Cfengine | VNC | Billix | Zenoss | Thin Clients | MySQL | Heartbeat

LINUX JOURNAL

Since 1994: The Original Magazine of the Linux Community

2009 | Supplement Issue | www.linuxjournal.com

SPECIAL SYSTEM ADMINISTRATION ISSUE

MAY THE SOURCE BE WITH YOU

Billix: the Sysadmin Toolkit for Your Pocket

Authentication with **Fedora Directory Server**

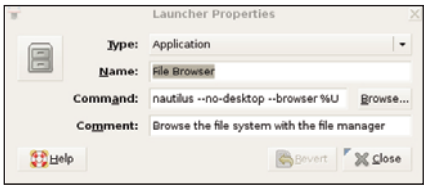
Remote Desktop Administration with **OpenSSH** or **VNC**

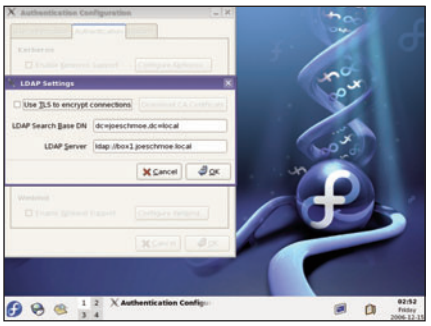
Manage Configuration Files with **Cfengine**

Setting Up a **PXE Boot Server**

Network Monitoring with **Zenoss**

CONTENTS SPECIAL SYSTEM ADMINISTRATION ISSUE





Linux Journal 2009 Lineup

JANUARY
Security

FEBRUARY
Web Development

MARCH
Desktop

APRIL
System Administration

MAY
Cool Projects

JUNE
Readers' Choice Awards

JULY
Mobile Linux

AUGUST
Kernel Capers

SEPTEMBER
Cross-Platform Development

OCTOBER
Hack This

NOVEMBER
Infrastructure

DECEMBER
Embedded

USPS *LINUX JOURNAL* (ISSN 1075-3583) (USPS 12854) is published monthly by Belltown Media, Inc., 2211 Norfolk, Ste 514, Houston, TX 77098 USA. Periodicals postage paid at Houston, Texas and at additional mailing offices. Cover price is $5.99 US. Subscription rate is $29.50/year in the United States, $39.50 in Canada and Mexico, $69.50 elsewhere. POSTMASTER: Please send address changes to *Linux Journal*, PO Box 16476, North Hollywood, CA 91615. Subscriptions start with the next issue. Canada Post: Publications Mail Agreement #41549519. Canada Returns to be sent to Bleuchip International, P.O. Box 25542, London, ON N6C 6B2

LINUX JOURNAL

At Your Service

MAGAZINE

PRINT SUBSCRIPTIONS: Renewing your subscription, changing your address, paying your invoice, viewing your account details or other subscription inquiries can instantly be done on-line, **www.linuxjournal.com/subs**. Alternatively, within the U.S. and Canada, you may call us toll-free 1-888-66-LINUX (54689), or internationally +1-818-487-2089. E-mail us at subs@linuxjournal.com or reach us via postal mail, Linux Journal, PO Box 16476, North Hollywood, CA 91615-9911 USA. Please remember to include your complete name and address when contacting us.

DIGITAL SUBSCRIPTIONS: Digital subscriptions of *Linux Journal* are now available and delivered as PDFs anywhere in the world for one low cost. Visit **www.linuxjournal.com/digital** for more information or use the contact information above for any digital magazine customer service inquiries.

LETTERS TO THE EDITOR: We welcome your letters and encourage you to submit them at **www.linuxjournal.com/contact** or mail them to Linux Journal, 1752 NW Market Street, #200, Seattle, WA 98107 USA. Letters may be edited for space and clarity.

WRITING FOR US: We always are looking for contributed articles, tutorials and real-world stories for the magazine. An author's guide, a list of topics and due dates can be found on-line, **www.linuxjournal.com/author**.

ADVERTISING: *Linux Journal* is a great resource for readers and advertisers alike. Request a media kit, view our current editorial calendar and advertising due dates, or learn more about other advertising and marketing opportunities by visiting us on-line, **www.linuxjournal.com/advertising**. Contact us directly for further information, ads@linuxjournal.com or +1 713-344-1956 ext. 2.

ON-LINE

WEB SITE: Read exclusive on-line-only content on *Linux Journal*'s Web site, **www.linuxjournal.com**. Also, select articles from the print magazine are available on-line. Magazine subscribers, digital or print, receive full access to issue archives; please contact Customer Service for further information, subs@linuxjournal.com.

FREE e-NEWSLETTERS: Each week, *Linux Journal* editors will tell you what's hot in the world of Linux. Receive late-breaking news, technical tips and tricks, and links to in-depth stories featured on **www.linuxjournal.com**. Subscribe for free today, **www.linuxjournal.com/enewsletters**.

LINUX JOURNAL

Executive Editor	Jill Franklin jill@linuxjournal.com
Associate Editor	Shawn Powers shawn@linuxjournal.com
Associate Editor	Mitch Frazier mitch@linuxjournal.com
Senior Editor	Doc Searls doc@linuxjournal.com
Art Director	Garrick Antikajian garrick@linuxjournal.com
Products Editor	James Gray newproducts@linuxjournal.com
Editor Emeritus	Don Marti dmarti@linuxjournal.com
Technical Editor	Michael Baxter mab@cruzio.com
Senior Columnist	Reuven Lerner reuven@lerner.co.il
Chef Français	Marcel Gagné mggagne@salmar.com
Security Editor	Mick Bauer mick@visi.com

Contributing Editors

David A. Bandel • Ibrahim Haddad • Robert Love • Zack Brown • Dave Phillips • Marco Fioretti
Ludovic Marcotte • Paul Barry • Paul McKenney • Dave Taylor • Dirk Elmendorf

Proofreader Geri Gale

Publisher	Carlie Fairchild publisher@linuxjournal.com
General Manager	Rebecca Cassity rebecca@linuxjournal.com
Sales Manager	Joseph Krack joseph@linuxjournal.com
Sales and Marketing Coordinator	Tracy Manford tracy@linuxjournal.com
Circulation Director	Mark Irgang mark@linuxjournal.com
Webmistress	Katherine Druckman webmistress@linuxjournal.com
Accountant	Candy Beauchamp acct@linuxjournal.com

Linux Journal **is published by, and is a registered trade name of, Belltown Media, Inc.**
PO Box 980985, Houston, TX 77098 USA

Advertising

E-MAIL: ads@linuxjournal.com
URL: www.linuxjournal.com/advertising
PHONE: +1 713-344-1956 ext. 2

Subscriptions

E-MAIL: subs@linuxjournal.com
URL: www.linuxjournal.com/subscribe
PHONE: +1 818-487-2089
FAX: +1 818-487-4550
TOLL-FREE: 1-888-66-LINUX
MAIL: PO Box 16476, North Hollywood, CA 91615-9911 USA
Please allow 4–6 weeks for processing address changes and orders
PRINTED IN USA

SHAWN POWERS

System Administration: Instant Gratification Edition

Welcome to the *Linux Journal* family! The issue you currently hold in your hands, or more precisely, the issue you're reading on your screen, was created specifically for you. It's so frustrating to subscribe to a magazine, and then wait a month or two before you ever get to read your first issue. You decided to subscribe, so we decided to give you some instant gratification. Because really, isn't that the best kind?

This entire issue is dedicated to system administration. As a Linux professional or an everyday penguin fan, you're most likely already a sysadmin of some sort. Sure, some of us are in charge of administering thousands of computers, and some of us just keep our single desktop running. With Linux though, it doesn't matter how big or small your infrastructure might be. If you want to run a MySQL server on your laptop, that's not a problem. In fact, in this issue we even talk about stored procedures on MySQL 5, so we'll show you how to get the most out of that laptop server.

If you do manage lots of machines, however, there are some tools that can make your job much easier. Cfengine, for instance, can help with configuration management for tens, hundreds or even thousands of remote computers. It sure beats running to every server and configuring them individually. But, what if you mess up those configuration files and half your servers go off-line? Don't worry, we'll teach you about Heartbeat. It's one of those high-availability programs that helps you keep going, even when some of your servers die. If a computer dies alone in your server room, does anyone hear it scream? Heartbeat does.

Do you have more than one server room? More than one location? Many of us do. Programs like Zenoss can help you monitor your whole organization from one central place. In the following pages, we'll show you all about it. In fact, using a VPN (which we talk about here) and remote desktop sessions (again, covered here), you probably can manage most of your network without even leaving home. If you haven't reconfigured a server from a coffee shop across the country, you haven't lived.

I do understand, however, that lots of sysadmins enjoy going in to work. That's great; we like office coffee too. Keep reading to find many hands-on tools that will make your work day more efficient, more productive and even more fun. Billix, for example, is a USB-booting Linux distribution that has tons of great features ready to use. It can install operating systems, rescue broken computers, and if you want to do something it doesn't do by default, that's okay, the article explains how to include any custom tools you might want.

Whether it's booting thin clients over a wireless bridge or creating custom PXE boot menus, the life of a system administrator is never boring. At *Linux Journal*, we try to make your job easier month after month with product reviews, tech tips, games (you know, for your lunch break) and in-depth articles like you'll read here. Whether you're a command-line junkie or a GUI guru, Linux has the tools available to make your life easier.

We hope you enjoy this bonus issue of *Linux Journal*. If you read this whole thing and are still anxious to get your hands on more Linux and open-source-related information, but your first paper issue hasn't arrived yet, don't forget about our Web site. Check us out at **www.linuxjournal.com**. If you manage to read the thousands and thousands of on-line articles and *still* don't have your first issue, I suggest checking with the post office. There must be something wrong with your mailbox!■

Shawn Powers is the Associate Editor for *Linux Journal*. He's also the Gadget Guy for LinuxJournal.com, and he has an interesting collection of vintage Garfield coffee mugs. Don't let his silly hairdo fool you, he's a pretty ordinary guy and can be reached via e-mail at shawn@linuxjournal.com. Or, swing by the #linuxjournal IRC channel on Freenode.net.

Cfengine for Enterprise Configuration Management

Cfengine makes it easier to manage configuration files across large numbers of machines.

SCOTT LACKEY

Cfengine is known by many system administrators to be an excellent tool to automate manual tasks on UNIX and Linux-based machines. It also is the most comprehensive framework to execute administrative shell scripts across many servers running disparate operating systems. Although cfengine is certainly good for these purposes, it also is widely considered the best open-source tool available for configuration management. Using cfengine, sysadmins with a large installation of, say, 800 machines, can have information about their environment quickly that otherwise would take months to gather, as well as the ability to change the environment in an instant. For an initial example, if you have a set of Linux machines that need to have a different /etc/nsswitch.conf, and then have some processes restarted, there's no need to connect to each machine and perform these steps or even to write a script and run it on the machines once they are identified. You simply can tell cfengine that all the Linux machines running Fedora/Debian/CentOS with XGB of RAM or more need to use a particular /etc/nsswitch.conf until a newer one is designated. Cfengine can do all that in a one-line statement.

Cfengine's configuration management capabilities can work in several different ways. In this article, I focus on a make-it-so-and-keep-it-so approach. Let's consider a small hosting company configuration, with three administrators and two data centers (Figure 1).

Each administrator can use a Subversion/CVS sandbox to hold repositories for each data center. The cfengine client will run on each client machine, either through a cron job or a cfengine execution dæmon, and pull the cfengine configuration files appropriate for each machine from the server. If there is work to be done for that particular machine, it will be carried out and reported to the server. If there are configuration files to copy, the ones active on the client host will be replaced by the copies on the cfengine server. (Cfengine will not replace a file if the copy process is partial or incomplete.)

A cfengine implementation has three major components:

- Version control: this usually consists of a versioning system, such as CVS or Subversion.
- Cfengine internal components: cfservd, cfagent, cfexecd, cfenvd, cfagent.conf and update.conf.

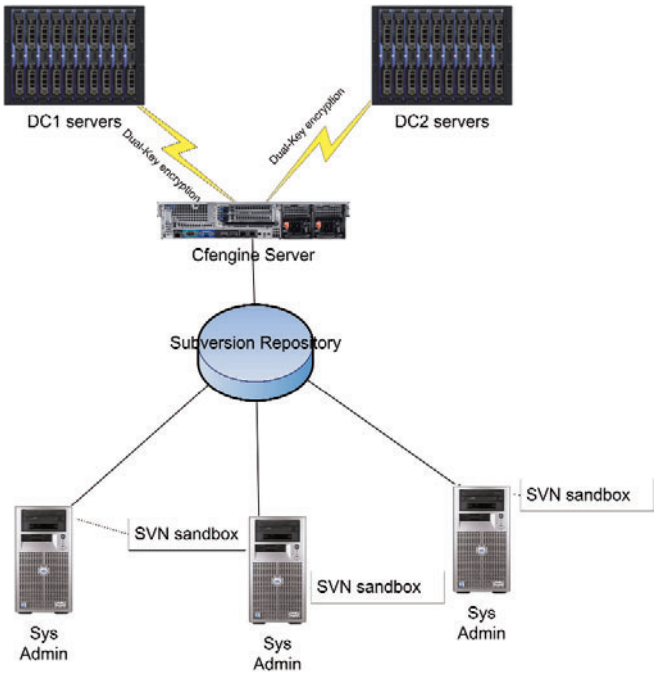


Figure 1. How the Few Control the Many

- Cfengine commands: processes, files, shellcommands, groups, editfiles, copy and so forth.

The cfservd is the master dæmon, configured with /etc/cfservd.conf, and it listens on port 5803 for connections to the cfengine server. This dæmon controls security and directory access for all client machines connecting to it. cfagent is the client program for running cfengine on hosts. It will run either from cron, manually or from the execution dæmon for cfengine, cfexecd. A common method for running the cfagent is to execute it from cron using the cfexecd in non-dæmon mode. The primary reason for using both is to engage cfengine's logging system. This is accomplished using the following:

```
*/10 * * * * /var/cfengine/sbin/cfexecd -F
```

as a cron entry on Linux (unless Solaris starts to understand */10). Note that this is fairly frequent and good only for a low number of servers. We don't want 800 servers updating within

the same ten minutes.

The cfenvd is the "environment dæmon" that runs on the client side of the cfengine implementation. It gathers information about the host machine, such as hostname, OS and IP address. The cfenvd detects these factors about a host and uses them to determine to which groups the machine belongs. This, in effect, creates a profile for each machine that cfengine uses to determine what work to perform on each host.

The master configuration file for each host is cfagent.conf. This file can contain all the configuration information and cfengine code for the host, a subset of hosts or all hosts in the cfengine network. This file is often just a starting point where all configurations are stored in other files and "imported" into cfagent.conf, in a very similar fashion to Nagios configuration files. The update.conf file is the fundamental configuration file for the client. It primarily just identifies the cfengine server and gets a copy of the cfagent.conf.

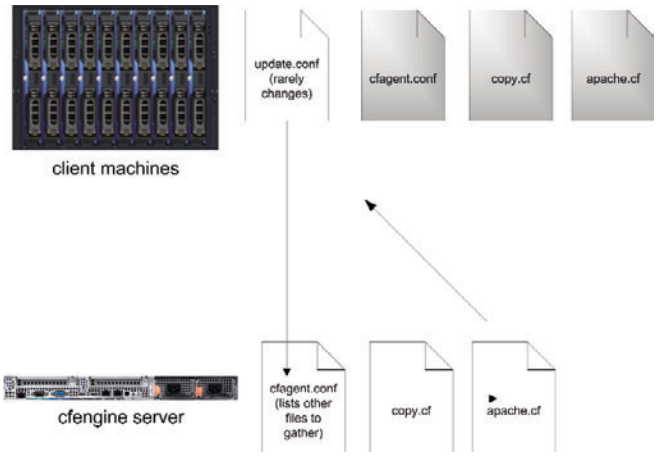


Figure 2. Automated Distribution of Cfengine Files

The update.conf file tells the cfengine server to deploy a new cfagent.conf file (and perhaps other files as well) if the current copy on the host machine is different. This adds some protection for a scenario where a corrupt cfagent.conf is sent out or in case there never was one. Although you could use cfengine to distribute update.conf, it should be copied manually to each host.

Cfengine "commands" are not entered on the command line. They make up the syntax of the cfengine configuration language. Because cfengine is a framework, the system administrator must write the necessary commands in cfengine configuration files in order to move and manipulate data. As an example, let's take a look at the files command as it would appear in the cfagent.conf file:

```
files:

/etc/passwd mode=644
    owner=root action=fixall
/etc/shadow mode=600
    owner=root action=fixall
```

This would set all machines' /etc/passwd and /etc/shadow files to the permissions listed in the file (644 and 600). It also would change the owner of the file to root and fix all of these settings if they are found to be different, each time cfengine runs. It's important to keep in mind that there are no group limitations to this particular files command. If cfengine does not have a group listed for the command, it assumes you mean any host. This also could be written as:

Because cfengine is a framework, the system administrator must write the necessary commands in cfengine configuration files in order to move and manipulate data.

```
files:
      any::
            /etc/passwd mode=644
                owner=root action=fixall
            /etc/shadow mode=600
                owner=root action=fixall
```

This brings us to an important topic in building a cfengine implementation: groups. There is a groups command that can be used to assign hosts to groups based on various criteria. Custom groups that are created in this way are called soft groups. The groups that are filled by the cfenvd dæmon automatically are referred to as hard groups. To use the groups feature of cfengine and assign some soft groups, simply create a groups.cf file, and tell the cfagent.conf to import it somewhere in the beginning of the file:

```
import:
       any::
            groups.cf
```

Cfengine will look in the default directory for the groups.cf file in /var/cfengine/inputs. Now you can create arbitrary groups based on any criteria. It is important to remember that the terms groups and classes are completely interchangeable in cfengine:

```
groups:
          development = ( nfs01 nfs02 10.0.0.17 )
          production = ( app01 app02 !development )
```

You also can combine hard groups that have been discovered by cfenvd with soft groups:

```
groups:
          legacy = ( irix compiled_on_cygwin sco )
```

Let's get our testing setup in order. First, install cfengine on a server and a client or workstation. Cfengine has been compiled on almost everything, so there should be a package for your OS/distribution. Because the source is usually the latest version, and many versions are bug fixes, I recommend compiling it yourself. Installing cfengine gives you both the server and client binaries

A great way to version control your config files is to use a cfengine variable for the filename being copied to control which version gets distributed.

and utilities on every machine, so be careful not to run the server dæmon (cfservd) on a client machine unless you specifically intend to do that. After the install, you should have a /var/cfengine/ directory and the binaries mentioned previously.

Before any host can actually communicate with the cfengine server, keys must be exchanged between the two. Cfengine keys are similar to SSH keys, except they are one-way. That is to say, both the server and the client must have each other's public key in order to communicate. Years of sysadmin paranoia cause me to recommend manually copying all keys and trusting nothing. Copy /var/cfengine/ppkeys/localhost.pub from the server to all the clients and from the clients to the server in the same directory, renaming them /var/cfengine/ppkeys/root-10.11.0.1.pub, where the IP is 10.11.0.1.

On the server side, cfservd.conf must be configured to allow clients to access particular directories. To do this, create an AllowConnectionsFrom and an admit section:

```
#cfservd.conf

control:
          AllowConnectionsFrom = ( 192.168.0.0/24 )
admit:
          /configs/datacenter1 *.example1.com
          /configs/datacenter2 *.example2.com
```

To test your example client to see whether it is connecting to the cfengine server, make sure port 5803 is clear between them, and run the server with:

```
cfservd -v -d2
```

And, on the client run:

```
cfagent -v --no-splay
```

This will give you a lot of debugging information on the server side to see what's working and what isn't.

