AUUGN

The Journal of AUUG Inc.
Volume 23 • Number 3
October 2002

ISSN 1035-7521

Print post approved by Australia Post - PP2391500002

Editorial

Con Zymaris
auugn@auug.org.au

I'm noticing a trend, and I don't think I'm the only one. The trend I'm seeing is the grass-roots re-energisation of the world of Unix, the world of open computing platforms and the world of AUUG. These three are fully, inexorably integrated. When one rises, they all rise. The catalyst which is causal and inspiration to this rise is open source software, in both methodology and philosophy. It's this *renaissance* of AUUG, which is my focus today. Renaissance. A word which means *rebirth* in Latin by way of simple translation, but which conveys much much more. Here's how it comes to mean what we understand of it today.

In the 6th Century, the Byzantine emperor Justinian ordered the closure of the Academy (*nee* Plato's) in Athens, as it was considered to be a pagan remnant. This, and the contemporaneous burning of the Great Library of Alexandria, and the evisceration of its last librarian, Hypatia, bought about what we colloquially know as the Dark Ages. Inspiration, methodology and philosophy were lost to the western world for almost a a thousand., but thankfully not to the Arab world.

Ancient Greek knowledge, garnered from translated Arabic texts, stored in libraries in Islamic Spain in the 12th century precipitated a *rebirth;* an explosion of scholastic and academic knowledge in western and southern Europe: *the* Renaissance, from which flowed the Enlightenment and political and social reforms, the formulation of the modern scientific method and our world as we know it.

These epochal changes flowed from the ideas and tenets of the ancients, to become the inspiration, methodology and philosophy of the modern world. Old, solid principles which had examined, contemplated and sometimes resolved many a complex problem, were worth re-introducing, to help avoid the often painful task of knowledge acquisition from first principles. The results, as we all know, were breathtaking.

The conceptual *bit-mask* of the Dark Ages may also be applied to our industry. I see AUUG fulfilling a spiritually equivalent role with respect to the re-introduction of the ideas, tenets and principles. Many or most of the problems and issues (technical, ethical and social) wrought by complex, vastly inter-connected computing systems, have been examined, contemplated and sometimes resolved by the people who constitute AUUG and its sister organisations. We were there. We have done this. We can do this again. Let's seize this renaissance of advanced computing systems based on Unix, and let us help the IT world by shedding light on how best to move forward. Let us accept this challenge and regain full relevance to both the advanced technical realm and also the popular. Let's teach to the world to *Carpe Unix!*

Cheers, Con

Contribution Deadlines for AUUGN in 2002

Volume 23 • Number 4 – December 2002: **November 15th, 2002**

Volume 24 • Number 1 – February 2003: March 15th, 2003

AUUG Incorporated gratefully acknowledges the support of its corporate sponsor:

AUUGN Editorial Committee

The AUUGN Editorial Committee can be reached by sending email to: auugn@auug.org.au

Or to the following address:
AUUG Inc
PO Box 7071
Baulkham Hills BC NSW 2153

Editor:
Con Zymaris

Sub-Editors:
Frank Crawford, Mark White

Contributors:
This issue would not have happened without the transcription and editorial efforts of Gary R. Schmidt" <grschmidt@acm.org>, Rik Harris <rik@kawaja.net>, Raymond Smith <zzrasmit@uqconnect.net>, David Lloyd <lloy0076@adam.com.au>,

Public Relations and Marketing:
Elizabeth Carroll

AUUGN Submission Guidelines

Submission guidelines for AUUGN contributions can be obtained from the AUUG World Wide Web site at:

www.auug.org.au

Alternately, send email to the above correspondence address, requesting a copy.

AUUGN Back Issues

A variety of back issues of AUUGN are still available. For price and availability please contact the AUUG Secretariat, or write to:
AUUG Inc
PO Box 7071
Baulkham Hills BC NSW 2153

Conference Proceedings

A limited number of copies of the Conference Proceedings from previous AUUG Conferences are still available. Contact the AUUG Secretariat for details.

Mailing Lists

Enquiries regarding the purchase of the AUUGN mailing list should be directed to the AUUG Secretariat.

Disclaimer

Opinions expressed by the authors and reviewers are not necessarily those of AUUG Inc., its Journal, or its editorial committee.

President's Column

Greg Lehey <Greg.Lehey@auug.org.au>

change, n. [a. AF. chaunge, OF. *change* (= Pr. *camge, camje*, Sp. *cange*):- late L. *cambi-um* exchange (Laws of Lombards), f. *cambire*, to change.] 1. a. The act or fact of changing; substitution of one thing for another; succession of one thing in place of another. 11. Sc. An ale-house; = change-house. --*Oxford English Dictionary*

We're certainly living in an era of change, and AUUG has not been excepted. When I joined AUUG a little over two years ago, I had little intention of taking a serious part in its organization. Now I'm reminded of a saying we had at university: "Too years ago I didn't know how to spel presadent, and now I are one".

It's not an easy office to slip into. I spent one year as an ordinary committee member and another as secretary, but I was still unprepared for the office of president, as some of you may have noted at the annual conference. It certainly makes me appreciate my predecessors, and I'd like to thank them all for their support. I'd particularly like to thank David Purdue, the immediate past president, for still being there to help me find my way.

In the last few years things have changed a lot. Some things have deteriorated: chapter activity has declined, and so have membership numbers. I suspect there's a strong correlation there. Other things have got better: we now have strong corporate sponsors, IBM and Sun, and we've just had one of the most successful annual conferences in years, with a 25% increase in delegates over the previous year--and that in one of the most dismal economic climates I can recall. In addition, membership involvement is increasing. We've had the first election to the Board of Directors that either Liz Carroll or I can remember.

Oh yes, the "Board of Directors". We used to call it the executive committee, but it's really Liz, our business manager, who does all the work, so we thought that the name change made sense.

There are other signs that things are looking up: recently the National Office of the Information Economy (http://www.noie.gov.au/) recognized our position as Australia's foremost UNIX and Open Source organization and asked us to help them in their evaluation of Open Source software for use in government departments. At the time of writing we don't have too much to report, but that will change in the course of the next few months.

In addition, happenings in South Australia show that the decline of the chapters isn't inevitable. The SA chapter closed down about seven years ago, but it was revived in April 2001, and it is now very active.

Things don't stop here, of course. Where are we going? There are a number of changes in the industry which have to affect AUUG.

Most notably, of course, there's "Open Source". Or is that Open Systems? Or Open Computing? Or Open Standards? Or just plain Free Software? The Board has discussed the terminology at length and we're still far from agreeing. Anyway, we're moving towards less proprietary solutions. Ten years ago, AUUG was a stronghold of proprietary UNIX. Nowadays you almost have to look for people who use any proprietary UNIX except for Solaris. The majority of the system-related papers at this year's conference were about Linux, followed by BSD (including Apple's MacOS X). Even the vendors of proprietary UNIX are jumping on the bandwagon, including both of our corporate sponsors.

This development has caused a lot of introspection on the board. Linux users already have their own very active organizations, and there are also a number of BSD groups round the country. The Linux users run their own very successful conference every summer, the linux.conf.au. One of the things we've asked in public is: ``is AUUG still relevant?".

Obviously I think we are. Linux is certainly not a passing fad: it's showing the way to the future of UNIX. But we've been in situations like these before, and we found it necessary to have a significant organization. Many of the Linux groups are proud of not having such an organization, and this is almost certainly the reason why we, and not the Linux groups, are representing Open Source to the NOIE.

Still, we have an issue with our relationships with the Linux communities. Effectively, they are us. My experience with LinuxSA shows them to be very much the same group of people who frequent the AUUG SA chapter, and many of them are more interested in BSD than in Linux. One of the challenges facing the Board in the coming year is to find better ways of cooperating with the Linux community. This is probably the key to our membership problems. They have a big advantage over AUUG: they don't have any membership fees. On the other hand, they don't have a number of our membership benefits either, in particular AUUGN. Is that enough? Do we need more membership benefits? If you have any ideas, please let us know (auugexec@auug.org.au).

Part of the problem is the question of our identity: who are we? On our web site we have two statements of our aims. The constitution says:

> *To promote knowledge and understanding of Open Systems including but not restricted to the UNIX system, networking, graphics, user interfaces and programming and development environments, and related standards.*

There's that "Open Systems" word again. We also have another statement, from Michael Paddon:

> *A bunch of people who gather together to talk about the cool stuff they're doing - preferably over a beer.*

So who are we really? I get asked that question a lot, and I haven't come up with a really good answer. A starting point might be to think of ourselves as a technically savvy group of professionals. Some of us may wear suits, but we're not IT bureaucrats. We may hack code for the fun of it, but most of us aren't just hobbyists. That doesn't mean we exclude people from either the IT management scene or the pure hobbyists--we have quite a number of members who fit each description--but those are not our core interests. Our members are some of the cleverest UNIX technical people anywhere in the world. Let's keep it that way.

/var/spool/mail/auugn

Editor: Con Zymaris <auugn@auug.org.au>

The mail keeps on coming, and it's becoming increasingly pertinent and useful. I found one of the compositions so useful and interesting, that I thought I'd include it here, *in toto*. Remember, to join the forum-of-ideas, talk to the *mailman*:
http://www.auug.org.au/mailman/listinfo/talk

From: Steve Jenkin <sjenkin@pcug.org.au>
Subject: SUMMARY Package building software
Many thanks to all those who replied. Credits listed below, then
summary, then original post...

No replies were received for HP-UX. Here are some old links I already had, no idea of their current state/usefulness.
http://www.yacc.com.au/hp.html - YACC's HP-UX Resource List
http://hpux.ee.ualberta.ca/ - Software Porting & Archive Centre for HP-UX

No replies for any of the other Sys V.2 derivaties, SCO or FSF/Tru64.

The closest thing to what I wanted was 'pkgsrc' [from NetBSD] that can package for Solaris & others, besides NetBSD.

Steve Landers, as always, had a very interesting & provocative viewpoint.

For applications, developers should just be delivering a _single_ file. In Tcl/Tk there is a module that allows a local 'loopback' mount of a file [the distribution], combine this with a 'stub' Tcl executable & you have a very powerful way to deliver complete applications without suffering many of the usual problems... Steve even has a way to distribute a single CD containing both Mac & Windows and it 'Just Works' (tm?) The technique is portable to other [scripting] environments, not looked to see it has been done. I don't see this technique as generally applicable to the 'systems' space where most of us play...

Hope this is useful.
SteveJ Wed 9-Oct-2002

Ben Elliston	- autoconf/automake/libtool
Brad Marshall	- Debian (.deb files)
Edwin Groothuis	- FreeBSD 'ports'
grant beattie	- NetBSD & 'pkgsrc'
Gregory Bond	- FreeBSD 'ports'
Jason Thomas	- Debian
Lee Sanders	- Win (.MSI)
Paul Armstrong	- Linux variants
Peter Nixon	- Where to load a page...
Rob Kearey	- Red Hat & rpm's
Scott Howard	- doco on Solaris Packaging
Joe SHEVLAND	- FreeBSD 'ports'
Steve Landers	- Tcl/Tk + MacOS (.app file trees)
Zebee Johnstone	- rpm

Ben Elliston
Autoconf isn't the package for packaging. If anything, it would be Automake. You might want to check the Automake mailing list archives.

As it happens, Tom Tromey and I are working on a new tool to eliminate autoconf/automake and libtool.

Brad Marshall
Debian uses .deb as the packaging format. Useful links are:-
http://www.debian.org/doc/maint-guide/ for the New Maintainers Guide,
http://www.debian.org/doc/developers-reference/ for the Developers Reference, and
http://www.debian.org/devel/ for general developer related stuff.

Edwin Groothuis
http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/ports.html
http://www.freebsd.org/doc/en_US.ISO8859-1/books/porters-handbook/

[building using ports] Was a piece of cake. For example, have a look at the Makefile in http://www.freebsd.org/cgi/cvsweb.cgi/ports/net/dhcpdump/

The most difficult part of making a FreeBSD port is the first port you make. All looks new, all looks difficult etc. But once you've completed your first you will never find it difficult again :-)

Oh euh, if you're doing this for a company, I would like to offer myskills in FreeBSD porting: http://k7.mavetju/unix/freebsd.php
Enough experience ;-)

[[And a good demonstration]]
This is only for the port, which is used to make the package. If you have read the ULRs you know the difference between ports and packages.

The makefile tells where to find the port (master_sites) and the name of the tarball (portname + version + extension which is defaulted to tar.gz). It has the list of man-pages in it (so they will be compressed and stored as cat-files too)

[...]

---

grant beattie
NetBSD's pkgsrc currently support NetBSD, Linux, Solaris and Darwin, with HP/UX and other support in progress.

I use it regularly on NetBSD and Solaris, it is far superior to the Solaris native package system, and it WORKS.

URL:
http://www.netbsd.org/Documentation/software/packages.html

and further information on using pkgsrc on non-NetBSD systems:

http://www.netbsd.org/zoularis/

It also has support for building native Solaris style binary packages which can then be installed with the usual pkgadd tools.

pkgsrc itself has support for building binary packages, as well as including a digital signature (from gpg).

Dependancies are automatically handled, which is great when you have a package with a lot of dependancies (mmm, mozilla, or mplayer for example) :-)

Its primary development platform is obviously NetBSD. It was originally based on the FreeBSD Ports collection, but has since been significantly expanded and multi-platform support is growing quickly.

It can also be used to install binary distributed software, such as Netscape 7 for Linux. It is certainly good to know that everything in /usr/pkg (or whatever LOCALBASE you use) is registered as belonging to a package.

Gregory Bond
FreeBSD (And I think NetBSD and OpenBSD) use a "ports" structure to build stuff from source, with the option of building a binary "package". See

http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/ports.html

for an overview.

Have I built a port for something not already in the ports tree? Sure, several. It can take ages if you are porting a large and difficult program, but that is true no matter how you package up the results.

Jason Thomas
debian has dpkg (which is the same thing as the rpm command – with different options) to install packages. apt-get is used to automate downloading and installing/updating packages from debian ftp/http/rsync servers.

there is a tool called 'alien' which will convert from debs to rpms to tarballs to unpacked debs/rpms/etc. This is available on redhat and debian, i think.

http://packages.debian.org/unstable/base/dpkg.html - not really helpful

http://www.advogato.org/proj/dpkg/ - this one is old

http://www.kitenet.net/programs/alien/

[and attached 2 man pages, not included here...]

Lee Sanders
For windows your package format is MSI. Many different ones, I would recommend
"Wise for Windows 4" or "Installshield Admin Studio"

We use Wise for Windows and have all the software used in the university environment packaged. Takes us less than 10 mins to install/setup the software for a machine.

Paul Armstrong
Debian uses deb packages.
Apt-get is simply a frontend for package management.
dpkg is the main suite of programs for working with (including building) packages.

Package system works wonderfully. Dependencies are checked fully on install and the package supplying the required item is listed.

> Are there any other Linux package variants?

Slackware uses tarballs. Gentoo uses a FreeBSD like build system.

FreeBSD uses tarballs with added information (a bit like Solaris). The main way of adding non-system software is through the ports system. This allows you to build the software from source and it installs it as a registered package so you can modify it as requried. The core operating system is not packaged at all. It is built from
source from which you can select not to build certain sections via
make.conf.

Rob Kearey

[Building rpms] Fairly straightforward, really. You can just use a skeleton spec file and flesh it out. It's generally just a matter of making a buildroot, autoconfig with patches, build and go, just as an example.

Also depends on what you mean by helper scripts. rpm support %pre and %post directives for scripts to be run before and after installation.

[On problems making packages] Well, not quite. You still need to do a filelist, &c, but you can generally do that via a src.rpm. It's important to keep in mind the difference between a src.rpm and the binary rpms it generates. For example:

foo-1.2.src.rpm

rpmbuild --rebuild foo-1.2.src.rpm

might generate

foo-1.2.i386.rpm
foo-1.2-devel.i386.rpm
foo-utils-1.2.i386.rpm

Typically, for a src.rpm, you have a tarball, and a list of patches. Rpm will unpack the source, apply patches, do an autoconfig (for example), apply post-patches if needed, build, make md5sums, and package up according to the spec file. rpm does however have helper apps to make buildroots, that kind of thing.

Any kind of packaging soon becomes a black art, really. rpm is very good at it, though :)

http://www.rpm.org.

Note that rpm has been revamped substantially for rpm 4.1.

Scott Howard
Not exactly what you're looking for, but..
http://www.docbert.org/Solaris/Pkg/

I really need to html'ize it one day...

Joe SHEVLAND
I'm sure someone will go into it in more detail, but in brief FreeBSD uses the Ports system to distribute application packages, which can have dependancies on other packages. The ports system will pull down the relevant source packages from a master site (with configurable alternatives), rebuild the software, branch off and build other dependancies and return to the original build, and finally register the package in a package database (and potentially a lot more features that I don't use, can't recall at the moment, or don't understand ;)

http://www.freebsd.org/ports/

is a good official starting point for further information on the system.

But no, I haven't ever rolled my own port, ...

Steve Landers
I don't know if this is exactly what you are looking for, but you might find http://www.digital-smarties.com/Tcl2002/tclkit.pdf interesting. I presented this paper 2 weeks ago at the 9th Annual Tcl/Tk conference in Vancouver.

Re other OS's doing packages - MacOS X has a really neat facility whereby an entire directory tree becomes an application. Sort of like a package, except that you don't need to "install" just copy your disk. >From memory they are called .app bundles or something similar.

Autoconf is, IMNSHO, part of the problem not part of the solution. Read my paper and you'll see why.

But, in summary, why should application developers care about all that gumph - it should be hidden.

>Is the problem a 30-yo paradigm of 'C' programs and re-inventing the wheel [vs

+building on what has gone before]??

I suspect the problem is system administrators thinking they are
software engineers or computing scientists ;-)

I would argue the set of skills needed to be a good system admin is
almost tangential to the set of skills needed to be a good computing
scientist. RPM (et al) is a system admin solution to a problem that
perhaps should be avoided altogether.

Zebee Johnstone

I use redhat to package stuff.

Basically you need a version of rpm on the build machine and the machine to install on.

You tar up the source, which can be either something to be configged and compiled or else a bunch of files. You write a SPEC file, which has pre and post install shellscript ability, including the ability to run scripts that are in your source. This SPEC file tells the rpm program how to get the files to package, and what to do with the files when installed.

You give the SPEC file config and install commands, and a list of files that will be in the final RPM. when you create the RPM it performs what config, compile, and install commands you have given it, then it gets the files you tell it to package, and packages them.

you can have several sub-packages, I have 4 colocated servers that require the same apache virtual sites, but have different IP addresses and mail requirements and so on, but the same file suffices for logrotation and other scripts. So each virtual server has one SPEC
file, with some files in the source the same and some individual ones for each colo. The SPEC file has a section for each colo, that runs the colo-specific scripts, and packages the files for each colo. It produces 4 different RPMs, but I only have to change things in one file and one source tarball.

I find it useful - decent version control, easy central configuration that can deal with some files on each machine being the same as other machines, and some files are different, plus it can do anything a shellscript can.

see http://www.rpm.org it has a documentation link that has good explanations.

Original Post:

I'm looking for software to build 'packages' on different Operating Systems - good/bad comments on them also solicited. Replies to me & I will summarise within a week or so - could you please indicate in your replies if I CANNOT include all/part of the reply in the summary. I expect to cut/paste the whole of received replies. [And I'd like to make a short 'white paper' from this as well. Any ideas on where it could be loaded?]

- and feel free to share relevant experiences with the whole list :-)

Linux:
Redhat et al use 'rpm' format & the rpm command can build packages. I've never tried to build a package in RPM and don't know if it needs 'helper' scripts or not.

Debian - haven't got a clue ;-) Know there is something like 'appt-get' & '.deb' files,

Are there any other Linux package variants?

NetBSD/FreeBSD/OpenBSD:
Don't know anything about them. I'm sure the Wasabi crew can (&will)
give me a good rundown :-) TIA

AIX:
http://www.bullfreeware.com/
Bull's Large Freeware and Shareware Archive for AIX

Have used the package builder on this site - works fine. Simple, easy - highly recommended.

Solaris:
Have used a 'home-brew' system from IP Australia that built on the standard Solaris commands. Yes, you can hand-build Solaris packages, BUT some helper scripts [that auto generate the directory structures, config files and other proto-files] it really becomes a breeze. Anyone seen a public domain Solaris 'package builder'?
HP-UX:
Have a friend at DFAT who builds HP-UX packages. Don't know how hard it was, or what sort of 'helper' scripts he needed.

Other Unixes:
SCO, Digital Tru64 (FSF), SysV.2 & V.4 variants..

Other O/S:
Do other O/S even 'do' packages??
NT has 'Service Packs' - but these seem to be the equivalent to 'tarballs' - no pkglist, remove/rollback or central inventory. I've been told that SMS has something like packages, but I've never seen it in action or heard more.

Other OSes?? MVS, VMS, do we care??

TIA
sj

To: talk auug <talk@auug.org.au>
From: Michael Paddon <michael@paddon.org>
Subject: AUUG - not just a Linux/BSD/Apple/Vendor group

Steve Jenkin writes:
It is 'Unix' specific, but not (just) Sys Admin, programming, interconnecting, metrics, performance, ... _and_ it makes us '_the_ place to meet peers when you earn your livlihood from Unix'

The trick is to find a tag that describes us well (ubergeekspace) but avoids unfortunate legalities or associations.

Upon reflection, there are several words that can describe us (at least broadly): open, interoperate, technology, professional, enthusiasts, etc.

There are words and phrases I think we should avoid due to excess baggage: unix, linux, internet, enterprise, open system, open source, etc. These are not bad terms, just potentially self limiting in various ways.

As a tag, therefore, I tend to like something along the lines of:
Open Computing Technology for Professionals and Enthusiasts

Of course, an infinite number of tag lines with different emphasis are possible. The key is to find a way to describe ourselves to the world in general (and prospective members in particular) that sums up our raison d'etre. This description should be as extroverted and inclusive as possible.

Michael

Public Notices

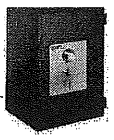
Upcoming Conferences & Events

LISA '02
16th Systems Administration Conference
November 3-8
Philadelphia, PA

IMW 2002
Internet Measurement Workshop 2002
November 6-8
France

OSDI '02
5th Symposium on Operating Systems Design and Implementation
December 9-11
Boston, MA

DATASAFE



Networked backup archiving device

Cybersource/DATASAFE is an entirely new idea in network backup, that guarantees you peace of mind. How does it do this? DATASAFE automatically archives 'snapshots' of your network servers daily. DATASAFE can be installed in minutes and is managed through your web-browser. And unlike tape backups, DATASAFE runs completely unattended, securing your data against the risk of a mislabeled tape or a forgotten backup. Best of all, DATASAFE offers unbeatable purchase value, and by requiring no regular maintenance, very low overall ownership costs.

And when it comes time to restore lost data, DATASAFE makes data retrieval easy -- simply select the files you want from the archive (using your web-browser) and click 'Restore.' You can even select the version and date of each file you want from the archive, going back months or perhaps years.

Specification
DATASAFE is a totally self-contained server appliance supplied as a rack mountable unit. It ships with either 80GB,160GB or 480GB of online archive storage. DATASAFE uses TCP/IP network protocols, and can operate with standard Microsoft Windows Networking (SMB/CIFS on Windows 95, 98, Me, NT, 2000, XP.) Contact Cybersource for Unix/Linux (NFS) and Apple Macintosh support.

DATASAFE comes with a twelve month return-to-manufacturer hardware warranty. Business and Extended Hours Service Level Agreements are also available. DATASAFE is available from Cybersource or through a national dealer channel. Contact Cybersource for more information.

Pricing
80GB Unit: $2950 (inclusive of GST; unlimited users, unlimited servers)
Can provide 6-12 months of online daily-backups for a small organisation

160GB Unit: $3950 (inclusive of GST; unlimited users, unlimited servers)
Can provide 3-6 months of online daily-backups for a medium-sized organisation

480GB Unit: $P.O.A (inclusive of GST; unlimited users, unlimited servers)
Can provide 1-3 months of online daily-backups for a large organisation

Web: http://www.cyber.com.au/cyber/product/datasafe/
Phone: +61 3 9642 5997 Mail: info@cyber.com.au

Cybersource

My Home Network (October 2002)

By: Frank Crawford <frank@crawford.emu.id.au>

It is strange how a little annoyance can lead to big change in an environment. In this case, my main server, bits, started to make a lot of noise, which turned out to the fan in the power supply. This system was made up of all the bits and pieces I'd collected over the years, in particular, the case from my first computer, circa 1987. It is amazing to think that a power supply can still be used after 15 years. Unfortunately, power supply technology has changed over the years, so there is no way I could find a replacement power supply, or a new one that would fit in the old case.

Now, I had been expecting this for some time, and one of my previous upgrades to the system was to install a motherboard which supported both AT and ATX power supplies. So, my first plan was to purchase a new case and ATX power supply. Technically this should work, and in practice it sort of did. The one thing I didn't consider is how the computer cases have changed to match motherboards over the last few years.

When I last upgraded 'bits', while there were a few extras, such as sound chips, were built-in, it was generally by supplying pins to connect external connectors to. Modern motherboards now have the connectors as part of the board and modern cases have cutouts in the box for these connectors. To make matters slightly worse, the previous keyboard standard was for "AT" style connectors (i.e. the big plugs) but is now "PS" style connectors (i.e. the little plugs), and the cutouts are very different. The final difference is that modern motherboards have fewer slots for additional cards because there are so many things built-in to the motherboard, so I was going from 6 slots to 4 slots in the case.

The most amazing thing about all this is that despite all the problems, I was able to find a case that suited and got my system back together in a fashion that worked, although not in the nicest fashion. For example, there were a number of cables that were hanging out the back and I had to leave out an old SCSI card that I no longer used. One thing that made this easier was that the cutouts for the connectors was a separate panel that could be changed depending on the motherboard. By leaving this panel off, I was able to feed lots of the cables out.

Of course, I wasn't really happy with this, and so I quickly made the decision that I would upgrade the motherboard. The one that was in 'bits' was an unnamed brand (well, just some strange one from Hong Kong) that supported Slot 1 cards, and standard SDRAM SIMMS (168 pins). I had a 400MHz Celeron processor on a conversion card for Slot 1 motherboards.

As well, it supported both AT and ATX power supplies, had audio and USB support on-board.

Since I was planning to upgrade this system, I decided to look at all my systems and plan possible upgrades for them. My current systems consisted of either Pentium III or Celeron from around 1998/99. They were all around 450MHz systems, with either 128MB or 256MB SDRAM. One of the systems supported an ATX power supply, the other two motherboards supported either AT and/or ATX power supplies.

At this point I should say that all these systems have been undergoing continual upgrades of such components as disk, video cards and memory, so that now the systems have somewhere between 20GB and 80GB disks, Matrox G550, G450 and Millennium video cards and 128MB to 256MB SDRAM (PC100/133). They all have cheap NICs supporting 10Mb Ethernet all with BNC connectors (i.e. coaxial cable). Given the history of changes there is a mixture of PCI and ISA cards.

Finally, I should also throw in some statistics. Current technology has CPU performance doubling every 18mths, disk capacity (but not necessarily performance) doubling every 12mths and network bandwidth every 9mths. To this I should add some comments once made to me, that I should only upgrade if the performance was double what I currently have. Given it is nearly 3 years since I upgraded, CPU performance should be up to 4 times, which it is, i.e. around 2GHz. On the other hand disk capacity should be nearly 8 times, but as I've been upgrading it progressively, it isn't as obvious. In fact, my disk capacity has increased from around 10GB to over 80GB, so again, I'm right on the curve. If I look at the network performance, it is far more problematic, as over the time it should have increased by a factor of 16. Now, my internal bandwidth hasn't changed (hopefully it will soon), but my external link has gone from 28Kbps to 512Kbps a factor of around 18! So, excluding CPU, I have been on track.

These statistics give some ball park figures for what I need to achieve, but in addition, I want to reuse as much of my current equipment. If possible, only changing the CPU. Given the change in technology over the last few years, just replacing the CPU is out of the question, unfortunately, I soon found out how much.

The first and obvious change was that AT power supplies are no longer supported, hence all systems need to become ATX based. This also implies that the system case needs to change, as the mounting point for the power supplies have changed, as has the power switches.

Now going back to the upgrade for 'bits', I made the decision to maximise the reuse of existing memory, etc, and reduce the cost as far as possible.
For a long time, I've favoured Gigabyte motherboards, and looking on their website, http://www.gigabyte.com, I hunted up a suitable motherboard. To keep the costs down, I looked at slightly older technology, going for a GA-6VEML motherboard, and a 1.3GHz Celeron processor. This is basically 12month old technology, but the motherboard had on-board support for 4xUSB, Audio, Video and NIC, basically everything I had as separate cards for the previous system.

By selecting this card, I was able to reuse my current memory modules, and only required 1 additional NIC (as 'bits' needed to support two NICs). In addition, the on-board IDE controller supports higher transfer speeds, better audio and better video.

Of course, finding this motherboard itself was the next issue, and again, keeping in mind the cheapest option, I elected to go to one of the computer markets. These markets cycles around to various locations, usually once a month, and I was able to check the dates via their web page, http://www.acomputermarket.com. There should be similar markets in your local area.

Okay, so, once this upgrade was complete (and it went fairly smoothly), it is on to looking at my other systems. Again, keeping prices to a minimum, and reusing as much as possible are an important issue, although maybe not to the extent of 'bits'.

