL) made up of one or more User Logic devices
- Control circuits allow explicit control of memory prefetch and data access
- Multiple banks of On-Board Memory maximizes local memory bandwidth
- GPIO ports allow direct MAP to MAP chain connections or direct data input
- Multiple DMA engines support
  - Distributed SRAM in User Logic
    - 264 KB @ 844 GB/s
  - Block SRAM in User Logic
    - 648 KB @ 260 GB/s
  - On-Board SRAM
    - 28 MB @ 9.6 GB/s
  - Microprocessor Memory
    - 8 GB @ 1400 MB/s

Figure 3. Block Diagram of MAP

Microprocessor with SNAP

The Dense Logic Devices (DLDs) used in these products are the dual-processor Intel IA-32 line of microprocessors. These third-party commodity boards are then equipped with the SRC-developed SNAP interface. SNAP allows commodity microprocessor boards to connect to, and share memory with, MAPs and Common Memory nodes that make up the rest of the SRC system.

The SNAP interface is designed to plug directly in to the microprocessors' memory subsystem, instead of its I/O subsystem, allowing SRC systems to sustain significantly higher interconnect bandwidths. SNAP uses separate input and output ports with each port currently sustaining a data payload bandwidth of 1.4GB/sec.

The intelligent DMA controller on SNAP is capable of performing complex DMA prefetch and data access functions, such as data packing, strided access and scatter/gather, to maximize the efficient use of the system interconnect bandwidth. Interconnect efficiencies more than ten times greater than a cache-based microprocessor using the same interconnect are common for these operations.

SNAP either can connect directly to a single MAP or to SRC's Hi-Bar switch for system-wide access to multiple MAPs, microprocessors or Common Memory.

SRC-6 System-Level Architectural Implementation

System-level configurations implement either a cluster of MAPstations or a crossbar switch-based topology. Cluster-based systems, as shown in Figure 4, utilize the microprocessor and DEL processor previously discussed in a direct connected configuration. Although this topology does have a microprocessor-DEL processor affinity, it also has the benefit of using standards-based clustering technology to create very large systems.

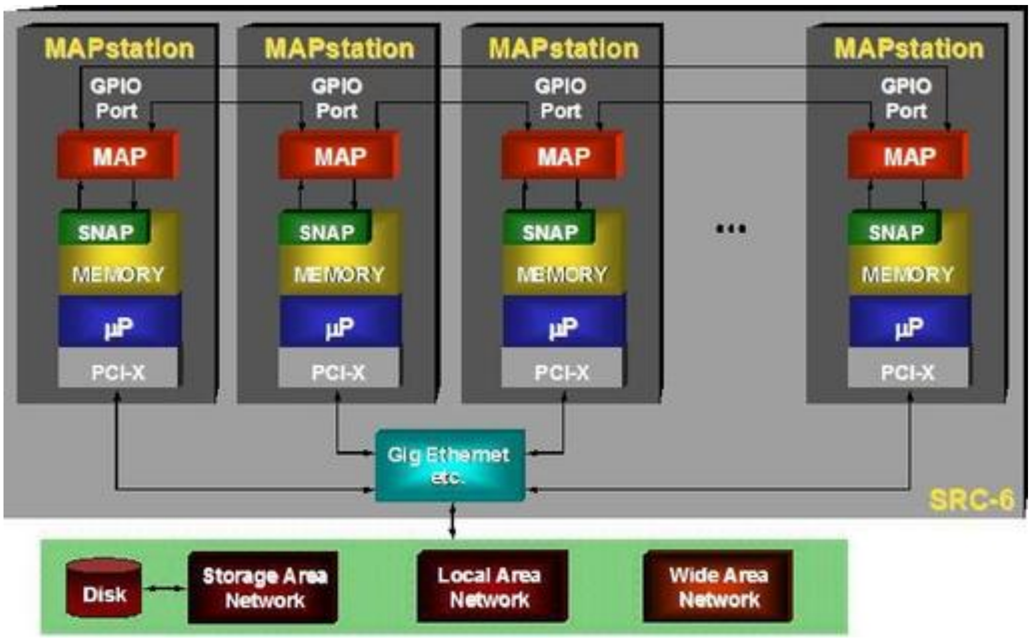


Figure 4. Block Diagram of Clustered SRC-6 System

When more flexibility is desired, Hi-Bar switch-based systems can be employed. Hi-Bar is SRC's proprietary scalable, high-bandwidth, low-latency switch. Each Hi-Bar supports 64-bit addressing and has 16 input and 16 output ports to connect to 16 nodes. Microprocessors, MAPs and Common Memory nodes can all be connected to Hi-Bar in any configuration as shown in Figure 4. Each input or output port sustains a yielded data payload of 1.4GB/sec for an aggregate yielded bisection data bandwidth of 22.4GB/sec per 16 ports. Port-to-port latency is 180ns with Single Error Correction and Double Error Detection (SECDED) implemented on each port.

Hi-Bar switches also can be interconnected in multitier configurations, allowing two tiers to support 256 nodes. Each Hi-Bar switch is housed in a 2U-high, 19-inch wide rackmountable chassis, along with its power supplies and cooling solution, for easy inclusion into rack-based servers.

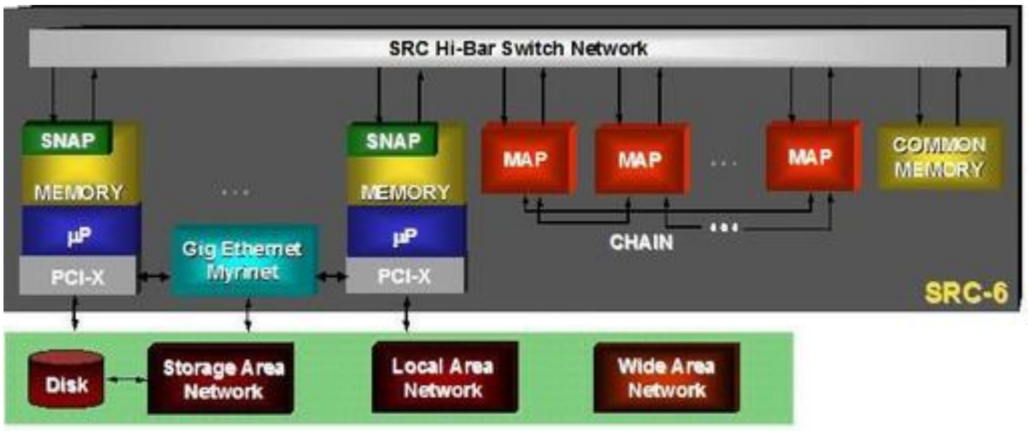


Figure 5. Block Diagram of SRC-6 with Hi-Bar Switch

SRC servers that use the Hi-Bar crossbar switch interconnect can incorporate Common Memory nodes in addition to microprocessors and MAPs. Each of these Common Memory nodes contains an intelligent DMA controller and up to 8GBs of DDR SDRAM. The SRC-6 MAPs, SNAPs and Common Memory node (CM) support 64-bit virtual addressing of all memory in the system, allowing a single flat address space to be used within applications. Each node sustains memory reads and writes with 1.4GB/sec of yielded data payload bandwidth.

The CM's intelligent DMA controller is capable of performing complex DMA functions such as data packing, strided access and scatter/gather to maximize the efficient use of the system interconnect bandwidth. Interconnect efficiencies more than ten times greater than a cache-based microprocessor using the same interconnect are common for these operations.

In addition, SRC Common Memory nodes have dedicated semaphore circuitry that also is accessible by all MAP processors and microprocessors for synchronization.

Programming Model for Reconfigurable Computing

Traditionally, the programming model for RC has been one of hardware design. Given that the tools required for the underlying FPGA technology of RC are all logic design tools from the Electronic Design Automation industry, there really has not been a programming environment recognizable to a software developer. The tools have supported Hardware Definition Languages (HDLs) such as Verilog, VHDL and Schematic Capture.

With the introduction of system-on-a-chip (SOC) technology and the complexity associated with hardware definition of such complexity, high-level languages have begun to be available. Java and C-like languages are becoming more common for use in programming RC chips. This is a significant step forward but continues to require quite a leap by application programmers.

The SRC programming model is the traditional software development model where C and Fortran are used to program the MAP processor, and any language capable of linking with the runtime libraries (written in C) can be compiled and run on the microprocessor portion of the system.

The SRC Carte programming environment was created with the design assumption that application programmers would be writing and porting applications to the RC platform. Therefore, the standard development strategies of design, code in high-level languages (HLLs), compile, debug via standard debugger, edit code, recompile and so on, until correct, are used to develop for the SRC-6 system. Only when the application runs correctly in a microprocessor environment is the application recompiled and targeted for the DEL processor, MAP.

Compiling to hardware in an RC system requires two compilation steps that are quite foreign to programming for an instruction processor. The output of the HLL compiler must be a hardware definition language. In Carte, the output either is Verilog or Electronic Design Interchange Format (EDIF). EDIF files are the hardware definition object files that define the circuits that will be implemented in the RC chips. If Verilog is generated, then that HDL must be synthesized to EDIF using a Verilog compiler such as Synplify from Synplicity.

A final step, place and route, takes the collection of EDIF files and creates the physical layout of the circuits on the RC chip. The output files for this process are a configuration bitstream, which can be loaded into an FPGA to create the hardware representation of the algorithm being programming into the RC processor.

The Carte programming environment performs the compilation from C or Fortran to bitstream for the FPGA without programmer involvement. It further compiles the codes targeted to microprocessors into objects modules. The final step for Carte is the creation of a unified executable that incorporates the microprocessor object modules, the MAP bitstreams, and all of the required runtime libraries into a single Linux executable file. Figures 6 and 7 present the Carte compilation process.

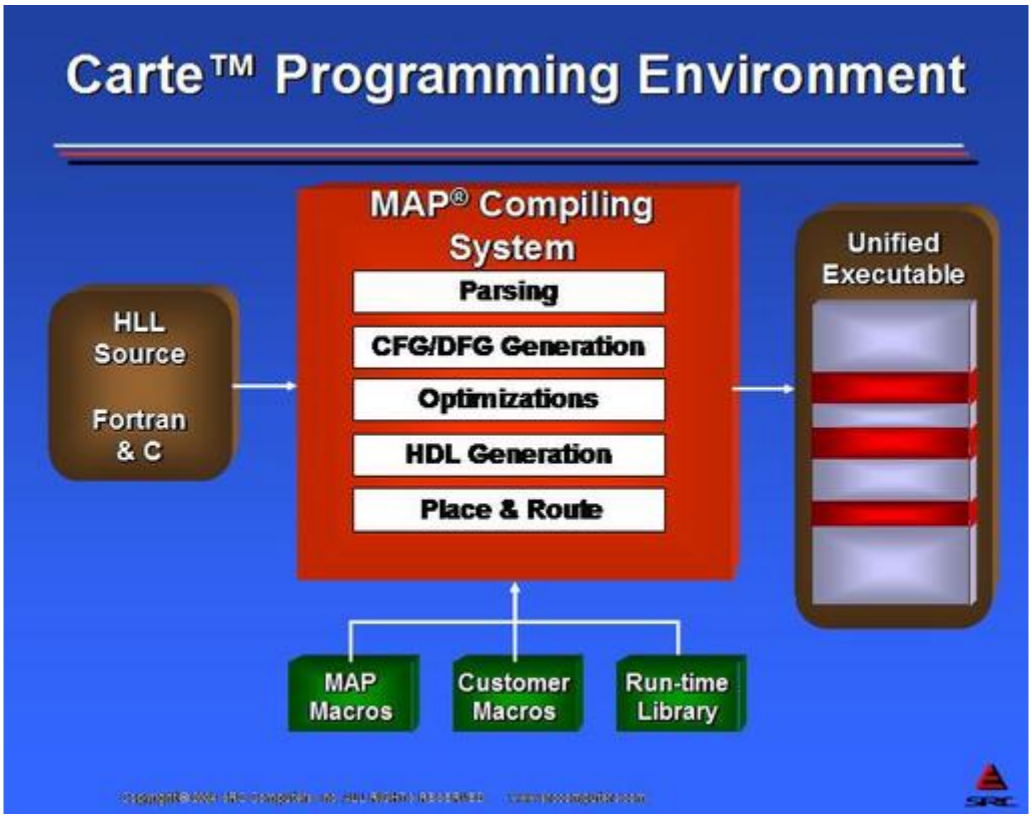


Figure 6. Carte Programming Environment

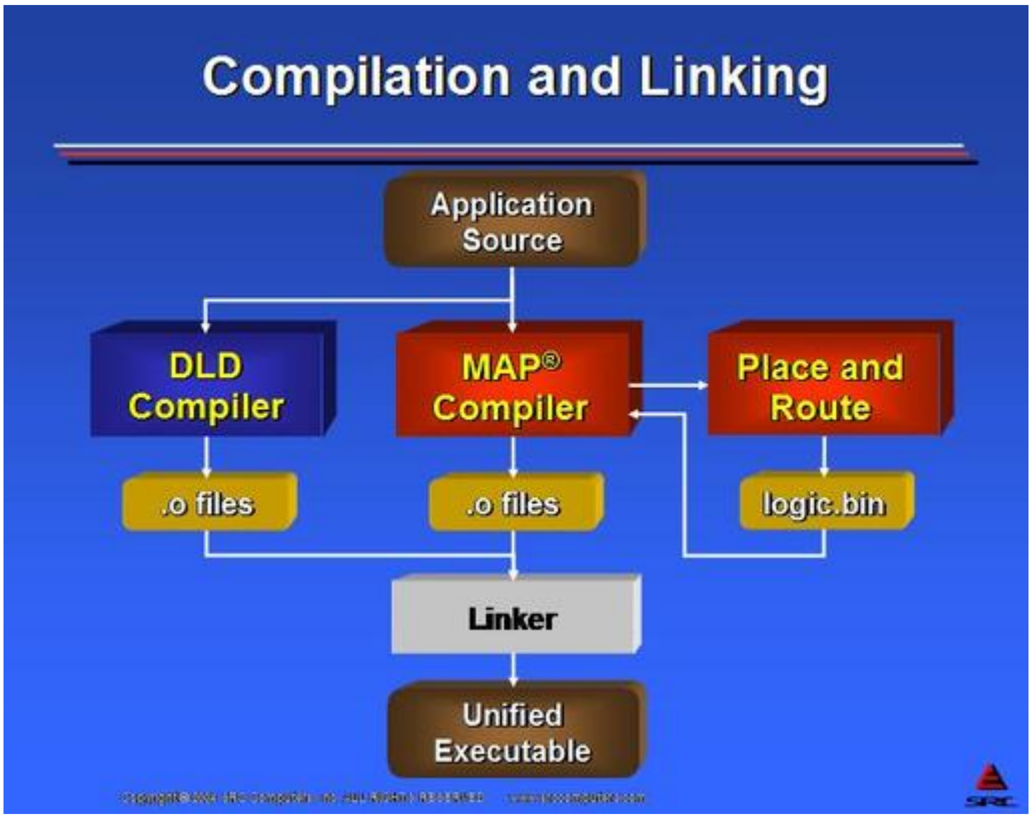


Figure 7. Carte Compilation

Open-Source Hardware Opportunity

Linux has led the way and benefited greatly from the Open Source movement where a large and dedicated group of software developers has created, modified and improved the Linux kernel and OS at a rate, quality and level of innovation that could not be matched by the work of a single commercial organization. Reconfigurable computing has the potential of enabling such innovation and technical advances in hardware design. Much of this article is spent explaining the concept of application programmers writing code and using standard programming methods to create application-specific hardware without requiring knowledge of hardware design. However, in RC the building blocks of the generated hardware created by application programmers is the functional unit. Functional units are basic computational units such as adders, floating-point multipliers or trigonometric functions. Functional units also can be specialty high-performance units, like triple DES functions, or nonstandard precision arithmetic units, such as 24-bit IEEE floating-point operators.

Functional units are created by logic designers. RC compilers, such as SRC's Carte MAP compiler, are capable of allowing customer-supplied functional units to be added to the standard set of operations supported by the compiler. When new and novel functional units are made available to application programmers, an even higher level of performance can be achieved.

It is in the creation and sharing of innovative hardware designs for functional units where an Open Hardware movement could bring substantial advances to computational science. The innovation and productivity seen in the open-source arena could be replicated as Open Hardware.

RC provides a vehicle for many more creative designers to create new and novel hardware that can be used by application developers. Through groups like Opencores.org, functional unit design can be shared and improved upon. The significant advances seen in the computational sciences, due to open-source software, easily could be seen through a movement focused on open hardware as well.

Code Example

To show the performance advantage of a DEL processor, a string-matching example is presented. The code for these examples is available on the *Linux Journal* FTP site—see the on-line Resources. This example came from the Web site of Christian Charras and Thierry Lecroq, referenced by NIST Dictionary of Algorithms and Data Structures. For comparison, the Brute Force and Boyer-Moore string-matching algorithms are implemented for the 2.8GHz Intel Xeon via Intel's C++ 8.0 compiler for Linux. The Brute Force algorithm is implemented for SRC's system using the Carte 1.8 Programming Environment. The Brute Force algorithm is a straightforward character-by-character comparison between a pattern and a text string. The Boyer-Moore is considered the most efficient string-matching algorithm. The example takes a randomly generated 20MB text string and searches for six and ten randomly generated patterns. Compilations are done with a -O3 optimization setting, and performance comparisons are shown in Table 1. Adding four additional search patterns to the test increases the microprocessor times but has no impact on the MAP execution times due to the pipelined logic. Though the Xeon runs at 2.8GHz, and the MAP runs at 100MHz, the parallelism seen in DEL can achieve a 99× performance advantage in MAP. This example required 60% of one FPGA in the MAP. A two-chip compile would deliver over 200× performance.

Table 1. String-Matching Performance

Implementation	Text Size	Patterns	Search Time	Speedup
Brute Force (Xeon)	20MB	6	0.827 sec	1.00×
Boyer-Moore (Xeon)	20MB	6	0.597 sec	1.38×
Brute Force (MAP)	20MB	6	0.0143 sec	57.75×
Brute Force (Xeon)	20MB	10	1.398 sec	1.00×

Implementation	Text Size	Patterns	Search Time	Speedup
Boyer-Moore (Xeon)	20MB	10	1.051	1.33×
Brute Force (MAP)	20MB	10	0.0141 sec	98.81×

To demonstrate the impact of adding additional computation into a pipelined loop, and the ability to introduce custom functional units, a second performance comparison is done in which a DES-encrypted string is passed to the search routine. The string must be decrypted prior to searching. In the case of the MAP implementation, a DES pipelined functional unit is introduced. The Verilog definition was obtained from Opencores.org and introduced into the search loop. Because the loop is pipelined, it continues to deliver a set of results per clock cycle. Therefore, the elapsed time for the 20MB text search, including a DES decryption, is unchanged from the search alone. This leads to a very dramatic 232× speedup over the microprocessor implementation. The ten-pattern MAP example uses only 74% of an FPGA, so a two-chip compile for the MAP would yield 460×.

Table 2. Performance for Searching an Encrypted String

Implementation	Text Size	Patterns	Search Time	Speedup
DES-Brute Force (Xeon)	20MB	6	2.77 sec	1.00×
DES-Boyer-Moor (Xeon)	20MB	6	2.63 sec	1.05×
DES- Brute Force (MAP)	20MB	6	0.0143 sec	193.09×
DES-Brute Force (Xeon)	20MB	10	3.31 sec	1.00×
DES-Boyer-Moor (Xeon)	20MB	10	3.11 sec	1.06×
DES- Brute Force (MAP)	20MB	10	0.0143 sec	231.76×

In the case of DES implemented on the Xeons, the code is an optimized code by Stuart Levy at Minnesota Supercomputer Center.

Conclusion

This article has explained reconfigurable computing, shown examples of the methods and the results that can be achieved. Significant performance gains can be demonstrated. In the present, RC has much to contribute to computational science, but the future holds advances well beyond the Moore's Law gains experienced in the world of microprocessors. RC is accessible to today's programmers using a familiar programming model and provides the framework within which a larger population of hardware designers can have an impact on high-performance computation through open-source creativity and productivity.

RC has been a long time in coming, but the enabling software and hardware technology has set the stage for RC to become part of every computer, from embedded processor to Peta-Scale supercomputer.

Resources for this article: /article/7867.

Dan Poznanovic (poz@srccomp.com) is VP Software Development at SRC Computers, Inc., and has been involved in the high-performance computing world since initially joining Cray Research, Inc., in 1987.

Finding Stubborn Bugs with Meaningful Debug Info

John Goerzen

Issue #129, January 2005

When a user reports a bug you can't duplicate, let your application help you find the problem. Add logging now and be a debugging master after the software gets deployed.

Bug tracking is often one of the most difficult processes in software development. Users may have situations different from developers, and bugs that are a big problem for users may not even be visible on developers' machines. Sometimes bugs can come and go, or networked programs may encounter bugs only when talking to specific servers or clients. In this article, I discuss techniques software developers can employ to help track down bugs more easily.

First, I discuss two ways to make it easier to receive and manage bugs, and then I show how to make your programs generate more useful debugging output. Then, I talk about tracking down troublesome bugs. Finally, I cover some practices that can help prevent bugs in the first place. Many of the techniques described in this article are employed in OfflineIMAP (see “Fast Convenient Mail for Travel: OfflineIMAP”, *LJ*, March 2004).

Tracking Bugs

Before examining how to make better bug reports possible, a critical first step is making sure you can deal with the bug reports you receive. For some small projects, simply publishing an e-mail address is sufficient. However, most projects need something more. Developers often get busy and forget about things. Bugs may be complicated to solve, requiring input from several people, or there simply may be a lot of bug reports.

A bug-tracking system (BTS) is a great way to help ensure that bugs are not forgotten. Most BTS implementations provide a way to track correspondence, handle attached files and delegate responsibility to particular people. Some also support categorization based on things such as severity, user environment and specific components.

If your project is hosted at a project hosting site such as SourceForge or Savannah, you already have a BTS available for your use. You should use it and encourage your users to submit bugs through that interface rather than to a mailing list.

If you need more flexibility, you can find BTS programs for Linux. Some of the most popular free software BTS programs are:

- Bugzilla, the BTS used by the Mozilla Project, is a flexible system primarily used through its Web interface.
- Request Tracker can be used as both a bug-tracking system and a support-tracking system. It features both Web and e-mail interfaces, though some administrative functions can occur only through the Web interface.
- Jitterbug is the BTS used by the Samba Project. It is similar in concept to Bugzilla but is more lightweight.
- Debbugs is the BTS used by the Debian Project. Debbugs has a Web interface, though it is read-only; all manipulations occur by e-mail. Debbugs is best suited for large projects with clearly identifiable components and responsibility for those components.

I personally prefer Request Tracker, because it seems to have a nice blend of features for a BTS. Your own requirements may differ.