Now, let's take a look at distributing a configuration file. Although cfengine has a full-featured file editor in the editfiles command, using this method for distributing configurations is not advised. The copy command will move a file from the server to the client machine with .cfnew appended to the filename. Then, once the file has been copied completely, it renames the file and saves the old copy as .cfsaved in the specified directory. Here's the copy command syntax:

```
copy:
      class::
               <<master-file>>

                     dest=target-file
                     server=server
                     mode=mode
                     owner=owner
                     group=group
                     backup=true/false
                     repository=backup dir
                     recurse=number/inf/0
                     define=classlist
```

Only the dest= is required, along with the filename to save at the destination. These can be different. Here's another example:

```
copy:
       linux::
          ${copydir}/linux/resolv.conf

               dest=/etc/resolv.conf
               server=cfengine.example1.com
               mode=644
               owner=root
               group=root
               backup=true
               repository=/var/cfengine/cfbackup
               recurse=0
               define=copiedresolvdotconf
```

The last line in this copy statement assigns this host to a group called copiedresolvdotconf. Although we don't have to do anything after copying this particular file, we may want to do some action on all hosts that just had this file successfully sent to them, such as sending an e-mail or restarting a process. As another example, if you update a configuration file that is attached to a dæmon, you may want to send a SIGHUP to the process to cause it to reread the configuration file. This is common with Apache's httpd.conf or inetd.conf. If the copy is not successful, this server won't be added to the copiedresolvdotconf class. You can query all servers in the network to see whether they are members and, if not, find out what went wrong.

A great way to version control your config files is to use a cfengine variable for the filename being copied to control which version gets distributed. Such a line may look something like this:

```
copy:
       linux::
           ${copydir}/linux/${resolv_conf}
```

Or, better yet, you can use cfengine's class-specific variables, whose scope is limited to the class with which they are associated. This makes copy statements much more elegant and can simplify changes as your cfengine files scale:

```
control:

    # ${resolve_conf} value depends on context,
```

```
    # is this a linux machine or hpux?
    linux:: resolve_conf = ( "${copydir}"/linux/resolv.conf )
    hpux:: resolve_conf = ( "${copydir}"/hpux/resolv.conf )
 copy:
       linux::

                ${resolve_conf}
```

Here is a full cfagent.conf file that makes use of everything I've covered thus far. It also adds some practical examples of how to do sysadmin work with cfengine:

```
# cfagent.conf

control:
        actionsequence = ( files editfiles processes )
        AddInstallable = ( cron_restart )
      solaris:: crontab = ( /var/spool/cron/crontabs/root
)
      linux:: crontab = ( /var/spool/cron/root )

files:
        solaris::
          ${crontab}
                        action=touch
       linux::
          ${crontab}
                        action=touch

editfiles:
       solaris::
            { ${crontab}
            AppendIfNoSuchLine "0,10,20,30,40,50 * * * *
            ➥/var/cfengine/sbin/cfexecd -F"
            DefineClasses "cron_restart"
       }
      linux::
           { ${crontab}
           AppendIfNoSuchLine "0,10,20,30,40,50 * * * *
           ➥/var/cfengine/sbin/cfexecd -F"
         #linux doesn't need a cron restart.
           }

shellcommands:
        solaris.cron_restart::
             "/etc/init.d/cron stop"
             "/etc/init.d/cron start"

import:
         any::
                  groups.cf
                  copy.cf
```

The above is a full cfagent configuration that adds cfengine execution from cron to each client (if it's Linux or Solaris). So effectively, once you run cfengine manually for the first time with this cfagent.conf file, cfengine will continue to run every five minutes from that host, but you won't need to edit or restart cron. The control section of the cfagent.conf is where you can define some variables that will control how cfengine handles the configuration file. `actionsequence` tells cfengine what order to execute each command, and `AddInstallable` is a variable that holds soft groups that get defined later in the file in a "define" statement, such as after the editfiles command where the line is `DefineClasses "cron_restart"`. The reason for using AddInstallable is sometimes cfengine skips over groups that are defined after command execution, and defining that group in the control section ensures that the command will be recognized throughout the configuration.

Being able to check configuration files out from a versioning system and distribute them to a set of servers is a powerful system administration tool. A number of independent tools will do a subset of cfengine's work (such as rsync, ssh and make), but nothing else allows a small group of system administrators to manage such a large group of servers. Centralizing configuration management has the dual benefit of information and control, and cfengine provides these benefits in a free, open-source tool for your infrastructure and application environments. ■

Scott Lackey is an independent technology consultant who has developed and deployed configuration management solutions across industry from NASA to Wall Street. Contact him at slackey@violetconsulting.net, www.violetconsulting.net.

Secured Remote Desktop/Application Sessions

Run graphical applications from afar, securely.

MICK BAUER

There are many different ways to control a Linux system over a network, and many reasons you might want to do so. When covering remote control in past columns, I've tended to focus on server-oriented usage scenarios and tools, which is to say, on administering servers via text-based applications, such as OpenSSH. But, what about GUI-based applications and remote desktops?

Remote desktop sessions can be very useful for technical support, surveillance and remote control of desktop applications. But, it isn't always necessary to access an entire desktop; you may need to run only one or two specific graphical applications.

In this month's column, I describe how to use VNC (Virtual Network Computing) for remote desktop sessions and OpenSSH with X forwarding for remote access to specific applications. Our focus here, of course, is on using these tools securely, and I include a healthy dose of opinion as to the relative merits of each.

Remote Desktops vs. Remote Applications

So, which approach should you use, remote desktops or remote applications? If you've come to Linux from the Microsoft world, you may be tempted to assume that because Terminal Services in Windows is so useful, you *have* to have some sort of remote desktop access in Linux too. But, that may not be the case.

Linux and most other UNIX-based operating systems use the X Window System as the basis for their various graphical environments. And, the X Window System was designed to be run over networks. In fact, it treats your local system as a self-contained network over which different parts of the X Window System communicate.

Accordingly, it's not only possible but *easy* to run individual X Window System applications over TCP/IP networks—that is, to display the output (window) of a remotely executed graphical application on your local system. Because the X Window System's use of networks isn't terribly secure (the X Window System has no native support whatsoever for any kind of encryption), nowadays we usually tunnel X Window System application windows over the Secure Shell (SSH), especially OpenSSH.

The advantage of tunneling individual application windows is that it's faster and generally more secure than running the entire desktop remotely. The disadvantages are that OpenSSH has a history of security vulnerabilities, and for many Linux newcomers, forwarding graphical applications via commands entered in a shell session is counterintuitive. And besides, as I mentioned earlier, remote desktop control (or even just viewing) can be very useful for technical support and for security surveillance.

Using OpenSSH with X Window System Forwarding

Having said all that, tunneling X Window System applications over OpenSSH may be a lot easier than you imagine. All you need is a client system running an X server (for example, a Linux desktop system or a Windows system running the Cygwin X server) and a destination system running the OpenSSH dæmon (sshd).

Note that I didn't say "a destination system running sshd *and* an X server". This is because X servers, oddly enough, don't actually run or control X Window System applications; they merely display their output. Therefore, if you're running an X Window System application on a remote system, you need to run an X server on your local system, not on the remote system. The application will execute on the remote system and send its output to your local X server's display.

Suppose you've got two systems, mylaptop and remotebox, and you want to monitor system resources on remotebox with the GNOME System Monitor. Suppose further you're running the X Window System on mylaptop and sshd on remotebox.

First, from a terminal window or xterm on mylaptop, you'd open an SSH session like this:

```
mick@mylaptop:~$ ssh -X admin-slave@remotebox
admin-slave@remotebox's password: **********
Last login: Wed Jun 11 21:50:19 2008 from
dtcla00b674986d
admin-slave@remotebox:~$
```

Note the -X flag in my ssh command. This enables X Window System forwarding for the SSH session. In order for that to work, sshd on the remote system must be configured with X11Forwarding set to yes in its /etc/ssh/sshd.conf file. On many distributions, yes is the default setting, but check yours to be sure.

Next, to run the GNOME System Monitor on remotebox, such that its output (window) is displayed on mylaptop, simply execute it from within the same SSH session:

```
admin-slave@remotebox:~$ gnome-system-monitor &
```

The trailing ampersand (&) causes this command to run in

the background, so you can initiate other commands from the same shell session. Without this, the cursor won't reappear in your shell window until you kill the command you just started.

At this point, the GNOME System Monitor window should appear on mylaptop's screen, displaying system performance information for remotebox. And, that really is all there is to it.

This technique works for practically any X Window System application installed on the remote system. The only catch is that you need to know the *name* of anything you want to run in this way—that is, the actual name of the executable file.

If you're accustomed to starting your X Window System applications from a menu on your desktop, you may not know the names of their corresponding executables. One quick way to find out is to open your desktop manager's menu editor, and then view the properties screen for the application in question.

For example, on a GNOME desktop, you would right-click on the Applications menu button, select Edit Menus, scroll down to System/Administration, right-click on System Monitor and select Properties. This pops up a window whose Command field shows the name gnome-system-monitor.

Figure 1 shows the Launcher Properties, *not* for the GNOME System Monitor, but instead for the GNOME File Browser, which is a better example, because its launcher entry includes some command-line options. Obviously, all such options also can be used when starting X applications over SSH.

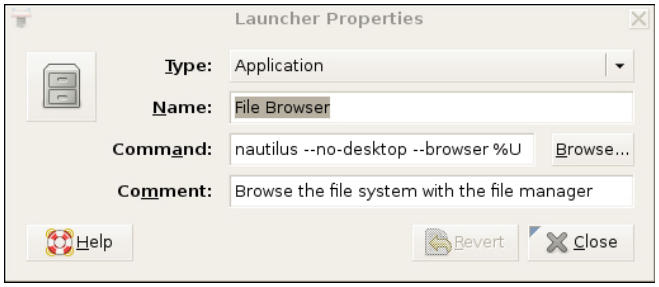


Figure 1. Launcher Properties for the GNOME File Browser (Nautilus)

If this sounds like too much trouble, or if you're worried about accidentally messing up your desktop menus, you simply can run the application in question, issue the command `ps auxw` in a terminal window, and find the entry that corresponds to your application. The last field in each row of the output from ps is the executable's name plus the command-line options with which it was invoked.

Once you've finished running your remote X Window System application, you can close it the usual way (selecting Exit from its File menu, clicking the x button in the upper right-hand corner of its window and so forth). Don't forget to close your SSH session too, by issuing the command `exit` in the terminal window where you're running SSH.

Virtual Network Computing (VNC)

Now that I've shown you the *preferred* way to run remote X Window System applications, let's discuss how to control an entire remote desktop. In the Linux/UNIX world, the most popular tool for this is Virtual Network Computing, or VNC.

But Don't Real Sysadmins Stick to Terminal Sessions?

If you've read my book or my past columns, you've endured my repeated exhortations to keep the X Window System off of Internet-facing servers, or any other system on which it isn't needed, due to X's complexity and history of security vulnerabilities. So, why am I devoting an entire column to graphical remote system administration?

Don't worry. I haven't gone soft-hearted (though possibly slightly soft-headed); I stand by that advice. But, there are plenty of contexts in which you may need to administer or view things remotely besides hardened servers in Internet-facing DMZ networks.

And, not all people who need to run remote applications in those non-Internet-DMZ scenarios are advanced Linux users. Should they be forbidden from doing what they need to do until they've mastered using the vi editor and writing bash scripts? Especially given that it *is* possible to mitigate some of the risks associated with the X Window System and VNC?

Of course they shouldn't! Although I do encourage all Linux newcomers to embrace the command line. The day may come when Linux is a truly graphically oriented operating system like Mac OS, but for now, pretty much the entire OS is driven by configuration files in /etc (and in users' home directories), and that's unlikely to change soon.

Originally a project of the Olivetti Research Laboratory (which was subsequently acquired by Oracle and then AT&T before being shut down), VNC uses a protocol called Remote Frame Buffer (RFB). The original creators of VNC now maintain the application suite RealVNC, which is available in free and commercial versions, but TightVNC, UltraVNC and GNOME's vino VNC server and vinagre VNC client also are popular.

VNC's strengths are its simplicity, ubiquity and portability—it runs on many different operating systems. Because it runs over a single TCP port (usually TCP 5900), it's also firewall-friendly and easy to tunnel.

Its security, however, is somewhat problematic. VNC authentication uses a DES-based transaction that, if eavesdropped-on, is vulnerable to optimized brute-force (password-guessing) attacks. This vulnerability is exacerbated by the fact that many versions of VNC support only eight-character passwords.

Furthermore, VNC session data usually is transmitted unencrypted. Only a couple flavors of VNC support TLS encryption of RFB streams, and it isn't clear how securely TLS has been implemented even in those. Thus, an attacker using a trivially hacked VNC client may be able to reconstruct and view eavesdropped VNC streams.

Luckily, as it operates over a single TCP port, VNC is easy to tunnel through SSH, through Virtual Private Network (VPN) sessions and through TLS/SSL wrappers, such as stunnel. Let's look at a simple VNC-over-SSH example.

Tunneling VNC over SSH

To tunnel VNC over SSH, your remote system must be running an SSH dæmon (sshd) and a VNC server application. Your local system must have an SSH client (ssh) and a VNC client application.

Our example remote system, remotebox, already is running sshd. Suppose it also has vino, which is also known as the GNOME Remote Desktop Preferences applet (on Ubuntu systems, it's located in the System menu's Preferences section). First, presumably from remotebox's local console, you need to open that applet and enable remote desktop sharing. Figure 2 shows vino's General settings.

If you want to view only this system's remote desktop without controlling it, uncheck Allow other users to control your desktop. If you don't want to have to confirm remote connections explicitly (for example, because you want to connect to this system when it's unattended), you can uncheck the Ask you for confirmation box. Any time you leave yourself logged in to an unattended system, be sure to use a password-protected screensaver!

vino is limited in this way. Because vino is loaded only after you log in, you can use it only to connect to a fully logged-in GNOME session in progress—not, for example, to a gdm (GNOME login) prompt. Unlike vino, other versions of VNC can be invoked as needed from xinetd or inetd. That technique is out of the scope of this article, but see Resources for a link to a how-to for doing so in Ubuntu, or simply Google the string "vnc xinetd".

Continuing with our vino example, don't close that Remote Desktop Preferences applet yet. Be sure to check the Require the user to enter this password box and to select a difficult-to-guess password. Remember, vino runs in an *already-logged-in* desktop session, so unless you set a password here, you'll run the risk of allowing *completely unauthenticated* sessions (depending on whether a password-protected screensaver is running).

Figure 2. General Settings in GNOME Remote Desktop Preferences (vino)

Figure 3. Advanced Settings in GNOME Remote Desktop Preferences (vino)

If your remote system will be run logged in but unattended, you probably will want to uncheck Ask you for confirmation. Again, don't forget that locked screensaver.

We're not done setting up vino on remotebox yet. Figure 3 shows the Advanced Settings tab.

Several advanced settings are particularly noteworthy. First, you should check Only allow local connections, after which remote connections still *will* be possible, but only via a port-forwarder like SSH or stunnel. Second, you may want to set a custom TCP port for vino to listen on via the Use an alternative port option. In this example, I've chosen 20226. This is an instance of useful security-through-obscurity; if our other

On the Web, Articles Talk!

Every couple weeks over at LinuxJournal.com, our Gadget Guy Shawn Powers posts a video. They are fun, silly, quirky and sometimes even useful. So, whether he's reviewing a new product or showing how to use some Linux software, be sure to swing over to the Web site and check out the latest video: **www.linuxjournal.com/video**.

We'll see you there, or more precisely, vice versa!

(more meaningful) security controls fail, at least we won't be running VNC on the obvious default port.

Also, you should check the box next to Lock screen on disconnect, but you probably should *not* check Require encryption, as SSH will provide our session encryption, and adding redundant (and not-necessarily-known-good) encryption will only increase vino's already-significant latency. If you think there's only a moderate level of risk of eavesdropping in the environment in which you want to use vino—for example, on a small, private, local-area network inaccessible from the Internet—vino's TLS implementation may be good enough for you. In that case, you may opt to check the Require encryption box and skip the SSH tunneling.

(If any of you know more about TLS in vino than I was able to divine from the Internet, please write in. During my research for this article, I found no documentation or on-line discussions of vino's TLS design details whatsoever—beyond people commenting that the soundness of TLS in vino is unknown.)

This month, I seem to be offering you more "opt-out" opportunities in my examples than usual. Perhaps I'm becoming less dogmatic with age. Regardless, let's assume you've followed my advice to forego vino's encryption in favor of SSH.

At this point, you're done with the server-side settings. You won't have to change those again. If you restart your GNOME session on remotebox, vino will restart as well, with the options you just set. The following steps, however, are necessary each time you want to initiate a VNC/SSH session.

On mylaptop (the system from which you want to connect to remotebox), open a terminal window, and type this command:

```
mick@mylaptop:~$ ssh -L 20226:remotebox:20226 admin-slave@remotebox
```

OpenSSH's -L option sets up a local port-forwarder. In this example, connections to mylaptop's TCP port 20226 will be forwarded to the same port on remotebox. The syntax for this option is "-L [localport]:[remote IP or hostname]:[remoteport]". You can use any available local TCP port you like (higher than 1024, unless you're running SSH as root), but the remote port must correspond to the alternative port you set vino to listen on (20226 in our example), or if you didn't set an alternative port, it should be VNC's default of 5900.

That's it for the SSH session. You'll be prompted for a password and then given a bash prompt on remotebox. But, you won't need it except to enter `exit` when your VNC session is finished. It's time to fire up your VNC client.

vino's companion client, vinagre (also known as the GNOME Remote Desktop Viewer) is good enough for our purposes here. On Ubuntu systems, it's located in the Applications menu in the Internet section. In Figure 4, I've opened the Remote Desktop Viewer and clicked the Connect button. As you can see, rather than remotebox, I've entered localhost as the hostname. I've also entered port number 20226.

When I click the Connect button, vinagre connects to mylaptop's local TCP port 20226, which actually is being listened on by my local SSH process. SSH forwards this connection attempt through its encrypted connection to TCP 20226 on remotebox, which is being listened on by remotebox's vino process.

At this point, I'm prompted for remotebox's vino password (shown in Figure 2). On successful authentication, I'll have full access to my active desktop session on remotebox.

To end the session, I close the Remote Desktop Viewer's session, and then enter `exit` in my SSH session to remotebox—that's all there is to it.

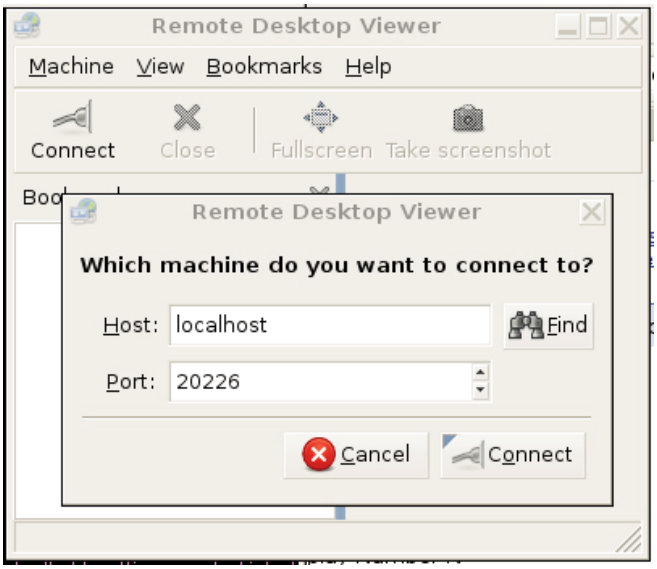


Figure 4. Using vinagre to Connect to an SSH-Forwarded Local Port

This technique is easy to adapt to other versions of VNC servers and clients, and probably also to other versions of SSH—there are commercial alternatives to OpenSSH, including Windows versions. I mentioned that VNC client applications are available for a wide variety of platforms; on any such platform, you can use this technique, so long as you also have an SSH client that supports port forwarding.

Conclusion

Thus ends my crash course on how to control graphical applications over networks. I hope my examples of both techniques, SSH with X forwarding and VNC over SSH, help you get started with whatever particular distribution and software packages you prefer. Be safe!■

Mick Bauer (darth.elmo@wiremonkeys.org) is Network Security Architect for one of the US's largest banks. He is the author of the O'Reilly book *Linux Server Security*, 2nd edition (formerly called *Building Secure Servers With Linux*), an occasional presenter at information security conferences and composer of the "Network Engineering Polka".

Resources

The Cygwin/X (information about Cygwin's free X server for Microsoft Windows): **x.cygwin.com**.

Tichondrius' HOWTO for setting up VNC with resumable sessions—Ubuntu-centric, but mostly applicable to other distributions: **ubuntuforums.org/showthread.php?t=122402**.