Instead of using older technology, I need to move up to what is current, and this brings in a much larger number of changes. To start with, to support the latest bus speeds, new types of memory are required, either RAMBUS or PC1600 (or later). In addition to support the power requirements of Pentium 4 systems, new ATX power supplies, with a rating of 300W and supporting a 12V output are required. The latest technology also requires a new socket (Socket 478) and a Front-Side Bus (FSB) of 400MHz or now 533MHz. This last point is important, as it relates to the maximum speed of the CPU. A FSB400 system can only support Pentium 4 speeds up to 2.4GHz, while the newer CPU (2.53GHz and later) require a FSB533 system. Lastly, all my ISA bus cards have to go, as most new systems only support PCI.

The final piece of the puzzle is the on-board components and the layout of the connectors. In practice, different models of motherboards have the connectors in different orders, often this will need to match the case you have. If you are buying a new case, then you need to match it up with the motherboard, but if you are upgrading, you need to be more selective.

So, what have I decided to do? Well, to start with, I plan to continue upgrading over a few months. I'll start with a motherboard that fits the current ATX case I have, but, I will have to change the ATX power supply to one that includes a 12V output, suitable for Pentium 4 support.
As well, I will go for a slightly older technology, one that supports my current memory DIMMs, so will be a 400MHz FSB, and limit the CPU to 2.4GHz. Such a motherboard is Gigabyte's GA-8ID533. In fact, if I'm lucky (and as of writing this I haven't had a chance to check), I may be able to swap the ATX power supply from 'bits', with my slightly older ATX system, drop in the new motherboard and CPU and increase the CPU

performance from 450MHz to 2.4GHz, with little or no other changes.

Of course the final step in my upgrade plans will include the purchase of a new case, including a suitable ATX power supply, a new FSB533 motherboard, more memory and the fastest CPU available at the time.

On current specs that would probably be a GA-8IEXP with a 2.53GHz Pentium 4, but I expect the new 3.02GHz Pentium 4 to be available by the time I purchase it.

I guess the final item is, how much did all this cost and did I really save anything over buying a whole new machine? That is a bit difficult to really tell, as the upgrades are progressive, and there are a number I am yet to purchased, but the upgrade to 'bits' cost about $100 for the case and about $250 for the motherboard and CPU and gave me a system that will continue to meet my expanded needs for 2-3 years. I expect to get away with a cost of around $500-$600 for the next round, and around $700-$1000 for the final upgrades (including memory, case, etc). This is a total cost of less than $2000 for three systems, that really bring them up to state of the art, and in one case (the last upgrade), something that is difficult to find as a complete system at the present time.

I should add as a closing comment, a number of my decisions here are based on personal preferences. They are items that have proved reliable for me over time and that have given me good service. I'm sure there are other combinations that would work just as well. The upgrades I'm suggesting are something that you should only consider if you are confident about your own abilities, but at the same time, is a good way to learn about what really makes up your computer system. You get to choose every component, and know why and how they work. As an aside, during this process I've found that there have been considerable advances in disk performance technologies over the last few years, and getting these to work cleanly with the new systems will give me a tremendous improvement in my server, with nothing more than swapping a few cables and tuning the system. I've also left out how the things that will be left over will go to improve my lowest end system and even give me enough to build yet another low end beast, if I am so inclined.

In a future column, I'll let you know how my upgrades go, and keep you informed of some of the other changes that go through, for example with disks and video. But for now let me know what you are planning and what you think the best upgrade path is.

AUUGN Bookreviews

Mark White <mark.white@auug.org.au>

AUUGN Book Reviews

This edition of AUUGN sees the re-introduction after a brief hiatus of reviews of books that will be of interest and relevance to the AUUG community. For this issue, we present reviews of two books from O'Reilly and Associates.

We hope to grow this section of AUUGN over the next few issues to become a useful resource for AUUG members. Some assistance would be great - if you have ideas on books you would like to see reviewed, have read a good book lately and are interested in writing a few hundred words about it, or if you are interested in reviewing books on a particular topic, please drop me a line.

Thanks for this issue go to AUUGN Editor Con Zymaris for his support, also to Woodslane in Sydney for providing the review copies.

Enjoy!

Mark White
AUUGN Book Review Editor
mark.white@auug.org.au

Free as in Freedom: Richard Stallman's Crusade for Free Software.

Williams, Sam (2002): O'Reilly & Associates

Imagine for a moment you are a budding author, backed by a traditional publishing house to write your first book. It's to be a biography, and all you need to do is convince the subject to co-operate. Sounds simple, right?

Then imagine that your subject is Richard Stallman, who effectively finds the notion of traditional copyrights not only wrong, but personally offensive and indeed downright evil, and unequivocally tells you so. Loudly.

At least you know that you're interviewing the right guy.

Enter O'Reilly & Associates, publishers of many terrifically useful tomes about various open computing topics, Linux, perl, sendmail, DNS, UNIX etc. ,with an offer to publish the book both in hardcopy and online but under the author's choice of licenses - either the Open Publication License (OPL) or the GNU Free Documentation License (GDFL). RMS acquiesces, and the book is on.

Sam Williams has gone to great lengths here to cover not only Stallman's defining position on free software and the achievements in this area, but also in non-intrusively exploring the early stages of Stallman's life and evolution of his thoughts on the topic of software. Covered in detail are his stints at MIT's famed AI lab, his founding of the Free Software Foundation, the creation of St Ignucius, his technical achievements (emacs, the GNU compilers) and of course his rationale and history behind the GNU General Public License (GPL).

Given Stallman's propensity for debate (and vocal disagreements!) with people about his views, it's not surprising to see other figures in the open source movement pop up in the book. While not always glowing of the legendary RMS persona (Bob Young: 'The biggest problem with Richard is that he treats his friends worse than his enemies'), they are all but universal in their respect of his dedication and unwavering commitment to his beliefs.

While the contents of the book are of course available online (see http://www.faifzilla.org) under the GFDL, the hardcover package (somewhat of a novelty for O'Reilly books) is well-presented and sturdy - a good sign for an enduring work.

To Williams' credit, he has created a good read and quite a fair representation of one of the most visible proponents of open source. (Sorry, I meant 'free software'!). Free as in Freedom is well worth the time of any open source software enthusiast with an interest in digging a little deeper into the psyche of one of the seminal figures of the movement.

Reviewer: Mark White
mark.white@auug.org.au

PROGRAMMING PYTHON, 2ND EDITION.

Lutz, Mark (2002): O'Reilly & Associates

Mark Lutz' Programming Python was the first book published on the then new scripting language. Five years later, the second edition is basically an entirely new book.

With the availability of dozens of other books on Python, the material in Programming Python has been cut, restructured, and expanded and now focuses on applying Python to real tasks. This is not the book to buy if you're looking for a tutorial introduction to Python, a language or library reference, or a cookbook-style collection of sample code. Instead, it explores the language and in particular, the standard libraries, over a series of major topics each divided into a handful of chapters.

The second edition has also been updated to cover Python 2.0, which includes extensions to both the base language and the libraries not covered in the earlier version. Still, Python continues to evolve rapidly, and at the time of this review, 2.2.2 is about to be released. There should be few (if any) problems with the example code from the book but it does not cover newer features like iterators and generators, list comprehensions, rich comparisons and weak references.

The first section covers using Python for tasks for which you might normally use a Unix shell. It works through file and directory handling, environment variables, streams, and text handling, before looking at processes, threads, signals, pipes and so on.

One of the strengths of this section is its coverage of both Unix and Windows. The Python standard libraries are mostly portable, but as might be expected, these sorts of system-oriented tasks see the most difference between the ports to various operating systems, and the issues of writing portable code for Unix and Windows are well covered. Sadly, MacOS (or at least OS9 and earlier) is not given the same treatment.

Like most programming languages, Python has support for numerous GUI toolkits. Tkinter, an interface to TCL's Tk widgets, is the default toolkit, supplied with the interpreter source and available on most platforms. The next major section gives an overview of Tkinter programming, a tour of widget set from Python, and then some complete example programs: a text editor, drawing program, desktop clock, etc.

Starting with the obligatory "Hello World!" (for the record, 4 lines using readable syntax) it works through layout, resizing, events and widget sub-classing in 30 pages of dense but clear text. Geometry management is one of the harder aspect of Tk programming for novices, and the overview chapter in this section does a reasonable job of explaining it without burying the reader in details.

Network programming is the subject of the third major section. Starting with sockets and multiplexing I/O, it moves on to the rich support for various Internet protocols in the standard library. Client-side use of FTP, HTTP, news, and email protocols is covered, resulting in a complete GUI email client.

Server-side web programming forms a large part of this section, starting with simple CGI scripts, and moving up to another version of the email client, this time as a web application. Integration of database access, web application frameworks and a brief mention of XML processing round out the section.

The final chapters cover some loose ends: more on database access, Python data structures, text processing (and parsing), and integration with C and C++. The book closes with some thoughts on Python's strengths and its role as a programming language, and a couple of appendices for reference material.

Frequent sidebars throughout the book point the reader to more detailed coverage of relevant topics, and expand on portability issues or hidden complexities not addressed in the main text.

Programming Python is a difficult book to position: neither tutorial, nor an advanced programmer's reference. I'd recommend it for experienced programmers who're able to pick up the syntax from a quick reference, but need to reduce the learning curve of the standard libraries, or as a second Python book for the less experienced, following a tutorial introduction to the language.

Areas that could use more coverage are XML processing (which is the most glaring omission), Python's built-in testing and documentation features, and the use of CORBA, DCOM or SOAP for writing or integrating distributed applications.

The author, Mark Lutz, has a distinct style that fits well with the never-too-serious Python community. Throughout the book, footnotes help the reader decipher the ongoing Monty Python jokes, but he manages to balance the humour well, and it never seems to detract from the technical material.

Reviewer: David Arnold
davida@pobox.com

Dynamically Tune up a File System

Author: Joseph Gan <joseph.gan@abs.gov.au>

Nowadays, the performance becomes a issue in terms of the file system tuning in Solaris. Changing some of file system's parameters that usually will destroy the data on the file system.

For instance, changing the cache segment block size in the volume of a T3 requires that you delete the existing volume; deleting volume will destroy the data. And the volume on the T3 has to be reinitialised, which can take a significant amount of time for a large disk space.

And also, change the segment size of a LUN in a raid box which needs to delete the existing LUN etc.

Even if changing the parameter of a metadevice, or re-name a metadevice under SDS (Solstics DiskSuite) which needs to un-mount the file system.

How to dynamically change the parameters of the file system without destroy the data on it?

First, the file system has to be created and mounted as one-way mirror metadevice, in the following example, d100 which contends d101 as its submirror:

```
      # metastat d100
      d100: Mirror
          Submirror 0: d101
            State: Okay
          Pass: 1
          Read option: roundrobin (default)
          Write option: parallel (default)
          Size: 10261520 blocks

      d101: Submirror of d100
          State: Okay
          Size: 10261520 blocks
          Stripe 0: (interlace: 32 blocks)
              Device              Start Block  Dbase
State           Hot Spare
              c1t12d0s0                  0     No
Okay
              c1t13d0s0               1520     No
Okay
              c1t14d0s0               1520     No
Okay
              c1t15d0s0               1520     No
Okay
```

Next step is to create a new metadevice d102, which must be the same size of the submirror d101 with a set of new parameters.

For T3, you need a spare disk volume. For the raid box, you need a set of spare disks and so on.

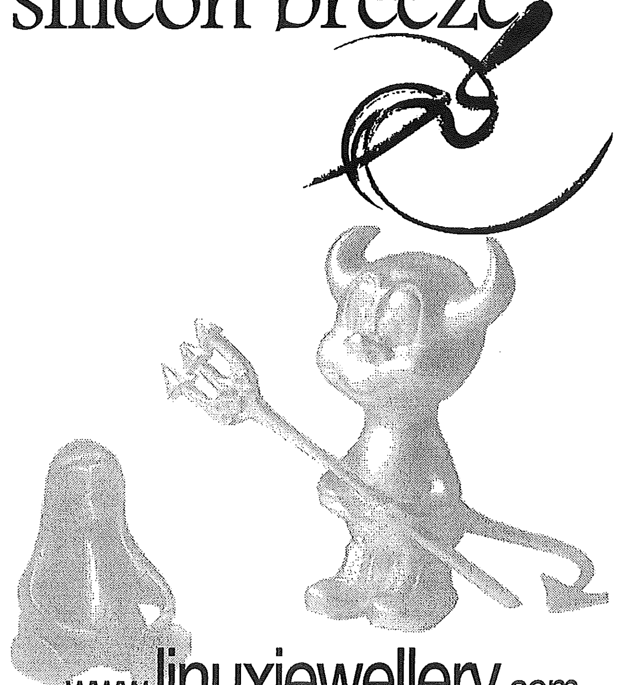
Then, add the new metadevice d102 as the second submirror to d100, resync will automatically take place.

After the resync has done, you have a two way mirrors. One submirror has the old parameters, and the other has the new parameters.

Finally, you can detach the old submirror d101 from d100, and remove metadevice d101 all together.

Now, you have got the file system with a set of new parameters dynamically.

silicon breeze



www.linuxjewellery.com
info@linuxjewellery.com

AUUG Corporate Members

as at 1st October 2002

- ac3
- Accenture Australia Ltd
- ADFA
- ANSTO
- ANU
- Australian Centre for Remote Sensing
- Australian Bureau of Statistics
- Australian Defence Force Academy
- Australian Industry Group
- Bradken
- British Aerospace Australia
- Bureau of Meteorology
- C.I.S.R.A.
- Cape Grim B.A.P.S
- Centrelink
- CITEC
- Corinthian Industries (Holdings) Pty Ltd
- CSC Australia Pty Ltd
- CSIRO Manufacturing Science and Technology
- Curtin University of Technology
- Cybersource Pty Ltd
- Deakin University
- Department of Land & Water Conservation
- Department of Premier & Cabinet
- Energex
- Everything Linux & Linux Help
- Fulcrum Consulting Group
- IBM Linux Technology Centre
- ING
- Land and Property Information, NSW
- LPINSW · Macquarie University
- Multibase WebAustralis Pty Ltd
- Namadgi Systems Pty Ltd
- National Australia Bank
- National Library of Australia
- NSW National Parks & Wildlife Service
- NSW Public Works & Services, Information Services
- Peter Harding & Associates Pty Ltd
- Rinbina Pty Ltd
- Security Mailing Services Pty Ltd
- St John of God Health Care Inc
- St Vincent's Private Hospital
- Stallion Technologies Pty Ltd
- Sun Microsystems Australia
- TAB Queensland Limited
- The University of Western Australia
- Thiess Pty Ltd
- Tower Technology Pty Ltd
- Uniq Advances Pty Ltd
- University of Melbourne
- University of New South Wales
- University of Sydney
- University of Technology, Sydney
- Victoria University of Technology
- Workcover Queensland

Advertisment: Job Positions

Senior Software Developer

Endace is a New Zealand startup company which grew out of the University of Waikato WAND network research group. We develop and market solutions for Internet monitoring and security. Our clients are located in all developed countries of the western world; the US/Canada, Europe and the Asia/Pacific. Our team is highly multicultural, bringing together the strength of people from a dozen different countries. This is a rare opportunity to work on leading edge computer and network technology while enjoying the best of New Zealand's rurual lifestyle.

For our Hamilton Technology team we are looking for a Senior Software Developer. The successful applicant will have:

- had a minumum of five years experience in software
- application development, two years in a commercial
- environment
- an in-depth knowledge of programming in C and other high-level programming languages, as well as assembler
- previous experience in developing UNIX kernel modules, device drivers on freeware operating systems (Linux, FreeBSD, others), or equivalent
- experience in embedded programming
- a solid understanding of software development under
- version control (CVS, RCS or SCCS) and release
- management and engineering
- a background in UNIX system, web server and Internet administration, including system security
- a good understanding of computer networking and the ability to learn new terms and technologies within a short time frame
- excellent interpersonal communication skills
- some experience in coordinating a project with other developers
- the desire to grow with the company and take over more responsibilities, as the need arises

For further information, please contact Endace Human Resources at <hr@endace.com> or phone Selwyn Pellett at +64-9-2610407.

Technical Support Engineer

Endace is a New Zealand startup company which grew out of the University of Waikato WAND network research group. We develop and market solutions for Internet monitoring and security. Our clients are located in all developed countries of the western world; the US/Canada, Europe and the Asia/Pacific. Our team is highly multicultural, bringing together the strength of people from a dozen different countries. This is a rare opportunity to work on leading edge computer and network technology while enjoying the best of New Zealand's rurual lifestyle.

For the support of our international customers out of our Hamilton Technology center, we are looking for applicants with:

- an outgoing, supportive and positive attitude
- excellent interpersonal communication skills
- the desire to succeed in an environment of high technology computer networking, without ever forgetting that it is all about people and solving their problems
- strong background in freeware UNIX configuration and administration, including Linux, FreeBSD and others
- strong background in Internet configuration and troubleshooting, computer and network security
- sound knowledge in general computer communications and the desire to learn fast and aquire new skills in this area
- the ability to work in stress situations, stay calm, friendly and supportive
- willigness to work at odd hours of the day, during Saturdays, to take phone calls and reply to emails, as needed
- the ability to travel overseas, as the need arises, to cope with time differences and jetlag

Preference will be given to people with previous employment in a technical support role and/or working at an Internet Service Provider, however, you are encouraged to send in your resume at all times.

For further information, please contact Endace Human Resources at <hr@endace.com> or phone Selwyn Pellett at +64-9-2610407.

SGI Shatters World Performance Record

Author: SGI, Inc.

[Editor's Note: This is purely a reprint of a release sent around by SGI, but I felt that it's of importance to the AUUGN readership, particularly in light of the commercialisation and commoditisation of a number of previously esoteric technology lines. And, aw-shucks... linear scalability of clustered CPUs is pretty damn cool :-)]

Company Attains Linear Scalability on 64-Processor Itanium 2-based System Running Linux INTEL DEVELOPER FORUM, SAN JOSE, Calif., (September 9, 2002)-- At the Fall 2002 Intel Developers' Forum, SGI (NYSE: SGI) today announced a breakthrough in advancing high-performance, scalable systems based on the Intel® Itanium® 2 processor and the Linux® operating system for high-end technical application users.

SGI has attained linear scalability on a 64-processor Itanium 2-based system and world-record results among microprocessor-based systems on the STREAM Triad benchmark, which tests memry bandwidth performance. Demonstrating linear scalability from 2 to 64 processors, the Itanium 2-based prototype system from SGI exceeded 120GB per second on a single system image. This result, derived from initial internal testing, marks a significant milestone for the industry: Early Itanium 2-based SGI® systems built on the innovative SGI® NUMAflexTM shared-memory architecture, have proven that Linux can scale well beyond the perceived limitation of eight processors in a single system image.

Additionally, results show that the upcoming Itanium 2-based SGI system has not only outperformed the IBM® eServer p690 and Sun Microsystems Sun FireTM 15K high-end microprocessor-based systems, it has also surpassed memory bandwidth performance on the CRAY C90TM, the CRAY SV1TM and the Fujitsu VPP5000 CMOS vector-based supercomputers.

SGI recently announced its expanded business strategy in which it will incorporate systems based on the Itanium processor family running Linux into its established line of high-performance computing systems. By utilizing Intel components within its NUMAflex modular architecture, SGI positions itself as the lead source for scalable Linux systems.

Early next year, SGI will release its fully optimized Itanium 2-based systems and its final STREAM benchmark results. The new Itanium 2- based SGI systems will enable maximum performance and scalability for a breadth of technical applications and will be fully compatible with two future Itanium processor family products (code-named "Madison" and "Montecito") to offer an unmatched investment protection.
"With its rich legacy in technical computing, SGI is well positioned to speed the adoption of the Itanium 2 processor in scalable high- performance environments," said Mike Fister, senior vice president and general manager, Enterprise Platforms Group, Intel. "We are delighted to see an Itanium 2-based system achieve such high efficiency in scaling Linux to this level, clearly demonstrating the benefits of combining the Itanium 2 processor's outstanding performance and scalability with SGI's NUMAflex architecture."

"This accomplishment will turn the heads of those who considered clustering a multitude of PCs as an optimal solution for integrating Linux into technical computing labs," said Jon "maddog" Hall, president and executive director, Linux International. "For those applications that need to scale, SGI has just proven that Linux need not be synonymous with clutter."

"This accomplishment demonstrates SGI's unique ability to innovate at the edge of the performance frontier," said Paul McNamara, vice president, Products and Platforms, SGI. "Our NUMAflex architecture enables us to create the world's most scalable shared-memory supercomputers, while delivering the rich application environment that is developing around the Intel Itanium line of processors and the Linux operating system."

Expanded Strategy

Under its new business strategy, SGI will expand its product and service offerings by adding its core technology into systems running the MIPS® processor and the IRIX® operating system--and soon, its Itanium 2-based systems running the Linux operating system. NUMAflex will remain the foundation of all SGI systems. Through its experience and expertise in high-performance computing, SGI will offer customers of the highest quality 64-bit operating environments.

About STREAM

STREAM is a highly regarded performance metric that measures the sustainable memory bandwidth, or flow, of a computing system. High- performance computing codes require a balance between the processor and memory subsystem to maintain a constant flow of data.

Book Review: Multitool Linux: Practical Uses for Open Source Software

Author: Aleksandar Stancin <sal@net-security.org>

Multitool Linux: Practical Uses for Open Source Software
Authors: Michael Schwarz, Jeremy Anderson, Peter Curtis and Steven Murphy
Pages: 576
Publisher: Addison Wesley
ISBN: 0-201-73420-6

Available for download is chapter 5 entitled "Samba: talking to Windows networks".

An interview of three of the four authors of the book is availabe, for their thoughts on the book click here.

Introduction

Linux is popular, not to say cool. A free alternative for Windows, Mac OS, and other OS's. Not long ago, Linux was mainly used by power users for various developing processes, system and network administrators for it's reliability, curios hackers and hacker wanna-be's alike. Nowadays, the situation is slightly different. Everyone wants to give it a spin, sooner or later, regardless of the user's knowledge. All major distributions are now as easy to install and perform basic system setup as any Windows version, not to mention a huge leap in the term of user friendliness. I can already hear old-school admins grunting at this one, but this is a book review not a rant, so leave it for some other more appropriate occasion. But, the real question is, what to do with Linux after you've successfully installed it? And, judging by a local linux newsgroup I follow and participate in, such questions arise more often, as the group is literally flooded by novice users asking for help, or pointers to which software to use and for what purpose. If you follow any Usenet group, you know how irritating it can get when people, again usually novice users, ask questions already answered in a message or thread dating one month ago, or so. The general idea of publishing a book that would serve such a purpose, giving example of what to do and how with linux seems like a good idea at this time. So, what we have here is a book that tries to do exactly that. To answer the 'what next' question for novice, and all linux users alike. Does it deliver? Read on.

Some tidbits about the authors and book

Multitool linux is written by four authors, M. Schwarz, J. Anderson, P. Curtis, and S. Murphy. As the cover says it, they are all experienced in programming in various developing environments, and have been working on various Unix systems for quite some time now. The book itself spreads on some 500+ pages of useful resources. Was that brief or what? Now, for the book itself...

Contents

To say that this book is packed with useful resources, ideas and pointers would be a heavy understatement. It boasts some 25 chapters, each with its subsections, so you can rest assured it isn't light bed time reading material. Let's shed some light on the material in it. As I've stated, it consists of 25 chapters, of which 24 are packed with information on various software packages.

The authors have tried to organize it into 3 major parts, the introduction, the toolbox and the afterword. As you already might guess, the introduction deals with some real basics of Linux, such as describing the concepts behind some of the licences (ie, GPL), Linux kernel, reconfiguring it, and such. Not much space is spent on it, as it's not a linux system administration book.

The real flesh of the book is the toolbox section, which contains all the really useful information in the book. It spreads from chapter 2 to 24, just to give you the idea. Now, what I'd usually do is describe each chapter in brief, but as there are far too many chapters to discuss, I'll stick with the authors organization of chapters into larger sections. Basic topics of network and communications are covered through chapters 2-9, containing information on subjects like remote control, masquerading, IPChains, even PLIP, Samba, undernets, secure web mail service and such. Surprising eh? Like I said, this is not "Running Linux" or "Linux System Administration Book", but a book of a different kind.
Next comes in line a section about privacy and security, spreading from chapter 10 - 14, containing such fine things as GPG and email security, sniffing, organising system security using software such as Tripwire, Snort, SSH, and configuring them. Mind you, there are far much better books written solely on system security, so don't look for too much here, it's only a section, but a decent one.

Then comes the part with miscellaneous tools described, in chapters 15 - 17. Now, the chapter 15 is a real gem here, as it describes in brief some valuable and irreplaceable tools for Linux, such as Vi, sed, dd, tar, CVS, Perl, various shells, grep, lsof, wget, various mail clients, GUI and non-GUI tools. Every novice user will find it very useful as all major Linux distributions install some or all of these by default, and are a priceless help for any linux user. It is concluded with a list of usefull linux related web sites. This one is a must read, not to be forgotten but used.

The following three sections are a lightweight reading in comparison to others, as they mainly deal with music and audio, graphics of any kind and video. Areas a lot of people work professionally in, and some of them make a living out of it, so it's a good thing to tag along and read them. You'll find useful software references and ideas about audio processing, vinyl clean up, mp3's, various cd burning tools, MIDI, speech synthesis, image processing, the lot.

What do I make of it

This book is intended as a guide for users who already installed their Linux boxes, but are unsure of their possibilities or don't know what to do with them. As such, it does good. As you can clearly see from the books contents it's packed with information about all major areas of interest for all sorts of users.

Anyone looking for a book of such subject can safely look here. If you're looking for one of those "Learn Linux in 24 Hours" or "Running Apache", look further, or prepare to be disappointed. Luckily, this book doesn't even try to pretend to be something it's not, so I must give credit to the authors for clearly stating it at the very beginning of it.

Now, very few, if any, books succeed in desire to be the jack of all trades when it comes to computers. Such is the case here. Even though it's literally packed with useful information, I can't really recommend it to a novice or inexperienced user, especially those users with little practical computer knowledge. In my honest opinion, far more knowledge is required to deal with this book and fully understand some of the subjects dealt with, than an average novice Linux user usually possesses. Sure, the authors have tried, but it seems to me that they aimed for the more intermediate users, than novice. I got no problem there, but one must keep in mind that such users already have and use software they need, and are advancing to some other topic that choosing how to encode mp3's. On the other hand, it deals with some pretty odd but amusing ideas of using email as a system console. How's that for an intermediate subject? Definitely.

So, on a scale od 1 - 5 (1 being the lowest possible grade) I can give it a 3.5, but cannot recommend it to novice users, only more intermediate ones. At first sight, it may seem like a book glued together of various unrelated subjects, but when you look into it better you'll see it's fairly good organized, but requires some knowledge. As it tries to cover as much various aspects as possible, you have to be prepared to do some extra researching and reading on your own in order to use it to a full extents. But, being a Linux user, that is expected of you, so there's no need for me to emphasize that fact, right?

This article is re-printed with permission. The originals can be found at:
http://www.net-security.org/review.php?id=12

Advocacy for Open Source in Government

Author: Con Zymaris <conz@cybersource.com.au>

The thoughts expressed here have had a multi-year gestation period, reaching a degree of cogency only in recent months, when a multitude of events, including the statement of intent to consider widespread deployments of open source technologies by close to 40 countries around the world, including the whole European Union, as well as an attempt (led my Microsoft no less) to stymie this brewing stampede, have caused me to consider this more completely.

This piece was written partly as a response to a column by Andrew Parsons in ZDNet, who wrote there recently about the misdirected push by sections of the our industry to push for Linux and Open Source adoption by government due to their zero licence cost. While much of Andrew's thesis can be quickly dismissed and countered, which I will do below, he is right on a couple of points. In his piece (Free source or free beer? source:
http://www.zdnet.com.au/newstech/os/story/0,2000024997,20267851,00.htm) Andrew rhetorically asks if it's a good idea to opt for the cheapest technology (i.e., the one with no licence fee costs) to which he doesn't render a complete response. I'll provide an answer in lieu of his, with the following rumination: the freedom from price of any software should not be the most significant aspect for the selection of software for government. What is of absolute importance is a combination of open and inter-operable formats, APIs and standards, along with transparency of software. What's at stake is not merely or primarily a discontinuity of commercial interests. It's far more fundamental.

The core concept here is that government functions differently to commerce, and rightly so. Open government is good government. The code which runs under the machinery of government should also be open; open to inspection; open to review; open to discussion, somewhat mirroring the Freedom of Information legislation which permeates through most advanced democracies, and much like open contracts and publically reviewable government processes, which are a pre-requisite for the viable governance of democratic society. We must always remember that governments are, and should continue to be, different to profit-driven businesses. Democratically elected governments have a much higher calling and should therefore be measured against a much more stringent gnomon. What might suffice as acceptable practice and probity in business, is never enough for the leaders and purveyors of citizenry.

Let's expand on these basic tenets of open government computing. Firstly, open standards for intercommunication and file formats should be an absolute must for any system implemented for government. Document standards should be fully disclosed, and preferably implemented by a range of developers. This is for reasons of longevity and permanence of the government record. We must not have documents which have been created by technologies that only one supplier can provide. This means that while Microsoft Word should never be used to create or store documents, open and clearly defined standards such as PDF and OpenOffice.org are fine. Further, systems technology which only runs on platforms from any single vendor (for example Microsoft's .Net) should likewise never be used. Once again, this has to do with ensuring the longevity and permanence of applications which can manipulate the data formats used, as well as encapsulate the machinery of government. As an alternative, standards-based Web-services, which are clearly defined and ratified by a non-vendor body such as the W3C, and implemented by many suppliers, are fine. Another example: VB.Net should not be used as it's only available from one supplier. Java is fine, as once again, it's available from many suppliers, including compliant and free open source versions.