Make It Easy to Submit Bugs

Sometimes I find a nasty bug in a program and want to report it. But to do so, I have to fill out a detailed questionnaire and perhaps divulge information I'd rather not. It should be easy for people to submit bugs and the information needed to track them down. If you take submissions on the Web, make the process simple. Don't require too much information, and accept submissions even if people don't know some information. Don't expect users to know anything about the different components of the project or which developers are responsible for a given problem.

Logging

When tracking down problems, you often want to know what state the program is in. Other times, you may want to know what actions were carried out prior to triggering the bug. Because users of programs don't necessarily have expertise with your code and a debugger, logging often is called for. Logging simply means writing out a record of the actions carried out. Simple programs might merely print out information, but usually you'll want something a little more capable.

Non-interactive programs, such as network servers, do not have a screen on which to display information. These programs often maintain a log file or use the syslog facility built in to Linux and UNIX systems.

Interactive programs may display information on-screen or also may generate a file. Having a log file available can make bug reporting easier, because the user simply can attach it to a bug report.

Sometimes, you might need quite a bit of data to figure out what's going on with a specific problem. However, all this data may be overkill for a normal session—it could flow right off a user's screen or fill up a hard disk. Therefore, many programs have a notion of a log level. The user can set, at runtime, how much information should be logged. Some programs even may have log categories, where users can configure which types of information are logged. OfflineIMAP uses this approach. For troublesome problems, users can turn on a communications log, which logs all data sent to or received from the IMAP server.

Python 2.3 introduced a useful module called logging. The logging module provides a uniform interface to several different ways of logging messages. Its supported logging methods include writing messages to files, network services, syslog, e-mailing messages and several others. The following is a simple example that illustrates use of the logging module:

```
#!/usr/bin/env python
import logging, sys

# Create the logger object
l = logging.getLogger('testlog')

# Create a handler and assign it to the object
handler = logging.StreamHandler(sys.stderr)
l.addHandler(handler)

# Levels are DEBUG, INFO, WARNING, ERROR, CRITICAL.
# Set the default level here. Any log messages
# beneath that level are dropped.
l.setLevel(logging.INFO)

# Try it out.

l.debug("Debug message -- system initialized.")
l.info("Here's some info.  I've just debugged.")
l.warning("I don't have many messages left.")
l.error("Only one more message to go.")
l.critical("Nothing else to do!")
```

This program begins by initializing the logger. It uses the StreamHandler to write logged text to standard error. It also sets the log level to INFO. Then it logs five messages. When you run this program, you see only the last four. The debug message was filtered out by setting the level to INFO. Many programs have a configuration or command-line option to set the level at runtime. You can use different logging methods simply by adding a different handler to your Logger object. The Python documentation has a reference for all the available handlers.

Check Input

Make sure the input you are receiving is valid. For instance, if you are expecting something on the command line, check to make sure you have the appropriate number of arguments before trying to use them (or trap the resulting exception). This gives users a better error message. Here's a sample Python program that demonstrates this:

```
#!/usr/bin/env python
import sys

try:
    print "You supplied: %s" % sys.argv[1]
except IndexError:
    print "You forgot an argument."
```

Handle Exceptions

Several programming languages, such as Java, Python and OCaml, include support for exceptions. With exceptions, you can catch errors at the place you choose, rather than having to check and handle errors with each call that may produce a problem. Sometimes, it might be correct to let exceptions go unhandled, but usually that is not the case. Exceptions should be caught and handled. Although it may be appropriate to terminate the program if you can't open the file a user asks for, it is still better to do so with an error message giving the filename and problem rather than let the user receive an ugly exception message.

Capture Exceptions

For exceptions that really are fatal to your program, you still may want to capture them. This would allow you, for instance, to log them to a file or display the exception in a pop-up box in the GUI application. This makes it easier for users to send the stack trace back to you. You also can use a generic exception catcher to perform other activities, perhaps output contents of various buffers to help you figure out what was going on at the time.

The following is an example that logs any exceptions along with some information about the program currently running. It then re-raises the exception and exits:

```
#!/usr/bin/env python
import logging, sys, StringIO, traceback, os
```

```
l = logging.getLogger('testlog')

handler = logging.StreamHandler(sys.stderr)
l.addHandler(handler)
formatter = logging.Formatter("LOG: %(message)s")
handler.setFormatter(formatter)

l.setLevel(logging.INFO)

def logexception():
    sbuf = StringIO.StringIO()
    traceback.print_exc(file = sbuf)
    excval = sbuf.getvalue()
    l.critical(" *** Exception Detected ***")
    l.critical("Current PID: %d" % os.getpid())
    l.critical("Program name: %s" % sys.argv[0])
    l.critical("Command line: %s" % \
               str(sys.argv[1:]))
    for line in excval.split("\n"):
        l.critical(line)

def main():
    print "Hello, I'm running."
    raise RuntimeError("Oops! I've had a problem!")

try:
    main()
except:
    logexception()
    raise
```

When you run this program, you should see something like this on your screen:

```
Hello, I'm running.
LOG:  *** Exception Detected ***
LOG: Current PID: 28441
LOG: Program name: /tmp/logerror.py
LOG: Command line: []
LOG: Traceback (most recent call last):
LOG:   File "/tmp/logerror.py", line 30, in ?
LOG:     main()
LOG:   File "/tmp/logerror.py", line 27, in main
LOG:     raise RuntimeError("Oops! I've had a problem!")
LOG: RuntimeError: Oops! I've had a problem!
LOG:
```

Here, the exception handler found the exception, grabbed the information about it and was able to log it. You also can see the traceback a second time. The raise statement at the end of the program causes the exception to be raised and handled in the normal fashion also. This means it aborts your program with a traceback. Depending on your requirements, you may opt to use sys.exit() to terminate instead.

Finding Reported Bugs

Now that you have some ways to help users submit good bug reports, let's look at ways to use those bug reports to track down problems. Armed with a log and perhaps traceback information, here are some questions to ask yourself:

- Can I duplicate the bug in my environment? If you can duplicate the problem on your own machine, you're a long way toward being able to resolve it easily. Use a debugger or other tool to track it down now that you can trigger it at will.

- Was the input and output what I expected? Perhaps the user supplied a value you didn't contemplate when you wrote the program. Or, perhaps a network client or server treats a protocol slightly differently from what you expected. Maybe the input or output is itself malformed, and the bug isn't even in your program. A debug log showing all I/O can be very helpful here.
- Was the program flow as expected? If your log calls to various functions or methods, you should be able to trace the flow of execution in a program. Perhaps certain conditions cause vital code to be skipped, leading to trouble later on.
- Where was the last point of correct execution? This may have been right before the error, or perhaps incorrect data was passed around for some time prior to a crash. Pinpointing the most recent time in the program's history where it was functioning normally can help track down the precise place where things went awry.
- If a traceback is on-hand, does the stack look normal? Check to make sure the function calls are as expected and that the data passed to them looks legitimate.

Preventing Bugs

All the techniques I've described in this article are useful, but they shouldn't be deployed in a vacuum. It's also important to adopt practices that help reduce the likelihood of bugs occurring. Here are some to consider:

- Adopt unit testing. Java, Python, OCaml, Perl and C all have unit testing frameworks available. Use them and exercise as many code paths as possible. This is especially important for a language such as Python where certain executions of a program may not even parse all of your code. It also can be important for Java; for instance, runtime exceptions can occur due to improper casting to or from Object.
- Avoid globals. Avoiding global (or class-global) variables helps isolate problems and helps prevent synchronization issues in multithreaded programs. Global variables can be the source of unexpected side effects in function calls, which can be hard to track down.
- Use the right tool for the job. Languages each have their own strengths and weaknesses, and no single language is the best tool for every task. For instance, although Perl makes it easy to parse delimited text files with regular expressions, OCaml provides tools specifically designed for writing a compiler. Problems that are expressed easily in one language may become much more difficult to express in another.

- But, don't use too many different tools. Most projects benefit from a standardized toolset. Pick a language and libraries that are most useful for the project at hand, and don't introduce new ones unless there's a compelling reason to do so.
- Use string and memory management tools. Many languages, including Java, Python, OCaml, Perl and Ruby, provide transparent memory management. You do not need to allocate and deallocate memory. You also do not need to concern yourself explicitly with end-of-string markers and string size limitations. Both of these are common problems with C programs that lead to runtime bugs or security holes. If you must use C, consider a garbage collection or memory pool library.
- Make it work first, then optimize. In many cases, it's better to develop working code first, then optimize it later. Many people optimize first, which does work in some cases. However, simple, bug-free code is usually more important than code that is as fast as it can possibly be.
- Write clean code. Split out code into functions. Write comments. Document what each function does and its effect on the environment.

Case Study: a Bug in OfflineIMAP

OfflineIMAP is a program that talks to IMAP servers and synchronizes an IMAP folder tree with a local tree. Many IMAP servers exist, and they don't all work exactly the same. Through its two-year history, OfflineIMAP has gained more and more of the debugging techniques discussed in this article. Problems that users encounter often are unreproducible with my particular setup, so detailed logging is a must. Some IMAP servers are buggy themselves, so the first question that has to be resolved with many reports is: is this even a bug in OfflineIMAP? In a surprising number of cases, the answer is no. OfflineIMAP uses certain IMAP features that most other IMAP clients do not, and those features tend to be poorly tested in some servers.

I'd like to walk you through one particularly stubborn OfflineIMAP bug I've been working on. About a year ago, someone reported a bug in OfflineIMAP using the Debian bug-tracking system. Unfortunately, I couldn't duplicate the problem, and the original submitter didn't have logging turned on when the problem happened. He also wasn't able to obtain debugging information. Given the information he did have, which included an error message, I was able to gather some information following the steps outlined earlier in this article. I didn't have information on the input and output, but the program flow and stack both looked normal. In the end, I was able to determine where the program crashed but not why, so the bug sat there for a while. Things were made more difficult because the bug was intermittent—sometimes the program would work fine, and occasionally it would crash.

Later, a second person experienced the same problem. He noticed the existing bug report on Debian and sent in his information. OfflineIMAP automatically tries to print out parts of a debug log if a fatal error occurs, and he was able to capture this output. This OfflineIMAP feature has proven valuable in the past, because it is not always possible to reproduce the situation leading to a problem.

In this case, the information helped. I was now able to see what OfflineIMAP was doing immediately prior to the bug occurring. But, it still was not enough information to discover the exact problem—everything still looked normal. However, the bug was intermittent, and he couldn't capture any additional information.

Eventually, a third person experienced the same problem. Again, he had some information but not quite enough to figure it out. Something else needed to happen, so I made the logging in the particular section of code more detailed. Hopefully, with the additional logging, the next time the problem is encountered, I'll have enough information to track it down.

Several things played an important role in this process. First, OfflineIMAP always generates a usable stack trace when a fatal error occurs. Even the least-detailed report showed exactly where the program was when it crashed. Secondly, error logs are helpful, but less so if people can't reproduce a particular bug easily. Printing out debugging information when a program crashes or malfunctions can be a useful way to help combat that problem.

Also, the bug-tracking system played an important role in tracking down the problem. Because Debian bug reports are public, the three submitters involved were able to identify an outstanding bug report and add their information to it. This helped everyone to manage the information related to the particular issue and also provided a place to start for the people who encountered the problem for the first time.

Conclusion

There are many ways to help your users report bugs in your program and track them down, but they should not be employed in a vacuum. Don't forget to make it easy to report and track bugs, and to write clean code in the first place. Finally, remember that none of these steps are a magic bullet. Taken together, they can simplify your bug-tracking process and help find many problems, but they won't necessarily solve everything.

Resources for this article: /article/7747.

John Goerzen is a longtime Linux programmer and the author of *Foundations of Python Network Programming*. He also serves as President of Software in the Public Interest, Inc. John welcomes your comments at jgoerzen@complete.org.

Archive Index Issue Table of Contents

Advanced search

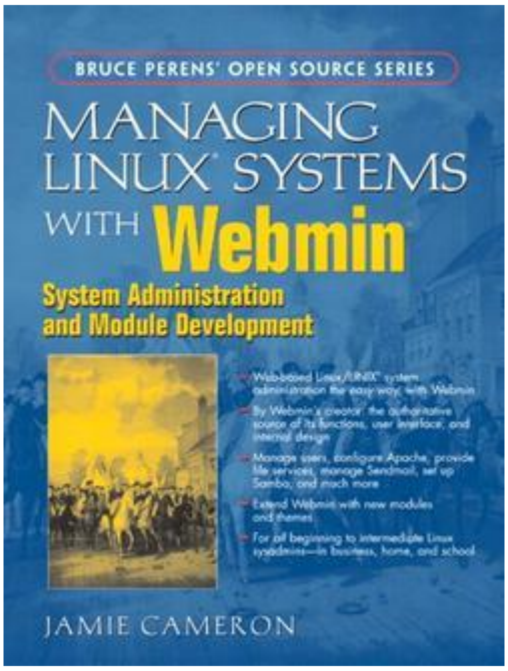
Using Webmin—By the Book

Frank Conley

Issue #129, January 2005

Webmin certainly is a convenient tool and a time saver, but it is not a substitute for understanding the inner workings of the OS.

Managing Linux Systems with Webmin: System Administration and Module Development **by Jamie Cameron**



Prentice Hall PTR, 2003

ISBN: 0131408828

$44.99 US

When it comes to Linux and UNIX, I'm not into convenience. Doing it all by hand or with a script, if the task is repetitive, always has been a better choice. Traditionally, I've been leery of GUIs that automate tasks. If you ever have experienced System Administration Manager (SAM) under HP-UX, you know what I mean. In general it works okay, but it has bitten back enough times to make me consider alternatives that offer a greater degree of reliability and consistency. My experience with open source and Linux has been mostly positive so far, and this fact has made me more receptive to trying things I otherwise might not have bothered with, such as Webmin.

Before installing Webmin, I read a couple of chapters in *Managing Linux Systems with Webmin* to get a feel for it, and then proceeded to check out the webmin.com Web site. The Web site is not merely the obvious place to start; it also is well documented and provided me everything I needed to know in getting started. I followed all the steps from installation to initial login. It seemed too easy.

Once I logged in I was presented with a number of icons that represent the management subsystems (Figure 1).

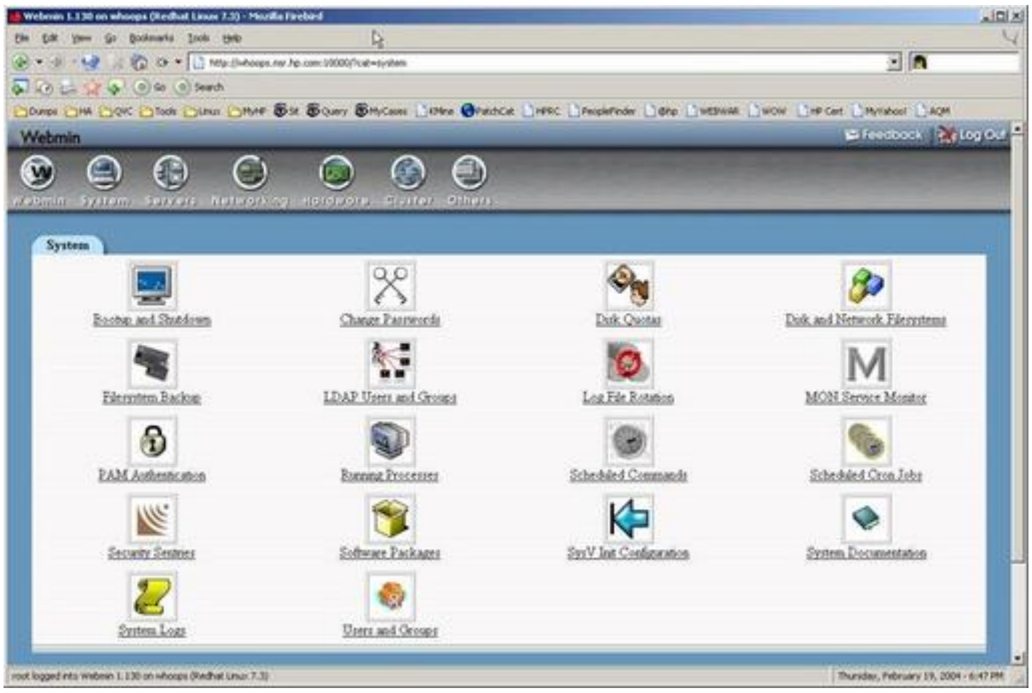
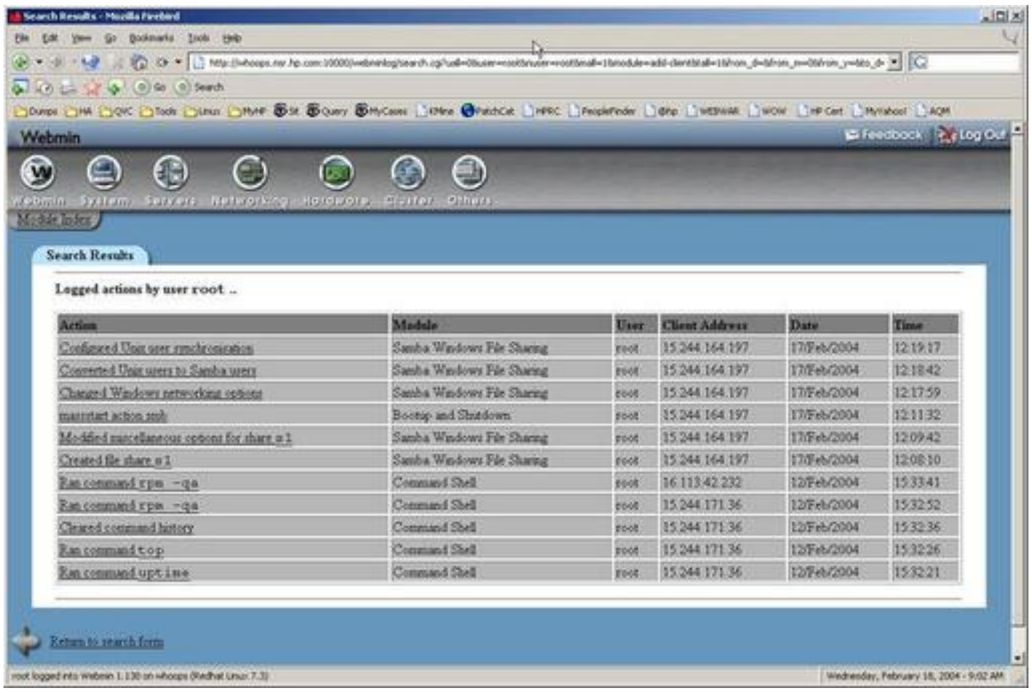


Figure 1. The System module contains a rich sysadmin task environment.

Eighty-plus modules are present in Webmin, and they vary in complexity. As with anything, it is a good idea to start with baby steps and then move on, but the first thing one should do upon entering the new environment is to secure it.

I was able to log in as root after the initial installation of Webmin, and this generates a certain amount of paranoia on my part. If you have put Webmin on a server, you should secure it, and this is where the book may come in handy. Chapter 3 walks the reader through a few steps to secure the box, but one is better advised to become familiar with Webmin configuration and access, topics covered in Chapters 51 and 52.

I've opted not to secure my machine as best I can, but that doesn't mean I don't want to know who was logged on when and why. By default, root's actions are logged, so if you have several folks using root, you can look at what they did. In Figure 2, one can see from the IP addresses that the root user was logged in from different clients.



Logged actions by user root ..

Action	Module	User	Client Address	Date	Time
Configured Unix user synchronization	Samba Windows File Sharing	root	15.244.164.197	17/Feb/2004	12:19:17
Converted Unix users to Samba users	Samba Windows File Sharing	root	15.244.164.197	17/Feb/2004	12:18:42
Changed Windows networking options	Samba Windows File Sharing	root	15.244.164.197	17/Feb/2004	12:17:59
massrtart action smb	Bootup and Shutdown	root	15.244.164.197	17/Feb/2004	12:11:32
Modified miscellaneous options for share a1	Samba Windows File Sharing	root	15.244.164.197	17/Feb/2004	12:09:42
Created file share a1	Samba Windows File Sharing	root	15.244.164.197	17/Feb/2004	12:08:10
Ran command rpm -qa	Command Shell	root	16.113.42.232	12/Feb/2004	15:33:41
Ran command rpm -qa	Command Shell	root	15.244.171.36	12/Feb/2004	15:32:52
Cleared command history	Command Shell	root	15.244.171.36	12/Feb/2004	15:32:36
Ran command top	Command Shell	root	15.244.171.36	12/Feb/2004	15:32:26
Ran command uptime	Command Shell	root	15.244.171.36	12/Feb/2004	15:32:21

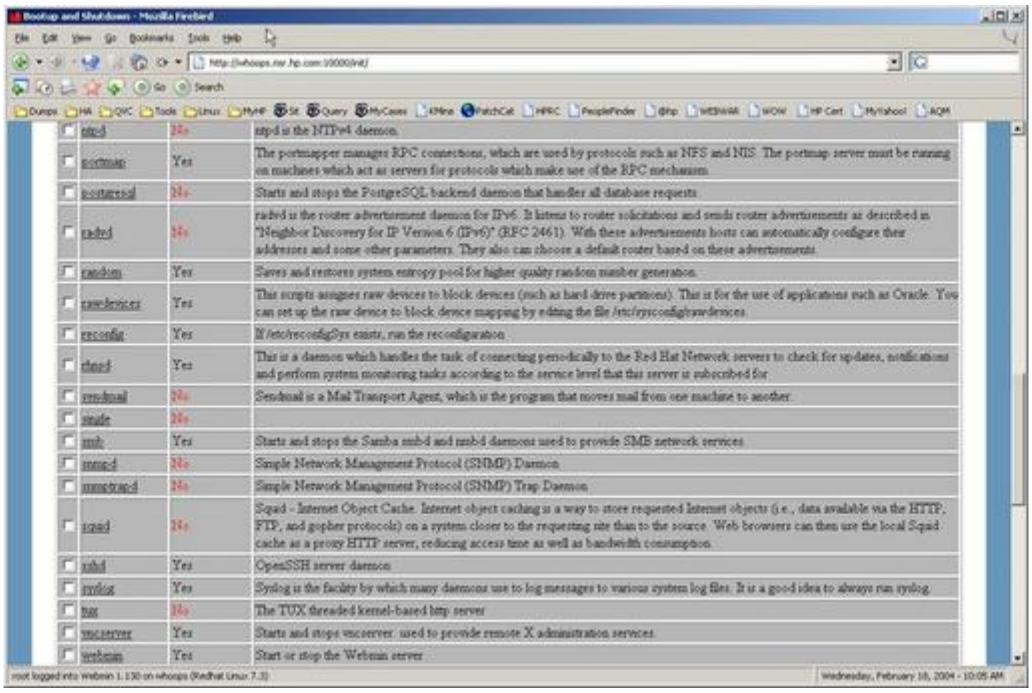
root logged into Webmin 1.130 on whoops (Redhat Linux 7.3)

Wednesday, February 18, 2004 - 9:02 AM

Figure 2. The logs clearly show what root has been doing and from where.