Wikipedia's VNC article, which may be helpful in making sense of the different flavors of VNC: **en.wikipedia.org/wiki/Vnc**.

Billix: a Sysadmin's Swiss Army Knife

Turn that spare USB stick into a sysadmin's dream with Billix.

BILL CHILDERS

Does anyone remember Linuxcare? Founded in 1998, Linuxcare was a company that provided support services for Linux users in corporate environments. I remember seeing Linuxcare at the first ever LinuxWorld conference in San Jose, and the thing I took away from that LinuxWorld was the Linuxcare Bootable Business Card (BBC). The BBC was a 50MB cut-down Linux distribution that fit on a business-card-size compact disc. I used that distribution to recover and repair quite a few machines, until the advent of Knoppix. I always loved the portability of that little CD though, and I missed it greatly until I stumbled across Damn Small Linux (DSL) one day.

After reading through the DSL Web site, I discovered that it was possible to run DSL off of a bootable USB key, and that old love for the Bootable Business Card was rekindled in a new way. It wasn't until I had a conversation with fellow sysadmin Kyle Rankin about the PXE boot environment he'd implemented, that I realized it might be possible to set up a USB key to do more than merely boot a recovery environment. Before long, I had added the CentOS and Ubuntu netinstalls to my little USB key. Not long after that, I was mentioning this

It wasn't until I had a conversation with fellow sysadmin Kyle Rankin about the PXE boot environment he'd implemented, that I realized it might be possible to set up a USB key to do more than merely boot a recovery environment.

in my favorite IRC channel, and one of the fellows in there suggested I put the code on SourceForge and call it Billix. I'd had a couple beers by then and thought it sounded like a great idea. In that instant, Billix was born.

Billix is an aggregation of many different tools that can be useful to system administrators, all compressed down to fit within a 256MB bootable USB thumbdrive. The 256MB size is not an arbitrary number; rather, it was chosen because USB thumbdrives are very inexpensive at that size (many companies now give them away as advertising gimmicks). This allows me to have many Billix keys lying around, just waiting to be used. Because the keys are cheap or free, I don't feel bad about leaving one in a server for a day or two. If your USB drive is larger than 256MB, you still can use it for its designed purpose—storing files. Billix doesn't hamper normal use of the USB drive in any way. There also is an ISO distribution of Billix if you want to burn a CD of it, but I feel it's not nearly as convenient as having it on a USB key.

The current Billix distribution (0.21 at the time of this writing) includes the following tools:

- Damn Small Linux 4.2.5
- Ubuntu 8.04 LTS netinstall
- Ubuntu 7.10 netinstall
- Ubuntu 6.06 LTS netinstall
- Fedora 8 netinstall
- CentOS 5.1 netinstall
- CentOS 4.6 netinstall
- Debian Etch netinstall
- Debian Sarge netinstall
- Memtest86 memory-checking utility
- Ntpwd Windows password changing utility
- DBAN disk wiper utility

So, with one USB key, a system administrator can recover or repair a machine, install one of eight different Linux distributions, test the memory in a system, get into a Windows machine with a lost password or wipe the disks of a machine before repurpose or disposal. In order to install any of the netinstall-based Linux distributions, a working Internet connection with DHCP is required, as the netinstall downloads the installation bits for each distribution on the fly from Internet-based mirrors.

Hopefully, you're excited to check out Billix. You simply can download the ISO version and burn it to a CD to get started, but the full utility of Billix really shines when you install it on a USB disk. Before you install it on a USB disk, you need to meet the following prerequisites:

- 256MB or greater USB drive with FAT- or FAT32-based filesystem.
- Internet connection with DHCP (for netinstalls only, not required for DSL, Windows password removal or disk wiping with DBAN).
- install-mbr (part of the mbr package on Ubuntu or Debian, needed for some USB drives).
- syslinux (from the syslinux package on Ubuntu or Debian, required to create the bootsector on the USB drive).
- Your system must be capable of booting from USB devices (most have this ability if they're made after 2005).

To install the USB-based version of Billix, first check your drive. If that drive has the U3 Windows software on it, you may want to remove it to unlock all of the drive's capacity (see the Resources section for U3 removal utilities, which are typically Windows-based). Next, if your USB drive has data on it, *back up the data*. I cannot stress this enough. You will be making adjustments to the partition table of the USB drive, so backing up any data that already is on the key is *critical*. Download the latest version of Billix from the Sourceforge.net project page to your computer. Once the download is complete, untar the contents of the tarball to the root directory of your USB drive.

Now that the contents of the tarball are on your USB drive, you need to install a Master Boot Record (MBR) on the drive and set a bootsector on the drive. The Master Boot Record needs to be set up on the USB drive first. Issue a `install-mbr -p1 <device>` (where `<device>` is your USB drive, such as /dev/sdb). *Warning:* make sure that you get the device of the USB drive correct, or you run the risk of messing up the MBR on your system's boot device. The -p1 option tells install-mbr to set the first partition as active (that's the one that will contain the bootsector).

Next, the bootsector needs to be installed within the first partition. Run `syslinux -s <device/partition>` (where `<device/partition>` is the device and partition of the USB drive, such as /dev/sdb1). *Warning:* much like installing the MBR, installing the bootsector can be a dangerous operation if you run it on the wrong device, so take care and double-check your command line before pressing the Enter key.

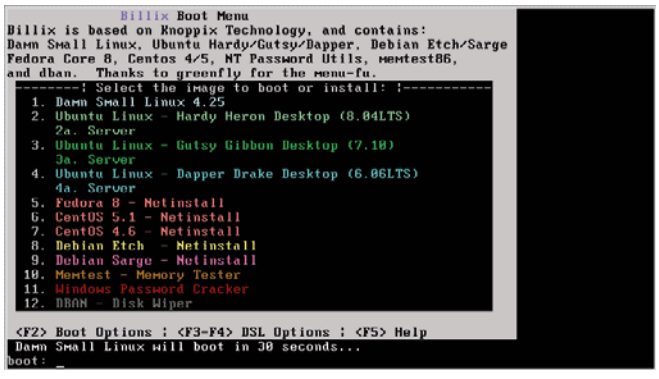


Figure 1. The Billix Boot Menu

Troubleshooting Billix

A few things can go wrong when converting a USB key to run Billix (or any USB-based distribution). The most common issue is for the USB drive to fail to boot the system. This can be due to several things. Older systems often split USB disk support into USB-Floppy emulation and USB-HDD emulation. For Billix to work on these systems, USB-HDD needs to be enabled. If your drive came with the U3 Windows-based software vault, this typically needs to be disabled or removed prior to installing Billix.

If you're seeing "MBR123" or something similar in the upper-left corner, but the system is hanging, you have a misconfigured MBR. Try `install-mbr` again, and make sure to use the -p1 switch. You will need to run syslinux again after running install-mbr. If all else fails, you probably need to wipe the USB drive and begin again. Back up the data on the USB drive, then use fdisk to build a new partition table (make sure to set it as FAT or FAT32). Use mkfs.vfat (with the -F 32 switch if it's a FAT32 filesystem) to build a new blank filesystem, untar the tarball again, and run install-mbr and syslinux on the newly defined filesystem.

At this point, your USB drive can be unmounted safely, and you can test it out by booting from the USB drive. Once your system successfully boots from the USB drive, you should see a menu similar to the one shown in Figure 1. Simply choose the number for what you want to boot, run or install, and that distribution will spring into action. If you don't select a number, Damn Small Linux will boot automatically after 30 seconds.

Damn Small Linux is a miniature version of Knoppix (it actually has much of the automatic hardware-detection routines of Knoppix in it). As such, it makes an excellent rescue environment, or it can be used as a quick "trusted desktop" in the event you need to "borrow" a friend's computer to do something. I have used DSL in the past to commandeer a system temporarily at a cybercafé, so I could log in to work and fix a sick server. I've even used DSL to boot and mount a corrupted Windows filesystem, and I was able to save some of the data. DSL is fairly full-featured for its size, and it comes with two window managers (JWM or Fluxbox). It can be configured to save its data back to the USB disk in a persistent fashion, so you always can be sure you have your critical files with you and that it's easily accessible.

All the Linux distribution installations have one thing in common: they are all network-based installs. Although this is a good thing for Billix, as they take up very little space (around 10MB for each distro), it can be a bad thing during installation as the installation time will vary with the speed of your Internet connection. There is one other upside to a network-based installation. In many cases, there is no need to update

Expanding Billix

It's relatively easy to expand Billix to support other Linux distributions, such as Knoppix or the Ubuntu live CDs. Copy the contents of the Billix USB tarball to a directory on your hard disk, and download the distro you want. Copy the necessary kernel and initrd to the directory where you put the contents of the USB tarball, taking care to rename any files if there are files in that directory with the same name. Copy any compressed filesystems that your new distro may use to the USB drive (for example, Knoppix has the KNOPPIX directory, and Puppy Linux uses PUP_XXX.SFS). Then, look at the boot configuration for that distro (it should be in isolinux.cfg). Take the necessary lines out of that file, and put them in the Billix syslinux.cfg file, changing filenames as necessary. Optionally, you can add a menu item to the boot.msg file. Finally, run `syslinux -s <device>`, and reboot your system to test out your newly expanded Billix.

I have a 2GB USB drive that has a "Super-Billix" installation that includes Knoppix and Ubuntu 8.04. An added bonus of having the entire Ubuntu live CD in your pocket is that, thanks to the speed of USB 2.0, you can install Ubuntu in less than ten minutes, which would be really useful at an installfest. There is good information on creating Ubuntu-bootable USB drives available at the Pendrive Linux Web site.

Alternatively, a really neat thing to do (but way beyond the scope of this article) is to convert Billix into a network-boot (via Pre-Execution Environment, or PXE) environment. I've actually got a VMware virtual machine running Billix as a PXE boot server.

the newly installed operating system after installation, because the OS bits that are downloaded are typically up to date. Note that when using the Red Hat-based installers (CentOS 4.6, CentOS 5.1 and Fedora 8), the system may appear to hang during the download of a file called minstg2.img. The system probably isn't hanging; it's just downloading that file, which is

Billix is an aggregation of many different tools that can be useful to system administrators, all compressed down to fit within a 256MB bootable USB thumbdrive.

fairly large (around 40MB), so it can take a while depending on the speed of the mirror and the speed of your connection. Take care not to specify the USB disk accidentally as the install target for the distribution you are attempting to install.

The memtest86 utility has been around for quite a few years, yet it's a key tool for a sysadmin when faced with a flaky computer. It does only one thing, but it does it very well: it tests the RAM of a system very thoroughly. Simply boot off the USB drive, select memtest from the menu, and press Enter, and memtest86 will load and begin testing the RAM of the system immediately. At this point, you can remove the USB drive from the computer. It's no longer needed as memtest86 is very small and loads completely into memory on startup.

The ntpwd Windows password "cracking" tool can be a controversial tool, but it is included in the Billix distribution because as a system administrator, I've been asked countless times to get into Windows systems (or accounts on Windows systems) where the password has been lost or forgotten. The ntpwd utility can be a bit daunting, as the UI is text-based and nearly nonexistent, but it does a good job of mounting FAT32- or NTFS-based partitions, editing the SAM account database and saving those changes. Be sure to read all the messages that ntpwd displays, and take care to select the proper disk partition to edit. Also, take the program's advice and nullify a password rather than trying to change it from within the interface—zeroing the password works much more reliably.

DBAN (otherwise known as Darik's Boot and Nuke) is a very good "nuke it from orbit" hard disk wiper. It provides various levels of wipe, from a basic "overwrite the disk with zeros" to a full DoD-certified, multipass wipe. Like memtest86, DBAN is small and loads completely into memory, so you can boot the utility, remove the USB drive, start a wipe and move on to another system. I've used to this to wipe clean disks on systems before handing them over to a recycler or before selling a system.

In closing, Billix may not make you coffee in the morning or eradicate Windows from the face of the earth, but having a USB key in your pocket that offers you the functionality to do all of those tasks quickly and easily can make the life of a system administrator (or any Linux-oriented person) much easier.■

Bill Childers is an IT Manager in Silicon Valley, where he lives with his wife and two children. He enjoys Linux far too much, and probably should get more sun from time to time. In his spare time, he does work with the Gilroy Garlic Festival, but he does not smell like garlic.

Resources

Billix Project Page: **sourceforge.net/projects/billix**

Damn Small Linux: **www.damnsmalllinux.org**

DBAN Project Page: **dban.sourceforge.net**

Knoppix: **www.knoppix.net**

Pendrive Linux: **www.pendrivelinux.com**

Syslinux: **syslinux.zytor.com/index.php**

Pxelinux: **syslinux.zytor.com/pxe.php**

U3 Removal Software: **www.u3.com/uninstall**

Zenoss and the Art of Network Monitoring

If a server goes down, do you want to hear it?

JERAMIAH BOWLING

If a tree falls in the woods and no one is there to hear it, does it make a sound? This the classic query designed to place your mind into the Zen-like state known as the silent mind. Whether or not you want to hear a tree fall, if you run a network, you probably want to hear a server when it goes down. Many organizations utilize the long-established Simple Network Management Protocol (SNMP) as a way to monitor their networks proactively and listen for things going down.

At a rudimentary level, SNMP requires only two items to work: a management server and a managed device (or devices). The management server pulls status and health information at regular intervals from the managed devices and stores the information in a table. Managed devices use local SNMP agents to notify the management server when defined behavior occurs (such as errors or "traps"), which are stored in the same table on the server. The result is an accurate, real-time reporting mechanism for outages. However, SNMP as a protocol does not stipulate how the data in these tables is to be presented and managed for the end user. That's where a promising new open-source network-monitoring software called Zenoss (pronounced Zeen-ohss) comes in.

Available for most Linux distributions, Zenoss builds on the basic operation of SNMP and uses a comprehensive interface to manage even the largest and most diverse environment. The Core version of Zenoss used in this article is freely available under the GPLv2. An Enterprise version also is available with additional features and support. In this article, we install Zenoss on a CentOS 5.1 system to observe its usefulness in a network-monitoring role. From there, we create a simulated multisystem server network using the following systems: a Fedora-based Postfix e-mail server, an Ubuntu server running Apache and a Windows server running File and Print services. To conserve space, only the CentOS installation is discussed in detail here. For the managed systems, only SNMP installation and configuration are covered.

Building the Zenoss Server

Begin by selecting your hardware. Zenoss lacks specific hardware requirements, but it relies heavily MySQL, so you can use MySQL requirements as a rough guideline. I recommend using the fastest processor available, 1GB of memory, fast enough hard disks to provide acceptable MySQL performance and Gigabit Ethernet for the network. I ran several test configurations, and this configuration seemed adequate enough for a medium-size network (100+ nodes/devices). To keep configuration simple, all firewalls and SELinux instances were disabled in the test environment. If you use firewalls in your environment, open ports 161 (SNMP), 8080 (Zenoss Management Page) and 514 (if you integrate syslog with Zenoss).

Install CentOS 5.1 on the server using your own preferences. I used a bare install with no X Window System or desktop manager. Assign a static IP address and any other pertinent network information (DNS servers and so forth). After the OS install is complete, install the following packages using the yum command below:

```
yum install mysql mysql-server net-snmp net-snmp-utils gmp httpd
```

If the mysqld or the httpd service has not started after yum installs it, start it and set it to run for your configured runlevel. Next, download the latest Zenoss Core .rpm from Sourceforge.net (2.1.3 at the time of this writing), and install it using rpm from the command line. To start all the Zenoss-related dæmons after the .rpm has been installed, type the following at a command prompt:

```
service zenoss start
```

Launch a Web browser from any machine, and type the IP address of the Zenoss server using port 8080 (for example, http://192.168.142.6:8080). Log in to the site using the default account admin with a password of zenoss. This brings up the main dashboard. The dashboard is a compartmentalized view of the state of your managed devices. If you don't like the default display, you can arrange your dashboard any way you want using the various drop-down lists on the portlets (windows). I recommend setting the Production States portlet to display Production, so we can see our test systems after they are added.

Almost everything related to managed devices in Zenoss revolves around classes. With classes, you can create an infinite number of systems, processes or service classifications to monitor. To begin adding devices, we need to set our SNMP community strings at the top-level /Devices class. SNMP community strings are like passphrases used to authenticate traffic between devices. If one device wants to communicate with another, they must have matching community names/strings. In many deployments, administrators use the default community name of public (and/or private), which creates a security risk. I recommend changing these strings and making them into a short phrase. You can add numbers and characters to make the community name more complex to guess/crack, but I find phrases easier to remember.

Click on the Devices link on the navigation menu on the left, so that /Devices is listed near the top of the page. Click on the zProperties tab and scroll down. Enter an SNMP

Available for most Linux distributions, Zenoss builds on the basic operation of SNMP and uses a comprehensive interface to manage even the largest and most diverse environment.

community string in the zSNMPCommunitiy field. For our test environment, I used the string whatsourclearanceclarence. You can use different strings with different subclasses of systems or individual systems, but by setting it at the /Devices class, it will be used for any subclasses unless it is overridden. You also could list multiple strings in the zSNMPCommunities under the /Devices class, which allows you to define multiple strings for the discovery process discussed later. Make sure your community string (zSNMPCommunity) is in this list.

Installing Net-SNMP on Linux Clients

Now, let's set up our Linux systems so they can talk to the Zenoss server. After installing and configuring the operating systems on our other Linux servers, install the Net-SNMP package on each using the following command on the Ubuntu server:

```
sudo apt-get install snmpd
```

And, on the Fedora server use:

```
yum install net-snmp
```

Once the Net-SNMP packages are installed, edit out any other lines in the Access Control sections at the beginning of the /etc/snmp/snmpd.conf, and add the following lines:

```
##      sec.name  source       community
com2sec local     localhost    whatsourclearanceclarence
com2sec mynetwork 192.168.142.0/24    whatsourclearanceclarence

##  group.name       sec.model   sec.name
group MyROGroup      v1          local
group MyROGroup      v1          mynetwork
group MyROGroup      v2c         local
group MyROGroup      v2c         mynetwork

##          incl/excl subtree  mask
view all    included  .1         80

##              context sec.model sec.level prefix  read   write  notif
access MyROGroup ""      any       noauth  exact   all    none   none
```

Do not edit out any lines beneath the last Access Control Sections. Please note that the above is only a mildly restrictive configuration. Consult the snmpd.conf file or the Net-SNMP documentation if you want to tighten access. On the Ubuntu server, you also may have to change the following line in the /etc/snmp/default file to allow SNMP to bind to anything other than the local loopback address:

```
SNMPDOPTS='-Lsd -Lf /dev/null -u snmp -I -smux -p /var/run/snmpd.pid'
```

Installing SNMP on Windows

On the Windows server, access the Add/Remove Programs utility from the Control Panel. Click on the Add/Remove Windows Components button on the left. Scroll down the list of Components, check off Management and Monitoring Tools, and click on the Details button. Check Simple Network Management Protocol in the list, and click OK to install. Close the Add/Remove window, and go into the Services console from Administrative Tools in the Control Panel. Find the SNMP service in the list, right-click on it, and click on Properties to bring up the service properties tabs. Click on the Traps tab, and type in the community name. In the list of Trap Destinations, add the IP address of the Zenoss server. Now, click on the Security tab, and check off the Send authentication trap box, enter the community name, and give it READ-ONLY rights. Click OK, and restart the service.

Return to the Zenoss management Web page. Click the Devices link to go into the subclass of /Devices/Servers/Windows, and on the zProperties tab, enter the name of a domain admin account and password in the zWinUser and zWinPassword fields. This account gives Zenoss access to the Windows Management Instrumentation (WMI) on your Windows systems. Make sure to click Save at the bottom of the page before navigating away.

Adding Devices into Zenoss

Now that our systems have SNMP, we can add them into Zenoss. Devices can be added individually or by scanning the network. Let's do both. To add our Ubuntu server into Zenoss, click on the Add Device link under the Management navigation section. Enter the IP address of the server and the community name. Under Device Class Path, set the selection to /Server/Linux. You could add a variety of other hardware, software and Zenoss information on this page before adding a system, but at a minimum, an IP address name and community name is required (Figure 1). Click the Add Device button, and the discovery process runs. When the results are displayed, click on the link to the new device to access it.