Secondly, there should be strong and viable competition amongst suppliers of platform and application software technology. Construing that the government have one, and only one supplier of desktop platform and productivity software, as it does now, is just not on. As indicated by the Sincere Choice project (http://www.sincerechoice.org/) software suppliers should compete fairly on the merit of their products, rather than by attempting to lock each other's products out of the market. At the risk of belabouring this point, open formats and platforms, particularly with Open Source reference implementations, assist greatly here.

Lastly, as a democratic society, we need to view the increasing adoption of the technologies which run counter to open government computing (i.e. closed file formats, platform specific technologies and the spread of technologies which have only one single, monopolistic supplier,) as alarming. Whilst this wasn't as big a problem 20 years ago with the minimal pervasiveness of computer technology, in 20 years from now, when e-government will be a fundamental component of many a democratic society, we will have serious and irreparable fractures unless the technology which deploys e-government in all its facets, is also open.

Note, no mention has been made of any financial basis for the adoption of Open Source software. This is a whole additional level and should form, as Andrew points out in his piece, a sideline issue. Having said that, Open Source is of great importance from a national economic perspective. By considering Open Source software systems (by way of import replacement for other expensive software systems) governments locally will not only be in line to save hundreds of millions of dollars each year in licence costs and help stem the current major trade deficit in IT, but also be at the forefront of this exciting next-wave now sweeping through our industry. Further, by establishing this country as a center of excellence for the development of this type of software, we can play a more prominent part and gain financially from the services and support revenue this realm fosters.

This combination of benefits haven't been lost on many functionaries of governments worldwide. Siegmar Mosdorf, German Secretary of State, voiced increasing support by the German Federal Government of Open Source and Linux software. Among others, he indicated the following reasons as to why governments worldwide should adopt Open Source software:

- protection from coercion by corporate entities (such as Microsoft) that control closed-source software upon which your government depends;
- reduced litigation;
- reduced international pressure regarding "piracy" issues;
- greater control of the software from a national security
- standpoint;
- greater national economic benefits for companies within your country to develop, improve and support software without the need for any external corporations outside your country; and finally
- cost saving reductions to your taxpayers.

(Source:
http://www.internetnews.com/intl-news/article/0,2171,6_408271,00.html)

Now, let's reach for the scalpel and bone-saw for some belated deconstruction of Andrew Parson's arguments in his recent missive on this subject. On the proponents of Open Source for governments, Andrew states:

"They attempt to put forward the claim that to allow governments to use commercially supplied software somehow curtails their choice in the matter?"

Incorrect. The fact of the matter is that there should be a viable set of options for technologies which meet the open and inter-operable standards requirements, as discussed above, and from a variety of suppliers. This isn't happening at present. For example, if you seek information about the list of platform and technology suppliers shortlisted to offer desktop operating system and office productivity software to governments locally, how many suppliers do you think will be on this list?

Andrew continues:

"Are these protagonists trying to state that these companies are not entitled to charge for their product? Are they not entitled to recoup some of their costs, and more importantly, as a BUSINESS are they not entitled to seek to make a profit so they can continue to BE a business?"

Incorrect. No one with a reasonable perspective could possibly expect that these suppliers should give their software away or to Open Source their code. That's never been on the agenda. What proponents of open government computing state is that government is a different realm, and unlike the commercial world, should mandate open and inter-operable document standards, platforms and communications technologies. If the closed-source vendors are able to meet these criteria with their products, then they too should be shortlisted for supply of technologies. It's then up to the individual purchasing officers to decide if the value proffered by these products is commensurate with the money asked for by these vendors, which must presumable therefore be above and beyond the quality or capabilities provided by the Open Source contenders.to justify the extra cost. If so, fine. If not, default to the Open Source option. This is open and fair competition.

Finally we have:

"I challenge YOU, the one who says they're over-priced to produce a product yourself that rivals the commercially available products. I challenge YOU to then provide the infrastructure and support levels that are normally available with these so- called more expensive programs."

I think Andrew has not been following events in our industry over these past few years. In response, I would point to suppliers like IBM or SGI, both of whom implement open and inter-operable (in this case Linux and Samba) technologies on high-end, enterprise-class computing hardware platforms, and charge a lot of money to very satisfied government customers for the supply and support of these products. These are just a few examples of hundreds of similar cases. Now, while these are products from commercial vendors, let's see if they meet our open government computing requirements. Are the products inter-operable and open, and offered from many other suppliers in a similar configuration? You bet. Are the core technologies open source? You bet. Are the customers happy? You bet? Are we, the taxpayers saving money? You bet. Are there any issues with this? I don't know. Ask Andrew.

With our eyes on the horizon, it is becoming obvious, with the increasing importance of computer software in all manner of government function, that open and inter-operable computing, often best implemented as Open Source software libre, is the best software model which allows the IT-functioning of modern government to progress in a manner commensurate with the aims and motives of open and democratic society. This is an issue that you, the citizens of society, can help raise, discuss and bring to bear. Assuming that someone else will take up the challenge is a myopic stance. As with many other aspects of the body-politik, vigilance is the price for eternal freedom.

This piece was originally published on ZDNet.

Multi-Account E-mail with *mutt*

Kamil Klimkiewicz <kamil@adrem.pl>

About three or four months ago I switched from Windows to Linux. I had been using Linux before but it was only my second operating system. When it became my primary one I had to deal with several problems. Most of them I was able to fix quickly. There was one thing which caused many troubles - I had three e-mail accounts.

Windows user could say, "Download some e-mail client and configure it to use several accounts." But there is something called the 'Unix philosophy'. It says that programmers should write small applications which do only one thing but do it well. What does it mean for us? It means that there is no single tool which fetches your mail from remote server, allows you to read and write mail and sends it to its target.

In this short article I will only show you how to configure tools called fetchmail and mutt. If you want to go deeper into this topic you should read: Mail-Administrator-HOWTO, Mail-User-HOWTO.
You can get them from http://www.linuxdoc.org.

1. Environment

Let's define our e-mail environment: we have three e-mail accounts, each placed on different server. We will call them 'First', 'Second' and 'Third.' Their addresses are: first@firstdomain.com, second@seconddomain.com, third@thirddomain.com. Moreover the first account uses IMAP protocol and the others POP3.

The local user who is going to receive all the messages is called 'john'. We need to set new value for $MAIL environment variable, since we won't use default '/var/spool/mail/john' (this is unsafe and less convenient.) To do this we add following lines to .bash_profile (of course if you use different shell you have to change different things):

```
MAIL=$HOME/Mail/Inbox
export MAIL (Don't forget to create directory
'$HOME/Mail'!.)
```

We will also use additional mailboxes for read messages (each account has its associated box.)

2. Fetchmail

Before we can read our mail we have to fetch it from remote server. To do this we will use a tool called fetchmail. It should be already installed on your system.

Configuring fetchmail is quite easy task. Moreover we can use utility 'fetchmailconf' which makes the process even easier. Configuration file we should edit is $HOME/.fetchmailrc. Simple one, appropriate for our environment, looks like this:

```
set postmaster "john"
set bouncemail
set properties ""
set daemon 300
```

poll First via firstdomain.com with proto IMAP user first there with password this_is_password is john here warnings 3600 poll Second via seconddomain.com with proto POP3 user second there with password this_is_password is john here warnings 3600 poll Second via thirddomain.com with proto POP3 user third there with password this_is_password is john here warnings 3600 To run fetchmail you only need to type fetchmail. It will be started in daemon mode and will check whether there is new mail every 5 minutes.

3. Mutt

Our messages are on local machine now, so we can read them using any Mail User Agent. I assume it is mutt because this article is intended to deal with mutt. Mutt needs to be configured before it can work like we want. First of all we have to put some basic definitions in its configuration file (it is usually called $HOME/.muttrc.) They can look like this:

```
set mbox = "~/Mail/Inbox"
set move = no
set folder = "~/Mail"
set record = +Sent
mailboxes +Inbox +First +Second +Third
```

This actually allows us to read the messages but every outgoing message will have something like john@localhost in its From header field. We should be able to change the sender address so the message can look like it was sent from firstdomain.com or seconddomain.com or whatever machine you have account on. To achieve this we use additional mailboxes (First, Second and Third) and mutt's so called hooks mechanism. The latter executes user defined commands when some action is being performed. There is folder-hook which is called when user changes mail folder (by pressing 'c' key.) To change the From field we need to modify from and realname mutt variables:

```
# Default action:
folder-hook . set from = first@firstdomain.com
folder-hook . set realname = First
# First account:
folder-hook First set from = first@firstdomain.com
folder-hook First set realname = First
# Second account:
folder-hook Second set from =
second@seconddomain.com
folder-hook Second set realname = Second
# Third account:
folder-hook Third set from = third@thirddomain.com
folder-hook Third set realname = Third
```

We should also define the alternates variable so mutt can recognize messages sent to and by us:

```
set alternates =
"first@firstdomain\.com|second@seconddomain\.com|t
hird@thirddomain\.com" Note: There is a web tool
called MuttrcBuilder at
http://mutt.netliberte.org
```

which you can use to configure your mutt.

This article is re-printed with permission. The originals can be found at:

http://www.linuxgazette.com/issue83/klimkiewicz.html

Amanda CD-RW Taper

Ben Elliston <bje@air.net.au>

I have enjoyed skimming Con's reviews of useful open source software packages in previous issues of AUUGN. I found a piece of software a few months ago that I think ranks a mention and thought I would write a few words about it. I hope others find it as useful as I did!

For those who are familiar with it, Amanda (http://www.amanda.org/) is quite an advanced backup system. It takes a little while to learn how to operate it on a day to day basis, but for those who know it or are prepared to learn, it's even useful for backing up home systems.

Tape drives are increasingly hard to justify for home use: they are so much more expensive than disks of the same capacity, media are expensive and the units can be somewhat prone to failure (in my experience, anyway!).

Amanda is reasonably modular. In particular, dump images are moved from a holding area to tape using a separate program known as "taper". Peter Conrad and Richard Kunze of Tivano Software in Germany have produced a replacement "taper" program that can use CD-RWs as a backup medium! The software and documentation can be found at http://www.tivano.de/software/amanda/index.shtml

In short, the taper places dump images collected by Amanda into an intermediate directory (not to be confused with the holding area) and then generates ISO file system images using as many of dump images as it can to fill each CD-RW (or CD-Rs, if you want to use cheaper media for permanent archival). The ISO images are then written to CD-RWs using the "cdrecord" program. The CD-RW taper emulates a CD-RW changer for multi-volume backups. CDs have to be changed manually. Because disk space is so cheap, it is also possible to just write dump images to the intermediate directory and leave them there.

All in all, this is a very useful utility for anyone who owns a $150 CD-RW drive!

Dynamically Expand a File-system to a Single Stripe instead of to Several Concatenated Stripes

Joseph Gan <joseph.gan@abs.gov.au>

A metadevice can be expanded by adding slices. Most UNIX file systems can be concatenated into a metadevice that contains an existing file system on the fly. If the file system type is UFS, it can be grown to fill the larger space while the file system is in use.

But the concatenation is good only for small random I/O and for even I/O distribution. On the other hand, striping is advantageous for large sequential I/O and for uneven I/O distribution.

Striping will increase performance by accessing data in parallel. However, if you want to expanded a file system to a single striped metadevice, you have to dismount the file system, then back it up to tapes and restore it back to the new partition.

How to dynamically expand a file system to a single striped metadevice without interrupting access to data?

Solstics DiskSuite (SDS) can do the job. Since SDS is co-packaged with Solaris and is available as an unbundled product. In the Solaris 9 however, now delivered as an integral part of the Solaris Operating Environment known as The Solaris Volume Manager (SVM).

First, the file system has to be created and mounted as one-way mirror metadevice, in this example, d80 mounted by /opt:

```
  # metastat d80
  d80: Mirror
      Submirror 0: d81
        State: Okay
      Pass: 1
      Read option: roundrobin (default)
      Write option: parallel (default)
      Size: 10261520 blocks

  d81: Submirror of d80
      State: Okay
      Size: 10261520 blocks
      Stripe 0: (interlace: 32 blocks)
          Device              Start Block  Dbase
State         Hot Spare
          c1t12d0s0                  0     No
Okay
          c1t13d0s0               1520     No
Okay
          c1t14d0s0               1520     No
Okay
          c1t15d0s0               1520     No
Okay
```

Next, use the metattach command to dynamically concatenate a new slice, /dev/dsk/c0t1d0s1, to the end of the existing submirror of d80, d81:

```
  # metattach d81 c0t1d0s1

  # metastat d80
  d80: Mirror
      Submirror 0: d81
        State: Okay
      Pass: 1
      Read option: roundrobin (default)
      Write option: parallel (default)
      Size: 10261520 blocks

  d81: Submirror of d80
      State: Okay
      Size: 10261520 blocks
      Stripe 0: (interlace: 32 blocks)
          Device              Start Block  Dbase
State         Hot Spare
          c1t12d0s0                  0     No
Okay
          c1t13d0s0               1520     No
Okay
          c1t14d0s0               1520     No
Okay
          c1t15d0s0               1520     No
Okay
      Stripe 1:
          Device              Start Block  Dbase
State         Hot Spare
          c0t1d0s1                   0     No
Okay
```

Then, use growfs command to expand the mounted file system (/opt) onto the raw metadevice /dev/md/rdsk/d80:

```
  # growfs -M /opt /dev/md/rdsk/d80

  /dev/md/rdsk/d80:       12336320 sectors in
8116 cylinders of 19 tracks, 80
  sectors
          6023.6MB in 129 cyl groups (63 c/g,
46.76MB/g, 5888 i/g)
  super-block backups (for fsck -F ufs -o b=#) at:
   32, 95872, 191712, 287552, 383392, 479232,
575072, 670912, 766752, 862592,
   958432, 1054272, 1150112, 1245952, 1341792,
1437632, 1533472, 1629312,
   1725152, 1820992, 1916832, 2012672, 2108512,
2204352, 2300192, 2396032,
   2491872, 2587712, 2683552, 2779392, 2875232,
2971072, 3064352, 3160192,
   3256032, 3351872, 3447712, 3543552, 3639392,
3735232, 3831072, 3926912,
   4022752, 4118592, 4214432, 4310272, 4406112,
4501952, 4597792, 4693632,
   4789472, 4885312, 4981152, 5076992, 5172832,
5268672, 5364512, 5460352,
   5556192, 5652032, 5747872, 5843712, 5939552,
6035392, 6128672, 6224512,
   6320352, 6416192, 6512032, 6607872, 6703712,
6799552, 6895392, 6991232,
   7087072, 7182912, 7278752, 7374592, 7470432,
7566272, 7662112, 7757952,
   7853792, 7949632, 8045472, 8141312, 8237152,
8332992, 8428832, 8524672,
   8620512, 8716352, 8812192, 8908032, 9003872,
9099712, 9192992, 9288832,
   9384672, 9480512, 9576352, 9672192, 9768032,
9863872, 9959712, 10055552,
   10151392, 10247232, 10343072, 10438912,
10534752, 10630592, 10726432,
   10822272, 10918112, 11013952, 11109792,
11205632, 11301472, 11397312,
   11493152, 11588992, 11684832, 11780672,
11876512, 11972352, 12068192,
   12164032, 12257312,
```

Now the file system (/opt) has been expanded dynamically, but it contends two stripes: stripe 0 which is the original one, and stripe 1 which is the expanded one.

Next step is to create a single stripe metadevice d82, which is the same size of the submirror d81:

In this example, we create a stripe with three 2.1Gb disks:

```
 # metainit d82 1 3 c0t11d0s2 c0t12d0s2 c0t13d0s2
d82: Concat/Stripe is setup

 # metastat d82
d82: Concat/Stripe
    Size: 12457920 blocks
    Stripe 0: (interlace: 32 blocks)
        Device              Start Block  Dbase
        c0t11d0s2                  0     No
        c0t12d0s2               1520     No
        c0t13d0s2               1520     No
```

Then, add the metadevice d82 as the second submirror to d80, resync will automatically take place:

```
 # metattach d80 d82
d80: submirror d82 is attached

 # metastat d80
d80: Mirror
    Submirror 0: d81
      State: Okay
    Submirror 1: d82
      State: Resyncing
    Resync in progress: 20 % done
    Pass: 1
    Read option: roundrobin (default)
    Write option: parallel (default)
    Size: 12336320 blocks
......
```

After the resync has done, we got the following two way mirrors:

```
  # metastat d80
  d80: Mirror

      Submirror 0: d81
        State: Okay
      Submirror 1: d82
        State: Okay
      Pass: 1
      Read option: roundrobin (default)
      Write option: parallel (default)
      Size: 12336320 blocks

  d81: Submirror of d80
      State: Okay
      Size: 12336320 blocks
      Stripe 0: (interlace: 32 blocks)
          Device              Start Block  Dbase
State        Hot Spare
          c1t12d0s0                  0     No
Okay
          c1t13d0s0               1520     No
Okay
          c1t14d0s0               1520     No
Okay
          c1t15d0s0               1520     No
Okay
      Stripe 1:
          Device              Start Block  Dbase
State        Hot Spare
          c0t1d0s1                   0     No
Okay

  d82: Submirror of d80
      State: Okay
      Size: 12336320 blocks
      Stripe 0: (interlace: 32 blocks)
          Device              Start Block  Dbase
State        Hot Spare
          c0t11d0s2                  0     No
Okay
          c0t12d0s2               1520     No
Okay
          c0t13d0s2               1520     No
Okay
```

Finally, we can detach the submirror d81 from d80, and remove metadevice d81 all
together:

```
  # metadetach d80 d81
  # metaclear d81

  # metastat d80
  d80: Mirror
      Submirror 1: d82
        State: Okay
      Pass: 1
      Read option: roundrobin (default)
      Write option: parallel (default)
      Size: 12336320 blocks

  d82: Submirror of d80
      State: Okay
      Size: 12336320 blocks
      Stripe 0: (interlace: 32 blocks)
          Device              Start Block  Dbase
State        Hot Spare
          c0t11d0s2                  0         No
Okay
          c0t12d0s2               1520     No
Okay
          c0t13d0s2               1520     No
Okay
```

Now, we have got the expanded file system (/opt) with a single stripe metadevice dynamically.

How to Install SuSE Linux 8 on your Xbox

Michael Steil <mist@c64.org>

Standard Linux distributions do work on the Xbox, with minor modifications. To run SuSE Linux as a server or for X Window and KDE or Gnome, you only have to change two lines in the Linux kernel, disable one init script, install two drivers for audio and networking and use our X Window configuartion file. This article is a step-by-step tutorial to install SuSE 8 with X-Window on the Xbox.

INGREDIENTS

You will need

- an Xbox that is equipped with a 10 GB drive (most are; bigger hard drives not tested) and with any modchip
- a SuSE 8 compatible PC with a network card
- SuSE Linux 8 (at least CD 1)
- the latest XBE bootloader as well as the patched SuSE kernel available from the Xbox Linux Project ("SuSE8-Xbox.tar.bz2")
- either a CD/RW (or DVD/RW) that works in your Xbox DVD drive or EvoX on your Xbox hard disk
- good Linux knowlegde

For network and audio support you will also need
- the SuSE 8 nForce device driver RPM, available on

nVidia's site (the SuSE support database links there)

To work interactively on the command line, you'll need
- an Xbox/USB adaptor, as described on the web site
- a USB keyboard

To work in X Window on your Xbox, you will need in addition:
- either another Xbox/USB adaptor or a hub to plug the Keyboard and the mouse into the first adaptor
- a USB mouse

Principle

We'll do a cross-install, i.e. we connect the Xbox hard disk to the PC, install Linux onto it, make some more modifications to the installation (drivers, patches), plug it into the Xbox and install the bootloader for the Xbox.

Cross-Installing SuSE

Before you start, it is strongly recommend to backup your Xbox hard disk, as described on the Xbox Linux site. Linux will be installed into unused parts of the hard disk, but...

You cannot just connect the Xbox hard disk to your PC, because it is locked. You have to "hot-swap" it. Place the PC and the Xbox next to each other and prepare an IDE cable to be able to connect the Xbox hard drive as secondary master. With the USB keyboard and USB mouse (if you want to use them on the Xbox later) connected (and with the PS/2 keyboard and mouse disconnected), turn the PC on and put the SuSE boot CD into the PC's CD drive. Stop at the boot loader menu and turn on the Xbox. When the Microsoft Dashboard (or EvoX) shows up, disconnect the Xbox hard disk's IDE cable and connect the PC's IDE cable while both computers are running. Chose "Installation" in the bootmanager. In the boot messages, the HD should be detected as a 10 GB drive at /dev/hda. If it's only 8 GB, you cannot install SuSE using this method (yet), sorry.

The USB mouse and the keyboard should get properly detected, just install according to your wishes (keyboard map, time zone, packages to install), but you will have to change the partitioning SuSE suggests or else it will overwrite your Xbox system data and savegames on the disk. Tell SuSE to discard the suggestion, and you want to define the partitions on your own in expert mode. Create /dev/hda1 starting at track 15534 (that's right after the last Xbox partition) with a size of 128 MB as a swap partition. Allocate the rest behind this partition (about 1.7 GB) as your root file system /dev/hda2; ReiserFS is preferred. Note that the SuSE installer can safely write a PC-like partition table, since the first sector of the Xbox hard disk is unused.

Start the installation and follow the onscreen-instructions. If the installer tells you that the disk might not be bootable, just ignore it, it should be bootable afterwards, unless your PC is really very old. When the installer wants to reboot, you either have to disconnect the IDE cable of the hard disk between the unmount messages and the actual reboot and reconnect it when the VGA card's bios prints it's initialization messages onto the screen, or you will have to unlock it again by connecting it to the Xbox and starting the Dashboard and hot-swap it to the PC while the VGA messages are shown (these should be some seconds), because a reboot locks the hard disk again.

The second phase of the installer should now start from hard disk. Configure your network now to match the settings you will expect in your Xbox. Just ignore all other hardware settings. The installer will reboot again, and you'll have to do the disconnect trick or unlock process again. Boot into your newly installed system.

Install the nForce driver RPM now. Edit /etc/modules.conf and disable your PC network interface and audio hardware. Enable the already existing nForce lines (just scan for "nForce"). If you want to use X, copy our XF86Config-4 to /etc/X11, else edit / etc/inittab and change the default runlevel from 5 to 3 ("id:3:initdefault:"). The init script "hwscan" in /etc/init.d would crash the Xbox, so disable it by simply renaming it. Copy the file /boot/initrd to a disk, because we will need it later. Your SuSE installation is now prepared for the Xbox hardware. Shut down and connect the hard disk to the Xbox.

You need a bootloader configuration now. Either you boot from CD or through EvoX. In either way, you need the bootloader default.xbe, the patched SuSE kernel, the initrd you copied from the SuSE hard disk before, and a linuxboot.cfg file. The package from the Xbox Linux web site contains the kernel and for each of both solutions a default.xbe and a linuxboot.cfg. Put the corresponding four files either onto a CD/RW (DVD/RW) containing a UDF filesystem ("mkisofs -udf") or copy them to E:\Linux\ using EvoX.

You should now be able to boot Linux. You should see the kernel initialization messages, the initrd messages and the init scripts, and you should finally get a "Login:" after a minute or two, either on the text screen, or, if you have X Window enabled, in the graphical environment. If you chose to access the Xbox only through the network, you can ssh into it now and do everything you can do on a Linux PC. If you want to work with the Xbox directly, you can now connect the USB keyboard (and mouse) through the adaptor(s), and you should be able to log in.

Issues

This is not yet perfect. Some issues:
- Starting SaX or installing software with YaST crashes (system configuration with YaST works, though).
- Pressing the eject button reboots.
- Pressing the power button doesn't shut Linux down.

- Trying to reboot Linux only makes it halt, you can reboot then by pressing the eject button.

Why SuSE?

I only had the current versions of Mandrake and SuSE here, and I don't have a broadband internet connection, so that I could download any other distributions. I tried Mandrake first, starting with root mounted as NFS, and with its own partition later. Too bad Mandrake is very pedantic about module versions, and there's something wrong with the compiler, so I was unable to run a kernel based on the Mandrake-patched 2.4.18 sources that loads nvnet.o. And without a Mandrake kernel, the kernel modules on hard disk would not load. So I tried SuSE, which made a lot less problems. If there are any Mandrake experts out there, I would still like to try Mandrake, too! And RedHat...

Again: The project has nothing to do with SuSE.

Final Words

This article has been written in OpenOffice.org 1.0.1 in SuSE 8 on the Xbox.

This article is re-printed with permission. The originals can be found at:

http://xbox-linux.sourceforge.net/articles.php?aid=2002247064525

The author has also informed AUUGN that there have been substantial additions to this project, and other distributions are now booting. Please refer to the website for more upto-date information:

http://xbox-linux.sourceforge.net/index.php

Chrooting All Services in Linux

Author:Mark Nielsen http://www.tcu-inc.com/mark

Introduction

What is chroot? Chroot basically redefines the universe for a program. More accurately, it redefines the "ROOT" directory or "/" for a program or login session. Basically, everything outside of the directory you use chroot on doesn't exist as far a program or shell is concerned.

Why is this useful? If someone breaks into your computer, they won't be able to see all the files on your system. Not being able to see your files limits the commands they can do and also doesn't give them the ability to exploit other files that are insecure. The only drawback is, I believe it doesn't stop them from looking at network connections and other stuff. Thus, you want to do a few more things which we won't get into in this article too much:

- Secure your networking ports.
- Have all services run as a service under a non-root account. In addition, have all services chrooted.
- Forward syslogs to another computer.
- Analyze logs files
- Analyze people trying to detect random ports on your computer
- Limit cpu and memory resources for a service.
- Activate account quotas.

The reason why I consider chroot (with a non-root service) to be a line of defense is, if someone breaks in under a non-root account, and there are no files which they can use to break into root, then they can only limit damage to the area they break in. Also, if the area they break into is owned mostly by the root account, then they have less options for attack. Obviously, there is something wrong if someone actually does break into your account, but it is nice to be able to limit the damage they can do.

PLEASE REMEMBER that my way of doing this is probably not 100% accurate. This is my first attempt at doing this, and if it just partially works well, it should be easy to finish out the rough edges. This is just a roadmap for a HOWTO I want to create on chroot.

How are we going to chroot everything?

Well, We create a directory, "/chroot" and we put all of our services under there in the following format:

- Syslogd will be at chrooted with each service.
- Apache will be at /chroot/httpd.
- Ssh will be at /chroot/sshd.
- PostgreSQL will be at /chroot/postmaster.
- Sendmail will be chrooted, but it won't be running under a non-root account, unfortuantely.
- ntpd will be chrooted to /chroot/ntpd
- named will be chrooted to /chroot/named

Each service should be completely isolated.

My Perl script to create chrooted environments.

`Config_Chroot.pl.txt` should be renamed `Config_Chroot.pl` after you download it. This perl script lets you list the services being installed, view the config files, configure a service, and start and stop the services. In general, this is what you should do.

1. Create the chroot directory
   `mkdir -p /chroot/Config/Backup`
2. Download `Config_Chroot.pl.txt` to `/chroot/Config_Chroot.pl`
3. Change the `$Home` variable in the perl script if you are not using `/chroot` as the home directory.
4. Download my config files,

http://www.linuxfocus.org/common/src/article225

Now, the important thing here is: **I have only tested in on RedHat 7.2 and RedHat 6.2.**

Modify the perl script for your distribution.

I ended up making a huge gigantic article on Chroot, but with my Perl script, it became much smaller. Basically, I noticed after chrooting many services, they all have very similar files and configurations that needed chrooted. The easiest way to figure out which files need copying for a particular service is to look at the manpage and also type "`ldd /usr/bin/file`" for programs that use library files. Also, you can chroot the the service you are installing and manually start it to see what errors you get or look at its log files.

In general, to install a service do this:

```
cd /chroot
./Config_Chroot.pl config  SERVICE
./Config_Chroot.pl install SERVICE
./Config_Chroot.pl start   SERVICE
```

Chrooting Ntpd

Ntpd is just a time service that lets you keep your computer and other computers in sync with the real time. It was a simple thing to chroot.

```
cd /chroot
# Uncomment the next line if you don't use my
config file.
#./Config_Chroot.pl config  ntpd
./Config_Chroot.pl install ntpd
./Config_Chroot.pl start   ntpd
```

Chrooting DNS or named

Already done, check out
http://www.linuxdoc.org/HOWTO/Chroot-BIND8-HOWTO.html
or
http://www.linuxdoc.org/HOWTO/Chroot-BIND-HOWTO.html

Or, if you want to use my script,

```
cd /chroot
# Uncomment the next line if you don't use my
config file.
#./Config_Chroot.pl config  named
./Config_Chroot.pl install named
./Config_Chroot.pl start   named
```

Chrooting Syslog with services and my complaints.

I want to chroot syslogd. My problem is, syslogd uses `/dev/log` by default, which can't be seen by chrooted services. Thus, I can't syslogd easily. Here are the possible solutions:

- Chroot syslogd with every service. I actually tested this, and yes, I was able to log stuff. I don't like this since I have a root running service.
- See if we can connect to an offsite logging facility.
- Just log files to a file and not through syslogd. This is probably the best security option, although if someone breaks, they could play around with the logs.
- Configure the main syslogd to look at several locations to get all the services. You use the `-a` option with syslogd to do this.