Exploring the Modules

Modules is a topic larger than what I can cover in the scope of this article, but I mention a few of the modules so we can sample the span of Webmin. The first of these modules after the Webmin (configuration) one is the System module. The first of the subsystems to look at is Bootup and Shutdown. The page should display Actions, Start At Boot and Description columns. This enables you to quickly disable services. For example, I typically disable sendmail by starting on this test box, unless I have a specific need for messaging. You should notice a red No in the Start At Boot column (Figure 3).



ntpd	No	ntpd is the NTPv4 daemon.
portmap	Yes	The portmapper manages RPC connections, which are used by protocols such as NFS and NIS. The portmap server must be running on machines which act as servers for protocols which make use of the RPC mechanism.
postgresql	No	Starts and stops the PostgreSQL backend daemon that handles all database requests.
radvd	No	radvd is the router advertisement daemon for IPv6. It listens to router solicitations and sends router advertisements as described in "Neighbor Discovery for IP Version 6 (IPv6)" (RFC 2461). With these advertisements hosts can automatically configure their addresses and some other parameters. They also can choose a default router based on these advertisements.
random	Yes	Saves and restores system entropy pool for higher quality random number generation.
rawdevices	Yes	This scripts assignes raw devices to block devices (such as hard drive partitions). This is for the use of applications such as Oracle. You can set up the raw device to block device mapping by editing the file /etc/sysconfig/rawdevices.
reconfig	Yes	If /etc/reconfigSys exists, run the reconfiguration.
rhnsd	Yes	This is a daemon which handles the task of connecting periodically to the Red Hat Network servers to check for updates, notifications and perform system monitoring tasks according to the service level that this server is subscribed for.
sendmail	No	Sendmail is a Mail Transport Agent, which is the program that moves mail from one machine to another.
single	No	
smb	Yes	Starts and stops the Samba smbd and nmbd daemons used to provide SMB network services.
snmpd	No	Simple Network Management Protocol (SNMP) Daemon
snmptrapd	No	Simple Network Management Protocol (SNMP) Trap Daemon
squid	No	Squid - Internet Object Cache. Internet object caching is a way to store requested Internet objects (i.e., data available via the HTTP, FTP, and gopher protocols) on a system closer to the requesting site than to the source. Web browsers can then use the local Squid cache as a proxy HTTP server, reducing access time as well as bandwidth consumption.
sshd	Yes	OpenSSH server daemon
syslog	Yes	Syslog is the facility by which many daemons use to log messages to various system log files. It is a good idea to always run syslog.
tux	No	The TUX threaded kernel-based http server
vncserver	Yes	Starts and stops vncserver. used to provide remote X administration services.
webmin	Yes	Start or stop the Webmin server

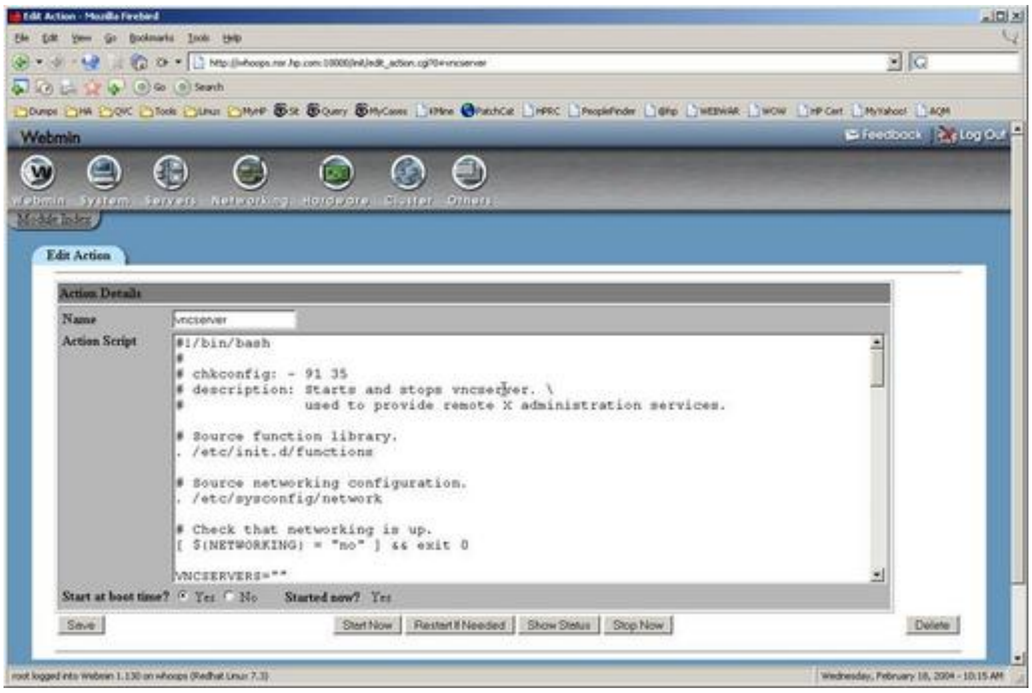
root logged into Webmin 1.130 on whoops (Redhat Linux 7.3)

Wednesday, February 18, 2004 - 10:05 AM

Figure 3. The list of services to start is long and varied.

Farther on down is vncserver, which is enabled to run at boot time. If I were to change this, I would check the box at left and then click on vncserver.

With that, a new page would load that displays, in editable form, the vncserver startup script (Figure 4). This page allows for stopping, starting, restarting, setting boot time start and editing the file. Once you save or delete the action, you return to the Bootup and Shutdown page.



Action Details

Name: vncserver

Action Script:

```
#!/bin/bash
#
# chkconfig: - 91 35
# description: Starts and stops vncserver. \
#              used to provide remote X administration services.

# Source function library.
. /etc/init.d/functions

# Source networking configuration.
. /etc/sysconfig/network

# Check that networking is up.
[ ${NETWORKING} = "no" ] && exit 0

VNCSERVERS=""
```

Start at boot time? [✓ Yes] No   Started now? Yes

root logged into Webmin 1.130 on whoops (Redhat Linux 7.3)

Wednesday, February 18, 2004 - 10:15 AM

Figure 4. Editing and testing services from within Webmin is straightforward and easy.

Jumping to another module, Hardware, and then to the Grub Boot Loader subsystem, I easily am able to set the boot kernel of choice and the delay I want. This is an easy task, one that could have been done equally as easily from the command line, but the GUI conveys more information in an intuitive manner than does text. If the GUI is reliable, it is desirable as a tool.

From Simple to Complex

Many of the subsystems are fairly intuitive, and it will be easy for the user to grasp and use them immediately, but some are complicated. The book comes in handy in navigating modules such as Apache, but it assumes that you already understand the applications.

A fair question to ask is whether the benefits of Webmin are worth the learning curve to use it. This would depend on the support environment in which you are working. Webmin certainly is a convenient tool and a time-saver, but it is not a substitute for understanding the inner workings of the OS. Will I continue to use it and learn more about it? Yes.

The book initially reminded me of a college textbook when I first picked it up. At 700+ pages there is a lot of information here and it is geared toward a serious user. It exhaustively covers the modules and walks you through what you need to know. I would recommend it to anyone who plans to use Webmin as a management tool.

Frank Conley is a UNIX/Linux Support Engineer and former system administrator. He has been toying with Linux since 1995, and long ago had the good sense to not see Linux as a toy, but as a very useful tool that he enjoys working with.

Archive Index Issue Table of Contents

Advanced search

Counting with uniq

Brian Tanaka

Issue #129, January 2005

Shell experts make the best of simple combinations of standard utilities. Learn one of the most common examples of using two common commands together.

One of the truly great qualities of UNIX-like operating systems is their ability to combine multiple commands. By combining commands, you can perform a wide array of tasks, limited only by your cleverness and imagination.

Although the number of potential command combinations is huge, my experience has shown that certain combinations come in handy more often than others. One I turn to frequently is combining the sort and uniq commands to count occurrences of arbitrary strings in a file. This is a great trick for new Linux users and one you never will regret adding to your skill set.

A Simple Example

Let's look at a simple example first to highlight the fundamental concepts. Given a file called fruit with the following contents:

```
apples
oranges
apples
```

you can discover how many times each word appears, as follows:

```
% sort fruit | uniq -c
  1 oranges
  2 apples
```

What's happening here? First, `sort fruit` sorts the file. The result ordinarily would go to the standard output (in this case, your terminal), but note the | (pipe) that follows. That pipe directs the output of `sort fruit` to the input of the next command, `uniq -c`, which prints each line preceded by the number of times it occurred.

A More-Advanced Example

It's not obvious from the simple example why this is so powerful. However, it becomes clearer when the file at hand is, for instance, an Apache Web server access log with hundreds of thousands of lines. The access log contains a wealth of valuable information. By using sort and uniq, you can do a surprising amount of simple data analysis on the fly from the command line. Imagine a coworker desperately needs to know the ten IP addresses that requested a PHP script called foo.php most often in January. Moments later, you have the information she needs. How did you derive this information so fast? Let's look at the solution step by step.

For the sake of this exercise your server is logging in the following format:

```
192.168.1.100 - - [31/Jan/2004:23:25:54 -0800] "GET /index.php HTTP/1.1" 200 7741
```

The log contains data from many months, not only January 2004, so the first order of business is to use grep to limit our data set:

```
% grep Jan/2004 access.log
```

We then look for foo.php in the output:

```
% grep Jan/2004 access.log | grep foo.php
```

If we are to count occurrences of IP addresses, we better limit our output to only that one field, like so:

```
% grep Jan/2004 access.log | grep foo.php | awk '{ print $1 }'
```

A discussion of awk is beyond the scope of this article. For now, you need to understand only that `awk '{ print $1 }'` prints the first string before any whitespace on each line. In this case, it's the IP address.

Now, at last, we can apply sort and uniq. Here's the final command pipeline:

```
% grep Jan/2004 access.log | grep foo.php | \
awk '{ print $1 }' |  sort -n | uniq -c | \
sort -rn | head
```

The backslash (\) indicates the command is continued on the next line. You can type the command as one long line without the backslashes or use them to break up a long pipeline into multiple lines on the screen.

You may have noticed that, unlike in our simple example, the first sort is a numeric sort (`sort -n`). This is appropriate because we are, after all, dealing with numbers.

The other difference is the inclusion of `| sort -rn | head`. The `sort -rn` command sorts the output of `uniq -c` in reverse numeric order. The `head` command prints only the first ten lines of output. The first ten lines are perfect for the task at hand because we want only the top ten:

```
  43 12.175.0.35
  16 216.88.158.142
  12 66.77.73.85
   9 66.127.251.42
   7 66.196.72.78
   7 66.196.72.28
   7 66.196.72.10
   7 66.147.154.3
   7 192.168.1.1
   6 66.196.72.64
```

You can change the functionality of this pipeline by making changes to any of the component commands. For instance, if you wanted to print the bottom ten instead of the top ten, you need change only `head` to `tail`.

Conclusion

Piping data through sort and uniq is exceedingly handy, and I hope reading about it whets your appetite for learning more about pipelines. For more information about any of the commands used in these examples, refer to the corresponding man pages.

Brian Tanaka has been a UNIX system administrator since 1994 and has worked for companies such as The Well, SGI, Intuit and RealNetworks. He can be reached at btanaka@well.com.

Archive Index Issue Table of Contents

Advanced search

A Memory-Efficient Doubly Linked List

Prokash Sinha

Issue #129, January 2005

Save precious bytes with a new twist on a standard data type.

In the quest to make small devices cost effective, manufacturers often need to think about reducing the memory size. One option is to find alternative implementations of the abstract data types (ADTs) we are used to for our day-to-day implementations. One such ADT is a doubly linked list structure.

In this article, I present a conventional implementation and an alternative implementation of the doubly linked list ADT, with insertion, traversal and deletion operations. I also provide the time and memory measurements of each to compare the pros and cons. The alternative implementation is based on pointer distance, so I call it the pointer distance implementation for this discussion. Each node would carry only one pointer field to traverse the list back and forth. In a conventional implementation, we need to keep a forward pointer to the next item on the list and a backward pointer to the previous item. The overhead is 66% for a conventional node and 50% for the pointer distance implementation. If we use multidimensional doubly linked lists, such as a dynamic grid, the savings would be even greater.

A detailed discussion of the conventional implementation of doubly linked lists is not offered here, because they are discussed in almost every data structure and algorithm book. The conventional and the distance pointer implementations are even used in the same fashion to have comparable memory and time usage statistics.

Node Definition

We define a node of pointer distance implementation like this:

```
typedef int T;
typedef struct listNode{
        T elm;
        struct listNode * ptrdiff;
};
```

The ptrdiff pointer field holds the difference between the pointer to the next node and the pointer to the previous node. Pointer difference is captured by using exclusive OR. Any instance of such a list has a StartNode and an EndNode. StartNode points to the head of the list, and EndNode points to the tail of the list. By definition, the previous node of the StartNode is a NULL node; the next node of the EndNode also is a NULL node. For a singleton list, both the previous node and next node are NULL nodes, so the ptrdiff field holds the NULL pointer. In a two-node list, the previous node to the StartNode is NULL and the next node is the EndNode. The ptrdiff of the StartNode is the exclusive OR of EndNode and NULL node: EndNode. And, the ptrdiff of the EndNode is StartNode.

Traversal

The insertion and deletion of a specific node depends on traversal. We need only one simple routine to traverse back and forth. If we provide the StartNode as an argument and because the previous node is NULL, our direction of traversal implicitly is defined to be left to right. On the other hand, if we provide the EndNode as an argument, the implicitly defined direction of traversal is right to left. The present implementation does not support traversal from the middle of the list, but it should be an easy enhancement. The NextNode is defined as follows:

```
typedef listNode * plistNode;
plistNode NextNode( plistNode pNode,
                    plistNode pPrevNode){
    return ((plistNode)
      ((int) pNode->ptrdiff ^ ( int)pPrevNode) );
}
```

Given an element, we keep the pointer difference of the element by exclusive ORing of the next node and previous node. Therefore, if we perform another exclusive OR with the previous node, we get the pointer to the next node.

Insertion

Given a new node and the element of an existing node, we would like to insert the new node after the first node in the direction of traversal that has the given element (Listing 1). Inserting a node in an existing doubly linked list requires pointer fixing of three nodes: the current node, the next node of the current node and the new node. When we provide the element of the last node as an argument, this insertion degenerates into insertion at the end of the list. We build the list this way to obtain our timing statistics. If the InsertAfter() routine does not find the given element, it would not insert the new element.

Listing 1. Function to Insert a New Node

```
void insertAfter(plistNode pNew, T theElm)
{
   plistNode pPrev, pCurrent, pNext;
   pPrev = NULL;
   pCurrent = pStart;

   while (pCurrent) {
      pNext = NextNode(pCurrent, pPrev);
      if (pCurrent->elm == theElm) {
         /* traversal is done */
         if (pNext) {
            /* fix the existing next node */
            pNext->ptrdiff =
                (plistNode) ((int) pNext->ptrdiff
                           ^ (int) pCurrent
                           ^ (int) pNew);

            /* fix the current node */
            pCurrent->ptrdiff =
              (plistNode) ((int) pNew ^ (int) pNext
                         ^ (int) pCurrent->ptrdiff);

            /* fix the new node */
            pNew->ptrdiff =
                (plistNode) ((int) pCurrent
                           ^ (int) pNext);
         break;
      }
      pPrev = pCurrent;
      pCurrent = pNext;
   }
}
```

First, we traverse the list up to the node containing the given element by using the NextNode() routine. If we find it, we then place the node after this found node. Because the next node has pointer difference, we dissolve it by exclusive ORing with the found node. Next, we do exclusive ORing with the new node, as the new node would be its previous node. Fixing the current node by following the same logic, we first dissolve the pointer difference by exclusive ORing with the next current node. We then do another exclusive ORing with the new node, which provides us with the correct pointer difference. Finally, since the new node would sit between the found current node and the next node, we get the pointer difference of it by exclusively ORing them.

Deletion

The current delete implementation erases the whole list. For this article, our objective is to show the dynamic memory usage and execution times for the implemented primitives. It should not be difficult to come up with a canonical set of primitive operations for all the known operations of a doubly linked list.

Because our traversal depends on having pointers to two nodes, we cannot delete the current node as soon as we find the next node. Instead, we always delete the previous node once the next node is found. Moreover, if the current node is the end, when we free the current node, we are done. A node is considered to be an end node if the NextNode() function applied to it returns a null node.

Use of Memory and Time

A sample program to test the implementation discussed here is available as Listing 2 from the *Linux Journal* FTP site (ftp.linuxjournal.com/pub/lj/listings/issue129/6828.tgz). On my Pentium II (349MHz, 32MB of RAM and 512KB of level 2 cache), when I run the pointer distance implementation, it takes 15 seconds to create 20,000 nodes. This is the time needed for the insertion of 20,000 nodes. Traversal and deletion of the whole list does not take even a second, hence the profiling at that granularity is not helpful. For system-level implementation, one might want to measure timings in terms of milliseconds.

When we run the same pointer distance implementation on 10,000 nodes, insertion takes only three seconds. Traversal through the list and deletion of the entire list both take less than a second. For 20,000 nodes the memory being used for the whole list is 160,000 bytes, and for 10,000 nodes it is 80,000 bytes. On 30,000 nodes it takes 37 seconds to run the insertion. Again it takes less than a second to finish either the traversal or the deletion of the whole list. It is somewhat predictable that we would see this kind of timing, as the dynamic memory (heap) used here is being used more and more as the number of nodes increases. Hence, finding a memory slot from the dynamic memory takes longer and longer in a nonlinear, rather hyperlinear fashion.

For the conventional implementation, the insertion of 10,000 nodes takes the same three seconds. Traversal takes less than a second for both forward and backward traversal. Total memory taken for 10,000 nodes is 120,000 bytes. For 20,000 nodes, the insertion takes 13 seconds. The traversal and deletion individually takes less than a second. Total memory taken for 20,000 nodes is 240,000 bytes. On 30,000 nodes it takes 33 seconds to run the insertion and less than a second to run the traversal and the deletion. Total memory taken by 30,000 nodes is 360,000 bytes.

Conclusion

A memory-efficient implementation of a doubly linked list is possible to have without compromising much timing efficiency. A clever design would give us a canonical set of primitive operations for both implementations, but the time consumptions would not be significantly different for those comparable primitives.

Prokash Sinha has been working in systems programming for 18 years. He has worked on the filesystem, networking and memory management areas of UNIX, OS/2, NT, Windows CE and DOS. His main interests are in the kernel and embedded systems. He can be reached at prokash@garlic.com.

At the Forge: Bloglines Web Services

Reuven M. Lerner

Issue #129, January 2005

More and more Web sites are offering machine-friendly versions of their services. Here's an example of a simple but useful service—updates on new Web site content.

Last month, we looked at ways in which we can gather, or aggregate, content from a number of different Web sites and put together a single summary of the day's news. Although it was amazing to see how much we could accomplish with a little bit of code, the application I presented is merely a toy when compared with actual aggregators. My example application supports only one user, is controlled by a primitive configuration file, doesn't categorize Weblogs into groups, checks for Weblog updates only when we explicitly ask it to do so and doesn't check for or handle errors.

Creating a robust, user-friendly aggregator is beyond the scope of this column, given the attention to technical and design details that would be necessary. But several days before I sat down to write this column, something amazing happened. The free, Web-based Bloglines.com aggregation service, which many people use to keep track of their favorite Weblogs, announced the availability of a Web service API that allows independent developers to create and deploy applications that use the data and applications developed by Bloglines. The publication and availability of the Bloglines API marks the growing popularity of Web services among well-known sites and opens the door to new applications built on the underlying Bloglines infrastructure.

This month, we take a look at the Bloglines API, including the creation of a simple application based on it. The API is brand new as of this writing (early October 2004) and undoubtedly will evolve as more people use it. If Weblogs interest you, and if you still are waiting to see practical uses for Web services, this combination of events might have come just in time.

What Is a Web Service, Anyway?

The basic idea behind Web services is quite simple: the Web's success is due in no small part to the fact that the client and server operating systems are irrelevant. So long as the client and server adhere to the HTTP and HTML specifications, they can communicate seamlessly. Linux has made inroads into the server space precisely for this reason.

Web services take this one step further, saying that computers and not people should be the biggest users of the Web. Although computers exchange information over HTTP, they send and receive data in XML, the markup language or meta-language, that has caught on like wildfire in recent years. If my computer can send XML in the HTTP request it sends to your computer, and your computer then returns XML in its HTTP response, we can exchange information regardless of what languages and operating systems we're using.

The original form of this service, known as XML-RPC, still exists and is great for fast, easy communication. But this idea was extended further, and a variety of data types, error-checking mechanisms and object serialization techniques were introduced that XML-RPC lacked. This extension became known as SOAP (Simple Object Access Protocol). SOAP theoretically can run on top of a variety of protocols, but it most often is sent on top of HTTP.

SOAP is a great solution to many problems, except that it is terribly complex, can be slow and is difficult to implement. And, both XML-RPC and SOAP require that the HTTP request include a well-formed XML request containing the query. One response to this growing complexity is REST (representational state transfer), in which all transactions are initiated by a simple HTTP GET request and all parameters are specified in the URL itself. The response then is an XML document containing the records and fields appropriate to the request. All of the Bloglines API calls are done with REST, although it's hard to say if this reflects the relatively simple queries now provided or if it's a design preference of the developers.

Although Web services probably are taking off behind corporate doors, only a few of the larger Web sites have made their plans and APIs public. The best-known examples are some of the largest and most profitable sites on the Web, including Amazon, eBay and Google. eBay charges for access to its Web services, with annual fees as well as per-transaction costs. By contrast, Amazon and Google have made their APIs freely available to the public, subject to usage restrictions and without making any promises regarding future availability.

In making its API public, Bloglines is indicating its interest in creating the same sort of developer community that Amazon, Google and eBay have created. This move also demonstrates its interest in remaining a leader in the world of

Weblog aggregation and applications. Given Google's purchase of Blogger several years ago and the extensive search features that Bloglines is making available with its API, we might be witnessing the beginning of a new type of application or platform battle, with the Google and Bloglines APIs competing for attention.

Presenting the Bloglines API

Bloglines aggregates content from a large number of Weblogs and frequently updated news sources. Bloglines is happy to accept feeds in a variety of formats, including Atom and several versions of RSS. Indeed, Bloglines offers subscribers the choice of which feed to use, if more than one is available. The Bloglines software then archives that content, providing a search interface for interested users. Bloglines provides some relevance features, telling subscribers which additional Weblogs might interest them. Finally, Bloglines lets you look at other users' subscriptions; if you are interested in seeing which Weblogs interest me, you can review my profile and see my subscriptions.