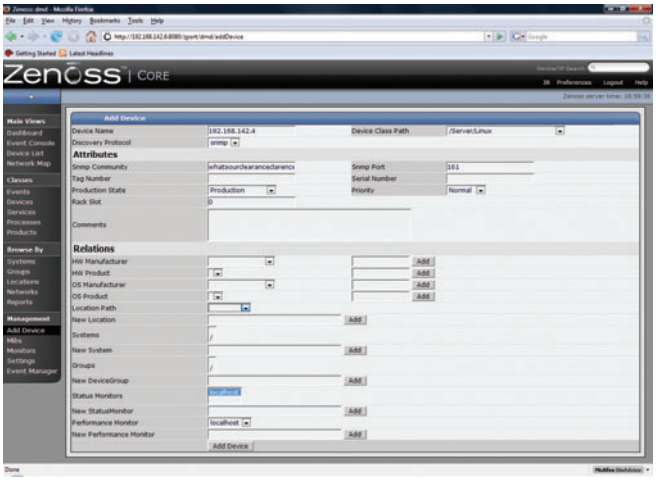


Figure 1. Adding a Device into Zenoss

To scan the network for devices, click the Networks link under the Browse By section of the navigation menu. If your network is not in the list, add it using CIDR notation. Once added, check the box next to your network and use the drop-down arrow to click on the Select Discover Devices option. You will see a similar results page as the one from before. When complete, click on the links at the bottom of the results page to access the new devices. Any device found will be placed in the /Discovered class. Because we should have discovered the Fedora server and the Windows server, they should be moved to the /Devices/Servers/Linux and /Devices/Servers/Windows classes, respectively. This can be done from each server's Status tab by using the main drop-down list and selecting Manage→Change Class.

If all has gone well, so far we have a functional SNMP monitoring system that is able to monitor heartbeat/availability (Figure 2) and performance information (Figure 3) on our systems. You can customize other various Status and Performance Monitors to meet your needs, but here we will use the default localhost monitors.

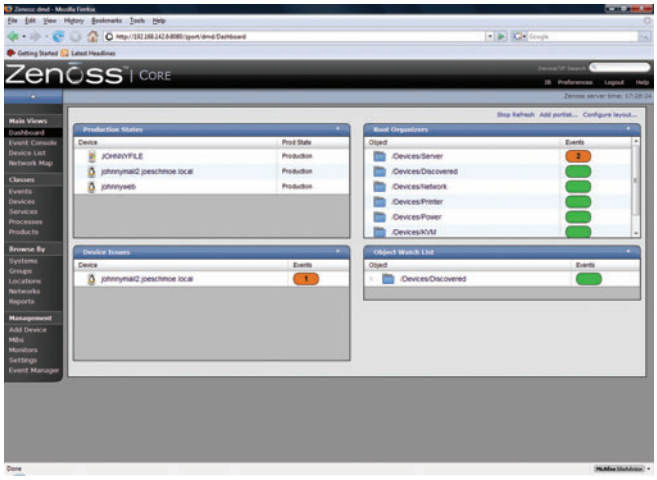


Figure 2. The Zenoss Dashboard

Figure 3. Performance data is collected almost immediately after discovery.

Creating Users and Setting E-Mail Alerts

At this point, we can use the dashboard to monitor the managed devices, but we will be notified only if we visit the site. It would be much more helpful if we could receive alerts via e-mail. To set up e-mail alerting, we need to create a separate user account, as alerts do not work under the admin account. Click on the Setting link under the Management navigation section. Using the drop-down arrow on the menu, select Add User. Enter a user name and e-mail address when prompted. Click on the new user in the list to edit its properties. Enter a password for the new account, and assign a role of Manager. Click Save at the bottom of the page. Log out of Zenoss, and log back in with the new account. Bring the settings page back up, and enter your SMTP server information. After setting up SMTP, we need to create an Alerting Rule for our new user. Click on the Users tab, and click on the account just created in the list. From the resulting page, click on the Edit tab and enter the e-mail address to which you want alerts sent. Now, go to the Alerting Rules tab and create a new rule using the drop-down arrow. On the edit tab of the new Alerting Rule, change the Action to email, Enabled to True, and change the Severity formula to >= Warning (Figure 4). Click Save.

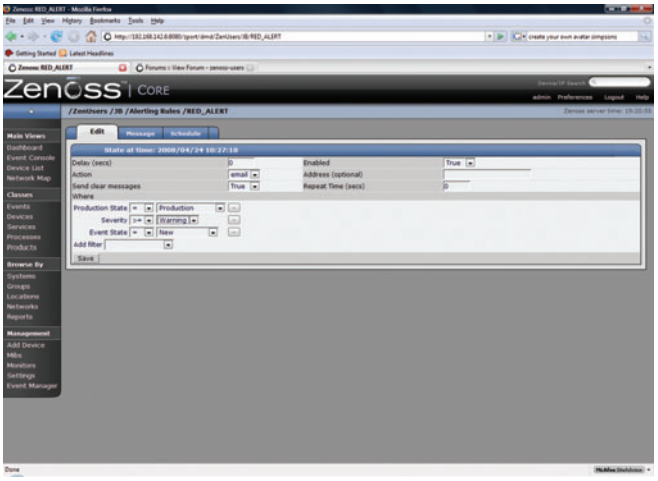


Figure 4. Creating an Alert Rule

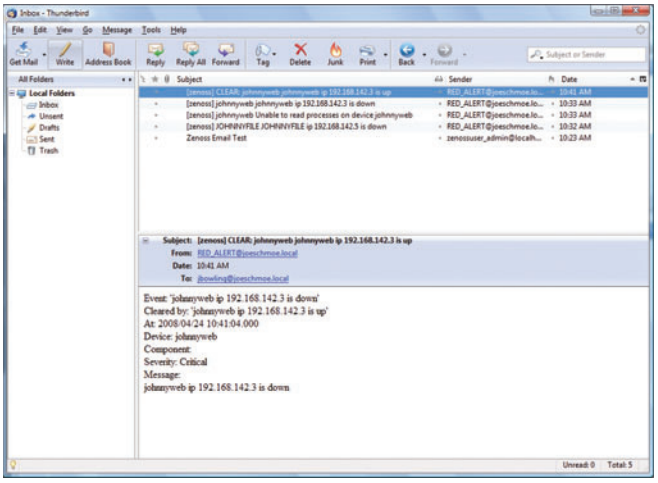


Figure 5. Zenoss alerts are sent fresh to your mailbox.

The above rule sends alerts when any Production server experiences an event rated Warning or higher (Figure 5). Using a filter, you can create any number of rules and have them apply only to specific devices or groups of devices. If you want to limit your alerts by time to working hours, for example, use the Schedule tab on the Alerting Rule to define a window. If no schedule is specified (the default), the rule runs all the time. In our rule, only one user will be notified. You also can create groups of users from the Settings page, so that multiple people are alerted, or you could use a group e-mail address in your user properties.

Services and Processes

We can expand our view of the test systems by adding a process and a service for Zenoss to monitor. When we refer to a process in Zenoss, we mean an active program, usually a dæmon, running on a managed device. Zenoss uses regular expressions to monitor processes.

To monitor Postfix on the mail server, first, let's define it as a process. Navigate to the Processes page under the Classes section of the navigation menu. Use the drop-down arrow next to OS Processes, and click Add Process. Enter Postfix as the process ID. When you return to the previous page, click on the link to the new process. On the edit tab of the process, enter `master` in the Regex field. Click Save before navigating away. Go to the zProperties tab of the process, and make sure the zMonitor field is set to True. Click Save again. Navigate back to the mail server from the dashboard, and on the OS tab, use the topmost menu's drop-down arrow to select Add→Add OSProcess. After the process has been added, we will be alerted if the Postfix process degrades or fails. While still on the OS tab of the

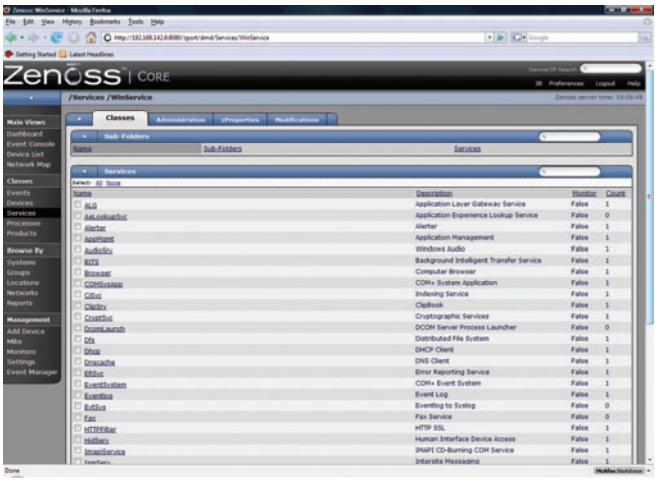


Figure 6. Zenoss comes with a plethora of predefined Windows services to monitor.

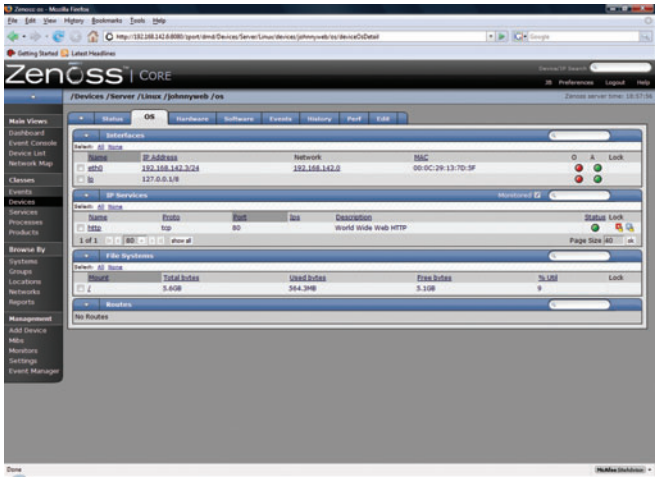


Figure 7. Monitoring HTTP as an IPService

Figure 8. Zenoss automatically maps your network for you.

server, place a check mark next to the new Postifx process, and from the OS Processes drop-down menu, select Lock OSProcess. On the next set of options, select Lock from deletion. This protects the process from being overwritten if Zenoss remodels the server.

Services in Zenoss are defined by active network ports instead of running dæmons. There are a plethora of services built in to the software, and you can define your own if you want to. The built-in services are broken down into two categories: IPServices and WinServices. IPservices use any port from 1-65535 and include common network apps/protocols, such as SMTP (Port 25), DNS (53) and HTTP (80). WinServices are intended for specific use with Windows servers (Figure 6).

Adding a service is much simpler than adding a process, because there are so many predefined in Zenoss. To monitor the HTTP service on our Web server, navigate to the server from the dashboard. Use the main menu's drop-down arrow on the server's OS tab arrow, and select Add→Add IPService. Type `HTTP` in the Service Class Field. Notice that the field begins to prefill with matches as you type the letters. Select TCP as the protocol, and click OK. Click Save on the resulting page. As with the OSProcess procedure, return to the OS tab of the server and lock the new IPService. Zenoss is now monitoring HTTP availability on the server (Figure 7).

Only the Beginning

There are a multitude of other features in Zenoss that space here prevents covering, including Network Maps (Figure 8), a Google Maps API for multilocation monitoring (Figure 9) and Zenpacks that provide additional monitoring and performance-capturing capabilities for common applications.

In the span of this article, we have deployed an enterprise-grade monitoring solution with relative ease. Although it's surprisingly easy to deploy, Zenoss also possesses a deep feature set. It easily rivals, if not surpasses, commercial competitors in the same product space. It is easy to manage, highly customizable and supported by a vibrant community.

Although you may not achieve the silent mind as long as you work with networks, with Zenoss, at least you will be able to sleep at night knowing you will hear things when they go down. Hopefully, they won't be trees.■

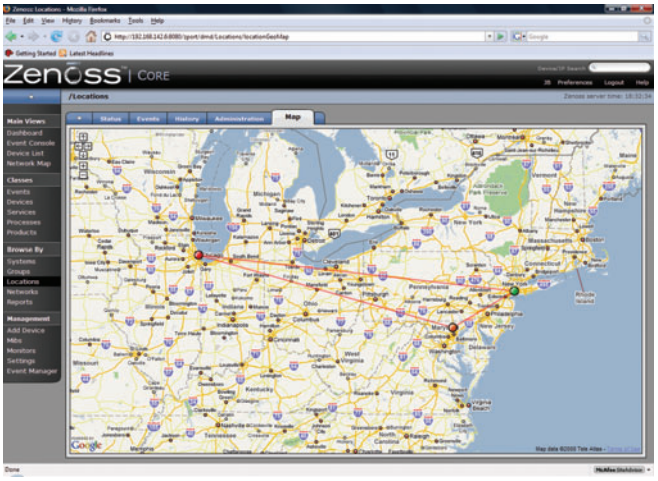


Figure 9. Multiple sites can be monitored geographically with the Google Maps API.

Jeramiah Bowling has been a systems administrator and network engineer for more than ten years. He works for a regional accounting and auditing firm in Hunt Valley, Maryland, and holds numerous industry certifications, including the CISSP. Your comments are welcome at jb50c@yahoo.com.

Resources

Zenoss: **www.zenoss.com**

Zenoss SourceForge Downloads Page: **sourceforge.net/project/showfiles.php?group_id=163126**

NET-SNMP: **net-snmp.sourceforge.net**

CentOS: **www.centos.org**

CentOS 5 Mirrors: **isoredirect.centos.org/centos/5/isos/i386**

Thin Clients Booting over a Wireless Bridge

How quickly can thin clients boot over a wireless bridge, and how far apart can they really be?

RONAN SKEHILL, ALAN DUNNE AND JOHN NELSON

In the 1970s and 1980s, the ubiquitous model of corporate and academic computing was that of many users logging in remotely to a single server to use a sliver of its precious processing time. With the cost of semiconductors holding fast to Moore's Law in the subsequent decades, however, the next advances in computing saw desktop computing become the standard as it became more affordable.

Although the technology behind thin clients is not revolutionary, their popularity has been on the increase recently. For many institutions that rely on older, donated hardware, thin-client networks are the only feasible way to provide users with access to relatively new software. Their use also has flourished in the corporate context. Thin-client networks provide cost-savings, ease network administration and pose fewer security implications when the time comes to dispose of them. Several computer manufacturers have leaped to stake their claim on this expanding market: Dell and HP Compaq, among others, now offer thin-client solutions to business clients.

And, of course, thin clients have a large following of hobbyists and enthusiasts, who have used their size and flexibility to great effect in countless home-brew projects. Software projects, such as the Etherboot Project and the Linux Terminal Server Project, have large and active communities and provide excellent support to those looking to experiment with diskless workstations.

Connecting the thin clients to a server always has been done using Ethernet; however, things are changing. Wireless technologies, such as Wi-Fi (IEEE 802.11), have evolved tremendously and now can start to provide an alternative means of connecting clients to servers. Furthermore, wireless equipment enjoys world-wide acceptance, and compatible products are readily available and very cheap.

In this article, we give a short description of the setup of a thin-client network, as well as some of the tools we found to be useful in its operation and administration. We also describe a test scenario we set up, involving a thin-client network that spanned a wireless bridge.

What Is a Thin Client?

A thin client is a computer with no local hard drive, which loads its operating system at boot time from a boot server. It is designed to process data independently, but relies solely on its server for administration, applications and non-volatile storage.

Following the client's BIOS sequence, most machines with network-boot capability will initiate a Preboot EXecution Environment (PXE), which will pass system control to the local network adapter. Figure 1 illustrates the traffic profile of the boot process and the various different stages, which are numbered 1 to 5. The network card broadcasts a DHCPDISCOVER packet with special flags set, indicating that the sender is trying to locate a valid boot server. A local PXE server will reply with a list of valid boot servers. The client then chooses a server, requests the name of the Linux kernel file from the server and initiates its transfer using Trivial File Transfer Protocol (TFTP; stage 1). The client then loads and executes the Linux kernel as normal (stage 2). A custom init program is then run, which searches for a network card and uses DHCP to identify itself on the network. Using Sun Microsystems' Network File System (NFS), the thin client then mounts a directory tree located on the PXE server as its own root filesystem (stage 3). Once the client has a non-volatile root filesystem, it continues to load the rest of its operating system environment (stage 4)—for example, it can mount a local filesystem and create a ramdisk to store local copies of temporary files. The fifth stage in the boot process is the initiation of the X Window System. This transfers the keystrokes from the thin client to the server to be processed. The server in return sends the graphical output to be displayed by the user interface system (usually KDE or GNOME) on the thin client.

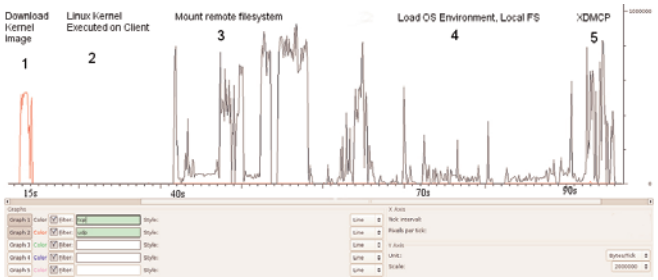


Figure 1. LTSP Traffic Profile and Boot Sequence

The X Display Manager Control Protocol (XDMCP) provides a layer of abstraction between the hardware in a system and the output shown to the user. This allows the user to be physically removed from the hardware by, in this case, a Local Area Network. When the X Window System is run on the thin client, it contacts the PXE server. This means the user logs in to the thin client to get a session on the server.

In conventional fat-client environments, if a client opens a large file from a network server, it must be transferred to the client over the network. If the client saves the file, the file must be transmitted over the network again. In the case of wireless networks, where bandwidth is limited, fat client networks are highly inefficient. On the other hand, with a

thin-client network, if the user modifies the large file, only mouse movement, keystrokes and screen updates are transmitted to and from the thin client. This is a highly efficient means, and other examples, such as ICA or NX, can consume as little as 5kbps bandwidth. This level of traffic is suitable for transmitting over wireless links.

How to Set Up a Thin-Client Network with a Wireless Bridge

One of the requirements for a thin client is that it has a PXE-bootable system. Normally, PXE is part of your network card BIOS, but if your card doesn't support it, you can get an ISO image of Etherboot with PXE support from ROM-o-matic (see Resources). Looking at the server with, for example, ten clients, it should have plenty of hard disk space (100GB), plenty of RAM (at least 1GB) and a modern CPU (such as an AMD64 3200).

The following is a five-step how-to guide on setting up an Edubuntu thin-client network over a fixed network.

1. Prepare the server. In our network, we used the standard standalone configuration. From the command line:

```
sudo apt-get install ltsp-server-standalone
```

You may need to edit /etc/ltsp/dhcpd.conf if you change the default IP range for the clients. By default, it's configured for a server at 192.168.0.1 serving PXE clients.

Our network wasn't behind a firewall, but if yours is, you need to open TFTP, NFS and DHCP. To do this, edit /etc/hosts.allow, and limit access for portmap, rpc.mountd, rpc.statd and in.tftpd to the local network:

```
portmap: 192.168.0.0/24
rpc.mountd: 192.168.0.0/24
rpc.statd:  192.168.0.0/24
in.tftpd:  192.168.0.0/24
```

Restart all the services by executing the following commands:

```
sudo invoke-rc.d nfs-kernel-server restart
sudo invoke-rc.d nfs-common restart
sudo invoke-rc.d portmap restart
```

2. Build the client's runtime environment. While connected to the Internet, issue the command:

```
sudo ltsp-build-client
```

If you're not connected to the Internet and have Edubuntu on CD, use:

```
sudo ltsp-build-client --mirror file:///cdrom
```

Remember to copy sources.list from the server into the chroot.

3. Configure your SSH keys. To configure your SSH server and keys, do the following:

```
sudo apt-get install openssh-server
sudo ltsp-update-sshkeys
```

4. Start DHCP. You should now be ready to start your DHCP server:

```
sudo invoke-rc.d dhcp3-server start
```

If all is going well, you should be ready to start your thin client.

5. Boot the thin client. Make sure the client is connected to the same network as your server.

Power on the client, and if all goes well, you should see a nice XDMCP graphical login dialog.