My only solution was to make sure syslogd is chrooted with every service. I would like some sort of solution which would log stuff in a non-root account using its own chrooted environment, like maybe a network port. It can probably be done, but I am going to stop where I am at and figure out a better solution later.

If you do not want to make a separate syslogd for each service, then with the main syslogd that you are running on your system, add the following command when syslogd starts:

```
syslogd -a /chroot/SERVICE/dev/log
```

If I had ssh and dns running, it might look like,

```
syslogd -a /chroot/ssh/dev/log -a
/chroot/named/dev/log -a /dev/log
```

Last note on Syslogd, I wish I could make it run under a non-root account. I tried a couple of simple things, but it didn't work and I gave up. If I could run syslogd under a non-root account with each service, that would satisfy my security issues. Possibly, even have it log offsite.

Chrooting Apache

This was extremely easy to do. Once I got it setup, I was able to execute Perl scripts. Now, my config file is rather long because I had to include Perl and the PostgreSQL libraries into the chrooted area. One thing to note, if you are connecting to a database, make sure your database service is running on the 127.0.0.1 loopback device and you specify the host to be 127.0.0.1 in your Perl scripts for the DBI module. Here is an example of how I connect to a database using persistent connections in apache:

```
$dbh ||= DBI-
>connect('dbi:Pg:dbname=DATABASE',"","",
{PrintError=>0});

if ($dbh )
   {$dbh->{PrintError} = 1;}
else
   {$dbh ||= DBI->connect('dbi:Pg:dbname=DATABASE;
      host=127.0.0.1',"","", {PrintError=>1});}
```

Source: http://httpd.apache.org/dist/httpd/

Compile and install apache on your main system at /usr/local/apache. Then use the perl script.

```
cd /chroot
# Uncomment the next line if you don't use my
config file.
# ./Config_Chroot.pl config  httpd
./Config_Chroot.pl install httpd
./Config_Chroot.pl start   httpd
```

I changed my httpd.conf file to have this stuff:

```
ExtendedStatus On

<Location /server-status>
   SetHandler server-status
   Order deny,allow
   Deny from all
   Allow from 127.0.0.1
</Location>

<Location /server-info>
   SetHandler server-info
   Order deny,allow
   Deny from all
   Allow from 127.0.0.1
</Location>
```

Then, just point your browser at `http://127.0.0.1/server-status` or `http://127.0.0.1/server-info` and check it out!

Chrooting Ssh

First off, ideally, you should port forward ssh on port 22 to port 2222. Then, when you start ssh, have it listen to port 2222 under a non-root account. For the initial ssh connection, we want to have secure accounts with passwords just to let the people in, but not do anything else. After they log in, then have a second ssh program running on port 127.0.0.1:2322 which will let them connect to the real system -- the second ssh program should ONLY listen on the loopback device. Now this is what you should do. We aren't going to do it. The only thing we are going to do is chroot ssh for this example. Exercises which are left up to the reader include putting sshd under a non-root account and to install a second sshd which listens on the loopback device to let people into the real system.

Again, we are going to just chroot ssh and let you worry about the consequences of doing that (you won't be able to see your entire system if you just do this). Also, ideally, it would be nice to set this up to record logs offsite. Also, we should use OpenSSH, but I am using the commercial SSH for simplicity (which is not a good excuse).

Source:
http://www.ssh.com/products/ssh/download.cfm

Install ssh at /usr/local/ssh_chroot. Then use the Perl script.

```
cd /chroot
# Uncomment the next line if you don't use my
config file.
# ./Config_Chroot.pl config  sshd
./Config_Chroot.pl install sshd
./Config_Chroot.pl start   sshd
```

I suppose one really good thing with putting ssh under a chrooted environment is that if you want to use it to replace an ftp server, people will have limited access to your area. Rsync and SCP go very well together for letting people upload files. I don't really like to put an ftp server up for people to log into. A lot of ftp servers are also chrooted, but they still transmit passwords in the clear, which I don't like.

Chrooting PostgreSQL

This was almost as simple as perl, except it required a few more libraries. Overall, it wasn't that hard to do. One thing I had to do was put PostgreSQL open to the network, but only on the loopback device. Since it was chrooted, other chrooted services couldn't get to it, like the apache web server. I did compile Perl into PostgreSQL, so I had to add a lot of Perl stuff to my config file.

Source:
ftp://ftp.us.postgresql.org/source/v7.1.3/postgresql-7.1.3.tar.gz

Compile and install apache on your main system at `/usr/local/postgres`. Then use the Perl script.

```
cd /chroot
# Uncomment the next line if you don't use my
config file.
# ./Config_Chroot.pl config  postgres
./Config_Chroot.pl install postgres
./Config_Chroot.pl start   postgres
```

Chrooting Sendmail

Go ahead and execute my script.

```
cd /chroot
# Uncomment the next line if you don't use my
config file.
# ./Config_Chroot.pl config  sendmail
./Config_Chroot.pl install sendmail
./Config_Chroot.pl start   sendmail
```

Now are there catches? Yes. It is still running as root. Darn. Also, certain files are recreated by the `/etc/rc.d/init.d/sendmail` file when it is started. Mine script doesn't handle that. Anytime you make changes to sendmail under `/etc/mail`, please copy the changes to `/chroot/sendmail/etc` also. Also, you will have to point `/var/spool/mail` to `/chroot/sendmail/var/spool/mail` so that the sendmail program and the users (when they log in) can see the same files.

The good thing is, you can always send mail out, it is just receiving it that is the problem. Thus, I was able to install sendmail with apache without any problems. Some of my perl scripts send mail out, and so, I needed the sendmail files copied into the chroot area for apache.

Other things to chroot.

Here is my philosophy:

1. Everything should be chrooted, including sendmail, ssh, apache, postgresql, syslog, and any service running on the computer.
2. Everything should be put under a non-root account (you might need to port forward protected ports to a non-protected port). This includes sendmail and syslog by the way.
3. Logs should be sent offsite.
4. A partition should be setup for each service to limit the amount of diskspace a hacker can use up if they decide to write files. You could use a loopback device to mount files as

filesystems for some of these services if you run out of partitions.
5. Root should own all files that do not change.

Now, when it comes to sendmail and syslogd, I still think they should be run under a non-root account. For sendmail, this should be possible, but I found it extremely difficult to run as a non-root account. I haven't been successful getting sendmail to run as a non-root account, and I think it is a serious mistake for it not to be. I know there are problems doing that, but I think they can ALL be taken care of. As long as file permissions are taken care of, I don't see why sendmail needs to be run as root. There might be some reason I am overlooking, but I doubt any of the obstacles can't be overcome.

For syslog, I haven't even tried, but I would say logs should should be logged. under a non-root account and I don't see why that shouldn't be possible. At least I was able to get syslog to be chrooted for each service.
All services should be setup as non-root accounts. Even NFS. Everything.

Suggestions

- Use two logins for ssh and have two running sshd daemons.
- Figure out how to get sendmail or some other mail program running as non-root.
- Strip out the unnecessary libraries under /lib. I just copied everything to make it easy on myself. Most of it you don't need.
- Do remote logging of syslogd and find out if we can attach syslogd to a network port and get all the services to connect to that network port on the loopback device. See if we can get syslogd to run as a non-root account.

Conclusion

I think chroot is cool for all services. I believe it is a big mistake not to chroot all services under non-root accounts. I wish a major distributions would do that, or a smaller distribution: ANY distribution. Mandrake started off by taking stuff from RedHat and expanding off of it, so perhaps, someone should take Mandrake and expand chroot off of them. Nothing prevents people from redoing other people's work in GNU/Linux, so I think it is possible. If some company wanted to chroot everything and create a systematic easy environment for people to manage their chrooted services, they would have a fantastic distribution! Remember, now that Linux is going mainstream, people don't want to see the command line, so if everything is done at a gui level, they don't need to see the guts and they really don't need to know what is going on -- they just need to be able to configure it and know that it just works!

I am in 100% complete support of the idea that all services should be chrooted with non-root accounts and that any distribution that doesn't do this is less than proper for me to use in a production environment. I am going to chroot everything, as much as possible -- eventually I will get there.

I plan on creating s HOWTO about chrooting. I am submitting a request to have someone help me convert this article into LyX format so that it can be put in the HOWTOs for Linux.

This article is re-printed with permission. The originals can be found at:

http://www.linuxfocus.org/English/January2002/article225.shtml

Avoiding Security Holes when Developing an Application - Part 5: Race Conditions

Authors: Frédéric Raynal <pappy@users.sourceforge.net>, Christophe Blaess <ccb@club-internet.fr>, Christophe Grenier <grenier@nef.esiea.fr>

Introduction

The general principle defining race conditions is the following: a process wants to access a system resource exclusively. It checks that the resource is not already used by another process, then uses it as it pleases. The race condition occurs when another process tries to use the same resource in the time-lag between the first process checking that resource and actually taking it over. The side effects may vary. The classical case in OS theory is the deadlock of both processes. More often it leads to application malfunction or even to security holes when a process wrongfully benefits from the privileges another.

What we previously called a *resource* can have different aspects. Most notably the *race conditions* discovered and corrected in the Linux kernel itself due to competitive access to memory areas. Here, we will focus on system applications and we'll deem that the concerned resources are filesystem nodes. This concerns not only regular files but also direct access to devices through special entry points from the `/dev/` directory.
Most of the time, an attack aiming to compromise system security is done against *Set-UID* applications since the attacker can benefit from the privileges of the owner of the executable file. However, unlike previously discussed security holes (buffer overflow, format strings...), race conditions usually don't allow the execution of "customized" code. Rather, they benefit from the resources of a program while it's running. This type of attack is also aimed at "normal" utilities (not *Set-UID*), the cracker lying in ambush for another user, especially *root*, to run the concerned application and access its resources. This is also true for writing to a file (i.e, `~/.rhost` in which the string "+ +" provides a direct access from any machine

without password), or for reading a confidential file (sensitive commercial data, personal medical information, password file, private key...)

Unlike the security holes discussed in our previous articles, this security problem applies to every application and not just to *Set-UID* utilities and system servers or daemons.

First example

Let's have a look at the behavior of a *Set-UID* program that needs to save data in a file belonging to the user. We could, for instance, consider the case of a mail transport software like *sendmail*. Let's suppose the user can both provide a backup filename and a message to write into that file, which is plausible under some circumstances. The application must then check if the file belongs to the person who started the program. It also will check that the file is not a symlink to a system file. Let's not forget, the program being *Set-UID root*, it is allowed to modify any file on the machine. Accordingly, it will compare the file's owner to its own real UID. Let's write something like:

```
1       /* ex_01.c */
2       #include <stdio.h>
3       #include <stdlib.h>
4       #include <unistd.h>
5       #include <sys/stat.h>
6       #include <sys/types.h>
7
8       int
9       main (int argc, char * argv [])
10      {
11          struct stat st;
12          FILE * fp;
13
14          if (argc != 3) {
15              fprintf (stderr, "usage: %s file
message\n", argv [0]);
16              exit(EXIT_FAILURE);
17          }
18          if (stat (argv [1], & st) < 0) {
19              fprintf (stderr, "can't find %s\n",
argv [1]);
20              exit(EXIT_FAILURE);
21          }
22          if (st . st_uid != getuid ()) {
23              fprintf (stderr, "not the owner of
%s \n", argv [1]);
24              exit(EXIT_FAILURE);
25          }
26          if (! S_ISREG (st . st_mode)) {
27              fprintf (stderr, "%s is not a normal
file\n", argv[1]);
28              exit(EXIT_FAILURE);
29          }
30
31          if ((fp = fopen (argv [1], "w")) ==
NULL) {
32              fprintf (stderr, "Can't open\n");
33              exit(EXIT_FAILURE);
34          }
35          fprintf (fp, "%s\n", argv [2]);
36          fclose (fp);
37          fprintf (stderr, "Write Ok\n");
38          exit(EXIT_SUCCESS);
39      }
```

As we explained in our first article, it would be better for a *Set-UID* application to temporarily drop its privileges and open the file using the real UID of the user having called it. As a matter of fact, the above situation corresponds to a daemon, providing services to every user. Always running under the *root* ID, it would check using the UID instead of its own real UID. Nevertheless, we'll keep this scheme for now, even if it isn't that realistic, since it allows us to understand the problem while easily "exploiting" the security hole.

As we can see, the program starts doing all the needed checks, i.e. that the file exists, that it belongs to the user and that it's a normal file. Next, it actually opens the file and writes the message. That is where the security hole lies! Or, more exactly, it's within the lapse of time between the reading of the file attributes with `stat()` and its opening with `fopen()`. This lapse of time is often extremely short but an attacker can benefit from it to change the file's characteristics. To make our attack even easier, let's add a line that causes the process to sleep between the two operations, thus having the time to do the job by hand. Let's change the line 30 (previously empty) and insert:

```
30          sleep (20);
```

Now, let's implement it; first, let's make the application *Set-UID root*. Let's make, **it's very important**, a backup copy of our password file `/etc/shadow`:

```
$ cc ex_01.c -Wall -o ex_01
$ su
Password:
# cp /etc/shadow /etc/shadow.bak
# chown root.root ex_01
# chmod +s ex_01
# exit
$ ls -l ex_01
-rwsrwsr-x 1 root  root   15454 Jan 30 14:14 ex_01
$
```

Everything is ready for the attack. We are in a directory belonging to us. We have a *Set-UID root* utility (here `ex_01`) holding a security hole, and we feel like replacing the line concerning *root* from the `/etc/shadow` password file with a line containing an empty password.

First, we create a `fic` file belonging to us:

```
$ rm -f fic
$ touch fic
```

Next, we run our application in the background "to keep the lead". We ask it to write a string into that file. It checks what it has to, sleeps for a while before really accessing the file.

```
$ ./ex_01 fic "root::1:99999:::::" &
[1] 4426
```

The content of the `root` line comes from the `shadow(5)` man page, the most important being the empty second field (no password). While the process is asleep, we have about 20 seconds to remove the `fic` file and replace it with a link (symbolic or physical, both work) to the `/etc/shadow` file. Let's remember, that every user can create a link to a file in a directory belonging to him even if he can't read the content, (or in `/tmp`, as we'll see a bit later). However it isn't possible to create a *copy* of such a file, since it would require a full read.

```
$ rm -f fic
$ ln -s /etc/shadow ./fic
```

Then we ask the shell to bring the `ex_01` process back to the foreground with the `fg` command, and wait till it finishes:

```
$ fg
./ex_01 fic "root::1:99999:::::"
Write Ok
$
```

Voilà! It's over, the `/etc/shadow` file only holds one line indicating *root* has no password. You don't believe it ?

```
$ su
# whoami
root
# cat /etc/shadow
root::1:99999:::::
#
```

Let's finish our experiment by putting the old password file back:

```
# cp /etc/shadow.bak /etc/shadow
cp: replace `/etc/shadow'? y
#
```

Let's be more realistic

We succeeded in exploiting a race condition in a *Set-UID root* utility. Of course, this program was very "helpful" waiting for 20 seconds giving us time to modify the files behind its back. Within a real application, the race condition only applies for a very short time. How do we take advantage of that ?

Usually, the cracker relies on a brute force attack, renewing the attempts hundreds, thousands or ten thousand times, using scripts to automate the sequence. It's possible to improve the chance of "falling" into the security hole with various tricks aiming at increasing the lapse of time between the two operations that the program wrongly considers as atomically linked. The idea is to slow down the target process to manage the delay preceding the file modification more easily. Different approaches can help us to reach our goal:

- To reduce the priority of the attacked process as much as possible by running it with the `nice -n 20` prefix;
- To increase the system load, running various processes that do CPU time consuming loops (like `while (1);`);
- The kernel doesn't allow debugging *Set-UID* programs, but it's possible to force a pseudo step by step execution sending *SIGSTOP-SIGCONT* signal sequences thus allowing to temporarily lock the process (like with the Ctrl-Z key combination in a shell) and then restart it when needed.

The method allowing us to benefit from a security hole based in race condition is boring and repetitive, but it really is usable! Let's try to find the most effective solutions.

Possible improvement

The problem discussed above relies on the ability to change an object's characteristics during the time-lapse between two operations, the whole thing being as continuous as possible. In the previous situation, the change did not concern the file itself. By the way, as a normal user it would have been quite difficult to modify, or even to read, the `/etc/shadow` file. As a matter of fact, the change relies on the link between the existing file node in the name tree and the file itself as a physical entity. Let's remember most of the system commands (`rm`, `mv`, `ln`, etc.) act on the file name not on the file content. Even when you delete a file (using `rm` and the `unlink()` system call), the content is really deleted when the last physical link - the last reference - is removed.

The mistake made in the previous program is considering the association between the name of the file and its content as unchangeable, or at least constant, during the lapse of time between `stat()` and `fopen()` operation. Thus, the example of a physical link should suffice to verify that this association is not a permanent one at all. Let's take an example using this type of link. In a directory belonging to us, we create a new link to a system file. Of course, the file's owner and the access mode are kept. The `ln` command `-f` option forces the creation, even if that name already exists:

```
$ ln -f /etc/fstab ./myfile
$ ls -il /etc/fstab myfile
8570 -rw-r--r--    2 root   root   716 Jan 25 19:07
/etc/fstab
8570 -rw-r--r--    2 root   root   716 Jan 25 19:07
myfile
$ cat myfile
/dev/hda5     /                  ext2      defaults,mand
1 1
/dev/hda6     swap               swap      defaults
0 0
/dev/fd0      /mnt/floppy        vfat      noauto,user
0 0
/dev/hdc      /mnt/cdrom         iso9660
noauto,ro,user  0 0
/dev/hda1     /mnt/dos           vfat      noauto,user
0 0
/dev/hda7     /mnt/audio         vfat      noauto,user
0 0
/dev/hda8     /home/ccb/annexe   ext2      noauto,user
0 0
none          /dev/pts           devpts
gid=5,mode=620  0 0
none          /proc              proc      defaults
0 0
$ ln -f /etc/host.conf ./myfile
$ ls -il /etc/host.conf myfile
8198 -rw-r--r--    2 root   root    26 Mar 11  2000
/etc/host.conf
8198 -rw-r--r--    2 root   root    26 Mar 11  2000
myfile
$ cat myfile
order hosts,bind
multi on
$
```

The `/bin/ls -i` option displays the inode number at the beginning of the line. We can see the same name points to two different physical inodes.

In fact, we would like the functions that check and access the file to always point to the same content and the same inode. And it's possible! The kernel itself automatically manages this association when it provides us with a file descriptor. When we open a file for reading, the `open()` system call returns an integer value, that is the descriptor, associating it with the physical file by an internal table. All the reading we'll do next will concern this file content, no matter what happens to the name used during the file open operation.

Let's emphasize that point: once a file has been opened, every operation on the filename, including removing it, will have no effect on the file content. As long as there is still a process holding a descriptor for a file, the file content isn't removed from the disk, even if its name disappears from the directory where it was stored. The kernel maintains the association to the file content between the `open()` system call providing a file descriptor and the release of this descriptor by `close()` or the process ends.

So there we have our solution! We can open the file and then check the permissions by examining the descriptor characteristics instead of the filename ones. This is done using the `fstat()` system call (this last working like `stat()`), but checking a file descriptor rather than a path. To access the content of the file using the descriptor we'll use the `fdopen()` function (that works like `fopen()`) while relying on a descriptor rather than on a filename. Thus, the program becomes:

```
1     /* ex_02.c */
2     #include <fcntl.h>
3     #include <stdio.h>
4     #include <stdlib.h>
5     #include <unistd.h>
6     #include <sys/stat.h>
7     #include <sys/types.h>
8
9      int
10     main (int argc, char * argv [])
11     {
12         struct stat st;
13         int fd;
14         FILE * fp;
15
16         if (argc != 3) {
17             fprintf (stderr, "usage: %s file
message\n", argv [0]);
18             exit(EXIT_FAILURE);
19         }
20         if ((fd = open (argv [1], O_WRONLY, 0))
< 0) {
21             fprintf (stderr, "Can't open %s\n",
argv [1]);
22             exit(EXIT_FAILURE);
23         }
24         fstat (fd, & st);
25         if (st . st_uid != getuid ()) {
26             fprintf (stderr, "%s not owner!\n",
argv [1]);
27             exit(EXIT_FAILURE);
28         }
29         if (! S_ISREG (st . st_mode)) {
30             fprintf (stderr, "%s not a normal
file\n", argv[1]);
31             exit(EXIT_FAILURE);
32         }
33         if ((fp = fdopen (fd, "w")) == NULL) {
34             fprintf (stderr, "Can't open\n");
35             exit(EXIT_FAILURE);
36         }
37         fprintf (fp, "%s", argv [2]);
38         fclose (fp);
39         fprintf (stderr, "Write Ok\n");
40         exit(EXIT_SUCCESS);
41     }
```

This time, after line 20, no change to the filename (deleting, renaming, linking) will affect our program's behavior; the content of the original physical file will be kept.

Guidelines

When manipulating a file it's important to ensure the association between the internal representation and the real content stays constant. Preferably, we'll use the following system calls to manipulate the physical file as an already open descriptor rather than their equivalents using the path to the file:

System call	*Use*
`fchdir (int fd)`	Goes to the directory represented by *fd*.
`fchmod (int fd, mode_t mode)`	Changes the file access rights.
`fchown (int fd, uid_t uid, gid_t gif)`	Changes the file owner.
`fstat (int fd, struct stat * st)`	Consults the informations stored within the inode of the physical file.
`ftruncate (int fd, off_t length)`	Truncates an existing file.
`fdopen (int fd, char * mode)`	Initializes IO from an already open descriptor. It's an *stdio* library routine, not a system call.

Then, of course, you must open the file in the wanted mode, calling `open()` (don't forget the third argument when creating a new file). More on `open()` later when we discuss the temporary file problem.

We must insist that it is important to check the system calls return codes. For instance, let's mention, even if it has nothing to do with *race conditions*, a problem found in old `/bin/login` implementations because it neglected an error code check. This application, automatically provided a *root* access when not finding the `/etc/passwd` file. This behavior can seem acceptable as soon as a damaged file system repair is concerned. On the other hand, checking that it was impossible to open the file instead of checking if the file really existed, was less acceptable. Calling `/bin/login` after opening the maximum number of allowed descriptors allowed any user to get *root* access... Let's finish with this digression insisting in how it's important to check, not only the system call's success or failure, but the error codes too, before taking any action about system security.

Race conditions to the file content

A program dealing with system security shouldn't rely on the exclusive access to a file content. More exactly, it's important to properly manage the risks of race conditions to the same file. The main danger comes from a user running multiple instances of a *Set-UID root* application simultaneously or establishing multiple connections at once with the same daemon, hoping to create a race condition situation, during which the content of a system file could be modified in an unusual way.

To avoid a program being sensitive to this kind of situation, it's necessary to institute an exclusive access mechanism to the file data. This is the same problem as the one found in databases when various users are allowed to simultaneously query or change the content of a file. The principle of file locking solves this problem.

When a process wants to write into a file, it asks the kernel to lock that file - or a part of it. As long as the process keeps the lock, no other process can ask to lock the same file, or at least the same part of the file. In the same way, a process asks for a lock before reading the file content to ensure no changes will be made while it holds the lock.

As a matter of fact, the system is more clever than that: the kernel distinguishes between the locks required for file reading and those for file writing. Various processes can hold a lock for reading simultaneously since no one will attempt to change the file content. However, only one process can hold a lock for writing at a given time, and no other lock can be provided at the same time, even for reading.

There are two types of locks (mostly incompatible with each other). The first one comes from BSD and relies on the `flock()` system call. Its first argument is the descriptor of the file you wish to access in an exclusive way, and the second one is a symbolic constant representing the operation to be done. It can have different values: `LOCK_SH` (lock for reading), `LOCK_EX` (for writing), `LOCK_UN` (release of the lock). The system call blocks as long as the requested operation remains impossible. However, you can do a binary OR | of the `LOCK_NB` constant for the call to fail instead of staying locked.

The second type of lock comes from System V, and relies on the `fcntl()` system call whose invocation is a bit complicated. There's a library function called `lockf()` close to the system call but not as fast. `fcntl()`'s first argument is the descriptor of the file to lock. The second one represents the operation to be performed: `F_SETLK` and `F_SETLKW` manage a lock, the second command stays blocks till the operation becomes possible, while the first immediately returns in case of failure. `F_GETLK` consults the lock state of a file (which is useless for current applications). The third argument is a pointer to a variable of `struct flock` type, describing the lock. The `flock` structure important members are the following:

Name	*Type*	*Meaning*
`l_type`	`int`	Expected action: `F_RDLCK` (to lock for reading), `F_WRLCK` (to lock for writing) and `F_UNLCK` (to release the lock).
`l_whence`	`int`	`l_start` Field origin (usually `SEEK_SET`).
`l_start`	`off_t`	Position of the beginning of the lock (usually 0).
`l_len`	`off_t`	Length of the lock, 0 to reach the end of the file.

We can see `fcntl()` can lock limited portions of the file, but it's able to do much more compared to `flock()`. Let's have a look at a small program asking for a lock for reading concerning files which names are given as an argument, and waiting for the user to press the Enter key before finishing (and thus releasing the locks).

```
/* ex_03.c */
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int
main (int argc, char * argv [])
{
  int i;
  int fd;
  char buffer [2];
  struct flock lock;

  for (i = 1; i < argc; i ++) {
    fd = open (argv [i], O_RDWR | O_CREAT, 0644);
    if (fd < 0) {
      fprintf (stderr, "Can't open %s\n", argv [i]);
      exit(EXIT_FAILURE);
    }
    lock . l_type = F_WRLCK;
    lock . l_whence = SEEK_SET;
    lock . l_start = 0;
    lock . l_len = 0;
    if (fcntl (fd, F_SETLK, & lock) < 0) {
      fprintf (stderr, "Can't lock %s\n", argv [i]);
      exit(EXIT_FAILURE);
    }
  }
  fprintf (stdout, "Press Enter to release the lock(s)\n");
  fgets (buffer, 2, stdin);
  exit(EXIT_SUCCESS);
}
```

We first launch this program from a first console where it waits:

```
$ cc -Wall ex_03.c -o ex_03
$ ./ex_03 myfile
Press Enter to release the lock(s)
```

>From another terminal...

```
$ ./ex_03 myfile
Can't lock myfile
$
```

locks.

With this locking mechanism, you can prevent race conditions to directories and print queues, like the lpd daemon, using a flock() lock on the /var/lock/subsys/lpd file, thus allowing only one instance. You can also manage the access to a system file in a secure way like /etc/passwd, locked using fcntl() from the *pam* library when changing a user's data.

However, this only protects from interferences with applications having correct behavior, that is, asking the kernel to reserve the proper access before reading or writing to an important system file. We now talk about cooperative lock, what shows the application liability towards data access. Unfortunately, a badly written program is able to replace file content, even if another process, with good behavior, has a lock for writing. Here is an example. We write a few letters into a file and lock it using the previous program:

```
$ echo "FIRST" > myfile
$ ./ex_03 myfile
Press Enter to release the lock(s)
```

>From another console, we can change the file:

```
$ echo "SECOND" > myfile
$
```

Back to the first console, we check the "damages":

```
(Enter)
$ cat myfile
SECOND
$
```

To solve this problem, the Linux kernel provides the sysadmin with a locking mechanism coming from System V. Therefore you can only use it with fcntl() locks and not with flock(). The administrator can tell the kernel the fcntl() locks are *strict*, using a particular combination of access rights. Then, if a process locks a file for writing, another process won't be able to write into that file (even as *root*). The particular combination is to use the *Set-GID* bit while the execution bit is removed for the group. This is obtained with the command:

```
$ chmod g+s-x myfile
$
```

However this is not enough. For a file to automatically benefit from strict cooperative locks, the *mandatory* attribute must be activated on the partition where it can be found. Usually, you have to change the /etc/fstab file to add the mand option in the 4th column, or typing the command:

```
# mount /dev/hda5 on / type ext2 (rw)
[...]
# mount / -o remount,mand
# mount /dev/hda5 on / type ext2 (rw,mand)
[...]
#
```

Now, we can check that a change from another console is impossible:

```
$ ./ex_03 myfile
Press Enter to release the lock(s)
```

>From another terminal:

```
$ echo "THIRD" > myfile
bash: myfile: Resource temporarily not available
$
```

And back to the first console:

```
(Enter)
$ cat myfile
SECOND
$
```

The administrator and not the programmer has to decide to make strict file locks (for instance /etc/passwd, or /etc/shadow). The programmer has to control the way the data is accessed, what ensures his application to manages data coherently when reading and it is not dangerous for other processes when writing, as long as the environment is properly administrated.

Temporary files

Very often a program needs to temporarily store data in an external file. The most usual case is inserting a record in the middle of a sequential ordered file, which implies that we make a copy of the original file in a temporary file, while adding new information. Next the unlink() system call removes the original file and rename() renames the temporary file to replace the previous one.

Opening a temporary file, if not done properly, is often the starting point of race condition situations for an ill-intentioned user. Security holes based on the temporary files have been recently discovered in applications such as *Apache*, *Linuxconf*, *getty_ps*, *wu-ftpd*, *rdist*, *gpm*, *inn*, etc. Let's remember a few principles to avoid this sort of trouble.