For now at least, much of this functionality remains under wraps, available only through the Bloglines Web site. But three particular pieces of functionality now are available from the Bloglines Web services API:

- Notifier: if you are a Bloglines subscriber and want to know when new content has arrived from one or more of the Weblogs to which you subscribe, now you can do it. This is the most established of the Bloglines Web services, and a number of tools for a number of operating systems and windowing toolkits rely on this interface to provide updates.
- Sync API: allows you to retrieve information about a particular user's subscriptions, as well as the latest entries from each of those subscriptions. You can think of this as the data underlying the HTML that Bloglines generates for the main Weblog listing it provides.
- Blogroll API: presents a way to retrieve and display a particular user's subscription list.

Notifier API

As I wrote above, Bloglines has decided to use REST for all of its Web services APIs. This means every request consists of a single URL, with all of the parameters and their values in the URL. Information is returned in whatever format the server deems appropriate. This stands in sharp contrast to SOAP, which specifies the name and type of each parameter and return value. A minor exception to this rule is that APIs requiring authentication expect the user name and password to arrive in HTTP Basic rather than in the URL itself. In the Bloglines universe, subscribers are identified by their e-mail addresses and user-selected passwords.

The easiest of the APIs to understand and use is the Notifier. To invoke the Notifier, simply go to the URL rpc.bloglines.com/update?user=reuven@lerner.co.il&ver=1. The response, while (incorrectly) tagged by the server as having a MIME type of text/html, contains a plain-text response of the format:

```
|A|B|
```

Notifiers can interpret the response as follows:

- Normally, A indicates the number of unread Weblog entries in the user's subscription.
- If the provided e-mail address is not registered with the system, then A contains -1.
- If B isn't empty, it then contains a URL pointing to an upgrade page. The documentation doesn't say much about what it means to have an upgrade page. I assume that such a page is meant for people rather than programs, because it would be impossible or at least quite difficult to identify all of the programs that use the Notifier API and that are in need of an upgrade.

We easily could implement the client side of the Notifier API in any modern high-level language. But at the time of this writing, versions of Bloglines client libraries exist in Perl, Python and Ruby. I use the Perl version (on CPAN as WebService::Bloglines), but you may feel more comfortable rolling your own version, using a different version or both.

Here is a simple command-line program that prints “You have new blogs!” if Bloglines reports that new messages are waiting and “No new blogs” if I already have read everything:

```
#!/usr/bin/perl

use WebService::Bloglines;

my $username = 'reuven@lerner.co.il';
my $password = 'MYPASS';

my $bloglines = WebService::Bloglines->new(
                           username => $username,
                           password => $password);

my $unread_blogs = $bloglines->notify();

if ($unread_blogs)
{
print "You have '$unread_blogs' new blogs!\n ";
}
else
{
print "No new blogs.\n"
}
```

The number returned by `$bloglines->notify()` is the number of unread postings, not of unread Weblogs. If there are 15 unread messages from five Weblogs, `$bloglines->notify()` returns 15, not 5. Moreover, the number reflects the state of the internal Bloglines database. That is, if you click on the Keep New check box at the bottom of a Weblog entry, it is included in the count of new messages returned by `$bloglines->notify()`.

If we enter an incorrect user name, our program exits with a fatal error and indicates that we gave it a bad user name. Giving a bad password has no consequences for the Notifier API, because that information is available publicly.

Blogroll API

Another offering from Bloglines, as we mentioned earlier, is the Blogroll API. A blogroll is a list of Weblogs that a particular author finds interesting and often reads. It's likely that if you enjoy reading someone's Weblog, you also would enjoy perusing that person's reading list. In the case of Bloglines, a blogroll simply is a list of subscriptions associated with a particular user.

So far, we have mentioned that someone's Bloglines user name is the same as his or her e-mail address. But this is not completely true—if you choose to use Bloglines for your own private purposes, never sharing information about your subscriptions with other people, you need nothing more than your e-mail address. But if you do want to expose your subscriptions, you must choose a user name with which they can be associated. In my case, my registration e-mail address is reuven@lerner.co.il and my user name is reuven. This distinction wasn't clear to me for the first few months that I used Bloglines, although it seems to be more obviously advertised now.

If a user has established a user name for public consumption and if that user has chosen to share his or her subscriptions, you can get a version of that user's Blogroll that uses HTML and JavaScript as follows: http://www.bloglines.com/public/reuven. If we want to retrieve the blogroll results in HTML, we can do so with the following style of URL: http://rpc.bloglines.com/blogroll?id=reuven&html=1.

But the whole idea of Web services is to make data machine-readable, such that it can be stored and processed by computers. OPML, the Outline Processor Markup Language, specified by Dave Winer in 2000, is the format used by Bloglines when it exports a list of subscriptions. It is not an official part of the Bloglines Web services specification, but you can retrieve it by going to the following type of URL: http://www.bloglines.com/export?id=reuven.

In all of the above examples, you can and should replace my Bloglines user name with that of the user whose blogroll you want to read. Not every user makes his or her subscription list public, so you may encounter error messages when trying to retrieve them. And once you retrieve the OPML, you need to process it, perhaps using a tool such as the publicly available XML::OPML module from CPAN.

Conclusion

As you can see, the Bloglines API for Web services opens the door to a host of third-party applications. It increasingly is possible to create useful applications that use HTML, XML and HTTP but that are not tied to a Web browser. The Notifier and Blogroll APIs are only the beginning. As we saw earlier, there is also a Sync API that effectively allows developers to create alternative GUIs and applications with the actual content Bloglines retrieves and stores. In my next column, we will look at the Sync API, building some basic applications on top of the Bloglines infrastructure.

Reuven M. Lerner, a longtime Web/database consultant and developer, now is a graduate student in the Learning Sciences program at Northwestern University. His Weblog is at altneuland.lerner.co.il, and you can reach him at reuven@lerner.co.il.

Kernel Korner: The Linux Test Project

Nigel Hinds

Issue #129, January 2005

Finding 500 bugs in 50 different kernel versions is the fruit of this thorough Linux testing and code coverage project.

The Linux Test Project (LTP) was developed to improve the Linux kernel by bringing automated testing to kernel design. Prior to the LTP, no formal testing environment was available to Linux developers. Although most developers unit-tested the effects of their own enhancements and patches, systematic integration testing did not exist. The LTP's primary goal is to provide a test suite to the Open Source community that helps to validate the reliability, robustness and stability of the Linux kernel. The suite tests kernel function and regression, with and without stress. The LTP is not a performance benchmark, but benchmarks often are used to drive the kernel during testing.

The LTP began as 100 test programs developed by SGI. Now, through the joint efforts of SGI, IBM, OSDL, Bull, Wipro Technologies and individual Linux developers, the LTP contains over 2,500 test programs, also called test cases, and a number of automation tools. The LTP supports multiple architectures, including x86, IA32/64, PPC32/64, and 32- and 64-bit s/390.

Although other test suites and projects exist, the LTP includes an environment for defining new tests, integrating existing benchmarks and analyzing test results. The Software Testing Automation Framework (STAF/STAX) is an open-source system that allows you to plan, distribute, execute and collect test results from a large pool of multiplatform test hosts. STAF/STAX also provides a powerful GUI-monitoring application that allows you to interact with and monitor the progress of your jobs. Test-coverage visualization tools let you see how much of a test's source code is executed by the kernel.

The IBM Linux Technology Center (LTC) has played a key role in using the LTP to uncover defects in the Linux kernel. Using the LTP, the LTC has tested more than 50 new kernel versions and found more than 500 defects. As covered in

Linda Scott's whitepaper (see the on-line Resources), a typical kernel test cycle uses the LTP for focus testing to isolate and validate Linux component and application stability. This includes regression testing on new kernels to ensure they meet the functionality of previous kernels. Integration testing then validates component interaction, driven by macro-benchmark workloads. Finally, reliability and stress testing validate systemic robustness with extended duration tests (96 hours to 30 days).

The remainder of this article describes how to download and run the LTP test suite using the automation tools. We also discuss some LTP tools that can be used to help improve kernel development and testing.

The Test Suite

The tests cover a wide range of kernel functions, including system calls, networking and filesystem functionality. The basic building block of the test suite is a test program that performs a sequence of actions and verifies the outcome. The test results usually are restricted to PASS or FAIL. Together, all the test programs and tools make up the LTP package.

The LTP is a GPL package and is available from SourceForge.net. A stable version of the LTP test suite source, ltp-*yyyymmdd*.tgz is released monthly. As of this writing, the latest version is ltp-20040405. After downloading the package, extract and install as follows:

```
tar zxf ltp-20040405.tgz
cd ltp-20040405
make
make install
```

You need root access to perform that last step and also to run the test suite. The test suite also is available in binary and source RPM format. For those of you who like living on the edge, development snapshots can be downloaded through anonymous CVS (see Resources).

Executing the Test Suite

Once installed, a number of options are available for running the LTP test suite. The most popular method is to use the runalltests.sh script, which executes about 800 of the original tests. The tests not included in runall are destructive, require monitoring or for some other reason cannot be automated. The runall script has a default behavior to run a single iteration of the test suite and produce verbose screen output. This output can be omitted with the quiet option (-q). As a simple introduction, we ignore the screen information for now and use the -l logfile_name and -p options to generate human-readable log results.

The test cases are executed by the test driver called Pan. Pan, included in the LTP package, is a lightweight driver used to run and clean up test programs. The runalltests script calls Pan to execute a set of test cases or a single test case. You can execute a set of test cases by providing runalltests with a -f scenario file. A scenario file is a simple ASCII text file that contains two columns. The first column has the name of a test case, and the second column has the command to be run. Comments start with a pound sign. For example:

```
# Testcase to test mmap function of the kernel
testcase1       mmap3 -l 100 -n 50

# Testcase to stress the kernel scheduler
testcase2       sched_stress.sh
```

The test driver uses the exit value of the test case to decide success or failure of a test. If the test case exits with a non-zero value, Pan records this as FAIL. If the test case exits with a value zero, the driver records it as PASS.

The simplest use of the test suite is to run it on your system to ensure that there are no failures:

```
runalltests.sh -l log -p -o output
```

For known failures, the LTP package includes an explanation and pointers to places for more information. Below is the partial log file from running ltp-20040506 on a 2.6.3 kernel:

```
Test Start Time: Mon May 17 14:20:45 2004
-----------------------------------------
Testcase                       Result     Exit Value
--------                       ------     ----------
abort01                        PASS       0
accept01                       PASS       0
access01                       PASS       0
...
rwtest01                       PASS       0
rwtest02                       PASS       0
rwtest03                       FAIL       2
rwtest04                       FAIL       2
rwtest05                       PASS       0
iogen01                        PASS       0
...
-----------------------------------------------
Total Tests: 797
Total Failures: 6
Kernel Version: 2.6.3-gcov
Machine Architecture: i686
Hostname: ltp2
```

In this partial log, 797 tests were run and six failed. rwtest03 and rwtest04 are I/O tests that failed due to mmap running out of resources. This problem has been resolved. The remaining failures, not shown in the log, are described below:

- setegid01: verify that setegid does not modify the saved gid or real gid—failed because of a bug in glibc 2.3.2.
- dio18,dio22: I/O testing—failed because of data comparison mismatch.
- nanosleep02: verify that nanosleep will suspend and return remaining sleep time after receiving signal—failed due to lack of microsecond clock precision.

Writing test programs is fairly straightforward. The test cases are written in ANSI C and BASH and use the LTP Application Program Interfaces (APIs) provided by the LTP library libltp to report test status. Templates are provided that show you how to develop test cases using libltp. The test cases can use the interface to print results messages, break out of testing sequence and report a test status such as PASS or FAIL. Manual pages for using these APIs are provided in the test suite package and also on the LTP Web site. For more on the esoteric uses of LTP and a tutorial on developing tests that can be included in the LTP, see the Iyer and Larson papers in Resources.

Automation Tools

Although not required to run the test suite, the LTP has a number of related tools and projects that facilitate test automation. Two of these projects are the Software Testing Automation Framework (STAF/STAX) and the Open Source Development Lab (OSDL) Test Platform.

LTC uses STAF/STAX to manage a pool of test machines. Using the STAF/STAX Web interface you can find and configure test machines, then run and monitor any set of test programs and return the results. STAF is an open-source, multiplatform, multilanguage testing framework. It is based on the concept of reusable services, such as process control, logging and event handling that automate testing activities. At its core, STAF is a message routing dæmon that maintains a network of local and remote services and routes requests to those services. A network of STAF-enabled machines is built by running STAF clients on dedicated networked hosts. STAX is a GUI-based execution engine built on STAF, XML and Python. It provides an interface for testers to distribute, execute and process test results.

The OSDL Scalable Test Platform provides a framework for developers to execute tests against specific kernels and kernel patches through a Web-based interface. LTP is one of the tests that OSDL executes. Using the Web interface, you also can search for historic test results. The LTP Web site has detailed information regarding this framework.

Expanding the Test Suite with Coverage Analysis

The often unspoken assumption with software testing is that the test cases cover a majority of the software source code written. A test covers a line of code if running the test executes the line. Coverage analysis measures how much of the target code is run during a test and is a useful mechanism for evaluating the effectiveness of the LTP test suite. Given two test cases that run successfully, the test with the higher code coverage provides somewhat more assurance that the code is bug-free. Of course, bugs still may exist in the untested code, and even 100% coverage does not guarantee bug-free code.

Cornett's paper provides a good introduction to the many types of coverage. We've based the LTP coverage on the GCC compiler that provides statement and branch coverage. As stated earlier, statement coverage reports which lines of the source code are executed. Branch-conditional coverage reports which Boolean conditions in a control statement, such as “if” or “while”, are tested and taken. In the code below, branch-conditional coverage would tell us when the branch was taken, statement1 executed, and when it was due to condition1 or condition2 being true:

```
   if ( condition1 || condition2 )
       statement1;
   else
       statement2;
```

GCC coverage works by passing the options -fprofile-arcs -ftest-coverage to the compiler and the GCOV program that processes coverage data. GCOV produces a source file annotated with the number of times each line of code and branch condition was executed. GCC coverage was intended originally for user-space programs and needed to be adapted for the kernel because coverage data is produced only when a program terminates, which the kernel never does.

Also, because the kernel is not linked with standard C libraries, many of the GCOV structures are not present in the kernel. The LTP has published a GCOV-kernel patch to the Linux kernel that addresses these issues and allows developers to use the existing GCOV tools to gather coverage data from a running kernel. Installation instructions as well as a detailed description of the functionality provided by the patch can be found at the LTP Web site and in an Ottawa Linux Symposium paper by Paul Larson and others.

The GCOV-kernel patch is published on the LTP Web site as a separate package, but it is included in the LTP development tree. In addition to kernel code changes, when installed, the patch configures the Makefiles to pass the coverage options to GCC when the kernel is compiled. The coverage options instruct the compiler to generate code and data structures to capture information that is used to determine which lines of kernel code were executed. The user-space tool GCOV combines the source files and the files generated by running a GCOV-enabled program—in our case this is the kernel —to produce a new source code file with the count for each line of C code, representing the number of times the line was executed. Because the program output needed for GCOV is not created until the program ends, and the kernel does not terminate, the patch also creates a /proc/gcov/... tree that GCOV can use at any time to get counter data from the kernel.

To facilitate coverage analysis further, the LTP has developed a utility, LCOV, to create more useful graphic GCOV output. LCOV can be downloaded from the GCOV-kernel Web site. LCOV automates the process of extracting the coverage data from the kernel, running GCOV and producing HTML. Once the GCOV-kernel patch is applied and compiled, the coverage system can be used as follows.

First, load the gcov-proc kernel module:

```
insmod gcov-proc.o
```

Clear the GCOV counters:

```
lcov --reset
```

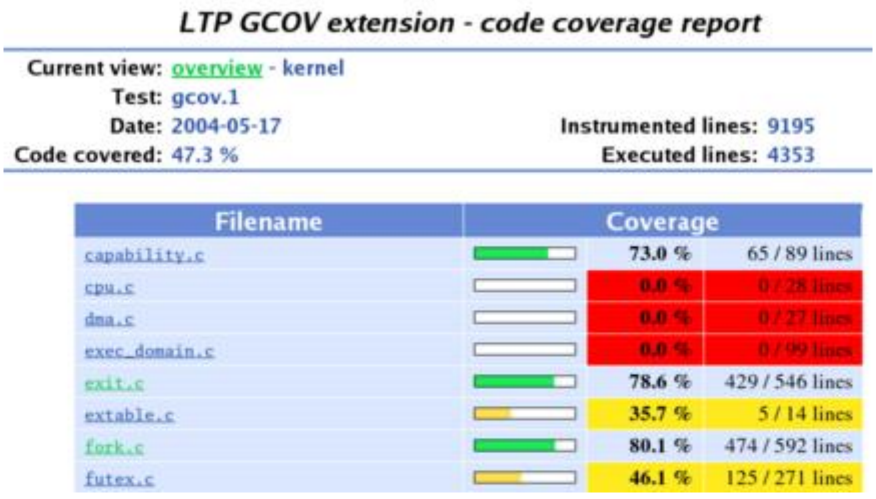
Run the LTP test suite or your favorite test program next, then capture GCOV data:

```
lcov -c -o coverage.info
```

Create the HTML coverage tree:

```
genhtml coverage.info
```

genhtml is one of the LCOV tools and generates HTML output at both the directory and file level, as illustrated in Figures 1 and 2. Figure 1 is a partial screenshot of the Linux kernel source subdirectory. In this example, the total amount of code covered for the directory is 47.3%. Each line shows a filename. A color-coded meter is used to represent the amount of coverage for the file: green for files with 50% coverage or greater, yellow for files with coverage between 10% and 50% and red for files with less than 10% coverage. The last two columns show the percent coverage for the file and number of lines executed over the total number of lines instrumented. Both figures are displayed on a color-coded background. Figure 2 is a partial view of a printk.c file. This is a graphical view of the original GCOV output. A similar color coding is used to allow you to identify under-utilized code quickly. At this time, LCOV output shows only statement coverage and not branch coverage.

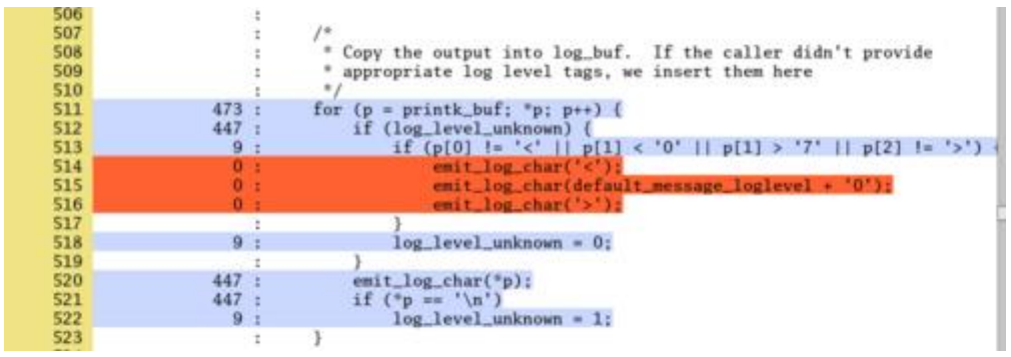


LTP GCOV extension - code coverage report

Current view: overview - kernel
Test: gcov.1
Date: 2004-05-17 Instrumented lines: 9195
Code covered: 47.3 % Executed lines: 4353

Filename	Coverage		
capability.c	🟩	73.0 %	65 / 89 lines
cpu.c		0.0 %	0 / 28 lines
dma.c		0.0 %	0 / 27 lines
exec_domain.c		0.0 %	0 / 99 lines
exit.c	🟩	78.6 %	429 / 546 lines
extable.c	🟨	35.7 %	5 / 14 lines
fork.c	🟩	80.1 %	474 / 592 lines
futex.c	🟨	46.1 %	125 / 271 lines

Figure 1. genhtml produces a code coverage report from GCOV data.



```
          :     /*
          :      * Copy the output into log_buf.  If the caller didn't provide
          :      * appropriate log level tags, we insert them here
          :      */
      473 :     for (p = printk_buf; *p; p++) {
      447 :         if (log_level_unknown) {
        9 :             if (p[0] != '<' || p[1] < '0' || p[1] > '7' || p[2] != '>')
        0 :                 emit_log_char('<');
        0 :                 emit_log_char(default_message_loglevel + '0');
        0 :                 emit_log_char('>');
          :             }
        9 :             log_level_unknown = 0;
          :         }
      447 :         emit_log_char(*p);
      447 :         if (*p == '\n')
        9 :             log_level_unknown = 1;
          :     }
```

Figure 2. Color codes show under-utilized code in the source file.

Conclusion

As Linux plays an increasing role in the enterprise computing space, robustness and reliability requirements have led to more formal testing methods. The LTP is a functional regression testing suite used to help improve Linux reliability. For any kernel development project, running the LTP test suite gives you a method to help ensure your changes don't break the kernel. As you test your kernel modifications, a GCOV-enabled kernel and accompanying LTP tools will help you visualize the effectiveness of your testing and help focus the test team on areas with low coverage.

In addition to test results that show kernel regressions and code coverage gaps, the LTP and coverage analysis potentially provide a method for measuring kernel improvement over time. Consider the simple argument: combined with higher coverage of the kernel code, fewer kernel failures suggest that Linux kernel reliability is improving. A study of how well LTP tracks improvement in Linux is part of our future work.

Finally, we would like to encourage developers to submit their tests to be included in the LTP suite. As always, suggestions and comments are welcome, and should be sent to the mailing lists found on the LTP Web site.

Resources for this article: /article/7809.

Nigel Hinds is a member of the technical staff at IBM T. J. Watson Research Center. He develops testing tools and maintains the kernel coverage system for the Linux Testing Project. His other interests include networking and distributed systems. He can be reached at nhinds@us.ibm.com.

Archive Index Issue Table of Contents

Advanced search

Cooking with Linux: Forgotten Security

Marcel Gagné

Issue #129, January 2005

You aren't supposed to use the same password on multiple accounts, but with all the servers and Web sites that require a password, what's a security-conscious chef to do? Here's how to make password wrangling both convenient and secure.

Where is that wine order from Henri's Fine Wines, François? We seem to be getting low on a couple of my favorites. Henri is usually right on top of these things. Did he not give you an order to approve? Ah, excellent. Then you have the order? No? What do you mean, it is somewhere safe? You either have it or you don't? I see. You thought it was important, so you encrypted the order and threw away the original message. Let me guess, *mon ami*, you do not remember the password you used to encrypt the message. That's what I thought. All right, show me what program you used.