Once the thin-client network was up and running correctly, we added a wireless bridge into our network. In our network, a number of thin clients are located on a single hub, which is separated from the boot server by an IEEE 802.11 wireless bridge. It's not an unrealistic scenario; a situation such as this may arise in a corporate setting or university. For example, if a group of thin clients is located in a different or temporary building that does not have access to the main network, a simple and elegant solution would be to have a wireless link between the clients and the server. Here is a mini-guide in getting the bridge running so that the clients can boot over the bridge:

- Connect the server to the LAN port of the access point. Using this LAN connection, access the Web configuration interface of the access point, and configure it to broadcast an SSID on a unique channel. Ensure that it is in Infrastructure mode (not ad hoc mode). Save these settings and disconnect the server from the access point, leaving it powered on.

- Now, connect the server to the wireless node. Using its Web interface, connect to the wireless network advertised by the access point. Again, make sure the node connects to the access point in Infrastructure mode.

- Finally, connect the thin client to the access point. If there are several thin clients connected to a single hub, connect the access point to this hub.

We found ad hoc mode unsuitable for two reasons. First, most wireless devices limit ad hoc connection speeds to 11Mbps, which would put the network under severe strain to boot even one client. Second, while in ad hoc mode, the wireless nodes we were using would assume the Media Access Control (MAC) address of the computer that last accessed its Web interface (using Ethernet) as its own Wireless LAN MAC. This made the nodes suitable for connecting a single computer to a wireless network, but not for bridging traffic destined to more than one machine. This detail was found only after much sleuthing and led to a range of sporadic and often unreproducible errors in our system.

The wireless devices will form an Open Systems Interconnection (OSI) layer 2 bridge between the server and the thin clients. In other words, all packets received by the wireless devices on their Ethernet interfaces will be forwarded over the wireless network and retransmitted on the Ethernet

adapter of the other wireless device. The bridge is transparent to both the clients and the server; neither has any knowledge that the bridge is in place.

For administration of the thin clients and network, we used the Webmin program. Webmin comprises a Web front end and a number of CGI scripts, which directly update system configuration files. As it is Web-based, administration can be performed from any part of the network by simply using a Web browser to log in to the server. The graphical interface greatly simplifies tasks, such as adding and removing thin clients from the network or changing the location of the image file to be transferred at boot time. The alternative is to edit several configuration files by hand and restart all dæmon programs manually.

Evaluating the Performance of a Thin-Client Network

The boot process of a thin client is network-intensive, but once the operating system has been transferred, there is little traffic between the client and the server. As the time required to boot a thin client is a good indicator of the overall usability of the network, this is the metric we used in all our tests.

Our testbed consisted of a 3GHz Pentium 4 with 1GB of RAM as the PXE server. We chose Edubuntu 5.10 for our server, as this (and all newer versions of Edubuntu) come with LTSP included. We used six identical thin clients: 500MHz Pentium III machines with 512MB of RAM—plenty of processing power for our purposes.

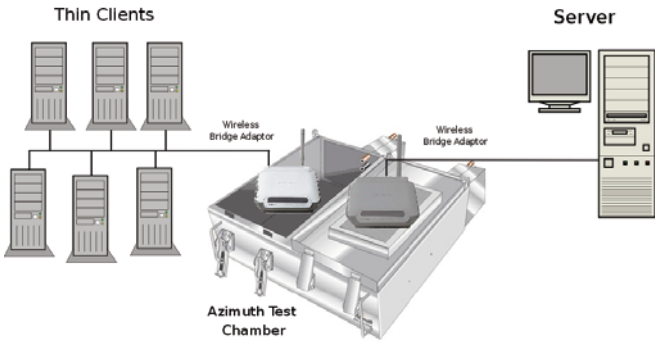


Figure 2. Six thin clients are connected to a hub, and in turn, this is connected to wireless bridge device. On the other side of the bridge is the server. Both wireless devices are placed in the Azimuth chamber.

When performing our tests, it was important that the results obtained were free from any external influence. A large part of this was making sure that the wireless bridge was not affected by any other wireless networks, cordless phones operating at 2.4GHz, microwaves or any other sources of Radio Frequency (RF) interference. To this end, we used the Azimuth 301w Test Chamber to house the wireless devices (see Resources). This ensures that any variations in boot times are caused by random variables within the system itself.

The Azimuth is a test platform for system-level testing of 802.11 wireless networks. It holds two wireless devices (in our case, the devices making up our bridge) in separate chambers and provides an artificial medium between them, creating complete isolation from the external RF environment. The Azimuth can attenuate the medium between

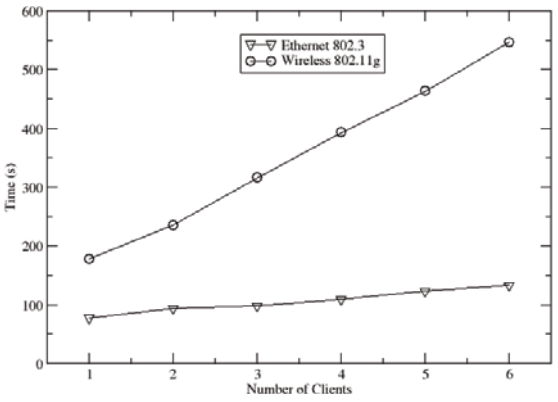


Figure 3. A Boot Time Comparison of Fixed and Wireless Networks with an Increasing Number of Thin Clients

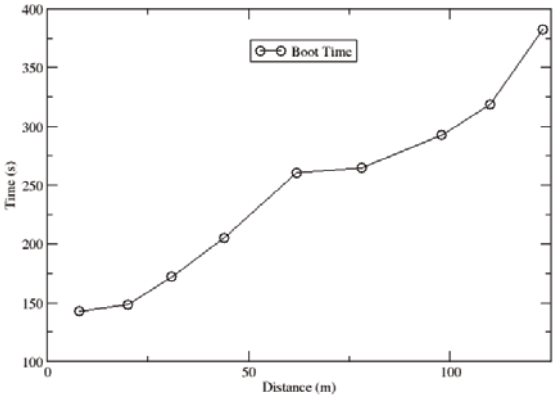


Figure 4. The Effect of the Bridge Length on Thin-Client Boot Time

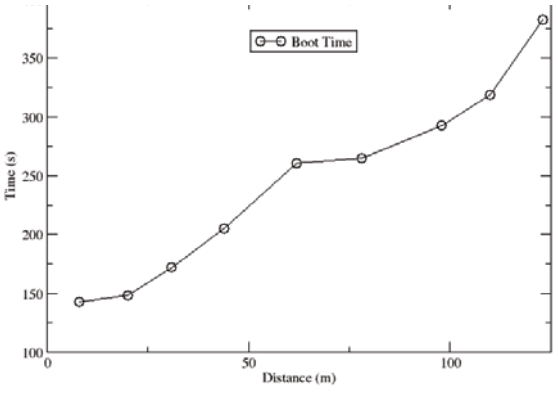


Figure 5. Boot Time in the Presence of Background Traffic

the wireless devices and can convert the attenuation in decibels to an approximate distance between them. This gives us the repeatability, which is a rare thing in wireless LAN evaluation. A graphic representation of our testbed is shown in Figure 2.

We tested the thin-client network extensively in three different scenarios: first, when multiple clients are booting simultaneously over the network; second, booting a single thin client over the network at varying distances, which are simulated by altering the attenuation introduced by the chamber;

and third, booting a single client when there is heavy background network traffic between the server and the other clients on the network.

Conclusion

As shown in Figure 3, a wired network is much more suitable for a thin-client network. The main limiting factor in using an 802.11g network is its lack of available bandwidth. Offering a maximum data rate of 54Mbps (and actual transfer speeds at less than half that), even an aging 100Mbps Ethernet easily outstrips 802.11g. When using an 802.11g bridge in a network such as this one, it is best to bear in mind its limitations. If your network contains multiple clients, try to stagger their boot procedures if possible.

Second, as shown in Figure 4, keep the bridge length to a minimum. With 802.11g technology, after a length of 25 meters, the boot time for a single client increases sharply, soon hitting the three-minute mark. Finally, our test shows, as illustrated in Figure 5, heavy background traffic (generated either by other clients booting or by external sources) also has a major influence on the clients' boot processes in a wireless environment. As the background traffic reaches 25% of our maximum throughput, the boot times begin to soar. Having pointed out the limitations with 802.11g, 802.11n is on the horizon, and it can offer data rates of 540Mbps, which means these limitations could soon cease to be an issue.

In the meantime, we can recommend a couple ways to speed up the boot process. First, strip out the unneeded services from the thin clients. Second, fix the delay of NFS mounting in klibc, and also try to start LDM as early as possible in the boot process, which means running it as the first service in rc2.d. If you do not need system logs, you can remove syslogd completely from the thin-client startup. Finally, it's worth remembering that after a client has fully booted, it requires very little bandwidth, and current wireless technology is more than capable of supporting a network of thin clients.

Acknowledgement

This work was supported by the National Communications Network Research Centre, a Science Foundation Ireland Project, under Grant 03/IN3/1396.■

Ronan Skehill works for the Wireless Access Research Centre at the University of Limerick, Ireland, as a Senior Researcher. The Centre focuses on everything wireless-related and has been growing steadily since its inception in 1999.

Alan Dunne conducted his final-year project with the Centre under the supervision of John Nelson. He graduated in 2007 with a degree in Computer Engineering and now works with Ericsson Ireland as a Network Integration Engineer.

John Nelson is a senior lecturer in the Department of Electronic and Computer Engineering at the University of Limerick. His interests include mobile and wireless communications, software engineering and ambient assisted living.

Resources

Linux Terminal Server Project (LTSP): **ltsp.sourceforge.net**

Ubuntu ThinClient How-To: **https://help.ubuntu.com/community/ThinClientHowto**

Azimuth WLAN Chamber: **www.azimuth.net**

ROM-o-matic: **rom-o-matic.net**

Etherboot: **www.etherboot.org**

Wireshark: **www.wireshark.org**

Webmin: **www.webmin.com**

Edubuntu: **www.edubuntu.org**

PXE Magic: Flexible Network Booting with Menus

Set up a PXE server and then add menus to boot kickstart images, rescue disks and diagnostic tools all from the network.

KYLE RANKIN

It's funny how automation evolves as system administrators manage larger numbers of servers. When you manage only a few servers, it's fine to pop in an install CD and set options manually. As the number of servers grows, you might realize it makes sense to set up a kickstart or FAI (Debian's Fully Automated Installer) environment to automate all that manual configuration at install time. Now, you boot the install CD, type in a few boot arguments to point the machine to the kickstart server, and go get a cup of coffee as the machine installs.

When the day comes that you have to install three or four machines at once, you either can burn extra CDs or investigate PXE boot. The Preboot eXecution Environment is an open standard developed by Intel to allow machines to boot over a network instead of from local media, such as a floppy, CD or hard drive. Modern servers and newer laptops and desktops with integrated NICs should support PXE booting in the BIOS—in some cases, it's enabled by default, and in other cases, you need to go into your BIOS settings to enable it.

Because many modern servers these days offer built-in remote power and remote terminals or otherwise are remotely accessible via serial console servers or networked KVM, if you have a PXE boot environment set up, you can power on remotely, then boot and install a machine from miles away.

When the day comes that you have to install three or four machines at once, you either can burn extra CDs or investigate PXE boot.

If you have never set up a PXE boot server before, the first part of this article covers the steps to get your first PXE server up and running. If PXE booting is old hat to you, skip ahead to the section called PXE Menu Magic. There, I cover how to configure boot menus when you PXE boot, so instead of hunting down MAC addresses and doing a lot of setup before an install, you simply can boot, select your OS, and you are off and running. After that, I discuss how to integrate rescue tools, such as Knoppix and memtest86+, into your PXE environment, so they are available to any machine that can boot from the network.

PXE Setup

You need three main pieces of infrastructure for a PXE setup: a DHCP server, a TFTP server and the syslinux software. Both DHCP and TFTP can reside on the same server. When a system attempts to boot from the network, the DHCP server gives it an IP address and then tells it the address for the TFTP server and the name of the bootstrap program to run. The TFTP server then serves that file, which in our case is a PXE-enabled syslinux binary. That program runs on the booted machine and then can load Linux kernels or other OS files that also are shared on the TFTP server over the network. Once the kernel is loaded, the OS starts as normal, and if you have configured a kickstart install correctly, the install begins.

Configure DHCP

Any relatively new DHCP server will support PXE booting, so if you don't already have a DHCP server set up, just use your distribution's DHCP server package (possibly named dhcpd, dhcp3-server or something similar). Configuring DHCP to suit your network is somewhat beyond the scope of this article, but many distributions ship a default configuration file that should provide a good place to start. Once the DHCP server is installed, edit the configuration file (often in /etc/dhcpd.conf), and locate the subnet section (or each host section if you configured static IP assignment via DHCP and want these hosts to PXE boot), and add two lines:

```
next-server ip_of_pxe_server;
filename "pxelinux.0";
```

The next-server directive tells the host the IP address of the TFTP server, and the filename directive tells it which file to download and execute from that server. Change the next-server argument to match the IP address of your TFTP server, and keep filename set to pxelinux.0, as that is the name of the syslinux PXE-enabled executable.

In the subnet section, you also need to add dynamic-bootp to the range directive. Here is an example subnet section after the changes:

```
subnet 10.0.0.0 netmask 255.255.255.0 {
    range dynamic-bootp 10.0.0.200 10.0.0.220;
    next-server 10.0.0.1;
    filename "pxelinux.0";
}
```

Install TFTP

After the DHCP server is configured and running, you are ready to install TFTP. The pxelinux executable requires a TFTP server that supports the tsize option, and two good choices are either tftpd-hpa or atftp. In many distributions, these options already are packaged under these names, so just install your distribution's package or otherwise follow the installation instructions from the project's official site.

Depending on your TFTP package, you might need to add an entry to /etc/inetd.conf if it wasn't already added for you:

```
tftp            dgram   udp     wait    root  /usr/sbin/in.tftpd
/usr/sbin/in.tftpd -s /var/lib/tftpboot
```

As you can see in this example, the -s option (used for tftpd-hpa) specified /var/lib/tftpboot as the directory to contain my files, but on some systems, these files are commonly stored in /tftpboot, so see your /etc/inetd.conf file and your tftpd man page and check on its conventions if you are unsure. If your distribution uses xinetd and doesn't create a file in /etc/xinetd.d for you, create a file called /etc/xinetd.d/tftp that contains the following:

```
# default: off
# description: The tftp server serves files using
# the trivial file transfer protocol.
# The tftp protocol is often used to boot diskless
# workstations, download configuration files to network-aware
# printers, and to start the installation process for
# some operating systems.
service tftp
{
        disable = no
        socket_type             = dgram
        protocol                = udp
        wait                    = yes
        user                    = root
        server                  = /usr/sbin/in.tftpd
        server_args             = -s /var/lib/tftpboot
        per_source              = 11
        cps                     = 100 2
        flags                   = IPv4
}
```

As tftpd is part of inetd or xinetd, you will not need to start any service. At most, you might need to reload inetd or xinetd; however, make sure that any software firewall you have running allows the TFTP port (port 69 udp) as input.

Add Syslinux

Now that TFTP is set up, all that is left to do is to install the syslinux package (available for most distributions, or you can follow the installation instructions from the project's main Web page), copy the supplied pxelinux.0 file to /var/lib/tftpboot (or your TFTP directory), and then create a /var/lib/tftpboot/pxelinux.cfg directory to hold pxelinux configuration files.

PXE Menu Magic

You can configure pxelinux with or without menus, and many administrators use pxelinux without them. There are compelling reasons to use pxelinux menus, which I discuss below, but first, here's how some pxelinux setups are configured.

You need three main pieces of infrastructure for a PXE setup: a DHCP server, a TFTP server and the syslinux software.

When many people configure pxelinux, they create configuration files for a machine or class of machines based on the fact that when pxelinux loads it searches the pxelinux.cfg directory on the TFTP server for configuration files in the following order:

- Files named 01-MACADDRESS with hyphens in between each hex pair. So, for a server with a MAC address of 88:99:AA:BB:CC:DD, a configuration file that would target only that machine would be named 01-88-99-aa-bb-cc-dd (and I've noticed it does matter that it is lowercase).
- Files named after the host's IP address in hex. Here, pxelinux will drop a digit from the end of the hex IP and try again as each file search fails. This is often used when an administrator buys a lot of the same brand of machine, which often will have very similar MAC addresses. The administrator then can configure DHCP to assign a certain IP range to those MAC addresses. Then, a boot option can be applied to all of that group.
- Finally, if no specific files can be found, pxelinux will look for a file named default and use it.

One nice feature of pxelinux is that it uses the same syntax as syslinux, so porting over a configuration from a CD, for instance, can start with the syslinux options and follow with your custom network options. Here is an example configuration for an old CentOS 3.6 kickstart:

```
default linux
label linux
    kernel vmlinuz-centos-3.6
    append text nofb load_ramdisk=1 initrd=initrd-centos-3.6.img
    ➥network ks=http://10.0.0.1/kickstart/centos3.cfg
```

Why Use Menus?

The standard sort of pxelinux setup works fine, and many administrators use it, but one of the annoying aspects of it is that even if you know you want to install, say, CentOS 3.6 on a server, you first have to get the MAC address. So, you either go to the machine and find a sticker that lists the MAC address, boot the machine into the BIOS to read the MAC, or let it get a lease on the network. Then, you need to create either a custom configuration file for that host's MAC or make sure its MAC is part of a group you already have configured. Depending on your infrastructure, this step can add substantial time to each server. Even if you buy servers in batches and group in IP ranges, what

happens if you want to install a different OS on one of the servers? You then have to go through the additional work of tracking down the MAC to set up an exclusion.

With pxelinux menus, I can preconfigure any of the different network boot scenarios I need and assign a number to them. Then, when a machine boots, I get an ASCII menu I can customize that lists all of these options and their number. Then, I can select the option I want, press Enter, and the install is off and running. Beyond that, now I have the option of adding non-kickstart images and can make them available to all of my servers, not just certain groups.

With pxelinux menus, I can preconfigure any of the different network boot scenarios I need and assign a number to them.

With this feature, you can make rescue tools like Knoppix and memtest86+ available to any machine on the network that can PXE boot. You even can set a timeout, like with boot CDs, that will select a default option. I use this to select my standard Knoppix rescue mode after 30 seconds.

Configure PXE Menus

Because pxelinux shares the syntax of syslinux, if you have any CDs that have fancy syslinux menus, you can refer to them for examples. Because you want to make this available to all hosts, move any more specific configuration files out of pxelinux.cfg, and create a file named default. When the pxelinux program fails to find any more specific files, it then will load this configuration. Here is a sample menu configuration with two options: the first boots Knoppix over the network, and the second boots a CentOS 4.5 kickstart:

```
default 1
timeout 300
prompt 1
display f1.msg
F1 f1.msg
F2 f2.msg

label 1
    kernel vmlinuz-knx5.1.1
    append secure nfsdir=10.0.0.1:/mnt/knoppix/5.1.1
    ➥nodhcp lang=us ramdisk_size=100000 init=/etc/init
    ➥2 apm=power-off nomce vga=normal
    ➥initrd=miniroot-knx5.1.1.gz quiet BOOT_IMAGE=knoppix
label 2
    kernel vmlinuz-centos-4.5-64
    append text nofb ksdevice=eth0 load_ramdisk=1
    ➥initrd=initrd-centos-4.5-64.img network
    ➥ks=http://10.0.0.1/kickstart/centos4-64.cfg
```

Each of these options is documented in the syslinux man page, but I highlight a few here. The default option sets which label to boot when the timeout expires. The timeout is in tenths of a second, so in this example, the timeout is 30 seconds, after which it will boot using the options set under label 1. The display option lists a message if there are any to display by default, so if you want to display a fancy menu for these two options, you could create a file called f1.msg in /var/lib/tftpboot/ that contains something like:

```
----| Boot Options |-----
|                       |
| 1. Knoppix 5.1.1      |
| 2. CentOS 4.5 64 bit  |
|                       |
-------------------------

<F1> Main | <F2> Help
Default image will boot in 30 seconds...
```

Notice that I listed F1 and F2 in the menu. You can create multiple files that will be output to the screen when the user presses the function keys. This can be useful if you have more menu options than can fit on a single screen, or if you want to provide extra documentation at boot time (this is handy if you are like me and create custom boot arguments for your kickstart servers). In this example, I could create a /var/lib/tftpboot/f2.msg file and add a short help file.