Usually, temporary file creation is done in the /tmp directory. This allows the sysadmin to know where short term data storage is done. Thus, it's also possible to program a periodic cleaning (using cron), the use of an independent partition formated at boot time, etc. Usually, the administrator defines the location reserved for temporary files in the <paths.h> and <stdio.h> files, in the _PATH_TMP and P_tmpdir symbolic constants definition. As a matter of fact, using another default directory than /tmp is not that good, since it would imply recompiling every application, including the C library. However, let's mention that GlibC routine behavior can be defined using the TMPDIR environment variable. Thus, the user can ask the temporary files to be stored in a directory belonging to him rather than in /tmp. This is sometimes mandatory when the partition dedicated to /tmp is too small to run applications requiring big amount of temporary storage.

The /tmp system directory is something special because of its access rights:

```
$ ls -ld /tmp
drwxrwxrwt 7 root  root    31744 Feb 14 09:47 /tmp
$
```

The *Sticky-Bit* represented by the letter t at the end or the 01000 octal mode, has a particular meaning when applied to a directory: only the directory owner (*root*), and the owner of a file found in that directory are able to delete the file. The directory having a full write access, each user can put his files in it, being sure they are protected - at least till the next clean up managed by the sysadmin.

Nevertheless, using the temporary storage directory may cause a few problems. Let's start with the trivial case, a Set-UID *root* application talking to a user. Let's talk about a mail transport program. If this process receives a signal asking it to finish immediately, for instance *SIGTERM* or *SIGQUIT* during a system *shutdown*, it can try to save on the fly the mail already written but not sent. With old versions, this was done in /tmp/dead.letter. Then, the user just had to create (since he can write into /tmp) a physical link to /etc/passwd with the name dead.letter for the mailer (running under effective UID *root*) to write to this file the content of the not yet finished mail (incidently containing a line "root::1:99999:::::").

The first problem with this behavior is the foreseeable nature of the filename. You can to watch such an application only once to deduct it will use the /tmp/dead.letter file name. Therefore, the first step is to use a filename defined for the current program instance. There are various library functions able to provide us with a personal temporary filename.

Let's suppose we have such a function providing a unique name for our temporary file. Free software being available with source code (and so for C library), the filename is however foreseeable even if it's rather difficult. An attacker could create a symlink to the name provided by the C library. Our first reaction is to check the file exists before opening it. Naively we could write something like:

```
if ((fd = open (filename, O_RDWR)) != -1) {
   fprintf (stderr, "%s already exists\n",
      filename);
   exit(EXIT_FAILURE);
}
fd = open (filename, O_RDWR | O_CREAT, 0644);
...
```

Obviously, this is a typical case of *race condition*, where a security hole opens following the action from a user succeeding in creating a link to /etc/passwd between the first open() and the second one. These two operations have to be done in an atomic way, without any manipulation able to take place between them. This is possible using a specific option of the open() system call. Called *O_EXCL*, and used in conjunction with *O_CREAT*, this option makes the open() fail if the file already exists, but the check of existence is atomically linked to the creation.

By the way, the 'x' Gnu extension for the opening modes of the fopen() function, requires an exclusive file creation, failing if the file already exists:

```
FILE * fp;

if ((fp = fopen (filename, "r+x")) == NULL) {
   perror ("Can't create the file.");
   exit (EXIT_FAILURE);
}
```

The temporary files permissions are quite important too. If you have to write confidential information into a mode 644 file (read/write for the owner, read only for the rest of the world) it can be a bit of a nuisance. The

```
#include <sys/types.h>
#include <sys/stat.h>

mode_t umask(mode_t mask);
```

function allows us to determine the permissions of a file at creation time. Thus, following a umask(077) call, the file will be open in mode 600 (read/write for the owner, no rights at all for the others).

Usually, the temporary file creation is done in three steps:

1. unique name creation (random) ;
2. file opening using O_CREAT | O_EXCL, with the most restrictive permissions;
3. checking the result when opening the file and reacting accordingly (either retry or quit).

How create a temporary file ? The

```
#include <stdio.h>

char *tmpnam(char *s);
char *tempnam(const char *dir,
   const char *prefix);
```

functions return pointers to randomly created names. The first function accepts a NULL argument, then it returns a static buffer address. Its content will change at tmpnam(NULL) next call. If the argument is an allocated string, the name is copied there, what requires a string of at least L_tmpnam bytes. Be careful with buffer overflows! The man page informs about problems when the function is used with a NULL parameter, if _POSIX_THREADS or _POSIX_THREAD_SAFE_FUNCTIONS are defined.

The tempnam() function returns a pointer to a string. The dir directory must be "suitable" (the man page describes the right meaning of "suitable"). This function checks the file doesn't exist before returning its name. However, once again, the man page doesn't recommend its use, since "suitable" can have a different meaning according to the function implementations. Let's mention that Gnome recommends its use in this way:

```
char *filename;
int fd;

do {
   filename = tempnam (NULL, "foo");
   fd = open (filename,
        O_CREAT | O_EXCL | O_TRUNC | O_RDWR, 0600);
   free (filename);
} while (fd == -1);
```

The loop used here, reduces the risks but creates new ones. What would happen if the partition where you want to create the temporary file is full, or if the system already opened the maximum number of files available at once...

The

```
        #include <stdio.h>

        FILE *tmpfile (void);
```

function creates an unique filename and opens it. This file is automatically deleted at closing time.

With GlibC-2.1.3, this function uses a mechanism similar to `tmpnam()` to generate the filename, and opens the corresponding descriptor. The file is then deleted, but Linux really removes it when no resources at all use it, that is when the file descriptor is released, using a `close()` system call.

```
FILE * fp_tmp;

if ((fp_tmp = tmpfile()) == NULL) {
   fprintf (stderr,
      "Can't create a temporary file\n");
   exit (EXIT_FAILURE);
}

/* ... use of the temporary file ... */

fclose (fp_tmp);  /* real deletion from the system
*/
```

The simplest cases don't require filename change nor transmission to another process, but only storage and data re-reading in a temporary area. We therefore don't need to know the name of the temporary file but only to access its content. The `tmpfile()` function does it.

The `man` page says nothing, but the Secure-Programs-HOWTO doesn't recommend it. According to the author, the specifications don't guarantee the file creation and he hasn't been able to check every implementation. Despite this reserve, this function is the most efficient.

Last, the

```
      #include <stdlib.h>

      char *mktemp(char *template);
      int mkstemp(char *template);
```

functions create an unique name from a template made of a string ending with "XXXXXX". These 'X's are replaced to get an unique filename.

According to versions, `mktemp()` replaces the first five 'X' with the *Process ID* (PID) ... what makes the name rather easy to guess: only the last 'X' is random. Some versions allow more than six 'X'.

`mkstemp()` is the recommended function in the Secure-Programs-HOWTO. Here is the method:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>

void failure(msg) {
   fprintf(stderr, "%s\n", msg);
   exit(1);
}

/*
 * Creates a temporary file and returns it.
 * This routine removes the filename from the
 * filesystem thus it doesn't appear anymore
 * when listing the directory.
 */
FILE *create_tempfile(char *temp_filename_pattern)
{
   int temp_fd;
   mode_t old_mode;
   FILE *temp_file;

   /* Create file with restrictive permissions */
   old_mode = umask(077);
   temp_fd = mkstemp(temp_filename_pattern);
   (void) umask(old_mode);
   if (temp_fd == -1) {
      failure("Couldn't open temporary file");
   }
   if (!(temp_file = fdopen(temp_fd, "w+b"))) {
      failure("Couldn't create temporary file's
file descriptor");
   }
   if (unlink(temp_filename_pattern) == -1) {
      failure("Couldn't unlink temporary file");
   }
   return temp_file;
}
```

These functions show the problems concerning abstraction and portability. That is, the standard library functions are expected to provide features (abstraction)... but the way to implement them varies according to the system (portability). For instance, the `tmpfile()` function opens a temporary file in different ways (some versions don't use `O_EXCL`), or `mkstemp()` handles a variable number of 'X' according to implementations.

Conclusion

We flew over most of the security problems concerning race conditions to the same resource. Let's remember you must never assume that two consecutive operations are always sequentially processed in the CPU unless the kernel manages this. If race conditions generate security holes, you must not neglect the holes caused by relying on other resources, such as variables shared between threads or memory segments shared using `shmget()`. Selection access mechanisms (semaphore, for example) must be used to avoid hard to discover bugs.

Links

- *Secure-Programs-HOWTO* by David A. Wheeler: www.linuxdoc.org/HOWTO/Secure-Programs-HOWTO/

This article is re-printed with permission. The originals can be found at: http://www.linuxfocus.org/English/September2001/article198.shtml

Kerberos: The Watchdog of the Ether

Author: Raj Shekhar <rajshekhar3007@yahoo.co.in>

Introduction

The first computer networks were used to send e-mails and share files and printers between researchers and corporate employees. In such a scenario security was not given much thought. Now the computer networks (especially the Internet) are used by millions for banking, shopping and filing their tax returns, and network security has become a major problem. Network security can be divided into four areas.

Secrecy
Secrecy has to do with keeping information out of the reach of nosy unauthorized people.

Authentication
Authentication deals with determining the identity of the communication partner, whether it/he/she is an impostor or the real thing.

Non repudiation
Non repudiation deals with signatures: it uniquely identifies the sender of a message or file. How can you prove that the order for "10 million left shoes only" came from your customer when he claims he ordered "10 right shoes only".

Integrity control
Integrity control assures that vital data has not been modified. Integrity is critical for conducting commerce over the Internet. Without assured integrity, purchase orders, contracts, specifications, or stock purchase orders could be modified with devastating effects.

Need for Authentication

Why do we need an authentication service? An authentication service verifies the identity of the communication partner. Authentication is a fundamental building block of a secure network environment. If a server knows for certain the identity of its client, it can decide whether to provide it a particular service (for example.. printing facility) or not, whether to give the user special privileges etc. As an aside authentication and authorization are different. If user Foo says "delete file bar", then the problem of verifying whether the command came from Foo is authentication. The problem of verifying whether Foo has permission to delete file bar is authorization.

Let's take an example of Alice, who wishes to deal with Bob, her banker. In real life Bob and Alice can authenticate each other by recognizing each others faces, voices or handwriting . However if they wish to transact over network none of these options are available. How can Bob be sure that the request to transfer all of Alice's money to a secret Swiss bank account came from Alice and not from Eve?

This is where an authentication service comes in. Alice starts by sending out a message to Bob. As these messages are being sent, we have Eve, an intruder, who may intercept, modify or replay the messages to trick Alice and Bob or just to throw a spanner in the works. Nevertheless when the authentication is complete, Alice is sure she is talking to Bob and Bob is sure that he is talking to Alice.

Enter Kerberos

Kerberos was created by MIT as a solution to network security problems. It has its roots in Project Athena, started in 1983. The aim of Project Athena was to create an educational computing environment built around high-performance graphic workstations, high speed networking, and servers of various types. Project Athena used Kerberos as its authentication system. The name Kerberos comes from Greek mythology; it is the three-headed dog that guarded the entrance to Hades. The Kerberos protocol uses strong cryptography so that a client can prove its identity to a server (and vice verse) across an insecure network connection. After a client and server have used Kerberos to prove their identity, they can also encrypt all of their communications to assure privacy and data integrity as they go about their business.

From http://web.mit.edu/Kerberos/www/

> *Many of the protocols used in the Internet do not provide any security. Tools to "sniff" passwords off the network are in common use by systems crackers. Thus, applications which send an unencrypted password over the network are extremely vulnerable. Worse yet, other client/server applications rely on the client program to be "honest" about the identity of the user who is using it .Other applications rely on the client to restrict its activities to those which it is allowed to do, with no other enforcement by the server.*

The original design and implementation of Kerberos Versions 1 through 4 was the work of two former Project Athena staff members, Steve Miller of Digital Equipment Corporation and Clifford Neuman (now at the Information Sciences Institute of the University of Southern California), along with Jerome Saltzer, Technical Director of Project Athena, and Jeffrey Schiller, MIT Campus Network Manager. Many other members of Project Athena have also contributed to the work on Kerberos. The latest version of Kerberos 4 from MIT is patch level 10.It is officially considered "dead" by MIT; all current development is concentrated on Kerberos 5. The latest version of Kerberos 5 is 1.2.1.

Some Keywords First

The art of devising ciphers is known as **cryptography** and breaking them is known as **cryptanalysis**; together they are known as **cryptology** . The message to be encrypted is known as **plaintext** or **cleartext**. The plaintext is encrypted by using a function, which takes as a parameter a **key**. The output of the encryption process is known as **ciphertext** .When ciphertext is put through a **decryption function**, we get back the plaintext. Going back to our story of Alice and Bob, they (Alice and Bob) are sometimes referred to as **principals**, the main characters of the story.

Traditionally, the encryption key is same as the decryption key. The key is known only to the principals. Such a key is known as **shared secret key**. However in a cypto system proposed by Diffie and Hellman (researchers at Stanford University) in 1976, the encryption and decryption keys are different. The key to be used for encryption is made public so that messages to be sent to that user can be encrypted using the publicly available key. This key is known as the **public key**. Each user also has a **private key** ,known only to the user, which is used for decrypting messages sent to the user. This system is known as **public-key cryptography** , to contrast with shared-key cryptography. The RSA algorithm is an example of public-key cryptography.

And Some More ...

Before describing the authentication process, it is important to remove ambiguities in the terms to be used.

Often network applications are made of two parts,

- the part which requests a service, called the client side of the application
- the part which provides the service, called the server side of the application

In a sense, every entity that uses the Kerberos system is a client. To distinguish between the Kerberos client and the client of a service, the client using the Kerberos service is known as a **Kerberos client**. The term **application server** refers to the server part of the application, that the clients communicate with using Kerberos for authentication.

Kerberos is a **trusted third party authentication system**. It is trusted in the sense that each of its client believes the judgment of the Kerberos' as to the identity of each of its other client to be accurate. To prove to the application server that it (Kerberos client) is trusted by the Kerberos server, it uses a **ticket**. In order for the Kerberos client to use any application server, a ticket is required. The server examines the ticket to verify the identity of the user. If all checks out, then the client is accepted. Along with a ticket an **authenticator** is also used by a Kerberos client to prove its identity. The authenticator contains the additional information which, when compared against that in the ticket proves that the client presenting the ticket is the same one to which the ticket was issued.

Kerberos maintains a database of its clients and their private keys for authentication. Because Kerberos knows these private keys, it can create messages which convince one client that another is really who it claims to be. The designers did not expect the entire world to trust a single database, so they made provision for having different **realms**. The realm is an administrative entity that maintains authentication data. Each organization wishing to run a Kerberos server establishes its own "realm".

And Now the Details

Kerberos assumes that the Kerberos clients are not trustworthy and requires the client to identify itself every time a service is requested from some other Kerberos client. The technique used by Kerberos are unobtrusive. Kerberos follows the following guidelines:

- Passwords are never sent across the network in cleartext. They are always encrypted. Additionally, passwords are never stored on Kerberos clients or server in cleartext.
- Every client has a password i.e. every application server, user, Kerberos client has a password.
- The only entity that knows all the password is the Kerberos server and it operates under considerable physical security.

Both the client and the application server are required to have keys registered with the authentication server (AS). If the client is a user, his key is derived from a password that he chooses; the key for a service (for example. a printing daemon) is a randomly selected key. These keys are negotiated during the registration of the clients.

The authentication process proceeds as follows:

1. A client sends a request to the authentication server requesting "credentials" for a given application server. The credentials consist of a ticket for the server and a session key. The ticket contains, along with other fields, the name of the server, the name of the client, the Internet address of the client, a timestamp, a lifetime, and session key. This information (the ticket) is encrypted using the key of the server for which the ticket will be used. Once the ticket has been issued, it may be used multiple times by the named client to gain access to the named server, until the ticket expires.

   Why is a timestamp included? The timestamp is put to prevent someone else from copying the ticket and using it to impersonate the Kerberos client at a later time. This type of attack is known as a replay. Because clocks don't always work in perfect synchrony, a small amount of leeway (about five minutes is typical) is given between the timestamp and the current time.

2. The AS responds with these credentials (the ticket and the session key), encrypted in the client's key. The AS also includes its name in the credentials to convince the Kerberos client that the decryption by the server was successful and the message came from the server.

   The AS does not know whether the client is *actually the principal which initiated the request for a the ticket. It simply sends a reply without knowing or caring whether they are the same. This is acceptable because nobody but the Kerberos client whose identity was given in the request will be able to use the reply. Its critical information is encrypted in that principal's key.*

3. The Kerberos client decrypts the credentials using its key to extract the session key. Note that because the ticket is encrypted in the key of the application server, a Kerberos client cannot decrypt it.

4. In order to gain access to the application server, the Kerberos client builds an authenticator containing the client's name and IP address, and the current time. The authenticator is then encrypted in the session key that was received with the ticket for the server. The client then sends the authenticator along with the ticket to the server.

5. The service decrypts the ticket with its own key, extracting the session key and the identity of the Kerberos client which the server sent it inside the ticket. It then opens the authenticator with the session key. The authenticator and the ticket demonstrate the identity of the client.

6. The session key (now shared by the client and application server) is used to authenticate the client, and may optionally be used to authenticate the server. It may also be used to encrypt further communication between the two parties or to exchange a separate sub-session key to be used to encrypt further communication.

The Ticket Granting Ticket

One of the goals of the Kerberos system is to remain as unobtrusive as possible. In the above exchange, the Kerberos client has to enter in a password every time it has to decrypt the credentials passed to it by the AS . If the Kerberos client is a user it becomes quite irritating to enter his password to have a file printed or whenever he wants modify a file on the network (remember that the key is derived from the user's password). The obvious way around this is to cache the key derived from the password. But caching the key is dangerous. With a copy of this key, an attacker could impersonate the user at any time (until the password is next changed).

Kerberos resolves this problem by introducing a new agent, called the ticket granting server (TGS). The TGS is logically distinct from the AS, although they may reside on the same physical machine. (They are often referred to collectively as the KDC--the Key Distribution Center). The function of the TGS is as follows. Before accessing any regular service, the user requests a ticket to contact the TGS, just as if it were any other service. This usually occurs when the user first logins into the system. This ticket is called the ticket granting ticket (TGT). After receiving the TGT, any time that the user wishes to contact a service, he requests a ticket not from the AS, but from the TGS. Furthermore, the reply is encrypted not with the user's secret key, but with the session key that the AS provided for use with the TGS. Inside that reply is the new session key for use with the regular service. The rest of the exchange now continues as described above. The TGT is good only for a fairly short period, typically eight hours.

Cross Realm Authentication

The Kerberos protocol is designed to operate across organizational boundaries. A client in one organization can be authenticated to a server in another. Each organization wishing to run a Kerberos server establishes its own "realm". The name of the realm in which a client is registered is part of the client's name, and can be used by the application server to decide whether to honor a request.

By establishing "inter-realm" keys, the administrators of two realms can allow a client authenticated in the local realm to use its authentication remotely The exchange of inter-realm keys registers the ticket-granting service of each realm as a principal in the other realm. A client is then able to obtain a ticket-granting ticket for the other realm's ticket-granting service from its local realm. When that ticket-granting ticket is used, the other ticket-granting service uses the inter-realm key (which usually differs from its own normal TGS key) to decrypt the ticket-granting ticket, and is thus certain that it was issued by the client's own TGS. Tickets issued by the remote ticket-granting service will indicate to the end-service that the client was authenticated from another realm.

Conclusion

Kerberos is not a one-shot solution to the network security problem. Trust is inherent throughout the system: the client trusts Kerberos, if it correctly provides the client's encryption key. The application trusts the client if the client successfully provides a ticket that is encrypted using the server's key. In this trust lies the weakness of the system.

Specifically speaking, secret keys should be kept just that, secret. If an intruder somehow steals a principal's key, it will be able to impersonate the principal. "Password guessing" attacks are not solved by Kerberos. If a user chooses a poor password, it is possible for an attacker to successfully mount an dictionary attack. Kerberos makes no provisions for client's security; it assumes that it is running on trusted clients with an untrusted network. If the client's security is compromised, then Kerberos is compromised as well. However, the degree to which Kerberos is compromised depends on the host that is compromised. If an attacker breaks into a multi-user machine and steals all of the tickets stored on that machine, he can impersonate the users who have tickets stored on that machine .... but only until those tickets expire.

This article is re-printed with permission. The originals can be found at: http://www.linuxgazette.com/issue82/shekhar.html

Process Tracing Using Ptrace

Author: Sandeep S <sk_nellayi@rediffmail.com >

The ptrace system call is crucial to the working of debugger programs like gdb - yet its behaviour is not very well documented - unless you believe that the best documentation is kernel source itself! I shall attempt to demonstrate how ptrace can be used to implement some of the functionality available in tools like gdb.

1. Introduction

ptrace() is a system call that enables one process to control the execution of another. It also enables a process to change the core image of another process. The traced process behaves normally until a signal is caught. When that occurs the process enters stopped state and informs the tracing process by a **wait()** call. Then tracing process decides how the traced process should respond. The only exception is SIGKILL which surely kills the process.

The traced process may also enter the stopped state in response to some specific events during its course of execution. This happens only if the tracing process has set any event flags in the context of the traced process. The tracing process can even kill the traced one by setting the exit code of the traced process. After tracing, the tracer process may **kill** the traced one or leave to continue with its execution.

Note: Ptrace() is highly dependent on the architecture of the underlying hardware. Applications using ptrace are not easily portable across different architectures and implementations.

2. More Details

The prototype of ptrace() is as follows:

```
#include <sys/ptrace.h>
long int ptrace(
   enum __ptrace_request request,
   pid_t pid,
   void * addr,
   void * data)
```

Of the four arguments, the value of **request** decides what to be done. **Pid** is the ID of the process to be traced. **Addr** is the offset in the user space of the traced process to where the **data** is written when instructed to do so. It is the offset in user space of the traced process from where a word is read and returned as the result of the call.

The parent can fork a child process and trace it by calling ptrace with **request** as PTRACE_TRACEME. Parent can also trace an existing process using PTRACE_ATTACH. The different values of **request** are discussed below.

2.1 How does ptrace() work.

Whenever ptrace is called, what it first does is to lock the kernel. Just before returning it unlocks the kernel. Let's see its working in between this for different values of request.

PTRACE_TRACEME: This is called when the child is to be traced by the parent. As said above, any signals (except SIGKILL), either delivered from outside or from the **exec** calls made by the process, causes it to stop and lets the parent decide how to proceed. Inside ptrace(), the only thing that is checked is whether the ptrace flag of the current process is set. If not, permission is granted and the flag is set. All the parameters other than **request** are ignored.

PTRACE_ATTACH: Here a process wants to control another. One thing to remember is that nobody is allowed to trace/control the **init** process. A process is not allowed to control itself. The current process (caller) becomes the parent of the process with process ID **pid**. But a **getpid()** by the child (the one being traced) returns the process ID of the real parent.

What goes behind the scenes is that when a call is made, the usual permission checks are made along with whether the process is init or current or it is already traced. If there is no problem, permission is given and the flag is set. Now the links of the child process are rearranged; e.g., the child is removed from the task queue and its parent process field is changed (the original parent remains the same). It is put to the queue again in such a position that **init** comes next to it. Finally a SIGSTOP signal is delivered to it. Here **addr** and **data** are ignored.

PTRACE_DETACH: Stop tracing a process. The tracer may decide whether the child should continue to live. This undoes all the effects made by PTRACE_ATTACH/PTRACE_TRACEME. The parent sends the exit code for the child in **data**. Ptrace flag of the child is reset. Then the child is moved to its original position in the task queue. The pid of real parent is written to the parent field. The single-step bit which might have been set is reset. Finally the child is woken up as nothing had happened to it; **addr** is ignored.

PTRACE_PEEKTEXT, PTRACE_PEEKDATA, PTRACE_PEEKUSER: These options read data from child's memory and user space. PTRACE_PEEKTEXT and PTRACE_PEEKDATA read data from memory and both these options have the same effect. PTRACE_PEEKUSER reads from the user space of child. A word is read and placed into a temporary data structure, and with the help of put_user() (which copies a string from the kernel's memory segment to the process' memory segment) the required data is written to **data** and returns 0 on success.

In the case of PTRACE_PEEKTEXT and

PTRACE_PEEKDATA, **addr** is the address of the location to be read from child's memory. In PTRACE_PEEKUSER **addr** is the offset of the word in child's user space; **data** is ignored.

PTRACE_POKETEXT, PTRACE_POKEDATA, PTRACE_POKEUSER: These options are analogous to the three explained above. The difference is that these are used to write the **data** to the memory/user space of the process being traced. In PTRACE_POKETEXT and PTRACE_POKEDATA a word from location **data** is copied to the child's memory location **addr**.

In PTRACE_POKEUSER we are trying to modify some locations in the task_struct of the process. As the integrity of the kernel has to be maintained, we need to be very careful. After a lot of security checks made by ptrace, only certain portions of the task_struct is allowed to change. Here **addr** is the offset in child's user area.

PTRACE_SYSCALL, PTRACE_CONT: Both these wakes up the stopped process. PTRACE_SYSCALL makes the child to stop after the next system call. PTRACE_CONT just allows the child to continue. In both, the exit code of the child process is set by the ptrace() where the exit code is contained in **data**. All this happens only if the signal/exit code is a valid one. Ptrace() resets the single step bit of the child, sets/resets the syscall trace bit, and wakes up the process; **addr** is ignored.

PTRACE_SINGLESTEP: Does the same as PTRACE_SYSCALL except that the child is stopped after every instruction. The single step bit of the child is set. As above **data** contains the exit code for the child; **addr** is ignored.

PTRACE_KILL: When the child is to be terminated, PTRACE_KILL may be used. How the murder occurs is as follows. Ptrace() checks whether the child is already dead or not. If alive, the exit code of the child is set to **sigkill**. The single step bit of the child is reset. Now the child is woken up and when it starts to work it gets killed as per the exit code.

2.2 More machine-dependent calls

The values of **request** discussed above were independent on the architecture and implementation of the system. The values discussed below are those that allow the tracing process to get/set (i.e., to read/write) the registers of child process. These register fetching/setting options are more directly dependent on the architecture of the system. The set of registers include general purpose registers, floating point registers and extended floating point registers. These more machine-dependent options are discussed below. When these options are given, a direct interaction between the registers/segments of the system is required.

PTRACE_GETREGS, PTRACE_GETFPREGS, PTRACE_GETFPXREGS: These values give the value of general purpose, floating point, extended floating point registers of the child process. The registers are read to the location **data** in the parent. The usual checks for access on the registers are made. Then the register values are copied to the location specified by **data** with the help of getreg() and __put_user() functions; **addr** is ignored.

PTRACE_SETREGS, PTRACE_SETFPREGS, PTRACE_SETFPXREGS: These are values of **request** that allow the tracing process to set the general purpose, floating point, extended floating point registers of the child respectively. There are some restrictions in the case of setting the registers. Some are not allowed to be changed. The data to be copied to the registers will be taken from the location **data** of the parent. Here also **addr** is ignored.

2.3 Return values of ptrace()

A successful ptrace() returns zero. Errors make it return -1 and set **errno**. Since the return value of a successful PEEKDATA/PEEKTEXT may be -1, it is better to check the **errno**. The errors are

EPERM: The requested process couldn't be traced. Permission denied.

ESRCH: The requested process doesn't exist or is being traced.

EIO: The request was invalid or read/write was made from/to invalid area of memory.

EFAULT: Read/write was made from/to memory which was not really mapped.

It is really hard to distinguish between the reasons of EIO and EFAULT. These are returned for almost identical errors.

3. A small example.

If you found the parameter description to be a bit dry, don't despair. I shall not attempt anything of that sort again. I will try to write simple programs which illustrate many of the points discussed above.

Here is the first one. The parent process counts the number of instructions executed by the test program run by the child.

Here the test program is listing the entries of the current directory:

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <syscall.h>
#include <sys/ptrace.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <errno.h>
```

```
int main(void)
{
  long long counter = 0; /*  machine
                instruction counter */
  int wait_val; /* child's return
                 value */
  int pid; /* child's process id */
  puts("Please wait");

  switch (pid = fork()) {
  case -1:
    perror("fork");
    break;
  case 0:
    /*  child process starts */
    ptrace(PTRACE_TRACEME, 0, 0, 0);
    /*
     * must be called in order to
     * allow the control over the child
     * process
     */
    execl("/bin/ls", "ls", NULL);
    /*
     * executes the program and cause
     * the child to stop and send a
     * signal to the parent, the parent
     * can now switch to
     * PTRACE_SINGLESTEP
     */
    break;
    /*  child process ends  */
  default:
    /*  parent process starts       */
    wait(&wait_val);
    /*
     * parent waits for child to
     * stop at next instruction
     * (execl())
     */
    while (wait_val == 1407 ) {
      counter++;
      if (ptrace(PTRACE_SINGLESTEP,
              pid, 0, 0) != 0)
        perror("ptrace");
      /*
       * switch to singlestep tracing
       * and release child if unable
       * call error.
       */
      wait(&wait_val);
      /* wait for next instruction to
       * complete
       */
    }
    /*
     * continue to stop, wait and
     * release until the child is
     * finished; wait_val != 1407
     * Low=0177L and High=05 (SIGTRAP)
     */
  }
  printf("Number of machine instructions :
%lld\n", counter);
  return 0;
}
```

open your favourite editor and write the program. Then run it by typing

```
cc file.c
./a.out
```

You can see the number of instructions needed for listing of your current directory. cd to some other directory and run the program from there and see whether there is any difference. (note that it may take some time for the output to appear, if you are using a slow machine).

4. Conclusion

Ptrace() is heavily used for debugging. It is also used for system call tracing. The debugger forks and the child process created is traced by the parent. The program which is to be debugged is exec'd by the child (in the above program it was "ls") and after each instruction the parent can examine the register values of the program being run. I shall demonstrate programs which exploit ptrace's versatility in the next part of this series. Good bye till then.