Steganography, François? You used a picture of yourself and embedded the wine order inside it—I'm impressed! We will deal with this problem a little later, François. There isn't much time, and our guests will be here any moment. Ah, but they are already here.

Welcome, *mes amis* to *Chez Marcel*, the world's finest Linux French restaurant and the home the greatest wine cellar in the world. Of course, at this moment, it might be only the second best in the world. It seems my faithful waiter misplaced an order and didn't want to tell me. Yes, François, I know you know where it is. Just go down to the cellar and bring back the 2000 Douro from Portugal. This is a great red, *mes amis*, a rich and powerful wine with wonderful, dark fruit flavors and just a hint of mystery. *Vite*, François!

While François brings back the wine, let me tell you how he managed not to misplace the wine order. He used a program called Steghide, created by Stefan Hetzl, to encode and encrypt the list inside an image, an image of himself as it turns out (Figure 1).

Figure 1. Hidden somewhere in this image is a large order for wine.

This process is called steganography. Using this process, you can take any message and encode it inside another message (or in this case, a graphic image). In fact, you could create a whole Web site, full of images with secret messages in all of them, and no one would be the wiser. You can get a copy of Steghide at the Steghide home page (see the on-line Resources). Contributed binaries are easy to find. To build from source, Steghide requires the libmhash, libjpeg, zlib and libmcrypt development libraries; other than that, it's an easy build that you'll recognize as an extract and build five-step:

```
tar -xzvf steghide-0.5.1.tar.gz
cd steghide-0.5.1
./configure
make
su -c "make install"
```

In order to hide the wine replacement order, François used the following command to encode the document into his picture:

```
steghide embed -cf francois.jpg -ef wine_order.txt
```

Speaking of wine, François has returned. If you would be so kind, *mon ami*, and pour for our guests. Anyhow, immediately upon running the command, you are asked for a passphrase:

```
Enter passphrase:
Re-Enter passphrase:
embedding "wine_order.txt" in "francois.jpg"... done
```

The result is an image that still looks as it did before you hid your secret message inside, but its size will have changed. To recover the data from the image, you or the person to whom you sent the image can use the extract argument with the command:

```
steghide extract -sf francois.jpg
Enter passphrase:
```

If you successfully entered the right information, the hidden file is saved to disk. This is precisely where things start to go wrong. After forgetting the passphrase, there is no way to retrieve the information. In real life, some of us have, on occasion, lost our keys. Some people chronically lose their keys, and that's why an enterprising inventor came up with the idea of putting a beeper on a keychain. Assuming you don't lose the locator unit, you can push a button and your keys emit a high-pitched signal telling you which cushion they've slipped behind.

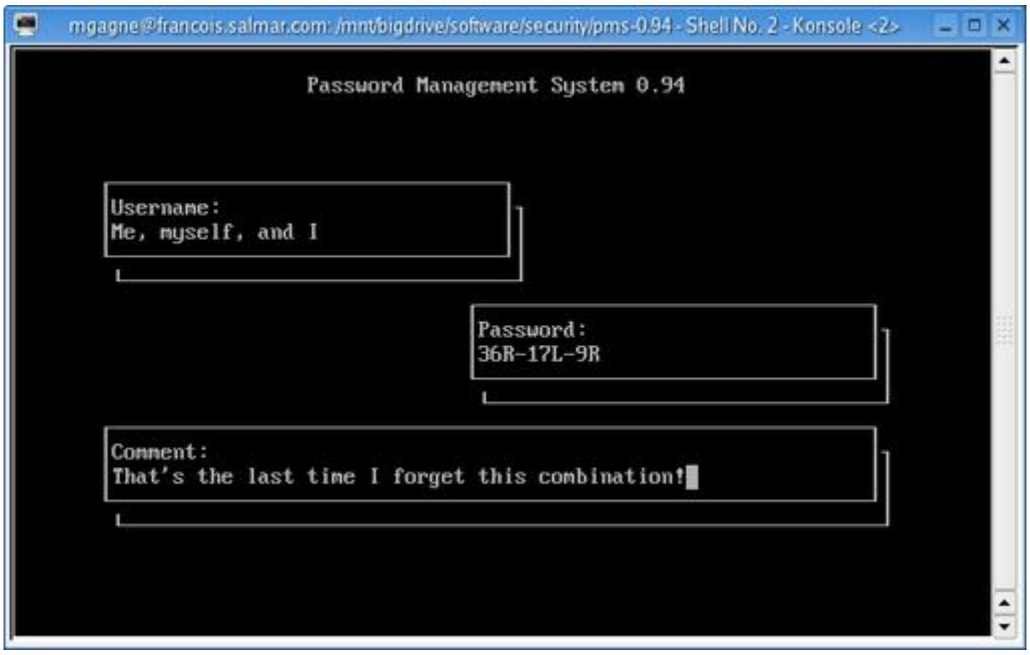
With passwords, there's a similar idea. The simplest of these is to write passwords down or keep them in a text file. That's not a particularly secure method. However, the idea of keeping a list of passwords or passphrases makes more and more sense as we are asked to remember dozens, sometimes hundreds of passwords. It might be a lot easier if all we had to remember was one password, and that's where password managers come into play.

The first one I ran across was Dennis Pries' Password Management System or PMS. I like this one because it can run in a text-only terminal window, which means you can access it through a shell login from wherever you might be. You can pick the program up from SourceForge (see Resources), where source and a Debian package are available.

To build PMS, you have to do a kind of double extract and build five-step. First, extract the tarred and gzipped bundle (`tar -xzvf pms-0.94.tar.gz`). Now, look inside that source directory and you'll find a contrib directory from which you can build cdk using the extract and build five-step on that source archive. Once cdk is installed, go back to the PMS source directory, then build and install that.

The command to use this password manager is `pms`. When you run it for the first time, it asks you for a master password. This is the only password or passphrase you need to remember from here, but make sure you do. Forget the master key and you won't be able to get at all those other passwords. Then, PMS provides you with a simple menu from which you can add, delete or rename a host. These would be the hosts that you need to log in to. Start by adding a host (for example, www.somewhere.dom) and then a comment (for example, main production system). You'll find yourself back at the main menu. From there, choose User Functions. That's the menu that lets you add or delete user names associated with whatever hostnames you added in the previous step. You also can show a user to display the password you thought was lost forever.

Before I move on, I should point out that the hostname and user name could be anything. For hostname, I could enter “school locker”, for user name “combination” and for password, the combination itself. Although it is intended for recording login information, it works very well for other things (Figure 2).


mgagne@francois.salmar.com: /mnt/bigdrive/software/security/pms-0.94 - Shell No. 2 - Konsole <2>

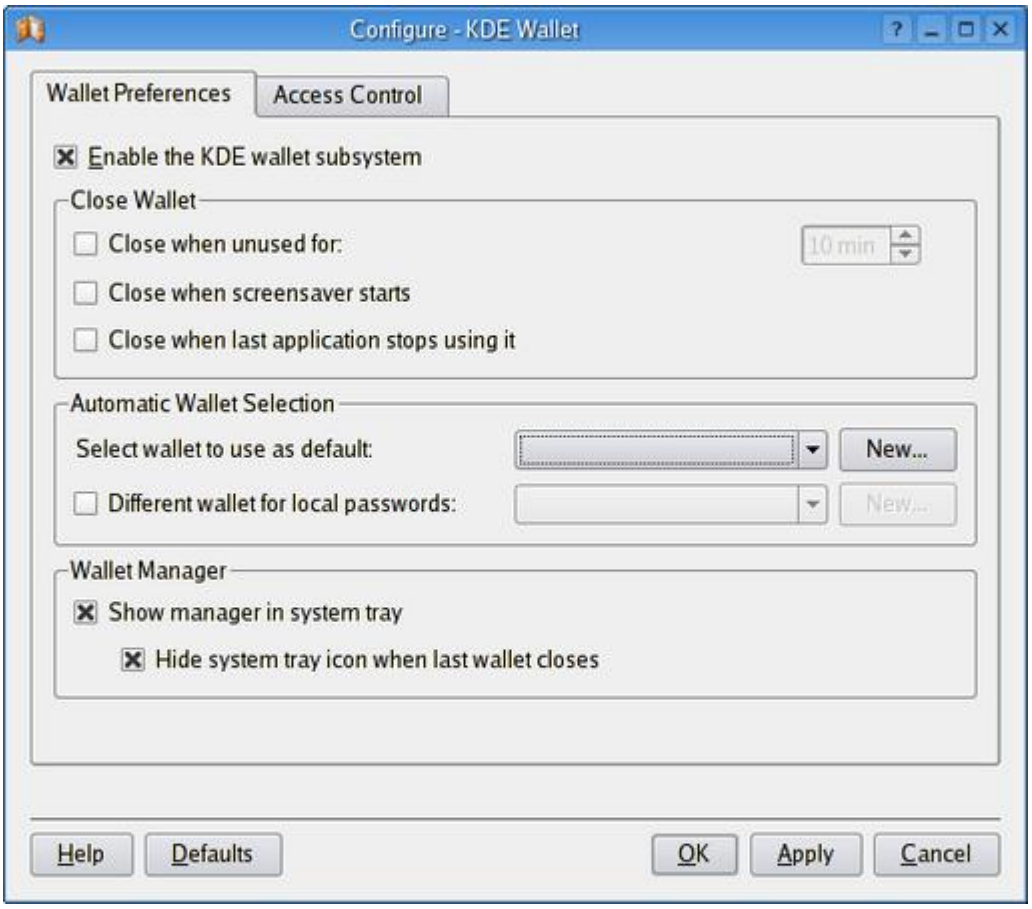
```
Password Management System 0.94

Username:
Me, myself, and I

Password:
36R-17L-9R

Comment:
That's the last time I forget this combination!
```

Figure 2. PMS isn't only for passwords. Store your locker combination too.

Another thing we tend to forget all the time are the various passwords we enter for the countless Web sites we visit. From on-line banking to newspapers that require you to have a free account to read the articles, the number of accounts we build up over time is staggering. Then, there are the passwords associated with our instant messaging accounts, e-mail accounts, FTP sites and more. If there were some way to maintain and store all this information transparently while we worked, it could simplify things. Is there such a thing that integrates into the desktop?

The answer is yes. With the release of KDE 3.2 and now 3.3, users of the desktop find that they have a password manager built in. It's George Staikos' KDE Wallet Manager, and the program that runs it is kwalletmanager. When you first start the program, no wallets are created. You will, however, see a small wallet icon appear in your system tray. If the wallet manager window is not already open, click on the icon, and a blank box, looking a great deal like an empty directory folder, appears. Click Settings on the menu bar and select Configure Wallet.

A new dialog box appears with most items grayed out. Click the check box that says Enable the KDE Wallet Subsystem. Several other options now are available to you (Figure 3).


Configure - KDE Wallet

Wallet Preferences | Access Control

[✓] Enable the KDE wallet subsystem

Close Wallet
[ ] Close when unused for: 10 min
[ ] Close when screensaver starts
[ ] Close when last application stops using it

Automatic Wallet Selection
Select wallet to use as default: New...
[ ] Different wallet for local passwords: New...

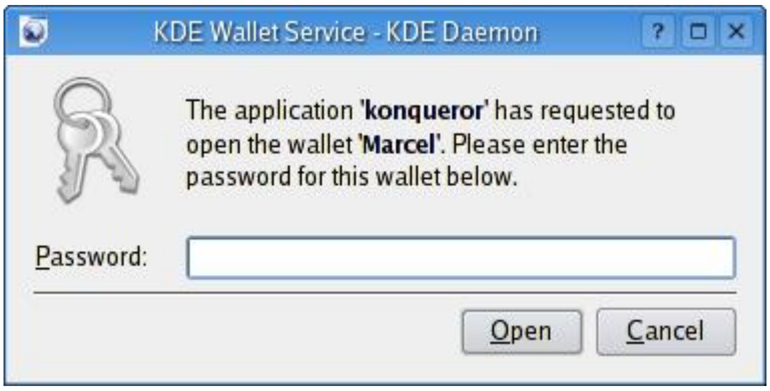
Wallet Manager
[✓] Show manager in system tray
[✓] Hide system tray icon when last wallet closes

Figure 3. Configuring the KDE Wallet Manager to Handle Your Passwords

Look at the middle section, labeled Automatic Wallet Selection. You're asked to select the wallet to use as the default. Right below that, you have the option of selecting another wallet for local passwords (more on that in a moment). If this is the first time you run the KDE Wallet, it's unlikely at this point that you have an existing wallet; click New and enter a name for this wallet when prompted to do so. You simply might choose to use your name as I did. Once you enter the name and click OK, the KDE Wallet Manager Wizard appears offering you the basic or advanced setup, with basic being the recommended choice. In the advanced setup, there are a few more information screens and you can choose at that time to create a separate wallet for local passwords. I chose basic and went for the single wallet.

Whichever you choose, at some point the wizard asks you for a master password to open the wallet. This is the super-password, the one you don't want to forget—the one that opens the door to all the others. Choose carefully, and make sure the check box labeled “Yes, I wish to use the KDE wallet to store my personal information” is checked on.

When you've finished the wizard you almost are done. A new dialog box pops up telling you that an application (the wizard) has asked to create a new wallet. You now must confirm this request with the password for that wallet. Take note of this dialog. You'll see something similar to it once per KDE session whenever an application wants to open the wallet to check a password. Until you log out, the wallet now stays open. In fact, visit a Web site where you are asked for a user name and password (such as your bank page). After you have entered the information and clicked Submit or Enter (depending on the form), a new dialog appears from the KDE Wallet Manager telling you that an application (in this case, Konqueror) has requested to open the default wallet (which you just created). Have a look at Figure 4 to see what I mean.



The application '**konqueror**' has requested to open the wallet '**Marcel**'. Please enter the password for this wallet below.

Figure 4. Your master password must be entered to open the wallet.

Enter your master password and click to continue. You'll get one final warning telling you that this encrypted information is about to be saved and asking for your confirmation. Click Yes. Now, look down in your system tray, and you'll see that the icon shows a slightly open wallet where before it was closed.

The beauty of this particular system is that all the information is entered magically for you next time you visit a site. This is true of any KDE application that asks you for a password, such as your instant messenger.

There is one catch, however, and it is a big one. As I mentioned, you'll need to enter your master password only once per KDE session, and that makes things easy. But beware: now that you've got your system automatically filling in passwords for you, securing your desktop becomes important. Make sure you lock your desktop before you walk away. Another way to do this is to go back into the KDE Wallet configuration dialog and look at some of the Close Wallet options. You can set it to close automatically after a defined period of time, like when the screensaver starts (when you normally would be away) or when the last application using it is closed. Doing it that way, you have one less thing to remember.

Judging by the clock on the wall, *mes amis*, it appears that closing time is once again upon us. As you can see, there are a number of alternatives for storing password information so that you do not have to remember dozens or hundreds of cryptic letter and number combinations. Perhaps if we can convince François to use a tool like this in the future, there won't be any more lost orders. In the meantime, I'm sure we can convince him to refill our guests' glasses one more time. And don't worry about the wine supply. I personally will make sure the wine cellar is fully stocked when we next meet. Until next time, *mes amis*, let us all drink to one another's health. *A votre santé Bon appétit!*

Resources for this article: /article/7860.

Marcel Gagné (mggagne@salmar.com) lives in Mississauga, Ontario. He is the author of *Moving to the Linux Business Desktop* (ISBN 0-131-42192-1), his third book from Addison-Wesley. In real life, he is president of Salmar Consulting, Inc., a systems integration and network consulting firm. He is also a pilot, writes science fiction and fantasy and folds a mean origami T-Rex.

Paranoid Penguin: Taking a Risk-Based Approach to Linux Security

Mick Bauer

Issue #129, January 2005

Risk is inevitable. Be pessimistic about individual programs failing, make plans for handling and containing problems, and you'll keep your system as a whole secure.

Since I started writing this column four years ago, the nature of Linux software vulnerabilities and threats hasn't changed that much. Buffer overflows, bad configurations (including file permissions) and inadequate input validation still form the lion's share of Linux vulnerabilities. If the same kinds of vulnerabilities keep cropping up, is the whole patch rat race futile? Or, is there a broader approach to Linux security that we can adopt?

This month, I discuss Linux security from a risk-based perspective and illustrate how by using the risk-based approach we can mitigate not only the Linux vulnerabilities we know about, but the ones nobody's discovered or publicized yet.

The Risk-Based Approach to Security

You may be wondering, what do I mean by a risk-based approach? Isn't all information security about risks? Indeed it is, but this term is actually short for risk management-based approach.

There are only a few ways to handle a given information security risk. We can avoid it by not doing anything that exposes us to that risk. We can eliminate it by addressing its root cause (which is, in practice, seldom within our control). We can mitigate it—that is, do something that lessens the impact of the risk in some way. Or, we can accept it.

One school of thought, now thankfully obsolete, holds that security is a binary equation: things either are secure or stupid, and if one judges a given activity or tool as not being secure, one simply should not do that thing or use that tool. In other words, in this school of thought, risk avoidance is the preferred approach to security.

As most of us acknowledge nowadays, however, absolute security does not exist. No magic combination of software choices, software/system configuration or network topology can make us invulnerable to security breaches. No combination, that is, that you actually can do any work with. Risk in some quantity or another is inevitable in networked computing.

The risk management-based approach to security acknowledges the need to seek balance between risk avoidance, risk mitigation and risk acceptance, by prioritizing risks based on their likelihood and potential impact. Risks that are both likely to occur and expensive to recover from are tagged as being the most important risks either to mitigate or avoid. Risks that either are highly unlikely or extremely cheap to recover from become reasonable candidates for risk acceptance. By the way, when I talk of the cost or impact of a risk occurring, I mean not only monetary cost but also lost time, reputation and productivity.

Figure 1 shows the general relationship between a risk's likelihood, its cost and its acceptability. The precise shape of the curve that defines the acceptable risk and unacceptable risk zones will vary from organization to organization. A financial institution, for example, will tend to have a much bigger red zone than a university network.

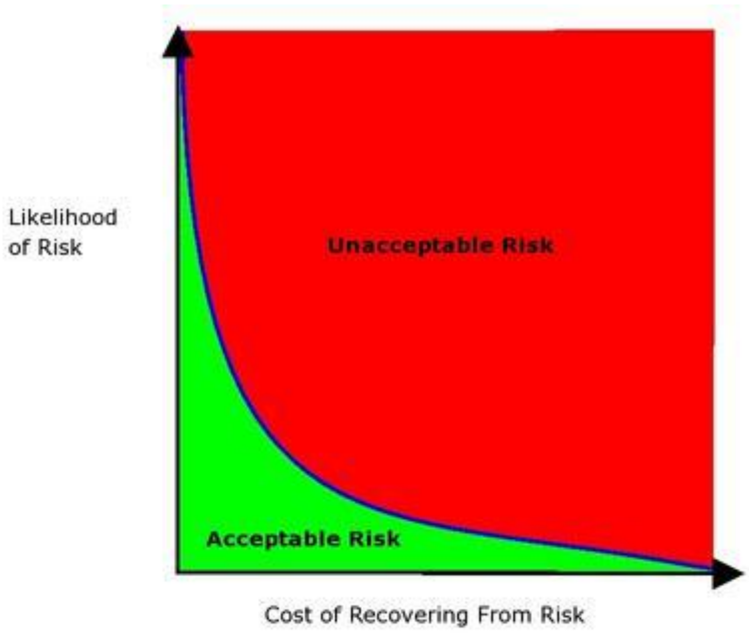


Figure 1. Risk Thresholds

Thus, to take a risk-based approach to security is to acknowledge that not all risks are created equal, and therefore, you must choose your fights. To do so effectively, however, requires you to be creative and honest in identifying and assessing the risks in a given undertaking. Denying a risk exists is far more dangerous than acknowledging and accepting that risk and making recovery plans should the worst occur.

This brings up another important aspect of the risk-based approach: risk acceptance should not mean complacency. Any risk that can't be avoided or mitigated must at least be taken into consideration in business continuation and recovery plans. Furthermore, very few information security risks can't be mitigated in some way or another; many types of risks can't be eliminated but can nonetheless be contained or attenuated.

Vulnerabilities and Threats

Okay, so Linux security is best handled with a risk-based outlook. What does that look like? Step one is to think about known and potential vulnerabilities in your Linux system. Most of Linux's application and system vulnerabilities fall into one of these categories:

- Buffer-overflow vulnerabilities (inadequate bounds checking).
- Poor input validation.
- Inappropriate file permissions.
- Inappropriate system privileges (avoidable use of root).
- Sloppy configuration.
- Insecure use of temporary files.
- Predictable or known default passwords.
- Administrative back doors (test or debug accounts).

The first vulnerability in this list, buffer overflows, is arguably the scariest. Buffer overflows frequently lead directly to remote root compromises. Like buffer-overflow conditions, many of these vulnerabilities are the direct result of programming errors such as insecure use of temporary files and administrative back doors. Others are more typically user-inflicted by predictable passwords or sloppy configuration.

No vulnerability, however, actually constitutes a threat unless someone attempts to exploit it. In other words, a threat equals a vulnerability plus an attacker.

Step two is to think about ways that these vulnerabilities might be exploited. Although Linux vulnerabilities haven't changed much over the years, the actors who attempt to exploit them have. They've become both more effective and dumber. The scary truth is, easy availability of exploit code and scripts has made it increasingly easy for unskilled attackers to conduct increasingly sophisticated attacks.

For example, traditionally, conducting a buffer-overflow attack has required considerable programming skill—besides being able to determine where in memory the overflowed data will end up, the attacker must write or procure exploit code, or shellcode written in assembler code specific to the target system's architecture, such as i386 or SPARC. Shellcode is the code that gets overflowed and executed, resulting in a shell, ideally with root privileges, on the target system.

In olden times, the difficulty of identifying offsets and writing working shellcode narrowed the field of potential buffer-overflow attackers considerably. Nowadays, however, if you want to exploit a well-known buffer-overflow vulnerability, all you need to do is perform the right type of Google search to obtain exploit tools, each complete with shellcode for a variety of target systems.

People who write exploit scripts and publish them on the Internet are a big enough problem. But they're not the only actors in the threat equation; if you're the kind of person who enjoys arming script kiddies, it's only a little more work to automate the exploit completely and package it up into a worm or virus.

Viruses, of course, can't propagate themselves; they're always embedded within something else, for example, e-mail attachments or executable files. Worms, which propagate themselves, are much scarier—they're essentially viruses with wings. In fact, if you were to watch your log files during a worm attack, you'd have a hard time distinguishing it from an attack conducted by a human being. A worm is sort of an attack robot.

Thus, attackers either can be humans or pieces of software. The good news is, because they exploit exactly the same sorts of vulnerabilities, the defenses are the same for each. The bad news is, attack scripts, worms and viruses have shortened exponentially the amount of time you've got between the time a vulnerability is discovered and publicized and the time your system probably will be probed for that vulnerability by an attacker.