Although this menu is rather basic, check out the syslinux configuration file and project page for examples of how to jazz it up with color and even custom graphics.

Extra Features: PXE Rescue Disk

One of my favorite features of a PXE server is the addition of a Knoppix rescue disk. Now, whenever I need to recover a machine, I don't need to hunt around for a disk, I can just boot the server off the network.

First, get a Knoppix disk. I use a Knoppix 5.1.1 CD for this example, but I've been successful with much older Knoppix CDs. Mount the CD-ROM, and then go to the boot/isolinux directory on the CD. Copy the miniroot.gz and vmlinuz files to your /var/lib/tftpboot directory, except rename them something distinct, such as miniroot-knx5.1.1.gz and vmlinuz-knx5.1.1, respectively. Now, edit your pxelinux.cfg/default file, and add lines like the one I used above in my example:

```
label 1
    kernel vmlinuz-knx5.1.1
    append secure nfsdir=10.0.0.1:/mnt/knoppix/5.1.1 nodhcp
    ➥lang=us ramdisk_size=100000 init=/etc/init 2
    ➥apm=power-off nomce vga=normal
    ➥initrd=miniroot-knx5.1.1.gz quiet BOOT_IMAGE=knoppix
```

Notice here that I labeled it 1, so if you already have a label with that name, you need to decide which of the two to rename. Also notice that this example references the renamed vmlinuz-knx5.1.1 and miniroot-knx5.1.1.gz files. If you named your files something else, be sure to change the names here as well. Because I am mostly dealing with servers, I added 2 after init=/etc/init on the append line, so it would boot into runlevel 2 (console-only mode). If you want to boot to a full graphical environment, remove 2

from the append line.

The final step might be the largest for you if you don't have an NFS server set up. For Knoppix to boot over the network, you have to have its CD contents shared on an NFS server. NFS server configuration is beyond the scope of this article, but in my example, I set up an NFS share on 10.0.0.1 at /mnt/knoppix/5.1.1. I then mounted my Knoppix CD and copied the full contents to that directory. Alternatively, you could mount a Knoppix CD or ISO directly to that directory. When the Knoppix kernel boots, it will then mount that NFS share and access the rest of the files it needs directly over the network.

Extra Features: Memtest86+

Another nice addition to a PXE environment is the memtest86+ program. This program does a thorough scan of a system's RAM and reports any errors. These days, some distributions even install it by default and make it available during the boot process because it is so useful. Compared to Knoppix, it is very simple to add memtest86+ to your PXE server, because it runs from a single bootable file. First, install your distribution's memtest86+ package (most make it available), or otherwise download it from the memtest86+ site. Then, copy the program binary to /var/lib/tftpboot/memtest. Finally, add a new label to your pxelinux.cfg/default file:

```
label 3
        kernel memtest
```

That's it. When you type 3 at the boot prompt, the memtest86+ program loads over the network and starts the scan.

Conclusion

There are a number of extra features beyond the ones I give here. For instance, a number of DOS boot floppy images, such as Peter Nordahl's NT Password and Registry Editor Boot Disk, can be added to a PXE environment. My own use of the pxelinux menu helps me streamline server kickstarts and makes it simple to kickstart many servers all at the same time. At boot time, I can not only indicate which OS to load, but also more specific options, such as the type of server (Web, database and so forth) to install, what hostname to use, and other very specific tweaks. Besides the benefit of no longer tracking down MAC addresses, you also can create a nice colorful user-friendly boot menu that can be documented, so it's simpler for new administrators to pick up. Finally, I've been able to customize Knoppix disks so that they do very specific things at boot, such as perform load tests or even set up a Webcam server—all from the network.■

Kyle Rankin is a Senior Systems Administrator in the San Francisco Bay Area and the author of a number of books, including *Knoppix Hacks* and *Ubuntu Hacks* for O'Reilly Media. He is currently the president of the North Bay Linux Users' Group.

Resources

tftp-hpa: **www.kernel.org/pub/software/network/tftp**

atftp: **ftp.mamalinux.com/pub/atftp**

Syslinux PXE Page: **syslinux.zytor.com/pxe.php**

Red Hat's Kickstart Guide: **www.redhat.com/docs/manuals/enterprise/RHEL-4-Manual/sysadmin-guide/ch-kickstart2.html**

Knoppix: **www.knoppix.org**

Memtest86+: **www.memtest.org**

Creating VPNs with IPsec and SSL/TLS

How to create IPsec and SSL/TLS tunnels in Linux.

RAMI ROSEN

VPN (Virtual Private Network) is a technology that provides secure communication through an insecure and untrusted network (like the Internet). Usually, it achieves this by authentication, encryption, compression and tunneling. Tunneling is a technique that encapsulates the packet header and data of one protocol inside the payload field of another protocol. This way, an encapsulated packet can traverse through networks it otherwise would not be capable of traversing.

Currently, the two most common techniques for creating VPNs are IPsec and SSL/TLS. In this article, I describe the features and characteristics of these two techniques and present two short examples of how to create IPsec and SSL/TLS tunnels in Linux and verify that the tunnels started correctly. I also provide a short comparison of these two techniques.

IPsec and Openswan

IPsec (IP security) provides encryption, authentication and compression at the network level. IPsec is actually a suite of protocols, developed by the IETF (Internet Engineering Task Force), which have existed for a long time. The first IPsec protocols were defined in 1995 (RFCs 1825–1829). Later, in 1998, these RFCs were depreciated by RFCs 2401–2412. IPsec implementation in the 2.6 Linux kernel was written by Dave Miller and Alexey Kuznetsov. It handles both IPv4 and IPv6. IPsec operates at layer 3, the network layer, in the OSI seven-layer networking model. IPsec is mandatory in IPv6 and optional in IPv4. To implement IPsec, two new protocols were added: Authentication Header (AH) and Encapsulating Security Payload (ESP). Handshaking and exchanging session keys are done with the Internet Key Exchange (IKE) protocol.

The AH protocol (RFC 2404) has protocol number 51, and it authenticates both the header and payload. The AH protocol does not use encryption, so it is almost never used.

ESP has protocol number 50. It enables us to add a security policy to the packet and encrypt it, though encryption is not mandatory. Encryption is done by the kernel, using the kernel CryptoAPI. When two machines are connected using the ESP protocol, a unique number identifies this connection; this number is called SPI (Security Parameter Index). Each packet that flows between these machines has a Sequence Number (SN), starting with 0. This SN is increased by one for each sent packet. Each packet also has a checksum, which is called the ICV (integrity check value) of the packet. This checksum is calculated using a secret key, which is known only to these two machines.

IPsec has two modes: transport mode and tunnel mode. When creating a VPN, we use tunnel mode. This means each IP packet is fully encapsulated in a newly created IPsec packet. The payload of this newly created IPsec packet is the original IP packet.

Figure 2 shows that a new IP header was added at the right, as a result of working with a tunnel, and that an ESP header also was added.

There is a problem when the endpoints (which are sometimes called peers) of the tunnel are behind a NAT (Network

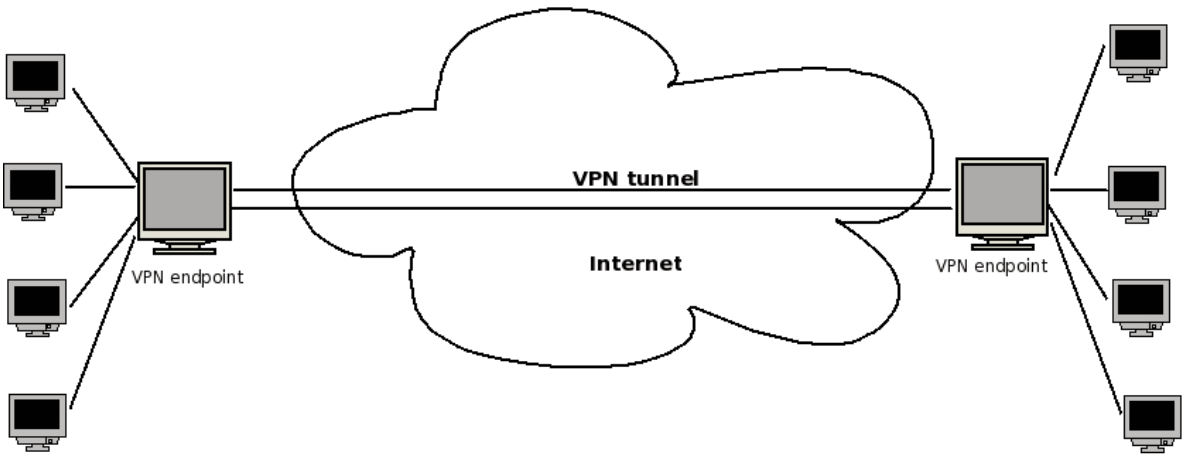


Figure 1. A Basic VPN Tunnel

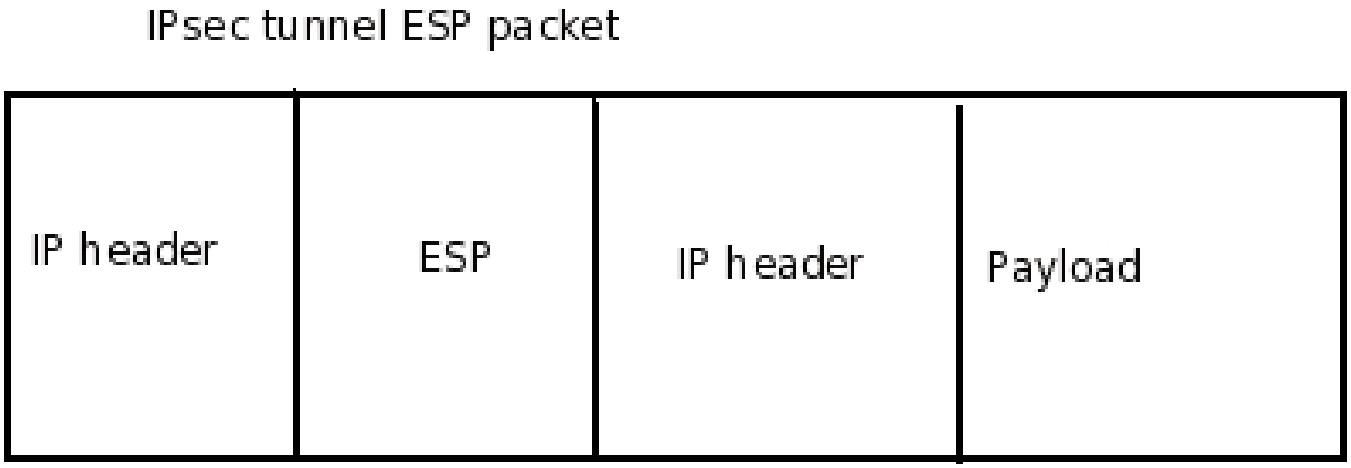


Figure 2. An IPsec Tunnel ESP Packet

Address Translation) device. Using NAT is a method of connecting multiple machines that have an "internal address", which are not accessible directly to the outside world. These machines access the outside world through a machine that does have an Internet address; the NAT is performed on this machine—usually a gateway.

When the endpoints of the tunnel are behind a NAT, the NAT modifies the contents of the IP packet. As a result, this packet will be rejected by the peer because the signature is wrong. Thus, the IETF issued some RFCs that try to find a solution for this problem. This solution commonly is known as NAT-T or NAT Traversal. NAT-T works by encapsulating IPsec packets in UDP packets, so that these packets will be able to pass through NAT routers without being dropped. RFC 3948, UDP Encapsulation of IPsec ESP Packets, deals with NAT-T (see Resources).

Openswan is an open-source project that provides an implementation of user tools for Linux IPsec. You can create a VPN using Openswan tools (shown in the short example below). The Openswan Project was started in 2003 by former FreeS/WAN developers. FreeS/WAN is the predecessor of Openswan. S/WAN stands for Secure Wide Area Network, which is actually a trademark of RSA. Openswan runs on many different platforms, including x86, x86_64, ia64, MIPS and ARM. It supports kernels 2.0, 2.2, 2.4 and 2.6.

Two IPsec kernel stacks are currently available: KLIPS and NETKEY. The Linux kernel NETKEY code is a rewrite from scratch of the KAME IPsec code. The KAME Project was a group effort of six companies in Japan to provide a free IPv6 and IPsec (for both IPv4 and IPv6) protocol stack implementation for variants of the BSD UNIX computer operating system.

KLIPS is not a part of the Linux kernel. When using KLIPS, you must apply a patch to the kernel to support NAT-T. When using NETKEY, NAT-T support is already inside the kernel, and there is no need to patch the kernel.

When you apply firewall (iptables) rules, KLIPS is the easier case, because with KLIPS, you can identify IPsec traffic, as this traffic goes through ipsecX interfaces. You apply iptables rules to these interfaces in the same way you apply rules to other network interfaces (such as eth0).

When using NETKEY, applying firewall (iptables) rules is much more complex, as the traffic does not flow through ipsecX interfaces; one solution can be marking the packets in the Linux kernel with iptables (with a setmark iptables rule). This mark is a member of the kernel socket buffer structure (struct sk_buff, from the Linux kernel networking code); decryption of the packet does not modify that mark.

Openswan supports Opportunistic Encryption (OE), which enables the creation of IPsec-based VPNs by advertising and fetching public keys from a DNS server.

OpenVPN

OpenVPN is an open-source project founded by James Yonan. It provides a VPN solution based on SSL/TLS. Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols that provide secure communications data transfer on the Internet. SSL has been in existence since the early '90s.

The OpenVPN networking model is based on TUN/TAP virtual devices; TUN/TAP is part of the Linux kernel. The first

When you apply firewall (iptables) rules, KLIPS is the easier case, because with KLIPS, you can identify IPsec traffic, as this traffic goes through ipsecX interfaces.

TUN driver in Linux was developed by Maxim Krasnyansky.

OpenVPN installation and configuration is simpler in comparison with IPsec. OpenVPN supports RSA authentication, Diffie-Hellman key agreement, HMAC-SHA1 integrity checks and more. When running in server mode, it supports multiple clients (up tp 128) to connect to a VPN server over the same port. You can set up your own Certificate Authority (CA) and generate certificates and keys for an OpenVPN server and multiple clients.

OpenVPN operates in user-space mode; this makes it easy to port OpenVPN to other operating systems.

Example: Setting Up a VPN Tunnel with IPsec and Openswan

First, download and install the ipsec-tools package and the Openswan package (most distros have these packages).

The VPN tunnel has two participants on its ends, called left and right, and which participant is considered left or right is arbitrary. You have to configure various parameters for these two ends in /etc/ipsec.conf (see man 5 ipsec.conf). The /etc/ipsec.conf file is divided into sections. The conn section contains a connection specification, defining a network connection to be made using IPsec.

An example of a conn section in /etc/ipsec.conf, which defines a tunnel between two nodes on the same LAN, with the left one as 192.168.0.89 and the right one as 192.168.0.92, is as follows:

```
...
conn linux-to-linux
        #
        # Simply use raw RSA keys
        # After starting openswan, run:
        # ipsec showhostkey --left (or --right)
        # and fill in the connection similarly
        # to the example below.
        left=192.168.0.89
        leftrsasigkey=0sAQPP...
        # The remote user.
        #
        right=192.168.0.92
        rightrsasigkey=0sAQON...
        type=tunnel
        auto=start
...
```

You can generate the leftrsasigkey and rightrsasigkey on both participants by running:

```
ipsec rsasigkey --verbose 2048 > rsa.key
```

Then, copy and paste the contents of rsa.key into /etc/ipsec.secrets.

In some cases, IPsec clients are roaming clients (with a random IP address). This happens typically when the client is a laptop used from remote locations (such clients are called Roadwarriors). In this case, use the following in ipsec.conf:

```
right=%any
```

instead of:

```
right=ipAddress
```

The %any keyword is used to specify an unknown IP address.

The type parameter of the connection in this example is tunnel (which is the default). Other types can be transport, signifying host-to-host transport mode; passthrough, signifying that no IPsec processing should be done at all; drop, signifying that packets should be discarded; and reject, signifying that packets should be discarded and a diagnostic ICMP should be returned.

The auto parameter of the connection tells which operation should be done automatically at IPsec startup. For example, auto=start tells it to load and initiate the connection; whereas auto=ignore (which is the default) signifies no automatic startup operation. Other values for the auto parameter can be add, manual or route.

After configuring /etc/ipsec.conf, start the service with:

```
service ipsec start
```

You can perform a series of checks to get info about IPsec on your machine by typing `ipsec verify`. And, output of `ipsec verify` might look like this:

```
Checking your system to see if IPsec has installed and started correctly:
Version check and ipsec on-path                                 [OK]
Linux Openswan U2.4.7/K2.6.21-rc7 (netkey)
Checking for IPsec support in kernel                            [OK]
NETKEY detected, testing for disabled ICMP send_redirects       [OK]
NETKEY detected, testing for disabled ICMP accept_redirects     [OK]
Checking for RSA private key (/etc/ipsec.d/hostkey.secrets)     [OK]
Checking that pluto is running                                  [OK]
Checking for 'ip' command                                       [OK]
Checking for 'iptables' command                                 [OK]
Opportunistic Encryption Support                                [DISABLED]
```

You can get information about the tunnel you created by running:

```
ipsec auto --status
```

You also can view various low-level IPSec messages in the kernel syslog.

You can test and verify that the packets flowing between the two participants are indeed esp frames by opening an FTP connection (for example), between the two participants and running:

```
tcpdump -f esp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 96 bytes
```

You should see something like this:

```
IP 192.168.0.92 > 192.168.0.89: ESP(spi=0xd514eed9,seq=0x7)
IP 192.168.0.89 > 192.168.0.92: ESP(spi=0x3a1563b9,seq=0x6)
IP 192.168.0.89 > 192.168.0.92: ESP(spi=0x3a1563b9,seq=0x7)
IP 192.168.0.92 > 192.168.0.89: ESP(spi=0xd514eed9,seq=0x8)
```

Note that the spi (Security Parameter Index) header is the same for all packets; this is an identifier of the connection.

If you need to support NAT traversal, add `nat_traversal=yes` in ipsec.conf; `nat_traversal=no` is the default.

The Linux IPsec stack can work with pluto from Openswan, racoon from the KAME Project (which is included in ipsec-tools) or isakmpd from OpenBSD.

Example: Setting Up a VPN Tunnel with OpenVPN

First, download and install the OpenVPN package (most distros have this package).

Then, create a shared key by doing the following:

```
openvpn --genkey --secret static.key
```

You can create this key on the server side or the client side, but you should copy this key to the other side in a secured channel (like SSH, for example). This key is exchanged between client and server when the tunnel is created.

This type of shared key is the simplest key; you also can use CA-based keys. The CA can be on a different machine from the OpenVPN server. The OpenVPN HOWTO provides more details on this (see Resources).

Then, create a server configuration file named server.conf:

```
dev tun
ifconfig 10.0.0.1 10.0.0.2
secret static.key
comp-lzo
```

On the client side, create the following configuration file named client.conf:

```
remote serverIpAddressOrHostName
dev tun
ifconfig 10.0.0.2 10.0.0.1
secret static.key
comp-lzo
```

Note that the order of IP addresses has changed in the client.conf configuration file.

The comp-lzo directive enables compression on the VPN link.

You can set the mtu of the tunnel by adding the tun-mtu directive. When using Ethernet bridging, you should use dev tap instead of dev tun.

The default port for the tunnel is UDP port 1194 (you can verify this by typing `netstat -nl | grep 1194` after starting the tunnel).

Before you start the VPN, make sure that the TUN interface (or TAP interface, in case you use Ethernet bridging) is not firewalled.

Start the vpn on the server by running `openvpn server.conf` and running `openvpn client.conf` on the client.