Sandeep S

I am a final year student of Government Engineering College in Thrissur, Kerala, India. My areas of interests include FreeBSD, Networking and also Theoretical Computer Science.

Exploring Perl Modules - Part 1: On-The-Fly Graphics with GD

Author: Pradeep Padala <p_padala@yahoo.com>

Welcome to Exploring Perl Modules!!!

Perl modules are considered to be one of the strongest points for perl's success. They contain a lot of re-usable code and of course are free. This is an attempt to trap the treasure trove. There are lot of tutorials and even books written on popular modules like CGI, DBI etc.. For less popular modules, users are left with documentation which is cryptic and sometimes incomplete.

I am starting a series of articles that will attempt to explain some of the less popular but useful modules. During the last year, I came across and programmed with numerous perl modules. I will explain the modules with numerous useful examples from my experience. We will take one module at a time and explore its various uses.

Who should be reading these

Well, you should know perl. We won't be delving much into the basics of perl. There are plenty of documentation, articles and books on perl. Learning

Perl [by Randal Schwartz & Tom Phoenix, ISBN 0596001320] is often recommended for beginners. Once you gain experience, you can try Programming Perl [by Larry Wall, Tom Christiansen and Jon Orwant, ISBN 0596000278].

If you are an average perl programmer and haven't used lot of modules, this is the right place. Modules provide a great way to re-use code and write efficient and compact applications. In each article we will graduate from simple examples to complex examples ending in a real-world application, if appropriate.

Introduction to Modules

Modules provide an effective mechanism to import code and use it. The following line imports a module and makes its functions accessible.

```
use modul;
```

For example if you want to use GD, you would write

```
use GD;
```

Finding and Installing Modules

Before we plunge into the details of programming, here are some instructions for finding and installing modules. We will be using various modules, and most of them are not installed by default. Some modules require libraries which may or may not have been installed. I will mention the things required whenever appropriate. Here are generic instructions for downloading and installing modules.

An easy way to install the module is by using the CPAN module. Run CPAN in interactive mode as

```
perl -MCPAN -e shell
```

Then you can do various tasks like downloading, decompressing and installing modules. For example, for installing GD you can use

```
install GD
```

If you are like me and and are accustomed to configure, make, make install method, here are the steps to install a module.

- Find the module in CPAN's list of all modules [at http://www.cpan.org/modules/00modlist.long.html].
- Download the latest version of the module. For example, the latest GD module can be downloaded from http://www.cpan.org/authors/id/LDS/GD-1.40.tar.gz
- Unzip the module
```
tar zxvf GD-1.40.tar.gz
```
- Build the module
```
perl Makefile.PL
```
(or)
```
perl Makefile.PL PREFIX=/my/perl/directory
```
(if you want to install in /my/perl/directory)
```
make
make test (optional)
```
- Install the modul make install

Ready to Go

So you have installed your favourite module and are raring to learn. In this article we will explore the perl GD module, which provides an interface to GD library [http://www.boutell.com/gd/]. We will also be using the CGI module for the web interface. You don't need to know a great deal of CGI to understand this article. I will explain things where necessary.

Graphics with GD

Let's start the wheels with a simple and effective example:

```
#!/usr/local/bin/perl -w
# Change above line to path to your perl binary

use GD

# Create a new image
$im = new GD::Image(100,100);

# Allocate some colors
$white =
  $im->colorAllocate(255,255,255);
$black = $im->colorAllocate(0,0,0);
$red = $im->colorAllocate(255,0,0);
$blue = $im->colorAllocate(0,0,255);

# Make the background transparent and
# interlaced
$im->transparent($white)
$im->interlaced('true');

# Put a black frame around the picture
$im->rectangle(0,0,99,99,$black);

# Draw a blue oval
$im->arc(50,50,95,75,0,360,$blue);

# And fill it with red
$im->fill(50,50,$red);

# Open a file for writing
open(PICTURE, ">picture.png") or
die("Cannot open file for writing");

# Make sure we are writing to a binary
# stream
binmode PICTURE;

# Convert the image to PNG and print it
# to the file PICTURE
print PICTURE $im->png;
close PICTURE;
```

This is the example given in the GD man page [see http://stein.cshl.org/WWW/software/GD/] with little modifications. This produces a small rectangle with a red oval with blue border. Let's dissect the program.

One of the first things you do with GD library, is create an image handle to work with. The line

```
$im = new GD::Image($width, $height)
```

creates and image with the specified width and height. You can also create an image from an existing image as well. It is useful for manipulating existing images. We will see an example on this in the later part of the article.

Next we need to allocate some colors. As you can guess, the RGB intensities need to be specified for initializing colors. Since we will be using lots of colors, let's write a small function which will initialize a bunch of colors for use.

```
# Save this as init_colors.pl
# Other scripts call this function

sub InitColors {
  my($im) = $_[0];
  # Allocate colors
  $white =
    $im->colorAllocate(255,255,255);
  $black = $im->colorAllocate(0,0,0);
  $red = $im->colorAllocate(255,0,0);
  $blue = $im->colorAllocate(0,0,255);
  $green
    = $im->colorAllocate(0, 255, 0);
  $brown =
    $im->colorAllocate(255, 0x99, 0);
  $violet =
    $im->colorAllocate(255, 0, 255);
  $yellow =
    $im->colorAllocate(255, 255, 0);
}
```

I often refer to this page [http://www.hypersolutions.org/pages/rgbhex.html] for some nice rgb combinations.

The next few lines are straightforward and pretty much self-explanatory. The last lines regarding the file creation require special mention. Since we will be writing an image to a file, we need to put the file handle in binary mode with

```
binmode MYFILEHANDLE;
```

This actually is a no-op on most UNIX-like systems.

Then we write to the file with the usual print command. GD can print the image in various formats. For example if you want to print a jpeg image instead of png, all you need to do is

```
print MYFILEHANDLE $im->jpeg;
```

Simple Drawing

GD offers some simple drawing primitives which can be combined to generate complex graphics. Examine the following script that gives a whirlwind tour of all the simple primitives.

```
#!/usr/local/bin/perl
# Change above line to path to your perl
binary

use GD;
do "init_colors.pl";

# Create a new image
$im = new GD::Image(640,400);

# Allocate some colors
&InitColors($im);

# Make the background transparent
# and interlaced
$im->transparent($white);
$im->interlaced('true');

$x1 = 10;
$y1 = 10;
$x2 = 200;
$y2 = 200;

# Draw a border
$im->rectangle(0, 0, 639, 399, $black);
# A line
$im->line($x1,$y1,$x2,$y2,$red);
# A Dashed Line
$im->dashedLine($x1 + 100, $y1, $x2,
    $y2, $blue);
# Draw a rectangle
$im->rectangle($x1 + 200, $y1, $x2 +
              200, $y2, $green);
# A filled rectangle
$im->filledRectangle($x1 + 400,
    $y1, $x2 + 400, $y2, $brown);
# A circle
$im->arc($x1 + 100, $y1 + 200 + 100,
    50, 50, 0, 360, $violet);

# A polygon
# Make the polygon
$poly = new GD::Polygon;
$poly->addPt($x1 + 200, $y1 + 200);
$poly->addPt($x1 + 250, $y1 + 230);
$poly->addPt($x1 + 300, $y1 + 310);
$poly->addPt($x1 + 400, $y1 + 300);
# Draw it
$im->polygon($poly, $yellow);

# Open a file for writing
open(PICTURE, ">picture.png") or
  die("Cannot open file for writing");

# Make sure we are writing to a binary
stream
binmode PICTURE

# Convert the image to PNG and print it
to the file PICTURE
print PICTURE $im->png;
close PICTURE;
```

The output looks like Figure 1

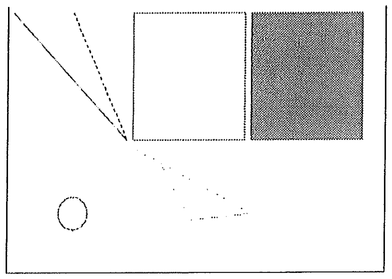


Figure 1: Basic Output from GD

The above script is self-explanatory. The polygon needs a little bit of explanation. In order to draw a polygon, you first have to make the polygon and then draw it. Of course, a polygon must have at least three vertices.

Drawing Text

So what about text? You can draw text in some of the simple fonts provided by GD or use a True Type font available on your system. There are two simple functions available to draw text.

```
# Draw the text
$im->string($font, $x, $y, $string, $color);

# Print text rotated 90 degrees
$im->stringUp($font, $x, $y, $string, $color);

The following script shows various simple fonts
provided by GD

#!/usr/local/bin/perl
# Change above line to path to your perl binary

use GD;
do "init_colors.pl";

# Create a new image
$im = new GD::Image(200, 80);

# Allocate some colors
&InitColors($im);

# Make the background transparent and interlaced
$im->transparent($white);
$im->interlaced('true');

# Create a Border around the image
$im->rectangle(0, 0, 199, 79, $black);
$x1 = 2;
$y1 = 2;

# Draw text in small font
$im->string(gdSmallFont, $x1, $y1, "Small font",
    $blue);
$im->string(gdMediumBoldFont, $x1, $y1 + 20,
    "Medium Bold Font", $green);
$im->string(gdLargeFont, $x1, $y1 + 40,
    "Large font", $red);
$im->string(gdGiantFont, $x1, $y1 + 60,
    "Giant font", $black);

# Open a file for writing
open(PICTURE, ">picture.png") or
    die("Cannot open file for writing");

# Make sure we are writing to a binary stream
binmode PICTURE;

# Convert the image to PNG and print it to the
# file PICTURE
print PICTURE $im->png;
close PICTURE;
```

The output picture looks like Figure 2.

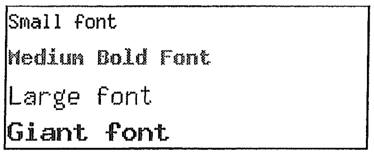


Figure 2: Playing with Fonts

As you can see, these fonts are quite limited and not so attractive. The following section shows the usage of True Type Fonts with GD

True Type Fonts

You can use the true type fonts available on your system to draw some nice text. The function stringFT is used to draw in TTF font.

```
# $fontname is an absolute or relative path to a
# TrueType font.
stringFT($fgcolor,$fgcolor,$fontname,$ptsize,
    $angle,$x,$y,$string);
```

Here's an example showing the usage

```
#!/usr/local/bin/perl
# Change above line to path to your perl binary

use GD;
do "init_colors.pl";

# Create a new image
$im = new GD::Image(270, 80);

# Allocate some colors
&InitColors($im);

# Make the background transparent and interlaced
$im->transparent($white);
$im->interlaced('true');

$im->rectangle(0, 0, 269, 79, $black);

$x1 = 10;
$y1 = 20;

# Draw text in a TTF font
$font =
    "/usr/X11R6/lib/X11/fonts/TTF/luxisri.ttf";
$im->stringFT($red, $font, 15, 0, $x1, $y1,
    "A TTF font");

$anotherfont =
    "/usr/share/fonts/default/TrueType/starbats.ttf
";
$im->stringFT($blue, $font, 20, 0, $x1,
    $y1 + 40, "Another one here !!!");

# Open a file for writing
open(PICTURE, ">picture.png") or
    die("Cannot open file for writing");

# Make sure we are writing to a binary stream
binmode PICTURE;

# Convert the image to PNG and print it to the
# file PICTURE
print PICTURE $im->png;
close PICTURE;
```

The output looks like this:

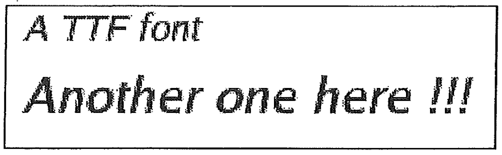


Figure 3: True Type Fonts

Let's go Online

Now that we have seen some basic uses of GD, let's turn our attention to web graphics. So how do you output an image through CGI? Simple. Add the following lines to the scripts instead of printing to a file.

```
# To disable buffering of image content.
select(STDOUT);
$| = 1;
undef $/;

print "Content-type: image/jpeg\n\n";
print $im->jpeg(100);
```

This is all you need to know about CGI for now. If you already know CGI, you can enhance your code for handling complex web interaction. Let's write a small program which reads an image and displays a resized version of it. It might be useful for showing thumbnails.

```
#!/usr/local/bin/perl -wT
# Change above line to path to your
# perl binary

use CGI ':standard';
use GD;

# create a new image
$image_file = "images/surfing.jpg";
$im =
    GD::Image->newFromJpeg($image_file);
($width, $height) = $im->getBounds();
$newwidth = $width / 3;
$newheight = $height / 3;
$outim =
    new GD::Image(
        $newwidth, $newheight);

# make the background transparent and
# interlaced
$outim->copyResized(
    $im, 0, 0, 0, 0, $newwidth,
    $newheight, $width,$height);

# make sure we are writing to a
# binary stream
binmode STDOUT;
select(STDOUT);
$| = 1;
undef $/;
print "Content-type: image/jpeg\n\n";
print $outim->jpeg();
```

In this example, the function newFromJpeg() reads a jpeg file. Then we then calculated the boundaries and resized it accordingly. A demo of the resizing can be found [at http://www.cise.ufl.edu/~ppadala/perl/exploring/cgi-bin/resize.cgi].

A Photo Album

With this resizing knowledge we can create a small online photo album. In this we use resizing to show smaller images and display the original image when the user clicks on the smaller images.

```
#!/usr/local/bin/perl -wT
# Change above line to path to your perl binary

use CGI ':standard';
use GD;

$imnum = param('imnum');
if(!defined($imnum)) {
    $imnum = 0;
}

$orig = param('orig');
if(!defined($imnum)) {
    $orig = 0;
}

select(STDOUT);
$| = 1;

@images = ("surfing.jpg", "boat.jpg",
    "boston-view.jpg", "seashore.jpg");
```

```
print "Content-type: text/html\n\n";
print "<font color=green>Click on the image" .
    " to make it bigger or smaller<br>" .
    "You can browse through the small images".
    "using the buttons or by clicking on the" .
    " numbers </font>\n";
print "<table><tr>\n";

if($imnum > 0 && $imnum < @images) {
    printf "<td><a href=album.cgi?imnum=%d>".
      "<img src=images/prev.giborder=0></a>\n",
      $imnum-1;
}

if($imnum >= 0 && $imnum < @images - 1) {
    printf "<td><a href=album.cgi?imnum=%d>".
      "<img src=images/next.gif border=0></a>\n",
      $imnum+1;
}

print "<td>";
for($i = 0; $i < @images; ++$i) {
    print "<a href=album.cgi?imnum=$i>".
        "$i|</a>\n";
}
print "</tr></table>\n";
if($imnum < 0 || $imnum >= @images) {
    print "<b>No such image</b>";
    exit;
}

if($orig) {
    print "<a href=album.cgi?imnum=$imnum>".
        "<img src=images/$images[$imnum]".
        "border=0></img></a>\n";
} else {
    $im =
      GD::Image->newFromJpeg(
        "images/$images[$imnum]");
    # create a new image
    ($width, $height) = $im->getBounds();
    $newwidth = 200;
    $newheight = 200;
    $outim = new GD::Image(
        $newwidth, $newheight);

    $outim->copyResized($im, 0, 0, 0, 0,
        $newwidth, $newheight, $width, $height);
    $tmpfile = "images/tmp$imnum.jpg";
    if ($tmpfile =~ /^([-\@\w.\/]+)$/) {
        # For the tainting stuff
        $tmpfile = $1;
    } else {
        print "Should never happen";
        exit; # Should never happen
    }
    open(TMP, ">$tmpfile")
        || die("Cannot open file");
    binmode(TMP);
    print TMP $outim->jpeg(100);
    close(TMP);
    chmod(0644, $tmpfile);
    print "<a href=".
        " album.cgi?imnum=$imnum&orig=1>".
        "<img src=$tmpfile border=0></a>";
}
```

This script uses a few CGI features. The function param returns theparameter value, if supplied. This value is used to display the proper image. If the user wants to see an original image, it is displayed.Otherwise a temporary resized image is created and displayed.

A demo of the album is [available at http://www.cise.ufl.edu/~ppadala/perl/exploring/cgi-bin/album.cgi>]

A Graphical Hit Counter

Now let us turn our attention to another popular web application "A Hit Counter". There are many counter scripts available on web. Here's our attempt to write one.

The counter works like this. Every time a web-page is accessed, the cgi script records the hit count and creates an image on-the-fly. So why wait? Let's write it.

```
#!/usr/local/bin/perl -wT
use CGI ':standard';
use GD;
use strict;

my($LOCK_SH, $LOCK_EX, $LOCK_NB,
    $LOCK_UN);

$LOCK_SH = 1;
$LOCK_EX = 2;
$LOCK_NB = 4;
$LOCK_UN = 8;

select(STDOUT);
$| = 1;

&main;

sub main {
  my($id, $iformat, $show);

  $id = param("id");
  $iformat = param("iformat");

  my($counter_value);
  $counter_value =
    &update_counter_value($id);

  chomp($counter_value);
  if($iformat eq "jpg"
    || $iformat eq "png") {
    &print_counter($iformat,
        $counter_value);
  } else {
    &print_error_image(
      "Image format $iformat not".
      " supported");
  }
}

sub print_counter {
  my($iformat, $counter_value) = @_;
  my($COUNTER_SIZE) = 4;

  my($im) = GD::Image->new(
      "${iformat}s/0.${iformat}");
  if(!defined($im)) {
    &print_error_image(
        "\$im couldn't be initialized");
    exit;
  }

  my($w, $h) = $im->getBounds();
  undef $im;

  my($printim) = GD::Image->new(
      $w * $COUNTER_SIZE, $h);
  $printim->colorAllocate(
      255, 255, 255);

  my($pos, $l, $temp, $digit, $x,
    $srcim);
  $x = 0;
  for($pos = $COUNTER_SIZE - 1;
      $pos >= 0;
      $pos--) {
    if($pos > length($counter_value) - 1) {
      $digit = 0;


    } else {
      $l = length($counter_value);
      $temp = $l - $pos - 1;
      $digit = substr($counter_value,
          $temp, 1);
    }
    $srcim = GD::Image->new(
        "${iformat}s/${digit}.".
        "${iformat}");
    $printim->copy($srcim, $x, 0,
        0, 0, $w, $h);
    $x += $w;
    undef $srcim;
  }
  if($iformat eq "jpg") {
    print "Content-type:".
        " image/jpeg\n\n";
    print $printim->jpeg(100);
  } else {
    print "Content-type:".
        " image/png\n\n";
    print $printim->png;
  }
}

sub print_error_image {
  my $error_string = $_[0];
  my $im =
    new GD::Image(
      gdMediumBoldFont->width *
        length($error_string),
      gdMediumBoldFont->height);
  $im->colorAllocate(255, 255, 255);
  my $red =
      $im->colorAllocate(255, 0, 0);
    $im->string(gdMediumBoldFont,
      0, 0, $error_string, $red);
  print "Content-type:".
      " image/jpeg\n\n";
  print $im->jpeg(100);
  exit;
}

sub update_counter_value {
  my($file_name, $counter_value);

  $file_name = "$_[0].counter";
  if ($file_name =~ /^([-\@\w.]+)$/) {
    # For the tainting stuff
    $file_name = $1;
  } else {
    exit; # Should never happen
  }
  if(open(COUNTERFILE,
      "+<$file_name") == 0) {
    # Getting accessed for the
    # first time
    open(COUNTERFILE, ">$file_name");
    print COUNTERFILE "1";
    close(COUNTERFILE);
    return 1;
  }

  flock(COUNTERFILE, $LOCK_EX);
  $counter_value = <COUNTERFILE>;
  seek(COUNTERFILE, 0, 0);
  ++$counter_value;
  print COUNTERFILE $counter_value;
  flock(COUNTERFILE, $LOCK_UN);

  close(COUNTERFILE);
  return($counter_value - 1);
}
```

This script can be used by adding a line like this in your web page.

```
<img src="counter.cgi?id=my_html_file.html&iformat=
jpg">
```

The id needs to be unique. A sample counter can be seen on my home page [at http://www.cise.ufl.edu/~ppadala].

Now to the innards of the script. The counter script has three important functions.

update_counter_value: This function reads the hit count from a file named html_file.counter and increments it. It creates the counter file, if one already doesn't exist. It also locks the file to avoid conflicts due to multiple simultaneous accesses.

print_counter: Prints the counter by attaching the counter digits in a new image. The digits are read from an appropriate directory.

print_error_image: This is a useful function to show error images. You can use it in your programs, for reporting errors through GD.

You need to have the digits (0-9) in jpg or png format. Sites like Counter Art dot Com [http://www.counterart.com/] provide free counter digits. In my next article, I'll discuss how to generate digits on the fly.

I developed a personal website statistics package woven around this counter concept. It provides much more than a simple counter. It logs the accesses, shows visitor statistics and much more. Check it out at [http://pstats.sourceforge.net].

You can also use the File::CounterFile module for managing the counter file.

Coming Up

I hope you enjoyed reading this article. In the coming months, we will look at GD::Graph and PerlMagick modules. Send me comments at this address p_padala@yahoo.com.

Have Fun !!!

Acknowledgements

My best friend ravi has become the official editor for all my writings. I am indebted to him for looking through all the gibberish I write and make sense out of it. Thanks ravi :-)

I thank Benjamin A. Okopnik for reviewing the article and pointing out some nice perl hacks.

Pradeep Padala

I am a master's student at University of Florida. I love hacking and adore Linux. My interests include solving puzzles and playing board games. I can be reached through p_padala@yahoo.com or my web site [at http://www.cise.ufl.edu/~ppadala].

An Introduction to GNU Privacy Guard

Author: David D. Scribner <dscribner@bigfoot.com>

Abstract

An article targeting users new to GnuPG on GNU/Linux (and UNIX) systems, and how it can play an important role in their personal and business lives in inceasing the security and communication of digital medium. The article explains some of what GnuPG can do, the very basics in using it, and why it can be so important in becoming a valuable utility in anyone's toolbox, both personally and professionally.

An Introduction to GNU Privacy Guard

'It's personal. It's private. And it's no one's business but yours.' - Philip Zimmermann

GNU Privacy Guard, or GnuPG (http://www.gnupg.org/), is the open-source equivalent of Philip Zimmermann's PGP (Pretty Good Privacy) encryption/authentication software released under GPL. Philip Zimmermann and others developed PGP in 1990 using the Rivest-Shamir-Adleman (RSA) public-key cryptosystem to answer the need for private and secure communications between individuals over digital medium. After its release to the public in 1991, it quickly grew to become the de facto standard worldwide for secure public-key encryption.

Even though the concept of public-key cryptography for encryption purposes was introduced close to three decades ago, and PGP has been around for over a third of that, you'll likely find that for some reason only a small number of PC users take full advantage of public-key security. I, myself, am guilty of this. When I first started using PGP 2.6 for DOS in the mid-nineties, it was used for little more than encrypting local documents and files for my own use. Few outside of technical circles had heard of it, and the majority of the people I talked to felt it was too cumbersome to use on a regular basis. That was then.

Today, I use GnuPG for a variety of tasks. Whether it's to sign and encrypt documents and contracts submitted to businesses, encrypt local files, or merely sign email and files to ensure others that no modifications have occurred to its content, I have found GnuPG to be a 'must have' utility kept close at hand when using my PCs.

Why did I switch from PGP to GnuPG? A lot of personal reasons, really, but it all basically came down to trust, and the fact that I believe in GPL.

GnuPG is also fully OpenPGP compliant, and was built from scratch from the ground up. It does not natively use any patented algorithms, supports a very wide array of current cipher technologies, is built to easily integrate future cipher technologies, and decrypts and verifies PGP versions 5.x, 6.x and 7.x messages. I still keep my early RSA keys around in case the need arises to decrypt a file from one of my old archives, but eventually even those files will be wiped, or re-encrypted with one of the newer, and perhaps stronger cipher algorithms available with GnuPG.

GnuPG is available for various flavors of Linux/UNIX, as well as Windows and Macintosh operating systems. As a GNU/Linux advocate, the focus of this article will be on using GnuPG with Linux, though the commands and procedures listed would apply across any of the supported platforms. As there is already very fine documentation for GnuPG, including the man page, The GNU Privacy Handbook, the GnuPG mini-HOWTO and the GnuPG FAQs, I won't go into great detail on all the many command-line operations or configuration settings. Instead, I'll try to explain in simple terms what GnuPG can do, the very basics in using it, and why it can be so important in becoming a valuable utility in your toolbox, both personally and professionally.

Privacy, do you need it?

When most people talk about privacy and strong encryption in the same sentence, they often think the only people needing such things must be doing something wrong, illegal, or involved with government espionage.

Spies, smugglers, mobsters, and terrorists come to mind, so you, not fitting into any of these categories and being an honorable and upstanding citizen certainly don't need to be involved with this kind of stuff, right?

You may also think, with Linux's advanced security features such as a protected filesystem structure, memory and strong policies in place, along with a good firewall and IDS, your private data (that list of important account numbers for example) is safe. Perhaps... at least until your fortress gets cracked. By then however, your private data may have been compromised, leaving you wondering what additional steps could have been taken to protect it.

Perhaps you conduct a lot of your business via email, sending quotes, contracts, or financial reports over the Internet to your corporation, boss, or business partners. If you've used the Internet for any length of time, surely you know that the email containing your documents takes multiple paths, and makes multiple 'pit stops' at various servers before it reaches its destination. Are you sure that when your document passes through these servers that your files and communications are not scanned, even forwarded on to a paying competitor?

Maybe your data isn't that confidential, but you would still like to ensure that the document or communication you're submitting reaches its final destination exactly as it was sent... no additions, no subtractions, no modifications at all. Can you be sure of this when sending it via standard email procedures?

Suppose you don't have any secrets, and there's not any real concern about someone modifying your email. Do you still want your ISP's service technician with nothing better to do snooping through your email on the server while it's sitting there waiting to be retrieved to know all about your life? If you're in a small or rural town, your ISP may be just a core crew that runs the show. Do you trust them fully with your account? If you sometimes email your family traveling itineraries, information about doctor visits or what stocks you've bought recently, do you really want your ISP to possibly know this information, too?

Even if you're using a large corporate ISP with established privacy policies, can you be sure that their policy is enforced all the way down, and at every level? Believe me, there's been more than one technician discovered snooping through user's mailboxes without their knowing. Some of these 'technicians' consider such a practice as entertaining as others might consider soap operas! As Philip Zimmermann stated in the original PGP User's Guide, 'It's personal. It's private. And it's no one's business but yours.' If you send plain-text email over the Internet, it may be personal, and it may be private, but it can easily become anyone else's business, too!

When you start thinking about it, you will no doubt find many areas in your life needing increased privacy, encryption and digital signatures, which may include private files, documents and email. If the need arises to share those files, documents and email with others, you may want to seriously consider incorporating public-key security into your routines.

Public-Key Security

The concept behind public-key security is that there is no need to disclose what should be kept most private, your secret key. With conventional encryption techniques, you encrypt a file for example with a password or passphrase. However, should you need to deliver that file to someone else, you will also need to disclose that password or passphrase to him or her so they can decrypt the file.

How can this be accomplished, and securely? Well, you could personally hand-deliver the key to the other party, along with the file, however this may not always be possible due to distance or other restrictions. You could email the file, then snail-mail, phone or fax the key to them, but between the time it leaves your hand, mouth or fax machine and is received by the other party, you still can't be 100% sure that it wasn't intercepted.

You could also email the file, and in a separate email, disclose the key.

However even this is not secure, and less so than one of the above methods at that! Short of hiring trusted 'key couriers' to deliver your password or passphrase such as government agencies might do, there is one alternative and that is to use two keys. One key, kept private, is used to decrypt or sign information, and the other key is made 'public' and is used to actually encrypt the data or verify the signature.

That is the basis behind GnuPG and PGP techniques. Using various cipher algorithms you create 'key rings' to hold your secret (private) and public keys. Your secret keys are protected by passphrases known only to you and should be kept secure. But your public key, which can be dispersed freely, instructs the GnuPG/PGP application used by others on how to encrypt the data, after which only one key can then decrypt it... your secret key.

Before you create your first key pair, you should come up with a good passphrase to protect it. The word 'passphrase' is used, as it should really be more than just a pass 'word.' There are utilities available to help you choose a passphrase if needed (Diceware is one such utility), but the general idea is to come up with a string of words or characters using mixed case, and one that includes numbers, punctuation marks and special characters that will provide strong resistance to cracking. The trick is to come up with a passphrase you're not likely to ever forget! You could use a single password, but if you're protecting your data with strong encryption ciphers, why make the weakest link (the passphrase) even weaker with something that might make it easier for someone to crack? You could also use a sentence perhaps, just be careful. Most all of the common and well known 'phrases' and quotes are already included in a cracker's toolkit, right along with all the dictionary words, and in multiple languages at that.

You can always change your passphrase after the secret key has been generated as well. Based on your use of GnuPG, with regards to privacy and surroundings, the passphrase should be changed regularly. Some suggest monthly, but if you ever find yourself doubting the security of your passphrase because someone was too near your shoulder when it was last entered, it should be changed the first chance you get. This feature is also handy if you find yourself wanting to experiment and work with GnuPG for a short session of learning... change your passphrase to something that can be quickly entered for the exercise, then change it back when your session is over.

More information on creating a strong passphrase can be found in the documentation or on the Internet, but the same common sense used in creating a strong password should be used in creating a stronger passphrase. Don't write it down. Don't tell it to anyone. And remember, if you forget your passphrase, you're sunk. There's truly no way to retrieve it.