Defense Scenario One: Firewall Policies

Now, let's begin matching these threats with defenses. This is where the risk-based approach becomes really important.

If you take an absolutist view of security, defense is simple. You choose software based not on the best combination of functionality, supportability and security, but solely based on security. Because security is your main software criterion, all you need to do is keep it patched, and all is well.

You probably also configure your firewall to trust nothing from the outside and to trust everything originating from the inside, because, of course, all outsiders are suspect and all insiders are trustworthy. In fact, software patches and firewall rules are so important in this view that practically nothing else matters.

And indeed, software patches and firewalls are important. But the degree to which we depend on patches and the way we use firewalls is somewhat different if we take the trouble to think about the real risks they're meant to address.

Consider the scenario I've sketched out in Figure 2. A firewall protects a DMZ network from the outside world, and it protects the internal network from both the outside world and the DMZ.

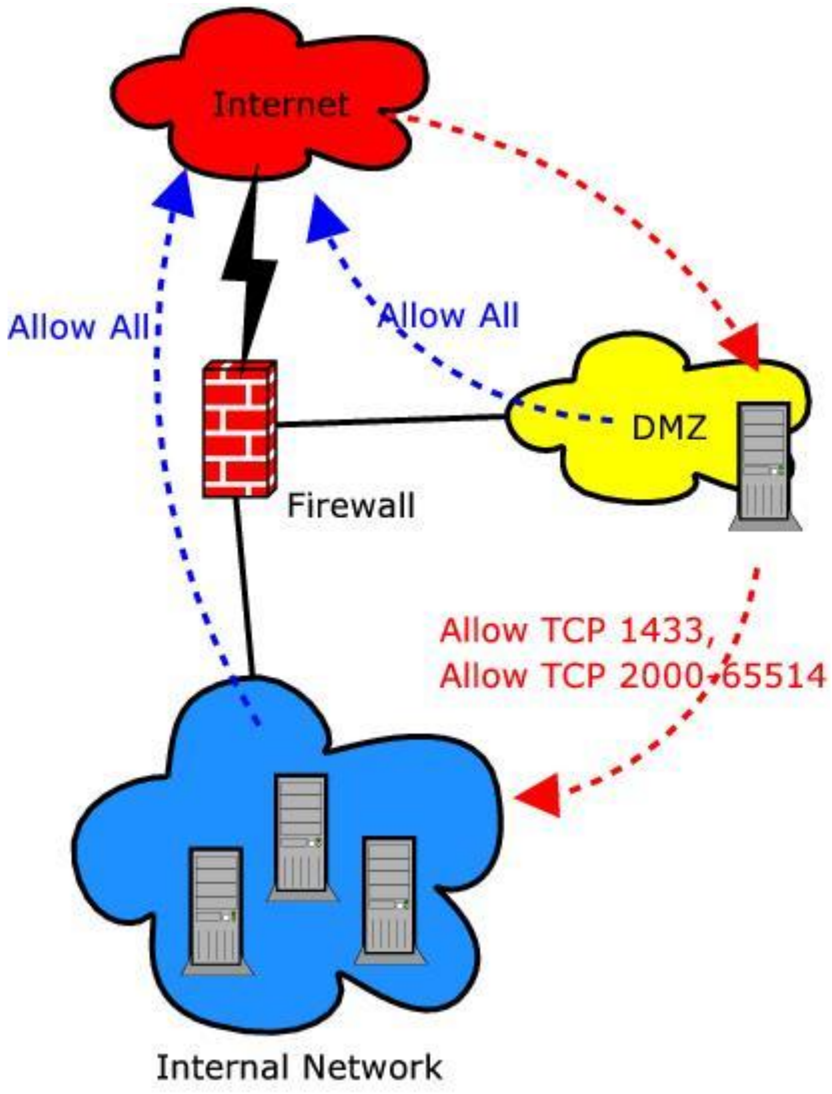


Figure 2. Simple Firewall Scenario

The firewall rules, shown in Figure 2 by the dotted lines, might look like this:

1. Allow all Internal hosts to reach the Internet via any port/protocol.
2. Allow all DMZ hosts to reach the Internet via any port/protocol.
3. Allow all Internet hosts to reach the DMZ via TCP port 80 (HTTP).
4. Allow the DMZ Web server to reach Internal hosts via TCP ports 1433 and 2000–65514.

On the face of it, this might seem reasonable enough. Internal users need to do all sorts of things on the Internet, so restricting that access is a hassle. The DMZ needs to do DNS queries for its logs, so why not give it outbound Internet access too? And there's a back-end application the DMZed Web server needs to access on the internal network that involves a database query on TCP 1433 plus a randomly allocated high port that falls within some finite range nobody's managed to document. So, the easiest thing to do is open up all TCP ports higher than 1999.

But let's consider three plausible risks:

1. Internet-based attacker compromises Web server and uses it to attack other systems on the Internet.
2. Worm infects internal system via an RPC vulnerability, and the infected system begins scanning large swaths of the Internet for other vulnerable systems.
3. Worm infects the internal system and starts backdoor listener on TCP 6666. Attacker compromises Web server, scans firewall, detects well-known worm's listener and connects to the internal system.

In the first risk scenario, we've got an obvious legal exposure. If our Web server is compromised, and our firewall isn't configured to restrict its access to the outside world, we may be liable if the Web server is used to attack other systems. Restricting the Web server's outbound access only to necessary services and destinations mitigates this risk. In practice, a typical DMZed Web server should require few if any data flows to the outside world—its job is responding to HTTP queries from the Internet, not initiating Internet transactions of its own.

In the second scenario, we have a similar exposure, though the network performance ramifications are probably greater than the legal ramifications (all that scanning traffic can clog our Internet uplink). Again, a more restrictive firewall policy around outbound access trivially mitigates this risk.

The third scenario may seem a little more outlandish than the others—what are the chances of a worm infection on the inside and a Web server compromise in the DMZ both happening at once? Actually, they don't have to occur simultaneously. If the worm sets up its backdoor listener on TCP 6666 but then goes dormant, it may not be detected for some time. In other words, the Web server's compromise doesn't need to occur on the same day, or even in the same month, as the worm infestation if the infected system isn't disinfected in time. As with the other two scenarios, a more restrictive firewall policy mitigates this risk and minimizes the chance of the internal worm infestation being exploited by outsiders.

Besides being mitigated by more restrictive rules, these three risks have another important commonality. You don't need to predict any of them accurately to mitigate them. Rather, it's enough to think “what if my inbound firewall rules fail to prevent some worm or virus from getting in, and unexpected types of outbound access are attempted?”

I can't stress strongly enough that it's important not to focus exclusively on attack prevention, which is what inbound firewall rules do. It's equally important to think about what might happen if your preventative measures fail. In information security, pessimism is constructive, not defeatist.

I also hope it's clear by now that my point isn't that firewall rules are the answer to all your Linux risks. The point is that effective firewall rules depend on you considering not only known threats, but potential threats as well.

Defense Scenario Two: Application Security

So, if firewalls aren't the panacea, what else must we do? Earlier in this column, I identified sloppy configurations as a major category of vulnerabilities; the flip side of this is that careful configurations are a powerful defense.

Suppose I've got an SMTP gateway in my DMZ that handles all e-mail passing between the Internet and my internal network. Suppose further that my organization's technical staff has a lot of experience with Sendmail and little time or inclination to learn to use Postfix, which I, as the resident security curmudgeon, consider to be much more secure. Management decides the gateway will run Sendmail. Is all lost?

It doesn't need to be. For starters, as I've stated before in this column, Sendmail's security record over the past few years actually has been quite good. But even if that changed overnight, and three new buffer-overflow vulnerabilities in Sendmail were discovered by the bad guys but not made known to the general public right away, our Sendmail gateway wouldn't necessarily be doomed—thanks to constructive pessimism on the part of Sendmail's developers.

Sendmail has a number of important security features, and two in particular are helpful in the face of potential buffer overflow vulnerabilities. Sendmail can be configured to run in a chroot jail, which limits what portions of the underlying filesystem it can see, and it can be configured to run as an unprivileged user and group, which minimizes the chances of a Sendmail vulnerability leading directly to root access. Because Sendmail listens on the privileged port TCP 25, it must be root part of the time, so in practice Sendmail selectively demotes itself to the unprivileged user/group—this is a partial mitigator, not a complete one.

Being chroot-able and running as an unprivileged user/group are two important security features common to most well-designed network applications today. Just as a good firewall policy aims for both prevention and containment, a good application configuration also takes into consideration the possibility of the application being abused or hijacked. Thus, the true measure of an application's securability isn't only about the number of CERT advisories it's been featured in, it also must include the mitigating features natively supported by the application.

Conclusion

The risk-based approach to security has two important benefits. First, rather than always having to say no to things, security practitioners using this approach find it easier to say “yes, if” (as in, “yes, we can use this tool if we mitigate Risk X, contain Risk Y” and so on). Second, by focusing not only on prevention of known threats but also by considering more generalized risks, the risk-based approach fosters defense in depth, in which layered controls minimize the chances of any one threat having global consequences (firewall rules plus chrooted applications plus intrusion detection systems and so on).

I hope you've found this discussion useful, and that it's lent some context to the more wire-headed security tools and techniques I tend to cover in this column. Be safe!

Mick Bauer, CISSP, is *Linux Journal*'s security editor and an IS security consultant in Minneapolis, Minnesota. He's the author of *Building Secure Servers With Linux* (O'Reilly & Associates, 2002).

Linux for Suits: Grass Roots vs. Giant Roars

Doc Searls

Issue #129, January 2005

While big-name companies scramble to protect business models, this company is making open-ended devices that give customers the right to control their own telephone and media experiences.

In January 2003, after a big PR buildup that included coverage in Reuters and *USA Today*, Kunitake Ando, president and CEO of Sony, announced in a keynote to the Consumer Electronics Show (CES) that his company and Matsushita would lead the rest of the industry's giants into a future of “always on” and interactive devices running a new Linux distro.

That following July, LinuxDevices wrote, “What began as a small but powerful call for an embedded Linux collaboration among Japanese consumer electronics (CE) manufacturers last December turned into a roar today, as eight consumer electronics powerhouses proclaimed the establishment of the CE Linux Forum (CELF).” Members included Sony, Matsushita, NEC, Philips, Samsung, Sharp and Toshiba. Later, the roster grew to include IBM, Mitsubishi, Metrowerks, Motorola, Nokia, LSI Logic, HP, Fujitsu, Hitachi, Phoenix, Sanyo and MontaVista.

Naturally, I expected to hear an even bigger roar for Linux at CES/2004 the following January. Instead, out of more than 3,000 exhibitors, only 11 bothered to mention Linux in their show guide texts. Except for RealNetworks, none of them were A-list companies. Today, the CELF site appears moribund. The copyright says 2003–2004. The last update was in February 2004. When I write to the e-mail address on the site, I hear nothing back.

So, rather than a great roar by a choir of giants, CELF instead looks like what old hands in Silicon Valley call a Barney agreement. That's where “partners” say “I love you, you love me”, and not much else happens.

Even if CELF eventually does produce a working distro, it reminds us in the meantime that Linux is fundamentally a grass-roots phenomenon. It's bottom-up, not top-down. I don't mean to discredit IBM, HP, Novell, Oracle or any of the other BigCos that promote Linux, support its development and fly the penguin flag. I do mean to credit the little guys who not only develop Linux, but deploy it in the marketplace. Especially the ones who deliver and not merely promise.

Take Unication, one of the 11 exhibitors waving the penguin flag at CES/2004. Unication is a Taiwan-based company that has, among other things, made Motorola pagers for the past 13 years. Unication recently has branched into building wireless gateways, servers, VoIP products and PDAs, all leveraging Linux's virtues as free and highly useful building material. The result is some very capable stuff, unencumbered by the product marketing strategies practiced by Unication's giant competitors.

Case in point: Unication's SC-203, an 802.11g-based router that delivers seven Web services in one device: Web, e-mail, printing, voice over IP (VoIP), VPN, firewall and media streaming, including MP3 audio and MPEG video. It's not hard to imagine the possibilities. Best of all, it doesn't come with any big vendor's digital rights management (DRM) or proprietary software lock-in. It's a box of open standards, ready to use. So, when you think about uses, you don't have to wonder what Sony or Philips or Apple or HP has stuck in there to please their lawyers and “content partners” while keeping you from doing any darn thing you want.

For example, the SC-203's VoIP SIP server, which works like a proxy server, supports VoIP calls over wired or wireless (Wi-Fi) connections. Users can make VoIP calls from any open Wi-Fi hotspot, through the SC-203. VoIP calls require paired FXS (phone) and FXO (line) devices. These are available from a variety of makers. Unication's server-side VoIP gateway is the WG-205, which comes in two models: one with two FXS ports and the other with one FXS (phone) and one FXO (line) port. The WG-205's default server IP points to the SC-203. Private IPs are permitted behind the SC-203 to make VoIP calls. The WG-205 either can connect to the SC-203 LAN port or connect independently through an ISP, as long as the Net is available.

The company also makes a Wi-Fi and Ethernet-based VoIP phone you can carry on the road. The U-Phone features include call holding, call forwarding, caller ID, configurable ring tones and melodies, a FIFO log of the last ten calls and the ability to redial any of them. It has a talk time of 2.5 hours and a standby time of 30 hours. So you carry a VoIP phone number wherever you go, ready for use when you're within Wi-Fi range of the Internet. You need some smarts to make it all work, but there's an influential market (that would include *Linux Journal* readers) for whom that's exactly the idea.

Several years ago, David Isenberg famously got himself fired from AT&T by writing a landmark essay titled “The Rise of the Stupid Network”. It was an argument for reversing the phone company's traditional approach, which was to maximize (preferably proprietary) intelligence inside the network and to restrict available use of the network by attached devices. Isenberg said the ideal model instead was making the network as stupid as possible in the middle, supporting unlimited and unrestricted intelligence connected from the outside. In other words, he wanted AT&T to build services that respected the wisdom of the Net's end-to-end architecture.

What Isenberg recommended was the corporate equivalent of a sex change, and AT&T couldn't bear to think of it. Smart middles always have been the telco and cable-carrier equivalent of the top-down producer-to-consumer worldview of every large producer in the Industrial Age. Smart-middle thinking is what accounts for the asymmetries of your cable and DSL connections. Asymmetrical bandwidth (fat-down, thin-up) certainly can be justified by typical usage patterns. But they're also deployed to restrict customers to the roles of consumers. There is no evidence that the prospect of consumers is even thinkable to most telephone and cable companies. Never mind that many of the Net's own infrastructural building materials, including countless servers running the LAMP suite on Linux, are products of the demand side, supplying itself.

So is it any surprise that we've hardly heard a peep about CELF? The very notion of “consumer electronics” presumes restricted autonomy on the demand side. Whatever CELF produces, we can be sure Linux in most cases will be too deeply embedded to serve as a freely useful platform for anybody other than the manufacturer. I hope they prove me wrong, but I'm not holding my breath.

Back at last year's CES, the coolest device I saw was Unication's Magic PDA. It's three devices in one: a PDA, an MP3 player and a VoIP phone. At the time of this writing, the phone uses only the G.711 codec, which typically is used for intranet calls. The next phase, I'm told, will implement all VoIP properties, including the support of G.723.1, G.729, G.726 and G.711u/a codecs, plus electronics to cancel echoes and suppress noise. What I instantly loved about it, however, was its ability to broadcast MP3 files on any frequency of the FM radio band, so you can listen to it on a car radio or home audio system.

I normally don't like to plug any company, but Solon Lee of Unication has been a helpful and patient correspondent while I worked on this piece, and I love to see pioneering companies like his compete in a market dominated by roaring giants. Especially when those giants make only noises where their products ought to be. If you can help Solon and his company find regional agents and distributors, anywhere in the world, drop a note to Unication at salesenquiry@uni.com.tw.

I'm also interested in progress reports on CELF and anything else anybody is doing with Linux at CES/2005. I'll be there, rooting for the roots.

Resources for this article: /article/7861.

Doc Searls (info@linuxjournal.com) is senior editor of *Linux Journal*. His monthly column is Linux for Suits and his biweekly newsletter is SuitWatch. He also presides over Doc Searls' IT Garage (garage.docsearls.com), a sister site to *Linux Journal* on the Web.

EOF: 441 Reasons to Go Linux

Brooke Partridge

Issue #129, January 2005

HP's community center in Mogalakwena, South Africa, produced an industry first—a four-headed PC.

Walk around any of the four computer labs in the Computer Science Department at the University of the North (UNIN), 150 miles from Johannesburg, South Africa. The first thing you notice is that half of their 250 computing desks have no computers on them.

The lab technician, Melt van Niekerk, will tell you that the aging machines they do own are used 12 hours a day by 2,800 students and the warranties expired two years ago. Freddy Nailana, Head of the Computer Science Department, explains that his budget was cut from 2.5 million rand (about $400,000 US) in 2001 to only 130,000 rand last year.

The challenges could not be more evident. How can UNIN increase student access to current technology for the lowest possible price, without sacrificing warranty and service?

At HP, we think we have the answer in the Multiuser 441 desktop solution. This desktop system allows four users to work simultaneously and independently on the same CPU, hence the name 441, or four-for-one. We began with a modified Linux kernel and added four monitors, keyboards, mice and sound and graphics cards. The result is a desktop system that costs about half the price of four standalone desktops.

Linux was the key to developing the 441 solution. It's inherently a multiuser, multitasking operating system. It has great security, stability, power and cost effectiveness—additional elements required by these markets.

When we launched the HP 441 Multiuser desktop solution in South Africa in March 2004, people understood they were looking at a unique approach to solving emerging-market challenges. The four-users-for-one-CPU configuration came with a low price and no special licenses. It also has a healthy bundle of preloaded educational software and a solid warranty, making it ideal for cash-strapped schools.

At HP, we've received a flood of inquiries from people as far afield as Brazil, Austria, Canada, Malaysia, New Zealand and Egypt—all of them asking, “When can I have one?”

This is a good problem to have. The demand we're seeing is a result of a unique approach to designing solutions for emerging markets. Much of the market research, product R&D, marketing, distribution and strategic rollout of the 441 has followed a nontraditional path within HP.

The Emerging Markets Solutions team at HP manages two “solution incubation sites”, one in India and another in South Africa. Known as HP i-communities (i for inclusion), both facilities are located in relatively remote, rural areas that have not seen the kind of information technology explosion that has transformed urban centers like Bangalore and Johannesburg.

The HP i-communities serve as test beds for products and services that are tailored specifically to the needs of rural emerging-market communities. Through our deep engagements there, we learned that we couldn't simply take existing HP products and make them available to people in remote areas. We had to work closely with local users to understand their needs—which are very different from those of customers in developed economies—and then design solutions to meet them.

Partnering with numerous public and private organizations, i-community teams have initiated an impressive array of technology-based social and economic development programs. The challenge—and opportunity—for HP is in commercializing these solutions for other emerging markets around the world.

The 441 is the first HP product commercialized for this environment. It was piloted at the Mogalakwena i-community in South Africa and is being used in call centers, tribal authority offices and schools around the province. Since early 2004, the 441 has been commercially available throughout South Africa.

The introduction of the 441 has even shown up on the radar screen of local and national government officials. The South African government is a strong proponent of open-source software. It is seen as a strategic way to lower development costs, localize standard applications into the nation's 11 official languages and build the skills set of local software developers.

The support of Linux from governments across emerging markets makes the 441 a very timely product. At HP, we're excited about the 441, but we're also excited about the process that led to its development. Deep engagement in rural, developing economies is pointing to a vast new market for the company. The new solutions developed for this market will draw on the strengths and advantages of open source.

With this kind of engagement and innovation, we believe we can fill the computer labs at the University of the North for a price they can afford.

Brooke Partridge is Director of Business and Market Development for HP's Emerging Market Solutions Organization, a team that develops and commercializes technology for developing economies.

Archive Index Issue Table of Contents

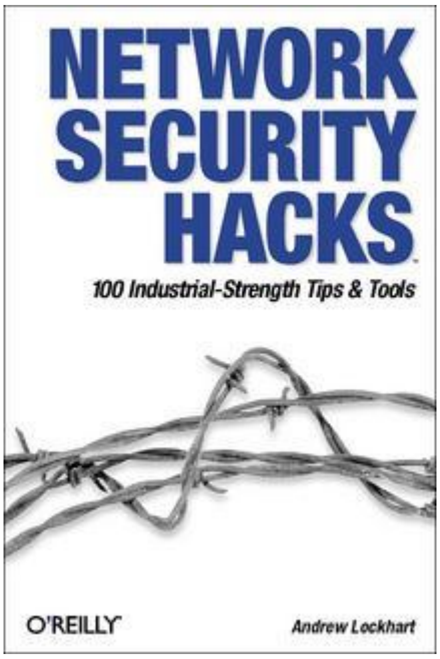
Advanced search

Network Security Hacks—100 Industrial-Strength Tips & Tools by Andrew Lockhart

Alex Weeks

Issue #129, January 2005

This book dispels the myth that simply having a firewall is a complete security design.



O'Reilly, 2004

ISBN: 0-596-00643-8

$24.95 US

With the ever-growing complexity of networks, administrators need an intricate array of tools and skills to ensure their network's security. Andrew Lockhart's practical *Network Security Hacks—100 Industrial-Strength Tips & Tools* provides an abundance of clever hacks to help fill your needs.

As with the other books in the series, this is a compilation of tips collected from real-world users who have faced the same problems that most of us deal with today.

The hacks range from automating simple system administration tasks, such as checking for patches that have been applied, to restricting permissions on filesystems that rarely change. Concepts are explained clearly, making them easy to understand, yet they still offer advice to seasoned professionals. I recommend *Network Security Hacks* for relatively inexperienced administrators as well as for experts.

This book dispels the myth that simply having a firewall is a complete security design. As a consultant for several large companies, I've seen how prevalent this idea really is. Instead, the book first approaches security by discussing how to harden your servers. Of course, it still offers tips on firewalls and packet filtering.

Unlike many others, Lockhart's book is comprehensive; covering tips for UNIX, Linux and Microsoft Windows systems. Because no system or network is impenetrable, meaning every system can be compromised, Lockhart offers a critical approach to minimizing the impact of a security breach. From hardening a server, applied encryption, trending and logging to intrusion detection and incident response, Andrew Lockhart's *Network Security Hacks—100 Industrial-Strength Tips & Tools* is an excellent resource.

LINUX JOURNAL

Advanced search

HP Compaq nx5000

Don Marti

Issue #129, January 2005

With the nx5000, HP gets closer than anyone has yet to a general-purpose business PC running Linux.

Product Information.

- Vendor: Hewlett-Packard

- URL: www.hp.com
- Price: $1,199+ US

The Good.

- Swap out the DVD-ROM/CD-RW drive for a spare battery.