You will get output like this on the client:

```
OpenVPN 2.1_rc2 x86_64-redhat-linux-gnu [SSL] [LZO2] [EPOLL] built on
Mar  3 2007
IMPORTANT: OpenVPN's default port number is now 1194, based on an official
port number assignment by IANA. OpenVPN 2.0-beta16 and earlier used 5000
as
the default port.
LZO compression initialized
TUN/TAP device tun0 opened
/sbin/ip link set dev tun0 up mtu 1500
/sbin/ip addr add dev tun0 local 10.0.0.2 peer 10.0.0.1
UDPv4 link local (bound): [undef]:1194
UDPv4 link remote: 192.168.0.89:1194
Peer Connection Initiated with 192.168.0.89:1194
Initialization Sequence Completed
```

You can verify that the tunnel is up by pinging the server from the client (ping 10.0.0.1 from the client).

The TUN interface emulates a PPP (Point-to-Point) network device and the TAP emulates an Ethernet device. A user-space program can open a TUN device and can read or write to it. You can apply iptables rules to a TUN/TAP virtual device in the same way you would do it to an Ethernet device (such as eth0).

IPsec and OpenVPN—a Short Comparison

IPsec is considered the standard for VPN; many vendors (including Cisco, Nortel, CheckPoint and many more) manufacture devices with built-in IPsec functionalities, which enable them to connect to other IPsec clients.

However, we should be a bit cautious here: different manufacturers may implement IPsec in a noncompatible manner on their devices, which can pose a problem.

OpenVPN is not supported currently by most vendors.

IPsec is much more complex than OpenVPN and involves kernel code; this makes porting IPsec to other operating systems a much heavier task. It is much easier to port OpenVPN to other operating systems than IPsec, because OpenVPN runs entirely in user space and is not involved with kernel code.

Both IPsec and OpenVPN use HMAC (Hash Message Authentication Code) to authenticate packets.

OpenVPN is based on using the OpenSSL library; it can run over UDP (which is the default and preferred protocol) or TCP. As opposed to IPsec, which runs in kernel, it runs in user space, so it is heavier than IPsec in terms of performance.

Configuring and applying firewall (iptables) rules in OpenVPN is usually easier than configuring such rules with Openswan in an IPsec-based tunnel.

Acknowledgement

Thanks to Mr Ken Bantoft for his comments.■

Rami Rosen is a computer science graduate of Technion, the Israel Institute of Technology, located in Haifa. He works as a Linux and Open Solaris kernel programmer for a networking startup, and he can be reached at ramirose@gmail.com. In his spare time, he likes running, solving cryptic puzzles and helping everyone he knows move to this wonderful operating system, Linux.

Resources

OpenVPN: **openvpn.net**

OpenVPN 2.0 HOWTO: **openvpn.net/howto.html**

RFC 3948, UDP Encapsulation of IPsec ESP Packets: **tools.ietf.org/html/rfc3948**

Openswan: **www.openswan.org**

The KAME Project: **www.kame.net**

MySQL 5 Stored Procedures: Relic or Revolution?

Stored procedures bring the legacy advantages and challenges to MySQL. GUY HARRISON

Stored procedures (or stored routines, to use the official MySQL terminology) are programs that are both stored and executed within the database server. Stored procedures have been features in closed-source relational databases, such as Oracle, since the early 1990s. However, MySQL added stored procedure support only in the recent 5.0 release and, consequently, applications built on the LAMP stack don't generally incorporate stored procedures. So, this is an opportune time to consider whether stored procedures should be incorporated into your MySQL applications.

Stored Procedures in the Client-Server Era

Database stored programs first came to prominence in the late 1980s and early 1990s, during the client-server revolution. In the client-server applications of that time, stored programs had some obvious advantages:

- Client-server applications typically had to balance processing load carefully between the client PC and the (relatively) more powerful server machine. Using stored programs was one way to reduce the load on the client, which might otherwise be overloaded.
- Network bandwidth was often a serious constraint on client-server applications; execution of multiple server-side operations in a single stored program could reduce network traffic.
- Maintaining correct versions of client software in a client-server environment was often problematic. Centralizing at least some of the processing on the server allowed a greater measure of control over core logic.
- Stored programs offered clear security advantages because, in those days, application users typically connected directly to the database, rather than through a middle tier. As I discuss later in this article, stored procedures allow you to restrict the database account only to well-defined procedure calls, rather than allowing the account to execute any and all SQL statements.

With the emergence of three-tier architectures and Web applications, some of the incentives to use stored programs from within applications disappeared. Application clients are now often browser-based, security is predominantly handled by a middle tier, and the middle tier possesses the ability to encapsulate business logic. Most of the functions for which stored programs were used in client-server applications now can be implemented in middle-tier code (PHP, Java, C# and so on).

Nevertheless, many of the traditional advantages of stored procedures remain, so let's consider these advantages, and some disadvantages, in more depth.

Using Stored Procedures to Enhance Database Security

Stored procedures are subject to most of the security restrictions that apply to other database objects: tables, indexes, views and so forth. Specific permissions are required before a user can create a stored program, and, similarly, specific permissions are needed in order to execute a program.

What sets the stored program security model apart from that of other database objects—and from other programming languages—is that stored programs may execute with the permissions of the user who *created* the stored procedure, rather than those of the user who is *executing* the stored procedure. This model allows users to perform actions via a stored procedure that they would not be authorized to perform using normal SQL.

This facility, sometimes called definer rights security, allows us to tighten our database security, because we can ensure that a user gains access to tables only via stored program code that restricts the types of operations that can be performed on those tables and that can implement various business and data integrity rules. For instance, by establishing a stored program as the only mechanism available for certain table inserts or updates, we can ensure that all of these operations are logged, and we can prevent any invalid data entry from making its way into the table.

In the event that this application account is compromised (for instance, if the password is cracked), attackers still will be able to execute only our stored programs, as opposed to being able to run any ad hoc SQL. Although such a situation constitutes a severe security breach, at least we are assured that attackers will be subject to the same checks and logging as normal application users. They also will be denied the opportunity to retrieve information about the underlying database schema (because the ability to run standard SQL will be granted to the procedure, not the user), which will hinder attempts to perform further malicious activities.

Another security advantage inherent in stored programs is their resistance to SQL injection attacks. An SQL injection attack can occur when a malicious user manages to "inject" SQL code into the SQL code being constructed by the applica-

tion. Stored programs do not offer the only protection against SQL injection attacks, but applications that rely exclusively on stored programs to interact with the database are largely resistant to this type of attack (provided that those stored programs do not themselves build dynamic SQL strings without fully validating their inputs).

Data Abstraction

It is generally a good practice to separate your data access code from your business logic and presentation logic. Data access routines often are used by multiple program modules and are likely to be maintained by a separate group of developers. A very common scenario requires changes to the underlying data structures while minimizing the impact on higher-level logic. Data abstraction makes this much easier to accomplish.

The use of stored programs provides a convenient way of implementing a data access layer. By creating a set of stored programs that implement all of the data access routines required by the application, we are effectively building an API for the application to use for all database interactions.

Reducing Network Traffic

Stored programs can improve application performance radically by reducing network traffic in certain situations.

It's commonplace for an application to accept input from an end user, read some data in the database, decide what statement to execute next, retrieve a result, make a decision, execute some SQL and so on. If the application code is written entirely outside the database, each of these steps would require a network round trip between the database and the application. The time taken to perform these network trips easily can dominate overall user response time.

Consider a typical interaction between a bank customer and an ATM machine. The user requests a transfer of funds between two accounts. The application must retrieve the balance of each account from the database, check withdrawal limits and possibly other policy information, issue the relevant UPDATE statements, and finally issue a commit, all before advising the customer that the transaction has succeeded. Even for this relatively simple interaction, at least six separate database queries must be issued, each with its own network round trip between the application server and the database. Figure 1 shows the sequences of interactions that would be required without a stored program.

On the other hand, if a stored program is used to implement the fund transfer logic, only a single database interaction is required. The stored program takes responsibility for checking balances, withdrawal limits and so on. Figure 2 shows the reduction in network round trips that occurs as a result.

Network round trips also can become significant when an application is required to perform some kind of aggregate processing on very large record sets in the database. For instance, if the application needs to retrieve millions of rows in order to calculate some sort of business metric that cannot be computed easily using native SQL, such as average time to complete an order, a very large number of round trips can result. In such a case, the network delay again may become the dominant factor in application response time. Performing the calculations in a stored program will reduce network overhead, which might reduce overall response time, but you need to be sure to take into account the differences in raw computation speed, which I discuss later in this article.

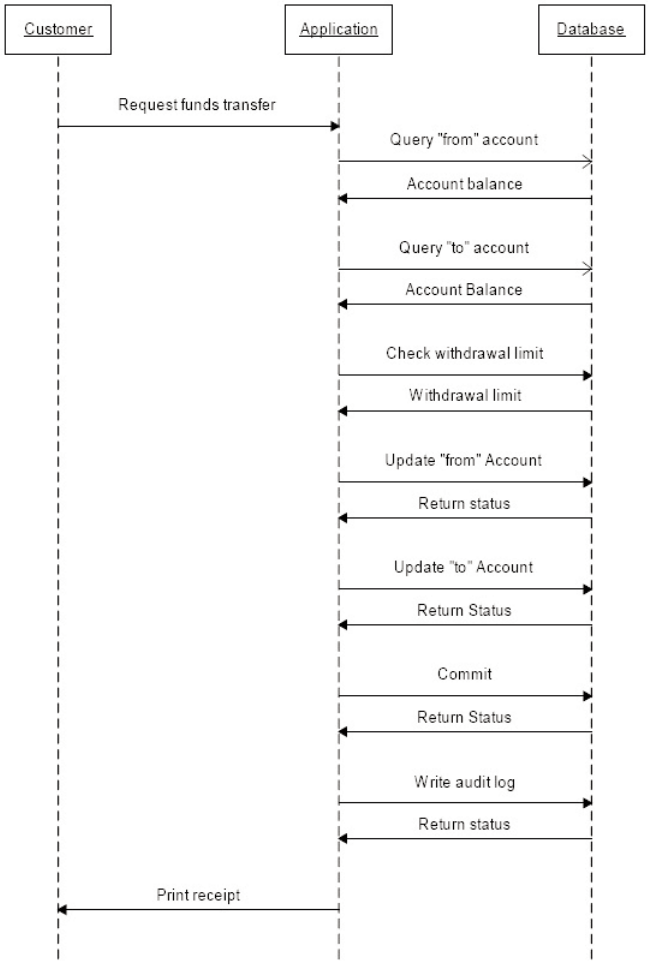


Figure 1. Network Round Trips without Stored Procedure

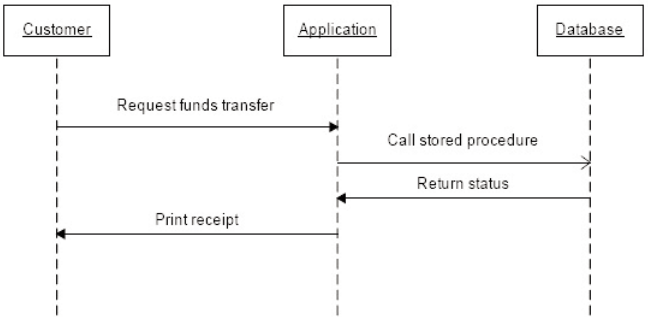


Figure 2. Network Round Trips with Stored Procedure

Creating Common Routines across Multiple Applications

Although it is commonplace for a MySQL database to be at the service of a single application, it is not at all uncommon for multiple applications to share a single database. These applications might run on different machines and be written in

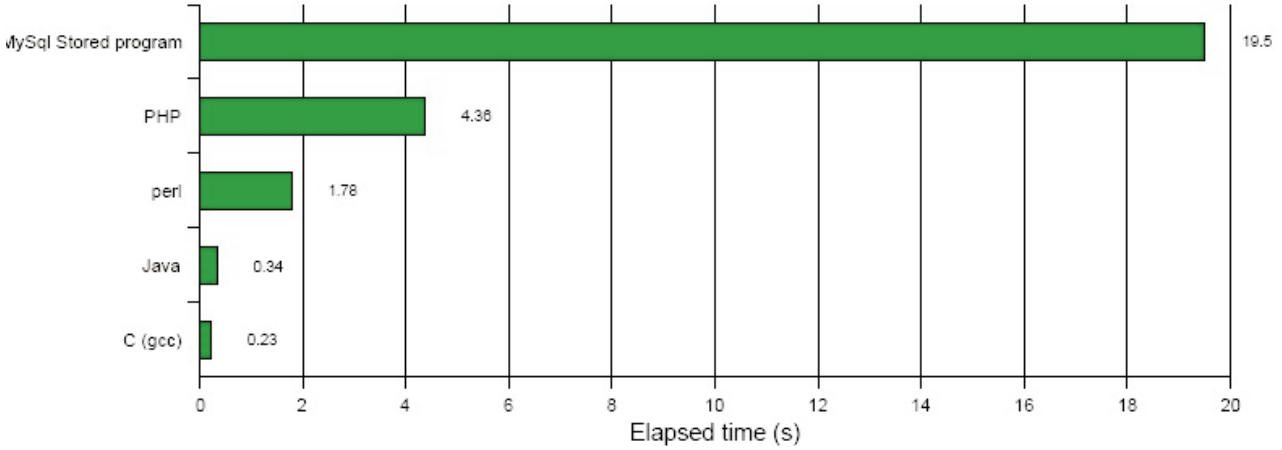


Figure 3. Stored procedures are a poor choice for number crunching.

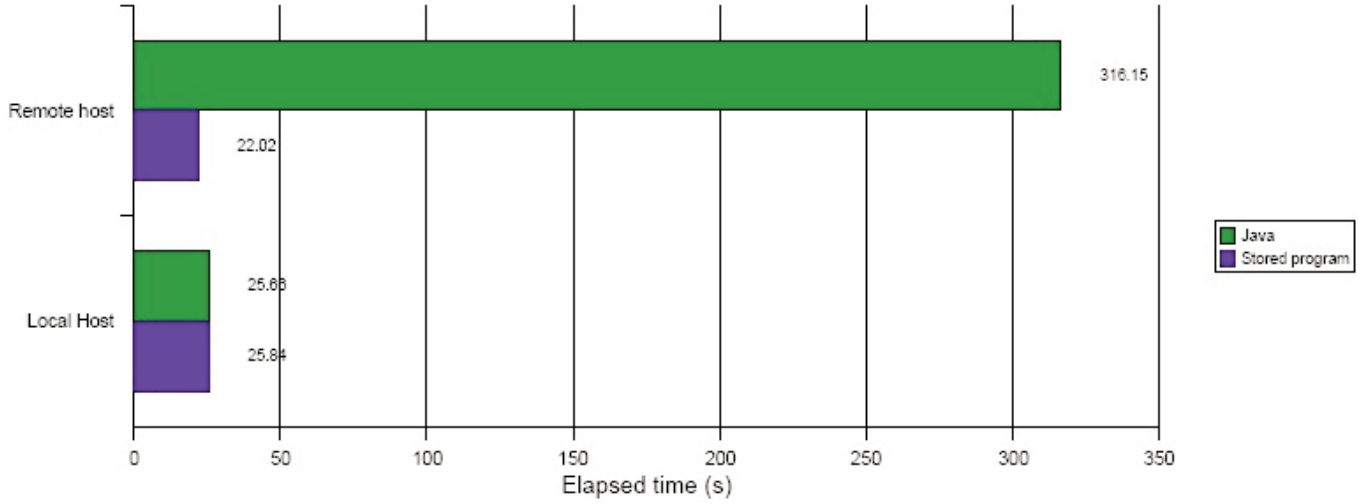


Figure 4. Stored procedures outperform when the network is a factor.

different languages; it may be hard, or impossible, for these applications to share code. Implementing common code in stored programs may allow these applications to share critical common routines.

For instance, in a banking application, transfer of funds transactions might originate from multiple sources, including a bank teller's console, an Internet browser, an ATM or a phone banking application. Each of these applications could conceivably have its own database access code written in largely incompatible languages, and without stored programs we might have to replicate the transaction logic, including logging, deadlock handling and optimistic locking strategies, in multiple places and in multiple languages. In this scenario, consolidating the logic in a database stored procedure can make a lot of sense.

Not Built for Speed?

It would be terribly unfair of us to expect the first release of the MySQL stored program language to be blisteringly fast. After all, languages such as Perl and PHP have been the subject of tweaking and optimization for about a decade, while the latest generation of programming languages—.NET and Java—has been the subject of a shorter, but more intensive optimization process by some of the biggest software companies in the world. So, right from the start, we might expect that the MySQL stored program language would lag in comparison with the other languages commonly used in the MySQL world.

Still, it's important to get a sense of the raw performance of the language. First, let's see how quickly the stored program language can crunch numbers. The first example compares a stored procedure calculating prime numbers against an identical algorithm implemented in alternative languages.

In this computationally intensive trial, MySQL performed poorly compared with other languages—five times slower than PHP or Perl, and dozens of times slower than Java, .NET or C (Figure 3).

Most of the time, stored programs are dominated by database access time, where stored programs have a natural performance advantage over other programming languages because of their lower network overhead. However, if you are writing a number-crunching routine, and you have a choice

between implementing it in the stored program language or in another language, such as PHP or Java, you may wisely decide against using the stored program solution.

If the previous example left you feeling less than enthusiastic about stored program performance, this next example should cheer you right up. Although stored programs aren't particularly zippy when it comes to number crunching, it is definitely true that you don't normally write stored programs simply to perform math; stored programs almost always process data from the database. In these circumstances, the difference between stored program and PHP or Java performance is usually minimal, unless network overhead is a big factor. When a program is required to process large numbers of rows from the database, a stored program can substantially outperform programs written in client languages, because it does not have to wait for rows to be transferred across the network—the stored program runs *inside* the database. Figure 4 shows how a stored procedure that aggregates millions of rows can perform well even when called from a remote host across the network, while a Java program with identical logic suffers from severe network-driven response time degradation.

Logic Fragmentation

Although it is generally useful to encapsulate data access logic inside stored programs, it is usually inadvisable to "fragment" business and application logic by implementing some of it in stored programs and the rest of it in the middle tier or the application client.

Debugging application errors that involve interactions between stored program code and other application code may be many times more difficult than debugging code that is completely encapsulated in the application layer. For instance, there is currently no debugger that can trace program flow from the application code into the MySQL stored program code.

Also, if your application relies on stored procedures, that's an additional skill that you or your team will have to acquire and maintain.

Object-Relational Mapping

It's becoming increasingly common for an Object-Relational Mapping (ORM) framework to mediate interactions between the application and the database. ORM is very common in Java (Hibernate and EJB), almost unavoidable in Ruby on Rails (ActiveRecord) and far less common in PHP (though there are an increasing number of PHP ORM packages available). ORM systems generate SQL to maintain a mapping between program objects and database tables. Although most ORM systems allow you to overwrite the ORM SQL with your own code, such as a stored procedure call, doing so negates some of the advantages of the ORM system. In short, stored procedures become harder to use and a lot less attractive when used in combination with ORM.

Are Stored Procedures Portable?

Although all relational databases implement a common set of SQL syntax, each RDBMS offers proprietary extensions to this standard SQL, and MySQL is no exception. If you are attempting to write an application that is designed to be independent of the underlying database, you probably will want to avoid these extensions in your application. However, sometimes you'll need to use specific syntax to get the most out of the server. For instance, in MySQL, you often will want to employ MySQL hints, execute non-ANSI statements, such as LOCK TABLES, or use the REPLACE statement.

Using stored programs can help you avoid RDBMS-dependent code in your application layer while allowing you to continue to take advantage of RDBMS-specific optimizations. In theory, stored program calls against different databases can be made to look and behave identically from the application's perspective. You can encapsulate all the database-dependent code inside the stored procedures. Of course, the underlying stored program code will need to be rewritten for each RDBMS, but at least your application code will be relatively portable.

However, there are differences between the various database servers in how they handle stored procedure calls, especially if those calls return result sets. MySQL, SQL Server and DB2 stored procedures behave very similarly from the application's point of view. However, Oracle and Postgres calls can look and act differently, especially if your stored procedure call returns one or more result sets.

So, although using stored procedures can improve the portability of your application while still allowing you to exploit vendor-specific syntax, they don't make your application totally portable.