So, with your freshly installed copy of GnuPG you can create a pair of these keys easily with:

```
$gpg --gen-key
```

By following the prompts, you will indicate what type of key you want to create (signing, encryption, or both) decide on a key size, expiration date, and enter your name, email address and an optional comment. You will also enter the good passphrase you decided to protect the pair with and the key generation will begin. After you've generated your first key pair, you can see that the keys are now in your keyring with the command: $gpg –list-keys

This, as the command indicates, lists all keys currently in your public keyring. If you're just starting out with GnuPG the only key listed will be the one just created. In this example, the output's format will resemble (with your own identity of course):

```
pub  1024D/F357CB52 2002-07-09 GnuPG User
<user@some-email.com> sub  2048g/A9CB69B3 2002-07-
09
```

In the example output above, you can see that the key listed has a key ID of F357CB52, and contains the name and email address of the key's owner.

If your public keyring contains several keys for other users, you can specify which key you want to view by simply adding the 'key specifier' of the key you want to view after the '--list-keys' directive. The 'key specifier' could be the owner's name, the key's ID, or the user's email address.

Depending on how many keys are on your keyring, an owner's name could be as simple as their first name, or their last name. If there's only one Kathy in your keyring, you could use Kathy. If there are two keys belonging to Kathy, but the email addresses are different, the email address of the chosen key could be used. If both keys have the same name and email addresses, the key's ID can be used to differentiate them.

When using an email address to identify a key, it's not necessary to use the entire email address, only the part that's unique to that user's key.

Whatever means you use to identify that key, all that's necessary is for it to be unique enough to identify the key you want. One note to make is that when using a key ID as a specifier, since the key ID is a hexadecimal number you can also prefix the ID with '0x'. If you were specifying the key ID F357CB52, it could be entered as 0xF357CB52. This is not required for GnuPG, but as you start working more with key servers, you will find that when using a key ID to search their database, they need to be prefixed with '0x'.

I highly suggest that the first thing you do after creating a key pair is generate a revocation certificate for that pair, then move it onto a diskette you can keep secured someplace safe (along with a printed hard copy of the certificate in case the medium becomes damaged). To create a revocation certificate for the key pair we just created (using the key ID as exampled above), issue the command:

```
$gpg --output revokedkey.asc --gen-revoke
0xF357CB52
```

This will generate a revocation certificate for the key ID F357CB52. The resulting file, 'revokedkey.asc' is the revocation certificate.

What is a revocation certificate and why do you need one? In the beginning, if you're just experimenting with GnuPG and haven't dispersed your public key, there might not be a need for one. But, once you've started using GnuPG to encrypt files and your public key is out there for others to use, a revocation certificate adds another layer of protection to your key pair.

Should your secret key become compromised or lost, or if you forget your passphrase, a revocation certificate posted to a keyserver or sent to your contacts to update their keyring will not only inform them of your key's demise, but also prevent them from encrypting new files to that public key (and likewise prevent you from using that key to sign or encrypt files yourself). If your key has been compromised, you can still use the secret key to decrypt files previously encrypted to or by you, and others can still verify your signatures created before the revocation, but it's an added safeguard you should take now.

If you've forgotten your passphrase, there's nothing you can do short of a brute-force attack of trying every passphrase combination you can think of that may have been used. And, without that passphrase (and access to the secret key), generating a revocation certificate is not possible. Hence the importance of generating one before it's too late!

Encryption & Digital Signatures

Since GnuPG uses the dual-key concept for increased security, how can this be used in your personal or business life? Let's take a look at the various areas where strong encryption and authentication could apply.

Asymmetric Encryption - Asymmetric encryption is the standard with dual-key concepts. As previously explained, anyone can use your public key to encrypt information, but only you can decrypt it. You might be working on a project meant for 'limited eyes only.' Protecting the data generated in that project via encryption could keep competitors from uncovering, and possibly beating you to market, with your project's outcome. Say it's a programming project for a new 'killer application' that will revolutionize the personal computer industry, a new product-packaging concept that will reduce costs for your company, or even a new and creative marketing scheme that will rake in big bucks. It would be a devastating blow to your project should your competitor catch wind of it before its time, so encrypting the files and communications relating to that project would definitely help keep them out of the loop.

Aside from other business data such as financial reports, projections, or statistics that might serve well being encrypted before being sent off over the Internet, as also mentioned it doesn't have to be 'secret' data that should be kept from prying eyes. It could be common data that you or I wouldn't even consider the need to secure, but with today's problems with identity theft, burglaries and crime on the rise, perhaps we should take a second look at some of this 'common' data we so easily send off without a second thought.

Phil Zimmermann likened sending plain-text email to a postcard. Actually, I think it's worse. I say this in consideration that at least those working for the US Postal Service are government-regulated employees. Surely those that consider government 'big brother' might disagree, but, having a career in the computer industry and not having known too many postal workers personally, I have met far more network and ISP technicians that I wouldn't trust with a ten-foot pole. I would dread the thought of having them know I was taking my family on a vacation on a particular date and wouldn't be back for a week... I might arrive home and find it empty! If it's none of their business, encrypt it!

The general command to encrypt a file using another's public key is:

```
$gpg --recipient username --encrypt filename
```

As an added measure of security explained later in the article, when encrypting a file to be sent to another GnuPG/PGP user, most prefer to also sign the file. The command for this would be: $gpg --recipient username --sign --encrypt filename The result of either of these two commands will encrypt a file ('filename') to a particular user's ('username') public key, producing a binary encrypted file ('filename.gpg'). If the '--sign' command was given, the encrypted file would incorporate a signature produced with your private key as well.

Symmetric (conventional) Encryption - Symmetric, or conventional, encryption is where encrypted data needs only one key to decrypt it, the same key it was encrypted with. The cipher used to encrypt the document or file is generated at the time of encryption, and during the process you will be asked for a passphrase. The passphrase should be different from the one you normally use with your secret key, particularly if the file will be passed on to another. You will still be faced with finding a secure way of providing the passphrase to the recipient, but if that person does not yet utilize a public-key system such as that provided by GnuPG or PGP, all that's necessary to decrypt this file with either of those utilities is the password or phrase used to encrypt it with.

Another use for symmetric encryption might be sensitive documents or files kept on your own system. Although it's advised that any truly sensitive data should not even be kept on a system accessible to the Internet, no doubt there's still data on your system that would best be protected if encrypted.

Take for example all those Internet sites and web mail accounts you have passwords for. Some users opt for simple passwords; some just use one password for all. Neither of these is secure, so short of keeping a list handy, trying to remember all these passwords can be a daunting task. Many browsers nowadays have the ability to store passwords, but if you change browsers, wipe your cookies clean or suffer a hard drive crash without having a current backup, restoring more than a few of these passwords would be difficult for most anyone.

That's where having a document listing these passwords and account information can come to the rescue. You can still use strong passwords without the worry of forgetting them, yet at the same time you don't want this document sitting on your hard drive unprotected, or worse, printed and sitting next to your keyboard. Encrypt it! When you run into a situation where you need a password that you can't remember, it only takes a second to decrypt the file to the terminal screen, keeping it safe from prying eyes when not needed.

No doubt you will think of plenty of other data you might find on your hard drive that would be better left protected with strong encryption, and especially if that hard drive is on a laptop that travels... it's bad enough if the laptop gets stolen, but in many instances the loss of sensitive data kept on the laptop can be worth more than the hardware by far!

To encrypt a file named 'filename' using symmetric encryption, issue the GnuPG command:

```
$gpg --symmetric filename
```

A copy of the original file in encrypted form, 'filename.gpg', will be generated.

Digital Signatures - Although digital signatures accompanying data don't yet carry the weight they deserve in most states, they can be of extreme value in so many instances that I'm actually surprised their use hasn't caught on more than they have already.

Using your secret key, you can have documents, files and messages 'signed' and time-stamped to authenticate that they actually did come from you. If the signature verifies, the receiver can be assured that the data was not modified in any way, and has arrived just as it was meant to be. This can be valuable for authors submitting stories or articles to publishers, programmers submitting files to the public, or businesses sending contracts, purchase orders or invoices to their partners. Literally just about any time you would normally send such a document signed with your own signature, a digitally signed document offers the same validity and authenticity in many cases, and prevents tampering to boot.

GnuPG's flexibility in signing permits signatures to be embedded in text messages as 'clear-signed'. Since the signature appears as ASCII text, this is useful for emails. The recipient can verify the signature to guarantee that the email has not been altered between the time it was sent or posted and the time it was received if they wish, but the message remains readable by all. To clear-sign a file named 'message', issue the GnuPG command:

```
$gpg --clearsign message
```

The resulting file, 'message.asc' can then be easily emailed as an attachment, or the contents included in the body of an email or posted to a newsgroup forum. For an example of clear-signed messages you can visit one of the GnuPG newsgroup forums, such as gnupg-users, where you'll find most every message signed by the GnuPG user submitting the post. Clear-signed signatures are often referred to as ASCII-armored signatures. I'll be covering ASCII-armoring in the next section.

A digital signature can also be included along with the document as a single file; the file is compressed along with the included signature.

Although the file's not actually encrypted, by using the GnuPG decrypt command and the sender's public key the file can be uncompressed, restored to its original format and the signature verified. The signature can also be verified without decompression in case you wish to keep the binary file intact as is.

Signatures can also be included as a detached file separate from the original file. This accommodates others that might not be using GnuPG or PGP utilities, allowing the original document or file to remain 'untouched.' Yet, because the signature hash was created from that file, it can still be verified as being unmodified and the time stamp checked.

This is popular with many binary files found on the Internet (though in the Linux world MD5 hash signatures are sometimes more common for this), or with documents where an embedded signature is not desired. In order to verify a signature, all anyone needs is that person's public key.

To create a detached binary signature ('filename.sig') for the file 'filename', the GnuPG command would be:

```
$gpg --detach-sign filename
```

Because signatures carry weight and can verify that the data hasn't changed, as previously mentioned it is strongly suggested that anytime you encrypt a file with another's public key, the file is also signed using your secret key prior to encryption. This adds protection to the file in that not only can that particular recipient be the only one to decrypt it, but during the process they are also assured that it came from you, and only you. The added authenticity helps thwart attempts by someone else trying to intercept and modify the file for nefarious means, as without your secret key there would not be a way to reapply the signature. To decrypt the file and make changes they would not only need the secret key and passphrase of the recipient, but to reapply the signature they would also need yours... very unlikely.
To verify a signature or a signed file, the GnuPG '--verify' command is used. If the file includes an embedded signature, it could be as simple as:

```
$gpg --verify signedfile
```

Or, if a detached signature accompanies a file, the command would include both the signature file, and the file signed, such as: $gpg --verify sigfile.asc signedfile

ASCII Armor

Although many applications support MIME/PGP standards and work very well with public-key encrypted or signed files in binary format, there may be some instances where ASCII format files are needed. Many email clients such as Mutt, Sylpheed and Kmail for example fully support MIME/PGP standards and work very well with GnuPG, but for those MUAs lacking full support, even though there's very likely a plug-in available, ASCII armor comes to the rescue.

A good example of situations where ASCII format files might be needed is email or newsgroup postings. You want your messages signed, permitting those using public-key utilities to authenticate your signature, yet still allow those that don't to easily view the message's content. Or, perhaps your company's policies prohibit binary attachments passing through their email servers to or from outside networks, requiring any binary files to first be converted to ASCII, similar to uuencoded files.

GnuPG takes into account these needs by allowing the user to specify that an ASCII armored file be produced for its output. The option for this is '--armor', which can be added to most any command where ASCII format is needed. Another command-line option you will often use in conjunction with '--armor' is '--output', which takes a filename as an argument.

You may have also seen public keys displayed as 'blocks' of garbled ASCII text on web sites. This makes it convenient to display the user's public key so visitors to the site can easily import the key into their own keyring. As you would expect, just about any output GnuPG generates, whether it be keys, encrypted files or signatures, can be output as ASCII-armored text. A few examples of producing ASCII-armored output with GnuPG are included below.

To encrypt the file 'filename' for recipient 'username', creating an armored ASCII text file ('filename.asc'):

```
$gpg --armor --recipient username --encrypt
filename
```

To encrypt the file 'filename' for recipient 'username', creating a signed armored ASCII text file ('filename.asc'):

```
$gpg --armor --recipient username --sign --encrypt
filename
```

To encrypt the file 'filename' using a symmetric cipher for encryption (does not include public key information), and creating an armored ASCII text file ('filename.asc'):

```
$gpg --armor --symmetric filename
```

To digitally sign the file 'message', creating an ASCII-armored version ('message.asc') of the file that contains the attached signature:

```
$gpg --armor --sign message
```

To create an ASCII-armored text file named 'sigfile.asc' as output for the detached signature, and is for the file 'filename', you could use the command:

```
$gpg --armor --output filename.asc --detach-sign
filename
```

Keyrings & Keyservers

As you've seen, the concept behind GnuPG and public-key security deals with keys, and keyrings. When you created your first pair of keys, you also created your first keyrings. If you take a look in your home directory, you'll find a hidden directory created for the GnuPG files (~/.gnupg). Within this directory there will be two keyring files, pubring.gpg and secring.gpg. The pubring.gpg file is your public keyring, and secring.gpg is your secret, or private, keyring.

Your secret keyring, containing only the one or two secret keys you create will remain small, however the public keyring will grow as you collect and add more public keys to it. Your secret keyring should be stringently protected at all costs, but your public keys, and keyrings if desired, can be shared.

As you continue working with GnuPG, you will note that many users publish their public keys on web sites or key servers. In the past, keys were exchanged via email, BBS or newsgroups postings. In recent years as the popularity of public-key security has gained ground, formal key servers have evolved to handle the need of dispersing these keys in a more convenient and centralized manner. After you've created the keys you'll be happy with and have worked with GnuPG to get to know your way around it a bit (and thoroughly memorized your passphrase!), you'll eventually want to start exchanging your public key(s) with others.

You'll also find various public keyrings dispersed on the Internet. Many of these keyrings hold the public keys of GnuPG/PGP users belonging to special interest groups. As you experiment with your secret and public keyrings, you may consider importing several keys from one of these keyrings to see what they're comprised of. You will likely find keys that have been signed by several other key holders, include additional email addresses, or even identifying photos. Since you may not actually know any of these people, you will probably limit your experiments to signing and encrypting files to your own key sets, but importing a few 'popular' keys from one of these keyrings will give you an idea of what yours can become like down the road! When you're done, they can be easily deleted from your keyring if you choose.

Say for example you find a user's key available for download on the Internet that you want to import into your own public keyring, and the key is named 'user-key.asc'. After downloading the file, to import this public key into your keyring you would use the command:

```
$gpg --import user-key.asc
```

Likewise, to make an ASCII-armored copy of your public key that can be emailed, included as a download link, or whose contents can be displayed on a web page so others can import it into their own keyring, use the command:

```
$gpg --armor --export username > keyname.asc
```

This will export your key in ASCII armor format as previously explained, for the username or key ID specified, redirecting the output to the file 'keyname.asc'.

Key servers, or 'keyservers' as they are sometimes referred to, have simplified the dispersing of public keys to the masses as it provides a somewhat centralized repository to post and retrieve public keys from.

Although there are several key servers out there, most all are connected and synchronize their data. If you post or update your key to one of these servers, you'll find that the key will be broadcast and updated to the other key servers, sometimes rather quickly.

A couple of the more popular key servers are MIT's PGP Key Server and the OpenPGP Key Server. For other key servers closer to your geographic location, you might take a look at the OpenPGP Server Lookup web site.

There, you'll find listed other key servers located around the world.

It's at these key servers that you can search for a public key belonging to someone by their name, email address or key ID. Keyservers are also good to use when verifying a key's fingerprint. If you were given a key by someone, or download one off the Internet from a web site, checking that key's fingerprint against the one listed on a keyserver to make sure they match is a good precaution to take as it will minimize the risk of receiving a bogus key. That's not to say that all keys on a keyserver are legitimate, but due to their circulation bogus keys can often be exposed for what they are.

Importing public keys from a key server into your keyring is as simple as specifying the key to import, and the key server to import it from. For example, the command:

```
$gpg --recv-keys --keyserver wwwkeys.pgp.net
user@some-email.com
```

will import from the keyserver, 'wwwkeys.pgp.net', the key for 'user@some-email.com' into your keyring. Exporting a key is just as easy.

The command:

```
$gpg --send-keys --keyserver wwwkeys.pgp.net
user@some-email.com
```

will export the key for 'user@some-email.com' currently residing in your public keyring to the keyserver 'wwwkeys.pgp.net'.

However, not everyone wishes to have his or her public key uploaded to a key server. So, if another user gives you their key for signing, always check with them prior to submitting it to one of the key servers. They may prefer that you email it back to them in a signed and encrypted message instead.

Web of Trust

Now that we've covered creating a key pair for your encryption and authentication needs, and have experimented a little with importing and exporting keys for yourself to use as a learning experience, you're probably eager to start exchanging keys with other users. One of the fundamental aspects of using public-key cryptology with GnuPG/PGP is the ability for users to 'sign' other users keys, adding yet another layer of legitimacy to that key. This also allows you to detect any tampering with the key in the future. However, signing another's key(s) is something that should not be taken lightly as it will affect the level of trust others have established in your key.

By having others sign your key, they are helping you establish what has become known as a 'web of trust.' Having several signatures on your key helps assure others receiving your key of its authenticity, especially if the key receiver also knows one or more of the others that has signed your key.

Keep in mind that when someone signs your key, they are only vouching for you in that your key is authentic and that it indeed belongs to you, because they have personally verified the fact. They are not vouching for your personality, ethics or morals, only that you are who you say you are identity-wise, and the key they've signed is yours.

Before you go signing another's key, it's important that you first verify that the key you were emailed, downloaded from a key server or were given actually belongs to that person. If you were to do this over the phone, or even in person, checking each character of the person's ASCII-armored key block would not only be exhausting, but extremely error prone as well. As such, each key has a 'fingerprint.' These fingerprints are derived from a hash of the key block, and provide a relatively sure way of verifying the key to the owner without error.

To view a key's fingerprint, the GnuPG command 'gpg --fingerprint keyID' is used. The output displayed can then be used to communicate the received key's fingerprint to the owner, who can then verify that the fingerprint is correct, and matches the fingerprint of their key. An example of a key's fingerprint, as output by GnuPG, might look similar to:

```
pub  1024D/F357CB52 2002-07-09 GnuPG User
<user@some-email.com> Key fingerprint = 5172 708D
CA58 C2D9 4082  BA0D 103D 1293 F357 CB52

sub  2048g/A9CB69B3 2002-07-09
```

Once you've confirmed that the fingerprints match, you can then sign that user's public key with your secret key to validate it. When you created your key pair and a public key was formed, it was automatically signed by your secret key when using GnuPG. To sign another's key however, you can use the command:

```
$gpg --sign-key keyID
```

Along with '--sign-key', one other command for signing keys is available that I want to bring to your attention, and that is '--lsign-key'. The '--lsign-key' command signs the public key for local use only (instead of flagging your signature as being exportable along with the key as the '--sign-key' command). As your signature is not exportable, you're not validating that user's key for others to rely on, but it will still be treated as a valid key on your local system.

Why would you sign a key for local use only? Perhaps one example would be that you imported a key used to verify the signature of a software package. You want to sign the key so that the risk of tampering with this key without your knowledge is minimized, and warnings issued by GnuPG about using a non-validated key won't be displayed. The command to locally sign a key is:

```
$gpg --lsign-key keyID
```

You will be prompted for your passphrase, after which your signature will then be attached to that key. To sign a key you could also edit that key.
The GnuPG command to edit keys is:

```
$gpg --edit-key keyID
```

This invokes the interactive menu system that allows you to modify the properties of a particular key. To sign the key with your secret key, in turn allowing your signature to be exported, use the menu command 'sign'.

Just as when using the '--sign' and '--lsign' commands from the shell's command prompt, you will be asked for your passphrase. When correctly supplied, it will attach your signature to the chosen public key when you then use the 'save' command, or 'quit' and respond that you want the changes saved.

The interactive menu system used when editing keys gives you a greater degree of flexibility when signing a key. For example, if you have a key that contains more than one user ID, you may want to sign only one or more Ids for that key, but not all of them. You might choose to do this if you weren't able to verify the email address associated with that ID, or want to keep the size of your keyring smaller than it would be if your signed keys had your signature attached to every ID (the key will still be considered valid, and as your keyring grows, signing only one ID on locally signed keys helps keep their size down). When you first display a key with '--edit-key', you will see something similar to:

```
pub 1024D/2B94CF89  created: 2001-01-01 expires:
never       trust: -/q

sub  1024g/12E688D4  created: 2001-01-01 expires:
never

(1)      Close Friend (GnuPG Gnut)
<close.friend@some-email.com>

(2).     Close Friend (Webmaster)
<close.friend@someother-email.com> Command>
```

The period after '(2)' indicates that this ID is the primary user ID. If you were to sign the key now, your signature would also be attached to that user ID as well. To mark a different ID, or several Ids for signing, the menu option 'uid' followed by the chosen number is used. Once marked, the ID number will have an asterisk placed after it. If you were to mark the first ID to attach your signature to, the resulting output would look like:

```
pub  1024D/2B94CF89  created: 2001-01-01 expires:
never       trust: -/q

sub  1024g/12E688D4  created: 2001-01-01 expires:
never (1)* Close Friend (GnuPG Gnut)
<close.friend@some-email.com>

(2).     Close Friend (Webmaster)
<close.friend@someother-email.com>
```

Signing the key at this point with the 'sign' menu command attaches your signature to the chosen ID. The menu command 'save' will then save the signed key and exit, or you could use the menu command 'quit' to exit without saving any changes.

Other commands you'll find useful in the interactive menu for editing keys are 'fpr' to list a key's fingerprint (which you'll do when verifying the key with the owner), 'check' to list the signatures on a key, 'adduid' and 'deluid' to add or delete a user ID on a key (name, comment or email address, which comes in handy if your email address changes), and 'trust', which is used to assign a trust value to a key. For a list of commands that can be used with this interactive menu, simply type the word 'help'.

After pressing the [Enter] key, a list of possible commands will be displayed.

When should you use the 'lsign' command in place of the 'sign' command? Anytime you cannot refutably verify that the key you're signing belongs to its owner! This is perhaps the most vitally important safeguard you can take in keeping your 'web of trust' sound. Unless you have personally checked and verified the identity of the person against the key you were given with complete complete confidence, or one of your trusted fellows has (more on this below, in 'Level of Trust'), don't sign it with an exportable signature! You may have found Werner Koch's (the main developer of GNU Privacy Guard) key on the Internet, but unless you can personally verify the key, don't sign it with an exportable signature. Doing so will only weaken the level of trust others have placed in you to vouch for another's key validity, in turn weakening your web of trust.

I would like to add that, while on the subject of building your web of trust, many 'famous' people have their public keys posted on key servers and the Internet (Linus Torvalds, Philip Zimmermann, Richard Stallman, Wichert Akkerman, Theodore T'so, Eris S. Raymond, etc.). If not common sense, then at least politeness should dictate that unless you actually know this person or will be conducting business with them, don't disturb their peace by invading their personal life in an attempt to verify their key just so you can sign it with your exportable signature. I'm sure they're kept busy enough as it is.

How does your web of trust grow? By getting to know other GnuPG/PGP users, and spreading the value that public-key encryption and authentication can bring to a person's personal and business dealings. You could add your public key's fingerprint to your email signature line perhaps (which sometimes brings about questions as to what it is by unknowing recipients, giving you the chance to evangelize), or once you've created a key pair you're happy with, publish your public key to a key server, or post the key block on your web site.

Many computer user groups may also hold key signing events where their member's keys are checked for accuracy and the holder's identity verified with a photo ID or driver's license. The keys are then later downloaded or emailed to the participants, key fingerprints checked, signed and returned to the user or key server as requested. Check your local user groups to see if they're holding one in your area. Linux user groups in particular often hold key signing events, and even parties in some localities, where the participants are invited to mingle and get to know each other in a more informal setting after the verification process is over.

As well, the Internet has come to the aid of some as there are sites (Biglumber.com for example) that will post lists of users wanting to have their keys verified, as well as users willing to sign another's key.

Checking the lists for your area might put you in touch with someone local so the verification process can be completed.

If GnuPG were to stop here however, your 'web of trust' would really be no more than a 'wagon wheel of trust', as the only trusted keys on your keyring would be those to which your signature is attached. As in real life, the concept of trust used with GnuPG extends beyond that. Say for example there was a local GNU/Linux convention that you attended with one of your good friends, Greg. At the convention the two of you run into a buddy of his, Tom, operating one of the booths and were introduced. You don't know Tom personally, but you do know Greg and trust that if he said this person was indeed Tom, you would have few doubts as to his identity.

GnuPG utilizes trust levels in much the same way. You may acquire keys for which you don't personally know the individual they belong to, yet due to the interaction with other keys on your keyring that are trusted, they may be 'introduced' and therefore treated as valid to varying degrees. This is what establishes the 'web' in the web of trust model. These keys may not be linked directly to your key, but by the interconnection with other keys on your keyring, they in fact become a part of your web.

Levels of Trust

Many new to public-key use are sometimes overwhelmed by the word 'trust' when referring to the keys in their public keyring. In simpler terms, there are basically two types of trust available to you, 'validity trust' and 'owner trust'.

Validity trust, also called calculated trust, is based on to what extent a key can be considered valid. When you sign a key it's considered fully valid. If you can't verify with absolute certainty that the key belongs to its owner, and the owner is who they say they are, you can either sign the key for local use only (so your signature is not exportable, but the key is still considered valid), or refuse to sign the key and allow the validity to be calculated. If you do verify the key as being authentic, you can sign the key with an exportable signature and the key is then not only considered valid by you, but can also be treated as valid by others that have placed trust in you.

This type of trust is referred to as owner trust, and as the name implies, has to do with a key's owner and is used to specify just how much trust you have in a person's key to validate, or introduce, other keys on your keyring. As mentioned, this is because although a person may not be able to personally verify the validity of another's key, GnuPG allows trusted keys to validate to various extents other keys containing that trusted key's signature. The validity of these introduced keys depends on the amount of owner trust placed on the introducing key. A greater amount of owner trust yields a greater amount of calculated validity trust.

The default configuration for GnuPG specifies that if you haven't personally signed a key yourself, it can still be considered valid as long as two conditions are met. The first condition specifies that either the key has been signed with another key on your keyring that has been granted full owner trust, or the key has been signed by at least three other keys on your keyring that have been granted marginal trust. The second condition requires that the path of signed keys leading back to your own key (and inclusive of) include no more than six keys, or five 'hops'.

These conditions can be altered in the GnuPG options, however, and can be made as tight (or flexible) as you wish them to be, but for most users the defaults are adequate and have withheld the test of time.

How do these levels of 'owner trust' work? Let's say you have personally validated and signed Greg's key and have assigned full owner trust to that key. You've since imported Sandra's key. Although you can't possibly validate her key personally since she lives in a distant city (and you've never met or spoken with her before), as Greg has signed her key it will be considered fully valid.

In a perfect world, the owner of every key on your keyring would be fully trusted, but that just isn't the case. It would be nice, and is why I try to stress the importance of keeping your web of trust sound, but there will be keys on your keyring whose owner you simply might not want to place trust in either due to their lack of knowledge in dealing with public-key security, or the seriousness they place in trust values. There might also be keys belonging to people that you just don't know enough to grant trust to, or people you simply don't know at all. They're all a part of your web of trust however, so GnuPG addresses this by allowing you to assign levels of owner trust to each of your keys.

Assigning owner trust to a key on your public keyring uses the same interactive menu system that can used to sign keys ('gpg --edit-key keyID'), and there can be one of five levels of trust placed on a key. If you recall, when you first chose to sign a key with the interactive editing menu, you were displayed with something similar to:

```
pub  1024D/2B94CF89  created: 2001-01-01 expires:
never       trust: -/f
sub  1024g/12E688D4  created: 2001-01-01 expires:
never
(1).    Close Friend (GnuPG Gnut)
<close.friend@some-email.com>
```

The two types of trust are displayed on the right. The owner trust level is displayed first, and the validity, or calculated trust level is displayed second. In this example, a '-' appears for the owner trust level, and an 'f' appears for the validity trust level, indicating the key has been signed, but no owner trust level has been established.

In dealing with owner trust, if the level is signified by a dash ('-'), it simply means that no level of trust has been assigned as yet. This is the default level placed on any new key imported into your keyring. If you don't know this person and have no level of trust to place on the key, simply leave this level of trust where it is. As there is no level of trust yet established, this key will never be used in validating another's key.

The second level is signified by an 'n' and means no trust, or to not trust this person to correctly verify others signatures. Since these levels can be changed at any time you might assign this level to someone that is new to public-key use until they've grasped the hang of it. Then again, the key may belong to someone you know that is careless in the way they carry out key verifications and don't wish to have possible falsely validated keys pollute your web of trust. With a trust level of this grade, this key won't be used in validating another's key either.

The third level is signified by an 'm' for marginal trust. Assigning marginal trust to a key indicates that you feel pretty confident that the key's owner understands the importance of proper key verification before placing their signature on a key. Bringing our example users back into the picture again, Greg mentioned to you that he has assigned full owner trust to Sandra's key on his keyring. You trust Greg's judgment of course, but since you don't personally know Sandra, her key might be a good candidate for marginal trust consideration until she's proved herself otherwise.

With a trust level of marginal, if your public keyring contains a key, which in turn has been signed by at least three other keys on your keyring of marginal standing, that key will be considered valid. Quite often, the majority of the public keys on your keyring will likely be granted this level as your web of trust grows and your keyring expands.