- 1400×1050 screen and keyboard with vi-friendly layout.
- Good sound from JBL speakers.

The Bad.

- Only two CDs of software installed.
- Heavy.
- Mystery lock up.

We covered HP's new Linux laptop at LinuxWorld in August 2004, and they loaned us one for a full review that fall. The nx5000 is a mid-size business notebook PC with a base weight of 5.75 pounds and a base price of $1,199. Display choices are XGA (1024×768) or SXGA+ (1400×1050), for an additional $75. Our review unit had the 1400×1050 screen and a combo DVD/CD-RW drive and weighed in at 2.85kg, or 6.28lb. The processor was a 1.4GHz Intel Pentium M. The nx5000 comes with 256MB to 2GB of memory, and ours had 512MB. A subset of SuSE 9.1 Professional is preinstalled and two CDs are included.

Physical layout is thoughtful, with a good-size keyboard. As on many laptops, the Ctrl keys are undersized and squeezed into the corner of the bottom row with a bunch of other modifiers. For regular use, you'll want to swap Caps Lock and Ctrl. The Escape key is undersized but in a convenient place above the tilde, and the backslash/pipe key is big and where it belongs, above Enter.

Overall size is larger than lightness-crazed road-warrior types will be comfortable with, but if you're transporting it only for commuting or occasional trips, the extra size and weight could be a good trade-off for you.

HP claims a full working day of battery life with both batteries installed, but laptop battery life is a notorious “your mileage may vary” measurement. Under light use with wireless on, including some text editing, Web surfing, listening to Internet radio and uploading the photo for this article, the nx5000 with one battery lasted more than 4.5 hours. HP installs Thomas Renninger's powersave dæmon, which is a nice touch.

Hardware Support

The Atheros a/b/g card was not configured out of the box, but a quick point-and-click session to set it to DHCP brought it up on an 802.11b network. The installed Atheros driver is proprietary. There's one other proprietary module installed, the slamr module for Winmodem support.

Laptop audio is usually tinny and awful, but the nx5000's speakers, branded with a JBL Pro logo, are consumer audio quality and plenty good enough for

Internet radio, playing games or watching a DVD. If you use the nx5000 as your home entertainment system too, the weight starts to look not so bad. Speaking of watching DVDs, the included DVD-playing software is InterVideo's LinDVD, which is DVD CCA-licensed. Playback was smooth, even with tasks running in the background. For day-to-day use, it might be more practical to swap out the DVD drive for the extra battery, use an independently developed DVD player that lets you play movies from the hard drive and simply connect an external drive when needed.

The volume controls for the speakers and headphones work separately, and the KDE volume control is configured to control speaker volume only. We had to go to the YaST volume settings control panel to turn up the headphones. This is a little confusing to start with, but the right thing for those times when you want to listen in private and not annoy everyone else in the library or café when you forget to plug in the headphones and an instant message comes in. A little user-interface help is needed here.

Devices

Everything onboard works, but how well is the system configured to work with the external USB devices that are your eyes and ears on the road?

We plugged in a brand-new Canon PowerShot S410 camera and fired up the preinstalled Digikam to pull the photos off with no configuration needed.

Plugging in a Sony DCR-HC20 MiniDV camcorder meant a little command-line work. The IEEE 1394 modules were installed, and following the instructions in Marcel's column in the December 2004 issue, we modprobed them in. Kino was not installed, but we brought in a copy from a SuSE 9.1 Professional DVD and captured some video.

It's understandable that a nonlinear video editing tool is missing from a default laptop install, but also strangely missing was OpenSSH, which people at *Linux Journal* use constantly. We pulled that in from the SuSE DVD as well. If you invest in this laptop, you also might want to pick up a copy of the full distribution so that you can get your favorite tools just as easily.

Support

Fortunately, for our ability to test HP's Linux support, we found one thing that didn't work—the front-panel volume and mute buttons. We got through to a Linux support person quickly, but the nx5000 he had seemed not to have the same load as the review unit, and the solution that got the review unit working didn't work for him. The solution was to run LinEAK, which was installed but not running in our KDE session.

Finally, right before we sent the laptop back, one person was able to get the nx5000 to lock up twice in one day simply by editing a file with Vim in a Konsole window. Ctrl-Alt-Backspace wouldn't work to kill X, and we weren't able to duplicate the problem.

With the nx5000, HP gets closer than anyone has yet to a general-purpose business PC running Linux. If you want to make your company's Linux desktop migration a treat of working in café and enjoying media instead of a chore, get one to evaluate. Dorm-room or small-apartment dwellers can consider this as a good main computer and entertainment system.

Don Marti is Editor in Chief of *Linux Journal*.

Archive Index Issue Table of Contents

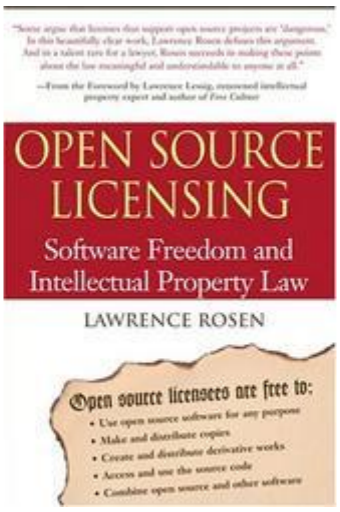
Advanced search

Open Source Licensing: Software Freedom and Intellectual Property Law by Lawrence Rosen

Don Marti

Issue #129, January 2005

If you ever have trouble getting the right to use free software at work because of license concerns, buy a copy of this book.


OPEN SOURCE LICENSING
Software Freedom and Intellectual Property Law
LAWRENCE ROSEN
Open source licensees are free to:
• Use open source software for any purpose
• Make and distribute copies
• Create and distribute derivative works
• Access and use the source code
• Combine open source and other software

Prentice Hall, 2004

ISBN: 0-13-148787-6

$39.99 US

Software licenses are like pluggable authentication modules—bad to try to re-implement yourself but important to get right if you want to be secure. Technology attorney Lawrence Rosen offers a manageable introduction to the subject in this book. If you ever have trouble getting the right to use free software at work because of license concerns, buy a copy of this book. If your company is planning to release free software, *Open Source Licensing* gives you the background to get the most out of your meetings with a lawyer about the license.

This book is a useful field guide to the rights and obligations that the common free software licenses offer and their strengths and weaknesses. It also covers the essentials of copyright and patent law as they apply to software. Rosen also introduces his new licenses, the Open Software License and Academic Free License, which he says fix yet-unexploited legal bugs in older licenses.

For someone who was motivated to write his own set of software licenses, Rosen is generous to the industry-standard GNU General Public License (GPL). He gives the GPL a clean legal bill of health, which makes this book helpful when deciding to use and contribute to GPL-covered software. But he does offer a clear explanation of why a software author would want the additional teeth that his new licenses offer. By binding users to a contract, he lets the licensor set the venue for any lawsuit over the license, insist on attorney's fees and obtain other advantages in court.

This book does an especially good job of covering how the common open-source licenses handle software patent threats and the differences in the patent defense measures in each license. However, it would have been helpful to include a discussion of one approach that patent holders have taken when contributing patented methods to GPL software—offering a patent license separate from the GPL but ostensibly compatible with it. Linux contributions from IBM, Red Hat and FSMLabs are licensed this way, under three different patent grants.

Although the book is strong on the legal side, it's weak on what many consider the overwhelming network effects of the GPL and the advantages of keeping new projects compatible with the existing universe of GPL code. It's surprising that a 2004 book that covers both the Mozilla Public License and the issue of relicensing doesn't mention that Mozilla began relicensing to include the GPL in 2001.

The business decisions about what software license to adopt are yours, and this book's power to dispel Fear, Uncertainty and Doubt about licenses and bring FUD victims into the software commons is invaluable. Reading *Open Source Licensing* is an ideal first step in the license decision process.

From the Editor: January 2005 - Security on the Go

Don Marti

Issue #129, January 2005

Now that work is just a verb, not a place, are all your security assumptions wrong?

It's time to question some security assumptions. Regular users' systems are always on an internal network with a firewall between them and the Internet. The only hosts reachable from the outside are a few bastion hosts. Bastion hosts run a strictly limited set of software, and only sysadmins have accounts on them. Computer security depends on physical security, because anyone who breaks into the server room can boot the server from a rescue disk and have his or her way with the files.

Meanwhile, in the real world, you have a copy of the project you're working on and a bunch of confidential e-mail on your laptop, and you're drinking La Minita at Dana Street Roasting Company while you peruse your project's Request Tracker and hold a Jabber meeting with people in three countries.

Public wireless cafés are a lot of great things, but secure corporate networks they're not. Because more and more companies would rather pay for laptops and drop-in office space than cubicles and desktops for all, you can wave bye-bye to the neat security chart with a bunch of stuff between the user and the menacing Internet Cloud.

Linux distributions are starting to offer good support for some encrypted partitions, which do the attacker no good without the key. Mike Petullo takes the process to its logical extreme and encrypts the root filesystem, which means you can encrypt everything (page 62).

The less we trust the network, the more we need encrypted e-mail. At *Linux Journal*, we rolled out GNU Privacy Guard (GPG) for everyone. Encrypted mail isn't the tweaky mess it used to be, now that the common mailers are integrating GPG support. Find out how to make secure mail a part of your work life in Roy Hoobler's article on page 52.

Now that everyone is outside all the time, the problem of removing unneeded software and keeping packages up to date is even more critical. Fortunately, many of the Linux distributions offer easy tools for installing new versions. Jeremy Turner shows off some screenshots on page 46. Meanwhile, we're still experimenting with SELinux, which could lock down even insecure versions of software to contain attacks. James Morris gives us a peek at the SELinux future on page 56.

The new mobile way of working isn't only a burden for sysadmins. Users often prefer to escape from cubicle-land. Why not make your company's Linux migration a productivity and multimedia treat, not a retraining chore? Just as Lincoln Durey's “Dear Laptop Vendor” was going to press in the fall of 2004, HP made the bold move of offering Linux preinstalled on a full-featured notebook computer. We had one at *Linux Journal* to try out, and yes, we're impressed. Get the details, including the results of a support call, on page 74.

Have fun keeping your systems secure for the real world, and if you see me editing the next issue at a coffeehouse, come over and say hi.

Don Marti is editor in chief of *Linux Journal*.

Letters

Readers sound off.

Cliché Busters

I'm in the building industry in the Chicago area and had an interesting conversation the other day with one of my suppliers. In the past I've heard the phrase “You'll never lose your job sticking with Microsoft.” I don't believe that's the case anymore; the supplier I just mentioned, this week, let go of its IT guy of the last 12 years because he wanted to stick with Microsoft and wouldn't consider any Linux alternatives.

—
Joseph B. Roth

Good Articles in October

Man, that October 2004 issue rocks. I particularly enjoyed Forster's “The Politics of Porting”. It's always a good story when someone bets it all and wins.

Hollenback's “Point-to-Point Linux” brought back memories of how we used Linux for T1 connectivity at Wayport. There is a single-port LMC (now SME) T1 card in every Wayport hotel. Hats off.

—
Jim Thompson

Choice in Fortran Compilers

Many Fortran programmers now use Linux, and there are about ten vendors selling Fortran 95 compilers for Linux. I would like to make *Linux Journal* readers aware of two open-source Fortran 95 compilers under development, g95 (see www.g95.org) and gfortran (see www.gfortran.org). g95 is already able to compile many large production codes, as listed on the g95 site. gfortran will become part of the Gnu Compiler Collection, gcc.

—

Vivek Rao

Portugal, Land of Fine Wine

I am an *LJ* subscriber and Linux user. Every month I read the magazine cover to cover and always enjoy every article, especially if it has some electronic device involved, like the USB programming articles or some embedded Linux device like the Linksys wireless router.

One article I always read with special attention is Cooking with Linux by Marcel Gagné. Besides the technical information, I always am excited to see what wine will he talk about each time. And until now, and I think I am not mistaken, he has never mentioned the fine wine from my country, Portugal. I expect Marcel to correct this fault in one of the next issues of *LJ*. I will not give any example of Portuguese wine here, because I am confident Marcel will find the finest brands of Portuguese wine. Regards to all. Keep the good work.

—
Luis Sismeiro

I believe he already may have some wine from Portugal in the cellar. Check page 26. —Ed.

Photo of the Month: *Vin de Pingouin*

Here is an image of a cute wine bottle; Francois may want to buy a case or two for the Linux chef. It was recently featured at a party to celebrate a successful prototype of my Wi-Fi HackTenna Project.

—
Pat Kane

Hosting Provider Supports SCO

Wow, EV1 has a full page ad (November 2004, page 87). Could this be the same EV1 caught consorting with SCO's “Linux License” scams last year?

—
Harold Stevens

Yes, EV1 gave hundreds of thousands of dollars to The SCO Group, the company known for its apparently baseless legal attacks on Linux. At the time, some posters to the EV1 message board wrote that they were Linux users canceling their accounts over this controversial decision. —Ed.

Security Advice

I read Don Marti's October 2004 column with interest. I agree that Linux is quite a way ahead of some OSes when it comes to security. SELinux is an excellent example of this. However, I differ with Don concerning his plan to implement SELinux for “simple bastion hosts such as name servers”. Although SELinux does bring a lot to the table, it is probably overkill for the majority of applications Linux is used for. When you increase the security of a system you reduce its usability. In the majority of cases, following a few simple steps will result in a very secure system without having to implement SELinux.

1) Perform OS installation disconnected from a network and install only those packages that are required for the system to function. 2) Install updated packages from media (CD-R, tape) created on another system. 3) Configure system services to run in as secure a mode as possible. For example, run BIND in a chroot()-ed environment. 4) Consider implementing a host-based firewall solution. 5) Enable comprehensive logging and develop processes that allow you to examine your logs thoroughly. 6) Keep your system packages up to date.

An excellent source of information to help secure Linux, and several other OSes (as well as Oracle and Apache) can be found on the Center for Internet Security Web site at www.cisecurity.org.

—
Keith Rice

On a name server, where no users need to run applications, the extra layer of protection could be worth the setup. See page 56 for more on SELinux. —Ed.

Less Clustering, More USB

Almost time to renew—I got my second reminder a while back. Then the November 2004 issue arrives and contains very little I can even understand, let alone need. Some of your articles are so focused that surely no more than two or three people in the world could benefit from them. Oscar?? Lots of cluster stuff. Event mechanisms???

Big debate on whether to renew. I can see what Marcel's article is about, but can't see anyone needing to do it. Some useful ideas on bash in the Paranoid Penguin article. No Best of Tech Support. Mostly a wasted edition.

Then I see Coming Next Month!!! Entertainment. Just the thing I need help with. Yes—renew. And tell Dave—he'll want to see this too. We'll just hope they address the problems I keep having: sound, video and USB. And hope for more desktop stuff: Bash, gimp, spam control and USB help.

I'd like to end with “Keep up the good work”, but have to make it, “Help me more.” The check is in the mail.

—
Bruce Bales

Penguin Cake for Web Class

After seeing the Tux cake in the November 2004 issue, I just had to send you a photo of the Tux cake one of my LAMP students made to celebrate the end of the Summer semester.

—
Darryl Bedford

Sweden.population++;

Great seeing Sweden featured in your *LJ* Index, October 2004. Unfortunately, the figures are a bit outdated as we are now officially 9+ million living here.

—
Nit Picker
aka Martin S.

Conference on Non-Linux Web Server?

I was cruising Netcraft and saw that the LinuxWorld Conference and Expo site is hosted on Microsoft Windows and IIS. I think a small letter-writing campaign could fix that problem. I think mentioning this in the editorial section would generate enough hate mail for these guys that they would correct their oversight.

I'm sure we could gather up resources to build, configure and maintain that server from volunteer resources. So they would be out only the hosting costs.

—
Dan

It's a good thing hate mail doesn't work, or everyone with software to promote would be flaming you. If you want to attend a conference with a Linux-based Web site, try linuxsymposium.org. —Ed.

Advanced search

UpFront

diff -u: What's New in Kernel Development

Zack Brown

Issue #129, January 2005

Markus Lidel has been named the official **Intelligent Input/Output** (I2O) maintainer. Designed by the I2O Special Interest Group, I2O is a hardware specification that allows hardware to offload I/O processing from the CPU, raising the performance of I/O on that hardware, while at the same time reducing its impact on the running system. Markus is a relative newcomer to Linux kernel development, first appearing in the kernel changelogs in March 2004 with some minor I2O patches for the 2.6.1 kernel. **Alan Cox**, as with so many other projects, was the de facto maintainer at the time and accepted many patches from Markus over the following few months. By July 2004 and the 2.6.8 release, however, Markus in turn was accepting I2O patches from other developers. In August 2004, **Warren Togami** nominated Markus to be the official maintainer, and after an acknowledgement from **Andrew Morton**, Markus updated the Maintainers file in September 2004 to list himself as the official I2O maintainer. Shortly thereafter he initiated a rewrite/reorganization of the I2O code, which was accepted into the 2.6.9-rc2 kernel, in accordance with Andrew's new looser policies on large changes within a stable kernel series.

Kernel development is often fraught with controversy and dispute. Recently the **pwc** driver, supporting Philips Web cameras, erupted in a dispute between the driver maintainer, **Nemosoft Unv**, and *Linux Journal* columnist and kernel lieutenant **Greg Kroah-Hartman**. To support various hardware, a hook to allow binary modules to link into the kernel had existed for a long time in the driver, against kernel policy. Eventually, Greg insisted on its removal, and Nemosoft asked Linus to remove the whole driver from the kernel. Now, under the GPL, the driver author has no legal right to insist on this, but **Linus Torvalds** felt it was important to honor the author's wishes, especially if the code in question was about to be unmaintained. This turned out to be an unpopular decision with people like Alan Cox, who saw the licensing issue as much more cut-and-dried. Nemosoft had released the code and couldn't just take it back. To illustrate his point, Alan said he might as well ask Linus to remove all of Alan's contributions over the years, which would in fact remove quite a significant percentage of the kernel and completely devastate the entire project. Alan's point simply was that it made no sense to honor developers' requests in such cases. After much arguing, **Luc Saillard** reverse engineered the binary module in question and posted a new version of the pwc driver, without the disputed hook.

David Engebretsen decided to stop being the **PowerPC** maintainer and has been succeeded by **Paul Mackerras** and **Anton Blanchard**, both of whom have contributed many PPC patches over the years, along with **Benjamin Herrenschmidt**, **Tom Rini** and many others. David maintained the PPC port as part of his job at IBM and led the team that did the original Linux port to the PPC64 architecture.

Jeff Garzik has created **blktool**, an easier, more generic version of the existing hdparm utility. Like hdparm, blktool can wreak havoc on disks if used improperly. Also like hdparm, blktool still is fairly IDE-centric, though Jeff is working on SCSI, I2O and RAID support. hdparm's IDE-centrism may stem from the fact that it was developed by **Mark Lord**, the original IDE subsystem maintainer back in the early days. Like hdparm, blktool provides a command-line interface to the nitty-gritty details of your disk, allowing the user to make fine adjustments in behavior, that could result in noticeable speedups. The precise interface used by blktool's command line is still in flux; in particular, Alan Cox feels that as long as a new tool is being developed, it should fix some of the problems hdparm displayed, particularly in the command-line interface. Jeff seems amenable to various alternatives, and it looks as though the final tool will have a variety of alternative mechanisms for users to get their points across. The real point seems to be that Jeff's initial attempt has found support among kernel developers, and various folks are hacking it into shape.

BZFlag: www.bzflag.org

Don Marti

Issue #129, January 2005

The name of this network 3-D tank battle game pays homage to Atari's 1980 arcade classic, *Battlezone*. Shoot the other team's tanks and capture their flag, or simply go “Rogue” and shoot whomever you want. There's a large player community and action happening on dozens of servers all the time.

The basics of operating your tank and shooting are easy to learn—but beware: some people are very, very good at this game.

Devil's Pie: www.burtonini.com/blog/computers/devilspie

Don Marti

Issue #129, January 2005

Isn't moving windows manually a drag? If you're running a simple window manager, such as Metacity, but you want the window matching and customizing features of a more complex window manager such as Sawfish, here's an add-on X utility for you. Write an XML config file to match windows you want to customize, and let Devil's Pie position or tweak them for you. For example, you can pin a certain application to appear on all desktops or move all terminal windows with a certain title to their own desktop.

LJ Index—January 2005

- 1. Millions of dollars spent for Oklahoma City's new public-safety Wi-Fi system: 90
- 2. Square miles covered by the new system: 650
- 3. Degree of accessibility to the Net over the system by the public: 0
- 4. Number of cities belonging to UTOPIA, the Utah Telecommunications Open Infrastructure Agency: 14
- 5. Thousands of households in UTOPIA's footprint: 140
- 6. Percentage of UTOPIA that's fiber optic: 100
- 7. Reported percentage of Linux “savings potentials” vs. Microsoft Windows: 30
- 8. Three-year savings on office apps, in thousands of Euros, for large-scale enterprises with 2,000 jobs: 525

- 9. Three-year savings on servers, in thousands of Euros, for large-scale enterprises with 2,000 jobs: 57
- 10. Three-year savings on content management, in thousands of Euros, for large-scale enterprises with 2,000 jobs: 32
- 11. Three-year savings on databases, in thousands of Euros, for large-scale enterprises with 2,000 jobs: 21
- 12. Number of enterprises surveyed for the above study: 50
- 13. Peak gigaflops of the new Cray XD1 Opteron/Linux-based supercomputer in a 12-processor chassis configuration: 58
- 14. Peak number of processors for the new Cray, with 12 chassis in one rack: 144
- 15. Peak gigaflops for a full rack configuration: 691
- 16. Price in millions of dollars of the new Cray: 2
- 17. Speed in MHz of the MIPS-based Sha hu (little tiger) system on a chip for embedded Linux designs, ready to run Linux: 400
- 18. Typical power draw of the Sha hu, in watts: 1
- 19. Millions of students to whom China wants to bring computing: 200
- 20. Number of computers China wants to deliver, per student: 1

- 1–3: CLS Communications, over Business Wire
- 3–6: UTOPIA
- 7–11: Research & Markets
- 12–16: InternetNews
- 17–20: LinuxDevices

On the Web

Embedded systems are what software wants to be when it grows up. Challenges such as using less memory, booting quickly and staying up for a long time without updates are requirements. Whether you're doing an in-car Linux system or developing phone switches for work, our Web site can help.