The fourth level, signified by an 'f' for full trust, is the level you would give to a key who's owner you have no doubts about their understanding of public-key use and verification checks, and is someone who's trust you would consider as good as your own. Our friend Greg would be a good example of someone worthy of full trust. He's been using GnuPG or PGP for quite a while, has established a good web of trust, and definitely takes every precaution in validating keys so as to not spoil that web of trust. With a full owner trust level, any key on your public keyring signed by this person's key will be considered fully valid.

The fifth level, signified by a 'u' for ultimate trust, is used only for keys in which there is a corresponding secret key available. In other words, your personal public keys will have the status of ultimate trust when placed on your keyring. Needless to say, any key on your public keyring signed by you is considered a fully valid key, and is why it's important that you properly validate keys via proper methods before you sign them. Get carried away by signing anyone and everyone's key without properly verifying them first and you will quickly pollute your web of trust. I'm sure I've said that enough so I'll leave that topic alone now.

GUI Front Ends

Since GnuPG is command-line oriented (which actually adds flexibility to the utility) some users may be disappointed or miss a graphical user interface. This may especially hold true for newer Linux users transitioning from a graphical desktop OS like Windows. Although there are graphical front-ends to the application similar to those in earlier PGP days, they are still in the development stage and may lack the 'polish' you expect, or have grown accustomed to. Although several GnuPG interfaces exist, I will only touch on a few of them here.

GPA - GPA, or GNU Privacy Assistant, is actually being developed by the GnuPG organization. Utilizing GTK (the GIMP Tool Kit) for its interface's widgets, GPA allows you to encrypt/decrypt files, view your keys, signatures, sign keys, edit trust levels, and import/export your keys to a public-key server among other things. Like many GUI front-ends, GPA is early work in progress, yet the version I'm currently using, 0.4.3, seems to be very stable. One area that GPA currently lacks is a system for the application's Help button (it's unresponsive). The interface is very intuitive however, so this certainly isn't a major drawback, but indicates there is still work to be done.

Seahorse - Seahorse provides a graphical interface much like GPA, with perhaps one of the nicest graphical front-ends to GnuPG that I've run across so far. Still very early in the development stage and currently at version 0.5.0, this GUI front-end provides much of the same functionality of GPA. Although the interface is very appealing to me, I did find that in its current state the application does suffer from a rare lockup now an then when used on my Debian 2.2 and Red Hat 7.1 systems. No data is corrupted, and killing the application is easy, but like most early projects this is to be expected. The author has recently announced a request to take over the Seahorse project, as development has stagnated for over a year now. Hopefully, someone will come to this worthwhile application's rescue.

TkPGP - Another nice GUI interface for GnuPG. TkPGP is written with Tcl/Tk, and also allows you to sign and encrypt documents in a text window, select files for encryption, decryption, signing and verification, but provides minimal key management functions. The application is capable of using a sizable number of GnuPG commands and configuration settings, providing a powerful interface to many of the GnuPG commands and options available, and includes the ability to remember your passphrase for a set time (handy when you're working with GnuPG for an extended period).

Other graphical front-ends exist for GnuPG and are worthy of consideration, so be sure to check into them before settling on one or two you like. There are also many character-based wrappers, written in Perl for example, that provide front-ends to this great utility. If you're operating at a character-based terminal and wanting a menu interface for GnuPG, be sure to check into these offerings as well. A rather comprehensive list of front-ends for GnuPG can be found at the GnuPG web site, or freshmeat.org.

Caveats

When working with sensitive data, whether you're using GnuPG or not, certain 'common sense' caveats should apply. In other words, you should be aware if your sensitive data is winding up in shared memory locations, temporary files, etc., and whether you are working with your documents over a network or on shared systems. The same concerns apply when using GnuPG or any other public-key encryption system.

Keep in mind that simply deleting a file does not actually remove the file's contents from the hard drive, and that it can often be recovered with a little work. Also, if the application you're using to create a sensitive document uses 'timed' saves or writes buffers to an unsecured location on your system, even after thorough deletion of the file there may be enough of the original document left behind in these areas to thwart your security efforts. Various file-wiping utilities exist for most platforms (such as 'shred' or 'wipe' for Linux), so if such a scenario applies to your situation, you should seriously consider looking into using one of these.

If you're using a shared terminal or operating over a network, you may consider using removable media such as a floppy or zip disk for your document's creation and encryption location... just don't leave it behind! You may also want to look into the features GnuPG has for encrypting text and messages straight from the terminal's keyboard. No plain-text file is left behind to worry about, as it's saved straight to an encrypted file.

Don't forget that your GnuPG secret keyring should be kept protected at all times. This file, secring.gpg, is kept in the ~/.gnupg directory by default. If your terminal is shared or if your home directory is stored on a server somewhere, you'll definitely want to configure an alternate location for this file in your ~/.gnupg/options configuration file. It could be kept on a floppy diskette, or even renamed and moved to another directory location if needed, as long as that location is secure.

For those that utilize shared terminals at work, a web cafe or truck stop, there's even a small distribution of Linux, Tinfoil Hat, developed especially to operate in conjunction with GnuPG. A bootable floppy that for the really paranoid even utilizes a wrapper for GnuPG called gpggrid to circumvent keystroke loggers, Tinfoil Hat Linux not only provides a reasonably secure place to store your keys and encrypted documents (reasonable as it's not handcuffed to your wrist), but also provides secure encryption and decryption of your files in RAM.

Keystroke loggers, or key loggers, can be software, or even small devices or plugs that attach between the system and the keyboard, logging every keystroke entered by the user (thereby capturing any text, including any passwords and passphrases that may have been entered). The hardware loggers, being small and usually attached at the back of a system, are many times overlooked and can go unnoticed. Software key loggers may save the keystrokes to a file, or transmit them over a network to another system.

If your data is sensitive, and your system accessible by others, be aware of the security measures that need to be taken into consideration in protecting your passphrase. A weak passphrase, or one that can be discovered easily, nullifies the security available with public-key encryption. The point is that your passphrase, as mentioned, is the weakest link in your encryption. Guard your passphrase well, and use common sense in keeping it safe or your efforts will be futile.

Also, if you haven't done so already, copy your keyring files to the same diskette as your key's revocation certificate and place it in a safe or deposit box for security's sake. Since the revocation certificate generated when you first created your key pair probably only includes 'generic' comments explaining your reasons for revoking the key, keeping a backup copy of the keys allows you to generate a new certificate with details should your secret key ever become compromised or lost.

Conclusion

We've now covered your introduction into using GnuPG. As you have seen, it's not difficult to use, and in fact most of it is rather intuitive.

When its features are used from within many other applications such as email clients or supportive applications or plug-ins, much of what goes on in the background is transparent, simplifying matters even more. The steps to using GnuPG are just as simple...

come up with a good passphrase, create a key pair and its revocation certificate, experiment with it, publish your public key to a key server when you're satisfied, establish and build your 'web of trust,' and last but certainly not least, always keep security in mind.

Although the use of public-key security has grown tremendously since those early PGP days, it still has a long way to go to gain popularity in many countries, perhaps due to the overall sense of freedom those countries enjoy within their boundaries. With the tremendous growth of the Internet, thefts of corporate data and the proliferation of crackers and script-kiddies, those boundaries have dissolved. Once you've started using GnuPG with regularity you'll perhaps wonder what you did without it. With the numerous worries about the need for increased security defenses in businesses, corporations and homes, you may even find that it brings a bit of stress relief to your life!

This article is re-printed with permission. The originals can be found at:

An Introduction to GNU Privacy Guard: http://articles.linuxguru.net/view/193/

Commentary: Why Linux will Conquer the World

Author: David Mohring <mohring@ihug.co.nz >

GNU/Linux clearly bears a strong resemblance to Unix. It offers many of the same features, while adding interesting additions of its own (free licensing, open sourced development, etc).

With the Linux platform the open source/free software community has already created a cross-market software unification infrastructure better than Microsoft has ever had (or is). This has result in rapid expansion in Linux's popularity which has eaten into Microsoft server market share as Linux also grows toward taking over the governmental,enterprise, desktop and development world.

There are a number of reasons for this:

1. The breadth of Linux's market presence.

Due to the liberal nature in which Linux is licensed, any real measurements of Linux's current level of deployment is as difficult to determine as the real number computers running pirated versions of Microsoft windows.

Trying to measure the current level of Linux deployment based around the number of computers/servers sold with operating systems installed is flawed. Linux based solutions are often efficient enough to be deployed on pre-existing hardware, whereas Microsoft is dropping support for NT4 and a Windows2000/XP based solutions almost always have a higher level of minimum requirements to do the same job. Also unlike Microsoft OEM license releases, there is no price advantage to purchasing the Linux with the computer, and Evans Data survey discloses that a full 38.9% of new Linux hardware deployments is assembled from parts. http://www.evansdata.com/computer.htm

The one exception to measuring the level of Linux based deployments is publicly accessible and query-able Internet servers. In the netcraft September 2001 web server survey. Linux based servers occupy 30% of the market compared to Microsoft's IIS webserver's 27.46% share. As of August 2002, the open source Apache webserver has 63.51% share compared to Microsoft's IIS 25.39%.

Even so, You would be hard pressed to find a software or hardware market where Linux does not have a rapidly increasing presence. Linux works on obsolete hardware (so you needn't throw the hardware away), common modern PC hardware, prototype wrist watchs,PDAs, the Playstation, PlaystationII, Dreamcast and even the XBox consoles, IBM mainframes, massive clusters, and a number of supercomputers . Linux runs on a vast number of different CPU chips, including the x86, Intel Itanium, AMD Hammer, ARM, Alpha, IBM AS/400, SPARC, MIPS, 68k, and Power PC. Linux securely hosts many databases, webservers, file and print servers, from many vendors, scaling both in price and ease, according to need. Linux now has two fully interoperating desktop systems and Libraries, KDE and GNOME, the latters Accessabilty Toolkit with the OpenOffice.org office suite has been singled out in this year's "Helen Keller Achievement Award in Technology".
http://newsvac.newsforge.com/article.pl?sid=02/09/13/1955240

Many vendors are now coming out with Linux based PDAs and embedded devices.

Granted, many companies, notably IBM, already offer many Linux based solutions. IBM has already turned all of its hardware and many software platforms into Linux hosting or hosted systems, however it is certainly not only vendor to do so. SGI, one of the leading Unix companies, is shattering world performance records, attaining linear scalability on a 64-processor Itanium2 based hardware running Linux.
http://www.sgi.com/newsroom/press_releases/2002/september/stream.html

With Linux many vendors already have the ultimate software and hardware reference platform. Most of all the commercial Unixs, Mini and Mainframe environments provide the ability to either directly run Linux binaries or host recompiled Linux source, Linux development will now dominate the enterprise

market. Even Microsoft's CEO Steve Ballmer has boasted that *'Microsoft has to make it as easy as possible to port Linux to Windows.'*
http://www.crn.com/Sections/BreakingNews/dailyarchives.asp?ArticleID=35789

The reason that the Linux and GNU libraries interpretation of the open Posix interface standards is becoming the defacto standard among most Unix and other OS vendors, rather than the BSD variants, is that it is freely available under commons preserving GPL and LGPL licensing. Unix/Posix popularity was driven by the degree to which it has provided a consistent stable and relatively secure interface while some other vendors change enterprise architectures on an almost yearly basis. This has made Posix a business requirement for forward looking implementors and vendors who wish to maintain a feature-rich, reusable, cross-language environments over a decent period of time, so the solution has a chance to be developed to become more stable and secure. Open source development under commons preserving licensing is providing an even more consistently stable and relatively secure interface for system hardware vendors,for open software/free licensed software developers and even provides a highly stable platform for proprietary software developers.

SAP, one of the leaders in enterprise and CRM systems is now using Linux as it's reference platform for all new SAP developments.
http://www9.sap.com/community/week35_1.asp

Oracle, one of the leaders in enterprise database systems, is now working with Redhat and others to make Linux even more secure, scaleable and faster platform for Oracle's own products.
http://www.oracle.com/linux/

It is inevitable that the inherent advantages will extend Linux beyond the datacenter and server market to migrate to the the X11 enterprise application server, the technical desktop, the specific role desktop, the business desktop, the home desktop, to the inevitable ubiquitous computing environment.

X11 extends Linux beyond the Linux/Unix desktop universe. There are X11-servers available for every desktop OS, from Microsoft's windows to MacOS9 and MacOSX.
http://www.rahul.net/kenton/xsites.html#XMicrosoft

Many of these X11-servers, such as under MacOSX's Xdarwin http://oroborosx.sourceforge.net/ or WinaXe on Microsoft Windows
http://www.apcmag.com/pics/ws/0101config5.gif
can fully intergrate the X11 applications into the host desktop environment, they look and feel like native applications.

In point of fact, X11 is the only distributed graphical interface to remain network/binary compatible back to 1986 X-clients. An organization can set up enterprise infrastructure behind a internal firewall and have it interfaced via X11 to the desktop - it can provide a consistent interface for decades. It is the only such system to remain virtually future-proof.

It is even possible to run some Microsoft Windows applications in a distributed thin-client environment under Linux, without needing Microsoft Operating System licenses for each client machine.
http://www.codeweavers.com/products/cxofficeserver/

Because you can distribute X11 applications across computers, it also can provide improved performance and a more secure environment for your organization's systems and data. Data intensive applications can run on or close to the the databases. Processor intensive applications can run on unused or shared special purpose computers. Exposed Internet clients such as web browsers, instant messaging, file sharing etc, can be run on "isolated" servers. Multimedia/display intensive applications can run locally, taking full advantage of full hardware and OpenGL acceleration such as DRI and GFX though interface libraries such as the multiplatform Simple DirectMedia Layer
http://www.libsdl.org/index.php

This multimedia ability, along with Linux's flexibility as been driving force for Linux's recent rapid adoption as a desktop for professional animation and digital effects in Hollywood and around the world.
http://www.linuxmovies.org/articles.html

Where locally hosted applications on other OS's are desirable, GNOME, GTK and KDE applications can be quickly ported to Windows and OSX.
http://homepage.ntlworld.com/steven.obrien2/

2. Sun Grants *full* rights to implement though JSPA agreement.

Unlike Microsoft's very limited submission to the ECMA (excluding ..NET notables, such as ASP.NET, WinForms and ADO.NET, to name a few) Sun has granted the Apache and all open source developers FULL rights to develop competing products.
http://jakarta.apache.org/site/jspa-agreement.html

THE AGREEMENT

The agreement responds to four concerns Apache raised in January and posted publicly on its Website at href="http://jakarta.apache.org/news/jspa-position.html

These four points were seen as critical to Apache's support of and continued participation in the Java Community Process:

- The right to freely implement specifications in open source
- The right for specification leads to

release reference implementations and test kits in open source

- The right for specifications to be created more publicly
- The right to free access to test kits by open source,

  non-profit, and academic groups

These issues are being addressed via two mechanisms. First, through a series of proposed revisions to the legal agreement signed when joining the Java Community Process, known as the JSPA. The JSPA is currently undergoing revision by the JCP Executive Committee (EC). As part of an early review, EC comments indicated a desire to address the concerns Apache raised. Apache is pleased to see Sun, who leads the JSPA revision process, propose modifications which address these concerns. This will help guide the JSPA revision process to conclusion.

Second, and for more immediate effect, Sun has agreed to proactively address Apache's issues for Sun-led JSRs in the following manner, as expressed in a letter of intent posted at

http://jcp.org/aboutJava/communityprocess/announce/LetterofIntent.html

For Sun-led specifications finalized from here forward (including revisions to existing specifications) the license terms will allow independent implemenations under open source licenses.

- The Test Compatibility Kit (TCK) binaries for these specifications will be made available at no cost to qualified open source, non-profit, and academic groups.
- A three-member board, including a representative

  from the Apache Software Foundation, will assure an impartial qualification process.
- Sun will provide substantial support to aid these

  qualified groups in the use and execution of the TCKs.

Apache welcomes other companies leading Java specifications -- many of whom have also desired more openness in the JCP -- in making similar pledges.

Microsoft CEO Steve Ballmer has made no similar acknowledgments, in fact quite the opposite ... http://swpat.ffii.org/players/microsoft/index.en.html

Responding to questions about the opening-up of the .NET framework, Ballmer announced that there would certainly be a "Common Language Runtime Implementation" for Unix, but then explained that this development would be limited to a subset, which was "intended only for academic use". Ballmer rejected speculations about support for free .NET implementations such as Mono: "We have invested so many millions in .NET, we have so many patents on .NET, which we want to cultivate."

There are those who claim that .NET is open to implentation, but until Microsoft make a legal public declaration similar to that of the JSPA, it is NOT.

Because of the above and Sun's progressive licensing toward both proprietary and competing implementation vendors http://java.sun.com/j2ee/licensees.html Java will dominate anywhere a 100% portable and truly secure distributed agent environment is required.

However it is easy enough to create portable Linux targeted source code portable to all Linux hosted architectures, where the Linux kernel can already provide a more secure environment than Microsoft's XP or but does not suffer the implicit overhead required by a virtual machine environment such as JVM or .NET.

3. Freely avalable development enviroment.

The GNU GCC project is in the singular position of being the leading development toolset for both the open source and proprietary Unix enterprise community. Even before Linux and FreeBSD was released, cross compiling and installing GNU utilities on proprietary Unixs was first things you did. Even Microsoft recognized this with the inclusion of the GNU GCC toolkit in it's Services For Unix (SFU) toolkit.
http://www.microsoft.com/windows/sfu/productinfo/overview/default.asp

Every iteration of GCC includes more support for the official ISO/ANSI standards for C,C++ and Sun's specifications for Java (yes! GCC compiles Java Classes into native code), with the aim of providing a consistent, stable and secure multi-targeted development platform.
Freely available development tools such as Emacs, XEmacs and Eclipse follows the principle of being able to fully customize the enviroment for productive professional development. Third party proprietary IDEs range from IBM's fully integrated J2EE website/webservices application development environment Websphere, to Borland's Kylix, the latter being freely available for developing GPL licensed projects. There are so many freely licensed development tools,environments,libraries and projects to choose from, that it is VERY rare for not to find an pre-existing open source stable solution for your problem; you do not have to start from scratch, just adopt and maybe adapt a pre-existing open source solution. http://freshmeat.net/

The role of open source development, under commons preserving GPL or LGPL licenses, in driving Linux adoption should not be underestimated. Though few companies will be anxious pay for software developed from scratch in the current slow economic environment, they don't have to. One of the key advantages of free licensing over proprietary solution development is the simplified legal invocations without the hassle of NDA'a and intellectual property cross licensing. It encourages both organizations and individuals to participate in the full knowledge that none of the participating parties can deny access at a later date though threat of intellectual property lawsuits or licensing.

The end user rights granted by the GPL and LGPL even extend to vendors of proprietary software, who may even be producing software that is direct competition with open source software. There is nothing to prevent proprietary software vendors from *linking* and distributing LGPL licensed code with their software, as long as are willing to distributed the LGPL'ed source code to the end users. There is nothing to prevent proprietary software vendors from *bundling* and distributing GPL licensed code with their software, as long as are willing to distributed the GPL'ed source code to the end users. As mentioned above Microsoft already does this with the GPL licensed GCC developer toolkit.

Even with a full GPL, the proprietary software vendor can strip out the required functionality from GPL sources and create separate standalone application that runs as a mini-server, callable via command line and passing data via pipes or even shared dynamic memory to the proprietary licensed application.

In comparison, on Microsoft's OSs, most application and middleware vendors that have gained a large enough market share of the desktop, soon find themselves directly competing with Microsoft. In many cases, Microsoft add the competing functionality to a freely bundled or pre-existing Microsoft products, often in a manner that is difficult for the competing vendor to interoperate with. The only chance of redress in years of lawsuits or decades of toothless antitrust cases. GPL Freedom is not only for the Free.

This means open source development method along with the commons preserving free licensing such as GPL and LGPL will provide the ultimate competitive advantage for the participants developing as well as the end customers, serving an incentive that is driving open source adoption far beyond what any NDA closed loop, RAND hobbled UN-"Shared" source option will provide. Given the assured lifespan of free licensed software, this is the pebble that *is* causing an avalanche.

In the meantime proprietary software,made much easier by adopted initiatives such as Linux Standard Base http://www.linuxbase.org/ ,will be there to fill gaps missing in ease of use or functionality.

4. Anybody can generate revenue from an open commons.
http://www.m-w.com/cgi-bin/dictionary?va=commons

Commons : the legal right of taking a profit in another's land in common with the owner or others

The GPL and similar licenses gives the inevitable strategic victory in the battle for mindshare with Microsoft. Microsoft's recent additions generate little for its end customers beyond its ability to lock it's customers into Microsoft's own products. Few of Microsoft's Internet exposed servers or clients are designed or implemented with security in mind, whereas Linux has range of security-compatible products (securer servers & applications , chroot & LSM etc) from which *END* organizations and individuals can *USE* to generate revenue. Enough of that revenue will be spent to make alliances with third parties to collectively develop new functionality, as well as directly suport the development infrastructure that has assisted in over one billion (a Gigabuck) worth of development to freely available source code available in Linux distributions today.
http://www.dwheeler.com/sloc/

GPL/Linux free licensed distributions provides a true free market, unconstrained by intellectual property monopolies that inflate the prices, to the developing vendors and the end consumer. Redhat could never be the next Microsoft, any attempt to lock customers into it's products would have both customers and vendors quickly switching distributions. But Redhat, and most of the other Linux distributions, have actually collaborated in a manner that actually encourages competition. http://www.linuxbase.org/

Contrast that with Microsoft's attempts to drive Microsoft's profitability by it's anti-competitive business practices (Don't take my word for it, read the Opinion of the Circuit Court of Appeals
http://www.naag.org/issues/microsoft/docs-state/pdf/ms-ct_of_apls-opinion.pdf
) Microsoft has no flexibility to offer when dealing with any real competition simply because they have so much of their revenue-generating business model based solely around retaining a absolutely dominant monopoly.

Conclusion

Customers benefit from seamless interoperability with other vendors products. (Microsoft has not even implemented a fully inter-operable version of the BSD licensed Kerberos). This is partially true in a world where networks are ubiquitous and computing devices extend far beyond the traditional computer. In such a market, breadth is a very good thing. It creates innovations in software design Even more important, it gives the customer the knowledge that they can choose a range of products to truly suit their individual needs. Even competing proprietary vendors can benefit from the freedom provided by the GPL and LGPL to insure that their own products and services can fully inter-operate, now and in the future.

Linux is simply the best positioned to be the cross-

platform, cross-market unification technology, guaranteed to grow correctly because of the way Linux is licensed. Many market vendors are already heavily involved with Linux, open source and free licensed software, benefiting from unification under free licensing. Microsoft could have provided that unification (based on Microsoft's early popularity among PC developers), but, right or wrong, Microsoft's executives greed ensured that it would not. Once again, Microsofts guilty of multiple violations of the Sherman Act
http://www.naag.org/issues/microsoft/docs-state/pdf/ms-ct_of_apls-opinion.pdf
(including five counts directly involving Java)
, yet once again, seems unable to comply with the most liberal of Department Of Justice Settlements
http://216.133.66.117/091802.pdf

Horace Greeley (1811-1872), Editor of the New York Tribune in an editorial in 1841 said: "*Do not lounge in the cities! There is room and health in the country, away from the crowds of idlers and imbeciles. Go west, before you are fitted for no life but that of the factory.* "

In the same way, I urge you to...
Do not lounge on the Microsoft platform! There is room and scope on Linux, away from the crowds of idlers and imbeciles, Go open, before you are fitted for no life but that of the helpdesk.

But more importantly, by 1871 Horace Greeley also wrote: "*This Daniel Boone business is about played out.*"

In the same way, the last decade's Linux customer base can be seen as the self reliant pioneers. The "Do It Yourself" attitude and habit was learned from a time when "doing for themselves" was the only option. This is no longer the case, there are plenty new settlers and far many more willing to migrate, who are all too willing to pay for hardware, support, customization, collective development and even quality proprietary licensed products.

Linux will conquer the world. Yet, only by attempting to pull the wool over your eyes does Microsoft stand a chance, of denying what they have known to be true from the outset...
http://www.opensource.org/halloween/

David Mohring - "Go Linux, Young Person!"

AMERICAN
BOOK STORE

*10% DISCOUNT
TO AUUG MEMBERS
ON OUR COMPLETE RANGE
OF COMPUTER, BUSINESS
AND GENERAL BOOKS*

173 Elizabeth St, Brisbane Queensland 4000
Ph: (07) 3229 4677 Fax: (07) 3221 2171 Qld Country Freecall: 1800 177 395
american_bookstore@compuserve.com

Name: ______________________________ Date: ____________

Address: ____

______________________________________ Post Code: __________

Phone Number: ______________________________

Payment Method: ❑ Cheque ❑ Money Order ❑ Amex ❑ Bankcard
❑ Diners ❑ Mastercard ❑ Visa

Card Number: ____

Expiry Date: ____________________ Signature: ______________________

This is a: ❑ Special Order ❑ Mail Order ❑ Book on Hold

QUANTITY	TITLE	PRICE

SUBTOTAL	$ ....
LESS 10% DISCOUNT	$ ....
POST & PACK	$ ....
TOTAL	$ ....

POSTAGE AND HANDLING FEES: 1 BOOK $6.00 2-4 BOOKS $7.00
BOOKS OVER $70.00 WE WILL SEND CERTIFIED - PLEASE ADD ANOTHER $1.50 OR WAIVE CERTIFIED DELIVERY.

FOR SPECIAL ORDERS, PLEASE ENCLOSE $10.00 PER BOOK AS A DEPOSIT.

AUUG Chapter Meetings and Contact Details

CITY	LOCATION	OTHER
ADELAIDE	We meet at Internode, Level 3/132 Grenfell St aka 'the old AAMI building', at 7 pm on the second Wednesday of each month.	Contact sa-exec@auug.org.au for further details.
BRISBANE	Inn on the Park 507 Coronation Drive Toowong	For further information, contact the QAUUG Executive Committee via email (qauug-exec@auug.org.au). The technologically deprived can contact Rick Stevenson on (07) 5578-8933. To subscribe to the QAUUG announcements mailing list, please send an e-mail message to: <majordomo@auug.org.au> containing the message "subscribe qauug <e-mail address>" in the e-mail body.
CANBERRA	Australian National University	
HOBART	University of Tasmania	
MELBOURNE	Various. For updated information See: http://www.vic.auug.org.au/auugvic/av_meetings.html	The meetings alternate between Technical presentations in the odd numbered months and purely social occasions in the even numbered months. Some attempt is made to fit other AUUG activities into the schedule with minimum disruption.
PERTH	The Victoria League 276 Onslow Road Shenton Park	
SYDNEY	TBA	

For up-to-date details on chapters and meetings, including those in all other Australian cities, please check the AUUG website at http://www.auug.org.au or call the AUUG office on **1-800-625655.**

Application / Renewal Individual or Student Membership of AUUG Inc.

Use this tax invoice to apply for, or renew, Individual or Student Membership of AUUG Inc. To apply online or for Institutional Membership please use **http://www.auug.org.au/info/**

This form serves as Tax Invoice.

Please complete and return to:

AUUG Inc, PO Box 7071, BAULKHAM HILLS BC NSW 2153, AUSTRALIA

If paying for your membership with a credit card, this form may be faxed to AUUG Inc. on +61 2 8824 9522.

Please do not send purchase orders.
Payment must accompany this form.

Overseas Applicants:

- Please note that all amounts quoted are in Australian Dollars.
- Please send a bank draft drawn on an Australian bank, or credit card authorisation.
- There is a $60.00 surcharge for International Air Mail
- If you have any queries, please call AUUG Inc on +61 2 8824 9511 or freephone 1800 625 655.

Section A:

Personal Details

Surname: ....

First Name: ....

Title: .... Position: ....

Organisation: ....

Address: ....

....

Suburb: ....

State: .... Postcode: ....

Country: .... Phone Work: ....

Phone Private: .... Facsimile: ....

E-mail: ....

Membership Number (if renewing): ....

Student Member Certification

For those applying for Student Membership, this section is required to be completed by a member of the academic staff.

I hereby certify that the applicant on this form is a full time student and that the following details are correct:

Name of Student: ....

Institution: ....

Student Number: ....

Signed: ....

Name: ....

Title ....

Date Signed: ....

Section B: Prices

Please tick the box to apply for Membership. Please indicate if International Air Mail is required.

Renew/New* Individual Membership	$110.00 (including $10 GST)	☐
Renew/New* Student Membership	$27.50 (including $2.50 GST)	☐
Surcharge for International Air Mail	$60.00	☐

* Delete as appropriate.

GST only applies to payments made from within Australia. Rates valid from 1st October 2002.

Section C: Mailing Lists

AUUG mailing lists are sometimes made available to vendors. Please indicate whether you wish your name to be included on these lists:

Yes ☐ No ☐

Section D: Payment

Pay by cheque

Cheques to be made payable to **AUUG Inc**. Payment in Australian Dollars only.

OR Pay by credit card

Please debit my credit card for A$ ....

Bankcard ☐ Mastercard ☐ Visa ☐

Card Number: .... Expires: ....

Name on card: .... Signature: ....

Date Signed: ....

Section E: Agreement

I agree that this membership will be subject to rules and bylaws of AUUG Inc as in force from time to time, and this membership will run from the time of joining/renewal until the end of the calendar or financial year as appropriate.

Signed: ....

Date Signed: ....

This form serves as Tax Invoice. AUUG ABN 15 645 981 718