- If you're curious about what is involved in developing for embedded platforms and targets, check out Dr Richard Sevenich's new Web series, a hands-on introduction to embedded development. Part 1 (www.linuxjournal.com/article/7848) explains how to pick a target for your embedded design. Future articles will discuss hardware setup, getting familiar with uClinux, adding a GUI and both porting and creating new applications. Follow his progress on-line or grab your own target device and become part of the project.
- We've been following Carbot, an in-car computer company, in its quest to reduce OS boot times, a major barrier in the widespread adoption of car-

ready computers and applications. The ultimate goal was to get a five-second BIOS boot time. In the final installment of this series (www.linuxjournal.com/article/7857), author Damien Stolarz reveals whether the team achieved its goal—"throw away all the useless BIOS functionality, show video at the bootloader (that is, GRUB splash screens) and start playing audio as early as we possibly could in the boot sequence".

- Our resident telecom and carrier-grade Linux specialist, Ibrahim Haddad, spends much of his time at Ericsson Canada working on ways to improve Linux for carrier-grade systems. His article "Critical Server Needs and the Linux Kernel" (www.linuxjournal.com/article/7855) outlines four features the kernel needs for deployment on server nodes in mission-critical environments: "a cluster communication protocol, support for multiple-FIB, a module to verify digital signatures of binaries at run time and an efficient low-level asynchronous event mechanism".

Ten Years Ago in *LJ*: January 1995

A UUCP connection is a good way to get your feet wet connecting to the outside world—it will teach you how to manage a news and mail feed.

—Russell Ochocki

We halted one of the PDPs, moved the data input lines over to our PC, and booted Linux.

—Vance Petree

Built with the CS/EE/Math student and serious developer in mind.

—Red Hat Software ad

They Said It

So when you find somebody smarter than you are, just coast along. Your management responsibilities largely become ones of saying "Sounds like a good idea—go wild", or "That sounds good, but what about xxx?" The second version in particular is a great way either to learn something new about "xxx" or seem *extra* managerial by pointing out something the smarter person hadn't thought about. In either case, you win.

—Linus Torvalds on kernel management (lwn.net/Articles/105375)

When you think about it, it makes sense. Linux and open-source products are cheaper, more robust and more secure. Having Microsoft tell us that their products have lower TCO is like them telling us that the Earth is flat. Right-thinking CIOs know that Linux and open-source software result in lower costs and are not likely to be hoodwinked by verbal sleight-of-hand or spurious, vendor-manipulated TCO studies.

—Del Elson, Open Source in Australia, *CXO Today* (www.cxotoday.com)

When open-source code is properly analyzed, there's nothing better. But just putting the code out in public is no guarantee.

—Bruce Schneier (www.schneier.com/blog/archives/2004/10/schneier_securi.html)

Current scholarly publishing models are not economically sustainable. Researchers and students have access to a diminishing fraction of relevant scholarship. But remedies and alternatives are being developed and tested.

—University of California Office of Scholarly Communications (osc.universityofcalifornia.edu)

Best of Technical Support

Our experts answer your technical questions.

Distributing /etc/shadow

I am hunting for a utility that I think already must exist somewhere. Here is the problem. Government computers need to have all passwords updated more frequently these days, including the root password. Until now, we had so many flavors of hardware and OSes that the thought of SSHing a copy of /etc/shadow or /etc/passwd to all machines was a moot point, simply because the different OSes required different entries for root. Overwriting the root entry on a machine with the syntax for the wrong OS was not worth it. I suppose the biggest problem with doing a blind overwriting of the files would result in possibly incorrect shells or login paths for root. However, we have been working at getting rid of all of the non-PC workstations we had (SGIs, Suns, HPs and so on) so we can attack the virus and patches problems with hopefully one OS to worry about. This means we simply can plop a new copy of the root entry for /etc/shadow or /etc/passwd to all machines via SSH.

Do you know if such a tool exists? I imagine some sort of script has been written that can be tweaked easily to propagate the changes. Some machines are on a domain with a DNS server. The ones not running DNS are running NIS. I am not familiar with the DNS ones yet, but I know the ones running NIS still have to have root changed locally. So far, we have been telnetting or SSHing to each machine one at a time to get the new root password in, because the root password won't map to each machine. The machines need the root accounts updated, especially if we were to need to go to single-user mode.

—
Irene Paradis

irene.paradis@us.army.mil

NIS has been used classically to solve this problem. However, there is really only one solution for the root password: you should update the /etc/shadow file. You also could use a RADIUS server.

—
Christopher Wingert

cwingert@qualcomm.com

You can use `rdist` to push many copies of a file out to your hosts. See www.magnicomp.com/rdist. Alternatively, you could disable, or "star out" the root password by putting a * in the encrypted password field of /etc/shadow, and use `sudo` for everything.

—
Don Marti

info@linuxjournal.com

How to Pass an Option to the Kernel?

What does the "Try `linux noacpi`, `linux disableapic` and `linux noacpi disableapic`" suggestion on page 72 of the October 2004 issue mean in response to a Fedora install question? My AMD dual-MP 2800+ regularly crashes and screen dumps. I just noticed a comment about acpi or apic—I need to read and record next time—on the last screen dump. Having just read the article, I was excited to reboot and try those commands, but I couldn't locate them.

—
Doug Baker

cfdbaker@qwest.net

You are asked to pass `noacpi` or `ldisableapic` or `noacpi disableapic` as a command-line option to the kernel. When the bootloader, GRUB or LILO, is asking which OS or kernel to boot, you can add these options. On LILO, press Ctrl-X to get a command line, and then type `linux noacp`. I am assuming that Linux is one of the options in the LILO menu. If this works outs for you, you can add this to /etc/lilo.conf permanently.

—
Usman Ansari

usmansansari@yahoo.com

On the GRUB bootloader, the default for Fedora, the process is similar. Check out the Unofficial Fedora FAQ at www.fedorafaq.org/#otherinstall.

—
Don Marti

info@linuxjournal.com

Testing CPU under Different Loads

I frequently test Linux machines as part of my job and am looking for a way to load the CPU smoothly from 0% to 100% to see what happens to certain applications. When I try to apply a smoothly ramping CPU load, I usually get either 0% or 100% CPU usage. If I try to sleep for very small increments, I get 0% alternating with 100%. Do you know of any tool or proven way to ramp the CPU?

—
Patrick Killelea

p@patrick.net

You could run a program that alternates some CPU-intensive task, such as generating pseudorandom numbers, with calls to usleep. Tweaking the values of BUFSIZE and USLEEP in this program lets me get a range of CPU loads:

```
/* Build with 'gcc -Wall load.c -o load' */
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>

#define BUFSIZE 1024
#define USLEEP 10000

char buf[BUFSIZE];

int main (int argc, char **argv)
{
        int f;
        f = open("/dev/urandom", O_RDONLY);
        while (1) {
                read(f, &buf, BUFSIZE);
                usleep(USLEEP);
        }
        return 0;
}
```

Thanks to Greg Kroah-Hartman for cleaning up the above code. See man usleep. To exercise individual CPUs on an SMP machine, try the CPU affinity system calls covered in Robert Love's article “CPU Affinity” in the July 2003 issue.

—
Don Marti

info@linuxjournal.com

Single-User Mode

How can I enter single-user mode, runlevel 1, at boot time?

—
Arthur Schroeder

showmeyr@yahoo.com

Edit your boot line in GRUB and add a `single` to the command line.

—
Christopher Wingert

cwingert@qualcomm.com

You can type `single` at the LILO or GRUB prompt to boot your Linux machine into single-user mode. If you always want to boot in single-user mode for some reason, you can modify LILO or GRUB and pass `single` as an option to the kernel. Or, you can modify the /etc/inittab file. There is a line at the top of this file—mine reads `id:3:initdefault` on my Red Hat 9.0 Linux box—in which you can replace 3 with 1.

—
Usman Ansari

usmansansari@yahoo.com

Luke 5:37–38

I am attempting to install Red Hat Linux 7.1 on my new Dimension 4600 Dell computer. The installation CD starts, and I have the option to choose the kind of installation I want. Whatever I choose, after the computer starts to recognize my hardware—it recognizes my CD-ROM and hard drives—it stops and freezes. I can do nothing but turn off my computer.

—
Joe Pietro

jm_pietro@hotmail.com

Before you waste too much time, you should use a newer Linux distribution. Red Hat 7.1 is several years old. Chances are you will have much better luck with a newer version. I suggest you use Fedora Core 2. Fedora Core, a branch of Red Hat, always has supported Dell hardware for the most part. You can download it from www.redhat.com.

—
Usman Ansari

usmansansari@yahoo.com

Red Hat 7.1 has no active source of security updates. It sounds like your hardware has some security sense. See fedoralegacy.org for support for older versions of Red Hat Linux. If you want a quick check on whether hardware is working and Linux-compatible before installing, try the bootable CD distribution Knoppix from knoppix.org first.

—
Don Marti

info@linuxjournal.com

Setting Serial Ports for USB-to-Serial Adapters

I have an application that attaches to multiple remote serial devices via multiple USB to serial adapters. Is there a way to specify that each USB device enumerates as a specific USB serial port, regardless of the order in which the USB ports are connected? For example, I always want USB port *x* to enumerate as /dev/usb/ttyUSB*y*. Because this application will be hosted in more than 200 locations, and it is possible that the USB serial adapter might be replaced or upgraded with a newer unit, solutions based on serial numbers of the USB device are not optimum.

—
Jeff Dennison

If you are using the 2.6 kernel, udev can do this matching for you. Simply define a rule based on something unique for a specific USB-to-serial device and use that to name the device. You mention that serial numbers will not work for you —try using the topology of the USB device or something else that you can determine is unique—uniqueness is the key here. If you are using the 2.4 kernel, good luck. You can muck around in the /proc/tty/drivers/usb-serial directory to try to determine which device is attached to which /dev/ttyUSB node, but it's a bit difficult—one big reason to switch to a 2.6 kernel.

—
Greg Kroah-Hartman

greg@kroah.com

Setting Compiler Options for Gentoo

I'm a newbie trying to install Gentoo from a live CD using a stage3 tarball. I've managed to get to the stage for optimizing my distro. I'm supposed to flag various options using GCC make. I need only enough to get working and understand the basics at this time. Any advice?

—
Rebelrouser

Rebelrouser@blueyonder.co.uk

Stick to the settings already given for your live CD if you do not know what to change. These settings already are present in the /etc/make.conf file. Consult the Gentoo installation guide for more information on this and how to install Gentoo properly.

—
Greg Kroah-Hartman

greg@kroah.com

Fedora Install Hangs

I am installing Fedora. During installation, at Display Setting for the monitor, I choose color depth 256 and click OK. But then my screen freezes and the display is unreadable (blue screen). I don't have any command prompt. Please help.

—
Chris

fiston63@hotmail.com

You can decline to configure the graphics card and X. Once you have booted after the installation is complete, try to configure X. Use the `lspci -vvv` command to see what kind of card you have. If support for your video card is not present, try the manufacturer's Web site for available drivers.

—
Usman Ansari

usmansansari@yahoo.com

Faking Out the Oracle Installer

Does anyone know of a way to fool the Oracle 10g installer into thinking Slackware is Red Hat, so it at least tries to install? If not, does anyone know how it detects that Red Hat isn't there?

—
Blake Tullysmith

bdt@vipretech.com

You can use a tool called strace on the installer:

```
# strace oracle-installer
```

From here, you can figure out what the program is looking for when it refuses to install.

—
Christopher Wingert

cwingert@qualcomm.com

New Products

RTLinuxPro 2.1, ACCPAC Advantage Series Version 5.3, SuSE Linux Professional 9.2 and more.

RTLinuxPro 2.1

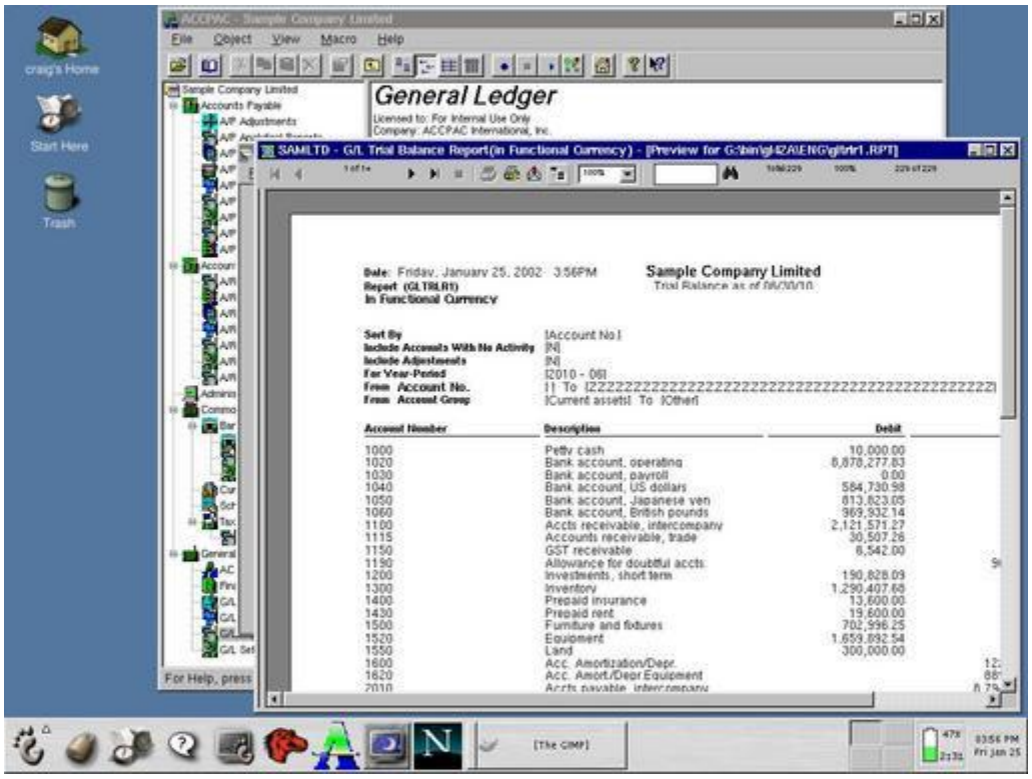
FSMLabs has released version 2.1 of the RTLinuxPro dual-kernel real-time operating system. New features and capabilities of the RTLinuxPro 2.1 include improved name-space partitioning from the companion OS for better reliability, error detection and debugging support; enhanced VME support for industrial and aerospace/defense applications; expanded support for new CPU architectures; and broadened POSIX APIs for code portability and standards-based interprocess, interthread and device communication. The RTLinuxPro Development Kit for embedded cross-development projects also has been updated for the RTLinuxPro 2.1 release and features the Visual SlickEdit IDE.

FSMLabs, Inc., 115 D Abeyta Avenue, Socorro, New Mexico 87801, 505-838-9109, www.fsmlabs.com.

ACCPAC Advantage Series Version 5.3

Version 5.3 of the ACCPAC Advantage Series accounting system offers a new transaction analysis feature that enables all users to codify all transactions and have these codes remain with the transaction for its duration. The presence of these tags enables report generation and detail analysis beyond traditional financial reporting. Version 5.3 also incorporates dozens of feature upgrades, including streamlined security options, throughout available accounting modules. Other new features include the ability to create and execute macros from within the ACCPAC Web application. Built-in kitting in the new inventory control module speeds up processing and fulfillment in conjunction with the improved order entry module.

ACCPAC International, 13700 International Place, Suite 300 Richmond, British Columbia, Canada V6V 2X8, 800-773-5445, www.accpac.com.


ACCPAC - Sample Company Limited
File
Object
View
Macro
Help
General Ledger
Licensed to: For Internal Use Only
Company: ACCPAC International, Inc.
Sample Company Limited
Accounts Payable
A/P Adjustments
SAMLTD - G/L Trial Balance Report(in Functional Currency) - [Preview for G:\bin\gl42A\ENG\glrtrl1.RPT]
For Help, press
craig's Home
Start Here
Trash
[The GIMP]
Fri Jan 25

```
Date: Friday, January 25, 2002  3:56PM          Sample Company Limited
Report (GLTRLR1)                                 Trial Balance as of 06/30/10
In Functional Currency

Sort By                              [Account No.]
Include Accounts With No Activity    [N]
Include Adjustments                  [N]
For Year-Period                      [2010 - 06]
From  Account No.                    [] To [ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ]
From  Account Group                  [Current assets] To [Other]

Account Number         Description                              Debit

1000                   Petty cash                           10,000.00
1020                   Bank account, operating           8,878,277.83
1030                   Bank account, payroll                     0.00
1040                   Bank account, US dollars            584,730.98
1050                   Bank account, Japanese yen          813,823.05
1060                   Bank account, British pounds        969,932.14
1100                   Accts receivable, intercompany    2,121,571.27
1115                   Accounts receivable, trade           30,507.26
1150                   GST receivable                        8,542.00
1190                   Allowance for doubtful accts.
1200                   Investments, short term             190,828.09
1300                   Inventory                         1,290,407.68
1400                   Prepaid insurance                    13,600.00
1430                   Prepaid rent                         19,600.00
1500                   Furniture and fixtures              702,996.25
1520                   Equipment                         1,659,892.54
1550                   Land                                300,000.00
1600                   Acc. Amortization/Depr.
1620                   Acc. Amort./Depr.Equipment
2010                   Accts payable, intercompany
```

SuSE Linux Professional 9.2

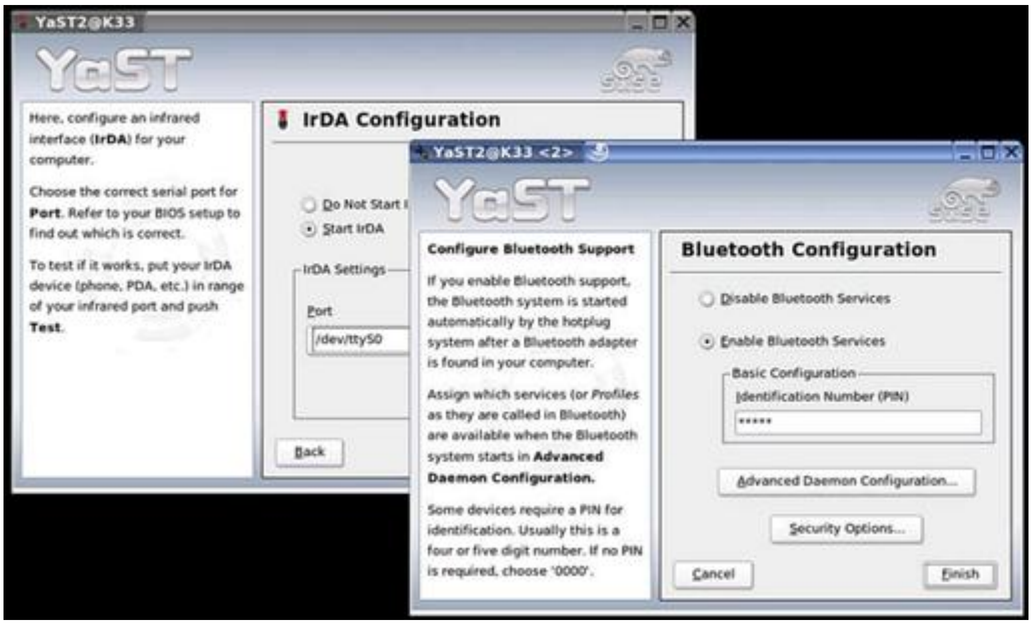
SuSE Linux Professional 9.2 is now available for standard 32-bit x86 processors as well as AMD and Intel 64-bit processors. SuSE 9.2 offers Bluetooth wireless support including automatic recognition of Bluetooth-enabled devices by way of YaST. Included Bluetooth software allows users to connect to and move easily among WLANs and other network connections. Other new features and applications of SuSE 9.2 include KDE 3.3 and GNOME 2.6 desktops; OpenOffice.org 1.1.3; Evolution 2.0; The GIMP 2.0; Inkscape, a vector graphics application; and Nvu, a Web authoring system. Included proprietary applications include TextMaker and PlanMaker, SEP backup software and a demo version of MainActor 5 video editing software.

SuSE, Inc., 1100 Sansome Street, San Francisco, California 94111, 888-875-4689, www.suse.com/us.

Yellow Dog Linux 4.0

Terra Soft Solutions announced the release of Yellow Dog Linux (YDL) 4.0, with 32-bit support for USB-G3s, G4s, G5 Power Macs and through the Freescale Board Support Package and Genesi Pegasos II ATX boards. YDL 4.0 is built on Fedora Core 2 and offers both KDE 3.3 and GNOME 2.6.0 desktops, with a new presentation for both the installer and post-installed desktop environment. Applications included in YDL 4.0 are OpenOffice 1.1.1, Rhythmbox 0.8.3 and Mozilla 1.7; development tools include glibc 2.3.3 and GCC 3.3.3, built on the 32-bit 2.6.8 kernel. Other new features for YDL 4.0 include expanded USB support for peripherals and built-in FireWire support, with bootable FireWire made possible through manual configuration. Dual-head monitor support also is available for PowerBooks and G4/G5 Power Macs. Wireless connectivity is available through the Netgear WG511 54Mb PCMCIA card with D-Link, and Linksys USB card support is planned for the near future.

Terra Soft Solutions, Inc., 451 North Railroad Avenue, Suite 102, Loveland, Colorado 80537, 970-278-9243, www.terrasoftsolutions.com.



YaST2@K33

Here, configure an infrared interface (**IrDA**) for your computer.

Choose the correct serial port for **Port**. Refer to your BIOS setup to find out which is correct.

To test if it works, put your IrDA device (phone, PDA, etc.) in range of your infrared port and push **Test**.

IrDA Configuration

○ Do Not Start I
[✓ Start IrDA]

IrDA Settings

Port: /dev/ttyS0

Back

YaST2@K33 <2>

Configure Bluetooth Support

If you enable Bluetooth support, the Bluetooth system is started automatically by the hotplug system after a Bluetooth adapter is found in your computer.

Assign which services (or *Profiles* as they are called in Bluetooth) are available when the Bluetooth system starts in **Advanced Daemon Configuration.**

Some devices require a PIN for identification. Usually this is a four or five digit number. If no PIN is required, choose '0000'.

Bluetooth Configuration

○ Disable Bluetooth Services
[✓ Enable Bluetooth Services]

Basic Configuration

Identification Number (PIN): *****

Advanced Daemon Configuration...

Security Options...

Cancel   Finish

Sugar Sales 2.0

Sugar Sales and Sugar Sales Professional 2.0, open-source customer relationship management (CRM) applications built on a LAMP platform, now are available from SugarCRM, Inc. New features for Sugar Sales Professional 2.0 include a quotes and price list module that allows a salesperson to access and manipulate data about product pricing and to generate a customized quote that can be saved in PDF format. Both CRMs offer lead management capabilities that enable users to track leads generated from marketing programs, external communications or customer inquiries. Both CRMs also include Web-based calendaring, automatic e-mail alerts for new sales assignments, customizable home pages and an increased number of predefined and dynamic reports.

SugarCRM, Inc., 10080 North Wolfe Road, SW3-303, Cupertino, California 95014, 408-873-9872, www.sugarcrm.com.

Archive Index Issue Table of Contents

Advanced search