Other Considerations

MySQL stored programs can be used for a variety of tasks in addition to traditional application logic:

- Triggers are stored programs that fire when data modification language (DML) statements execute. Triggers can automate denormalization and enforce business rules without requiring application code changes and will take effect for all applications that access the database, including ad hoc SQL.

- The MySQL event scheduler introduced in the 5.1 release allows stored procedure code to be executed at regular intervals. This is handy for running regular application maintenance tasks, such as purging and archiving.

- The MySQL stored program language can be used to create functions that can be called from standard SQL. This allows you to encapsulate complex application calculations in a function and then use that function within SQL calls. This can centralize logic, improve maintainability and, if used carefully, improve performance.

You Decide!

The bottom line is that MySQL stored procedures give you more options for implementing your application and, therefore, are undeniably a "good thing". Judicious use of stored procedures can result in a more secure, higher performing and maintainable application. However, the degree to which an application might benefit from stored procedures is greatly dependent on the nature of that application. I hope this article helps you make a decision that works for your situation.■

Guy Harrison is chief architect for Database Solutions at Quest Software (www.quest.com). This article uses some material from his book *MySQL Stored Procedure Programming* (O'Reilly 2006; with Steven Feuerstein). Guy can be contacted at guy.harrison@quest.com.

Getting Started with Heartbeat

Your first step toward high-availability bliss.

DANIEL BARTHOLOMEW

In every work environment with which I have been involved, certain servers absolutely always must be up and running for the business to keep functioning smoothly. These servers provide services that always need to be available—whether it be a database, DHCP, DNS, file, Web, firewall or mail server.

A cornerstone of any service that always needs be up with no downtime is being able to transfer the service from one system to another gracefully. The magic that makes this happen on Linux is a service called Heartbeat. Heartbeat is the main product of the High-Availability Linux Project.

Heartbeat is very flexible and powerful. In this article, I touch on only basic active/passive clusters with two members, where the active server is providing the services and the passive server is waiting to take over if necessary.

Installing Heartbeat

Debian, Fedora, Gentoo, Mandriva, Red Flag, SUSE, Ubuntu and others have prebuilt packages in their repositories. Check your distribution's main and supplemental repositories for a package named heartbeat-2.

After installing a prebuilt package, you may see a "Heartbeat failure" message. This is normal. After the Heartbeat package is installed, the package manager is trying to start up the Heartbeat service. However, the service does not have a valid configuration yet, so the service fails to start and prints the error message.

Sometimes, there's only one way to be sure whether a node is dead, and that is to kill it. This is where STONITH comes in.

You can install Heartbeat manually too. To get the most recent stable version, compiling from source may be necessary. There are a few dependencies, so to prepare on my Ubuntu systems, I first run the following command:

```
sudo apt-get build-dep heartbeat-2
```

Check the Linux-HA Web site for the complete list of dependencies. With the dependencies out of the way, download the latest source tarball and untar it. Use the ConfigureMe script to compile and install Heartbeat. This script makes educated guesses from looking at your environment as to how best to configure and install Heartbeat. It also does everything with one command, like so:

```
sudo ./ConfigureMe install
```

With any luck, you'll walk away for a few minutes, and when you return, Heartbeat will be compiled and installed on every node in your cluster.

Configuring Heartbeat

Heartbeat has three main configuration files:

- /etc/ha.d/authkeys
- /etc/ha.d/ha.cf
- /etc/ha.d/haresources

The authkeys file must be owned by root and be chmod 600. The actual format of the authkeys file is very simple; it's only two lines. There is an auth directive with an associated method ID number, and there is a line that has the authentication method and the key that go with the ID number of the auth directive. There are three supported authentication methods: crc, md5 and sha1. Listing 1 shows an example. You can have more than one authentication method ID, but this is useful only when you are changing authentication methods or keys. Make the key long—it will improve security and you don't have to type in the key ever again.

Listing 1. The /etc/ha.d/authkeys File

```
auth 1
1 sha1 ThisIsAVeryLongAndBoringPassword
```

The ha.cf File

The next file to configure is the ha.cf file—the main Heartbeat configuration file. The contents of this file should be the same on all nodes with a couple of exceptions.

Heartbeat ships with a detailed example file in the documentation directory that is well worth a look. Also, when creating your ha.cf file, the order in which things appear matters. Don't move them around! Two different example ha.cf files are shown in Listings 2 and 3.

The first thing you need to specify is the keepalive—the time between heartbeats in seconds. I generally like to have this set to one or two, but servers under heavy loads might not be able to send heartbeats in a timely manner. So, if you're seeing a lot of warnings about late heartbeats, try

Listing 2. The /etc/ha.d/ha.cf File on Briggs & Stratton

```
keepalive 2
deadtime 32
warntime 16
initdead 64
baud 19200
# On briggs the serial device is /dev/ttyS1
# On stratton the serial device is /dev/ttyS0
serial /dev/ttyS1
auto_failback on
node briggs
node stratton
use_logd yes
```

Listing 3. The /etc/ha.d/ha.cf File on Deimos & Phobos

```
keepalive 1
deadtime 10
warntime 5
udpport 694
# deimos' heartbeat ip address is 192.168.1.11
# phobos' heartbeat ip address is 192.168.1.21
ucast eth1 192.168.1.11
auto_failback off
stonith_host deimos wti_nps ares.example.com erisIsTheKey
stonith_host phobos wti_nps ares.example.com erisIsTheKey
node deimos
node phobos
use_logd yes
```

increasing the keepalive.

The deadtime is next. This is the time to wait without hearing from a cluster member before the surviving members of the array declare the problem host as being dead.

Next comes the warntime. This setting determines how long to wait before issuing a "late heartbeat" warning.

Sometimes, when all members of a cluster are booted at the same time, there is a significant length of time between when Heartbeat is started and before the network or serial interfaces are ready to send and receive heartbeats. The optional initdead directive takes care of this issue by setting an initial deadtime that applies only when Heartbeat is first started.

You can send heartbeats over serial or Ethernet links—either works fine. I like serial for two server clusters that are physically close together, but Ethernet works just as well. The configuration for serial ports is easy; simply specify the baud rate and then the serial device you are using. The serial device is one place where the ha.cf files on each node may differ due to the serial port having different names on each host. If you don't know the tty to which your serial port is assigned, run the following command:

```
setserial -g /dev/ttyS*
```

If anything in the output says "UART: unknown", that device is not a real serial port. If you have several serial ports, experiment to find out which is the correct one.

If you decide to use Ethernet, you have several choices of how to configure things. For simple two-server clusters, ucast (uni-cast) or bcast (broadcast) work well.

The format of the ucast line is:

```
ucast <device> <peer-ip-address>
```

Here is an example:

```
ucast eth1 192.168.1.30
```

If I am using a crossover cable to connect two hosts together, I just broadcast the heartbeat out of the appropriate interface. Here is an example bcast line:

```
bcast eth3
```

There is also a more complicated method called mcast. This method uses multicast to send out heartbeat messages. Check the Heartbeat documentation for full details.

Now that we have Heartbeat transportation all sorted out, we can define auto_failback. You can set auto_failback either to on or off. If set to on and the primary node fails, the secondary node will "failback" to its secondary standby state

when the primary node returns. If set to off, when the primary node comes back, it will be the secondary.

It's a toss-up as to which one to use. My thinking is that so long as the servers are identical, if my primary node fails, then the secondary node becomes the primary, and when the prior primary comes back, it becomes the secondary. However, if my secondary server is not as powerful a machine as the primary, similar to how the spare tire in my car is not a "real" tire, I like the primary to become the primary again as soon as it comes back.

Moving on, when Heartbeat thinks a node is dead, that is just a best guess. The "dead" server may still be up. In some cases, if the "dead" server is still partially functional, the consequences are disastrous to the other node members. Sometimes, there's only one way to be sure whether a node is dead, and that is to kill it. This is where STONITH comes in.

Fortunately, with logging enabled, troubleshooting is easy, because Heartbeat outputs informative log messages.

STONITH stands for Shoot The Other Node In The Head. STONITH devices are commonly some sort of network power-control device. To see the full list of supported STONITH device types, use the `stonith -L` command, and use `stonith -h` to see how to configure them.

Next, in the ha.cf file, you need to list your nodes. List each one on its own line, like so:

```
node deimos
node phobos
```

The name you use must match the output of `uname -n`.

The last entry in my example ha.cf files is to turn on logging:

```
use_logd yes
```

There are many other options that can't be touched on here. Check the documentation for details.

The haresources File

The third configuration file is the haresources file. Before configuring it, you need to do some housecleaning. Namely, all services that you want Heartbeat to manage must be removed from the system init for all init levels.

On Debian-style distributions, the command is:

```
/usr/sbin/update-rc.d -f <service_name> remove
```

Check your distribution's documentation for how to do the same on your nodes.

Now, you can put the services into the haresources file.

Listing 4. A Minimalist haresources File

```
stratton 192.168.1.41 apache2
```

Listing 5. A More Substantial haresources File

```
deimos \
         IPaddr::192.168.12.1 \

Filesystem::/dev/etherd/e1.0::/opt/storage::xfs \
         killnfsd \
         nfs-common \
         nfs-kernel-server
```

As with the other two configuration files for Heartbeat, this one probably won't be very large. Similar to the authkeys file, the haresources file must be *exactly* the same on every node. And, like the ha.cf file, position is *very* important in this file. When control is transferred to a node, the resources listed in the haresources file are started left to right, and when control is transfered to a different node, the resources are stopped right to left. Here's the basic format:

```
<node_name> <resource_1> <resource_2> <resource_3> . . .
```

The node_name is the node you want to be the primary on initial startup of the cluster, and if you turned on auto_failback, this server always will become the primary node whenever it is up. The node name must match the name of one of the nodes listed in the ha.cf file.

Resources are scripts located either in /etc/ha.d/resource.d/ or /etc/init.d/, and if you want to create your own resource scripts, they should conform to LSB-style init scripts like those found in /etc/init.d/. Some of the scripts in the resource.d folder can take arguments, which you can pass using a :: on the resource line. For example, the IPAddr script sets the cluster IP address, which you specify like so:

```
IPAddr::192.168.1.9/24/eth0
```

In the above example, the IPAddr resource is told to set up a cluster IP address of 192.168.1.9 with a 24-bit subnet mask (255.255.255.0) and to bind it to eth0. You can pass other options as well; check the example haresources file that ships with Heartbeat for more information.

Another common resource is Filesystem. This resource is for mounting shared filesystems. Here is an example:

```
Filesystem::/dev/etherd/e1.0::/opt/data::xfs
```

The arguments to the Filesystem resource in the example above are, left to right, the device node (an ATA-over-Ethernet drive in this case), a mountpoint (/opt/data) and the filesystem type (xfs).

For regular init scripts in /etc/init.d/, simply enter them by name. As long as they can be started with `start` and stopped with `stop`, there is a good chance that they will work.

Listings 4 and 5 are haresources files for two of the clusters I run. They are paired with the ha.cf files in Listings 2 and 3, respectively.

The cluster defined in Listings 2 and 4 is very simple, and it has only two resources—a cluster IP address and the Apache 2 Web server. I use this for my personal home Web server cluster. The servers themselves are nothing special—an old PIII tower and a cast-off laptop. The content on the servers is static HTML, and the content is kept in sync with an hourly rsync cron job. I don't trust either "server" very much, but with Heartbeat, I have never had an outage longer than half a second—not bad for two old castaways.

The cluster defined in Listings 3 and 5 is a bit more complicated. This is the NFS cluster I administer at work. This cluster utilizes shared storage in the form of a pair of Coraid SR1521 ATA-over-Ethernet drive arrays, two NFS appliances (also from Coraid) and a STONITH device. STONITH is important for this cluster, because in the event of a failure, I need to be sure that the other device is really dead before mounting the shared storage on the other node. There are five resources managed in this cluster, and to keep the line in haresources from getting too long to be readable, I break it up with line-continuation slashes. If the primary cluster member is having trouble, the secondary cluster kills the primary, takes over the IP address, mounts the shared storage and then starts up NFS. With this cluster, instead of having maintenance issues or other outages lasting several minutes to an hour (or more), outages now don't last beyond a second or two. I can live with that.

Troubleshooting

Now that your cluster is all configured, start it with:

```
/etc/init.d/heartbeat start
```

Things might work perfectly or not at all. Fortunately, with logging enabled, troubleshooting is easy, because Heartbeat outputs informative log messages. Heartbeat even will let you know when a previous log message is not something you have to worry about. When bringing a new cluster on-line, I usually open an SSH terminal to each cluster member and tail the messages file like so:

```
tail -f /var/log/messages
```

Then, in separate terminals, I start up Heartbeat. If there are any problems, it is usually pretty easy to spot them.

Heartbeat also comes with very good documentation. Whenever I run into problems, this documentation has been invaluable. On my system, it is located under the /usr/share/doc/ directory.

Conclusion

I've barely scratched the surface of Heartbeat's capabilities here. Fortunately, a lot of resources exist to help you learn about Heartbeat's more-advanced features. These include active/passive and active/active clusters with N number of nodes, DRBD, the Cluster Resource Manager and more. Now that your feet are wet, hopefully you won't be quite as intimidated as I was when I first started learning about Heartbeat. Be careful though, or you might end up like me and want to cluster everything.■

Daniel Bartholomew has been using computers since the early 1980s when his parents purchased an Apple IIe. After stints on Mac and Windows machines, he discovered Linux in 1996 and has been using various distributions ever since. He lives with his wife and children in North Carolina.

Resources

The High-Availability Linux Project: **www.linux-ha.org**

Heartbeat Home Page: **www.linux-ha.org/Heartbeat**

Getting Started with Heartbeat Version 2: **www.linux-ha.org/GettingStartedV2**

An Introductory Heartbeat Screencast: **linux-ha.org/Education/Newbie/InstallHeartbeatScreencast**

The Linux-HA Mailing List: **lists.linux-ha.org/mailman/listinfo/linux-ha**

Fedora Directory Server: the Evolution of Linux Authentication

Check out Fedora Directory Server to authenticate your clients without licensing fees.

JERAMIAH BOWLING

In most enterprise networks today, centralized authentication is a basic security paradigm. In the Linux realm, OpenLDAP has been king of the hill for many years, but for those unfamiliar with the LDAP command-line interface (CLI), it can be a painstaking process to deploy. Enter the Fedora Directory Server (FDS). Released under the GPL in June 2005 as Fedora Directory Server 7.1 (changed to version 1.0 in December of the same year), FDS has roots in both the Netscape Directory Server Project and its sister product, the Red Hat Directory Server (RHDS). Some of FDS's notable features are its easy-to-use Java-based Administration Console, support for LDAP3 and Active Directory integration. By far, the most attractive feature of FDS is Multi-Master Replication (MMR). MMR allows for multiple servers to maintain the same directory information, so that the loss of one server does not bring the directory down as it would in a master-slave environment.

Getting an FDS server up and running has its ups and downs. Once the server is operational, however, Red Hat makes it easy to administer your directory and connect native Fedora clients. In addition to providing network authentication, you easily can extend FDS functionality across other applications, such as NFS, Samba, Sendmail, Postfix and others. In this article, we focus solely on using FDS for network authentication and implementing MMR.

Installation

To begin, download a Fedora 6 ISO readily available from one of the many Fedora mirrors. FDS has low hardware requirements—500MHz with 256MB RAM and 3GB or more space. I recommend at least a 1GHz processor or above with 512MB or more memory and 20GB or more of disk space. This configuration should perform well enough to support small businesses up to enterprises with thousands users. As for supported operating systems, not surprisingly, Red Hat lists Fedora and Red Hat flavors of Linux. HP-UX and Solaris also are supported. With your bootable ISO CD, start the Fedora 6 installation process, and select your desired system preferences and packages. Make sure to select Apache during installation. Set your host and DNS information during the install, using a fully qualified domain name (FQDN). You also can set this information post-install, but it is critical that your host information is configured properly. If you plan to use a firewall, you need to enter two ports to allow LDAP (389 default) and the Admin Console (default is random port). For the servers used here, I chose ports 3891 and 3892 because of an existing LDAP installation in my environment. Fedora also natively supports Security-Enhanced Linux (SELinux), a policy-based lock-down tool, if you choose to use it. If you want to use SELinux, you must choose the Permissive Policy.

Once your Fedora 6 server is up, download and install the latest RPM of Fedora Directory Server from the FDS site (it is not included in the Fedora 6 distribution). Running the RPM unpacks the program files to /opt/fedora-ds. At this point, download and install the current Java Runtime Environment (JRE) .bin file from Sun before running the local setup of FDS. To keep files in the same place, I created an /opt/java directory and downloaded and ran the .bin file from there. After Java is installed, replace the existing soft link to Java in /etc/alternatives with a link to your new Java installation. The following syntax does this:

```
cd /etc/alternatives
rm ./java
ln -sf /opt/java/jre1.5.0_09/bin/java java
```

Next, configure Apache to start on boot with the chkconfig command:

```
chkconfig -level 345 httpd on
```

Then, start the service by typing:

```
service httpd start
```

Now, with the useradd command, create an account named fedorauser under which FDS will run. After creating the account, run `/opt/fedora-ds/setup/setup` to launch the FDS installation script. Before continuing, you may be presented with several error messages about non-optimal configuration issues, but in most cases, you can answer yes to get past them and start the setup process. Once started, select the default Install Mode 2 - Typical. Accept all defaults during installation except for the Server and Group IDs, for which we are using the fedorauser account (Figure 1), and if you want to customize the ports as we have here, set those to the correct

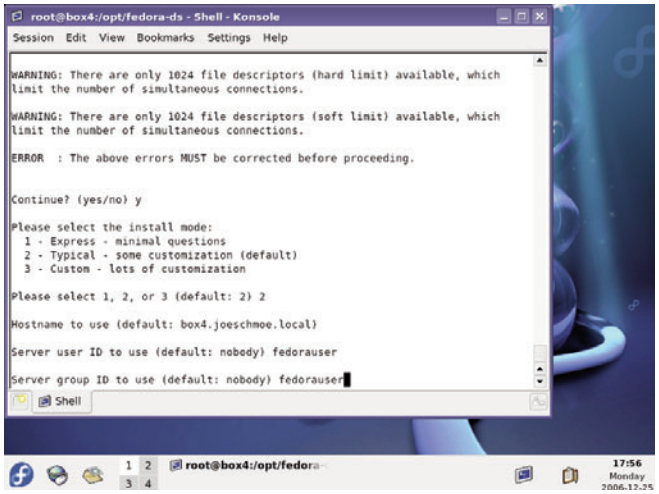


Figure 1. Install Options

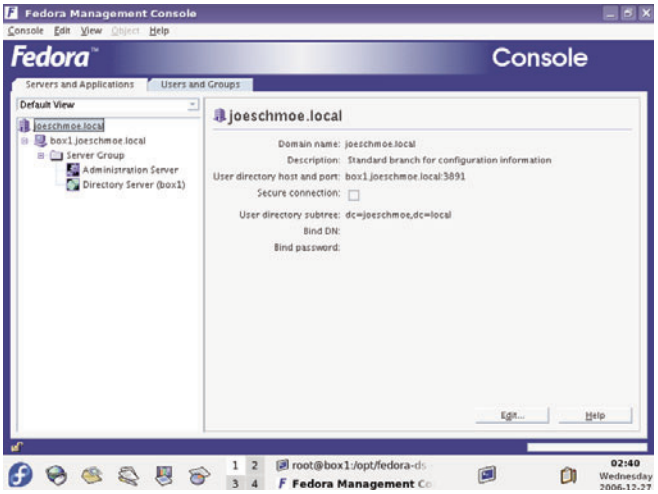


Figure 2. Main Console

numbers (3891/3892). You also may want to use the same passwords for both the configuration Admin and Directory Manager accounts created during setup.

When setup is complete, use the syntax returned from the script to access the admin console (`./startconsole -u admin http://fullyqualifieddomainname:port`) using the Administrator account (default name is Admin) you specified during the FDS setup. You always can call the Admin console using this same syntax from /opt/fedora-ds.

At this point, you have a functioning directory server. The final step is to create a startup script or directly edit /etc/rc.d/rc.local to start the slapd process and the admin server when the machine starts. Here is an example of a script that does this:

```
/opt/fedora-ds/slapd-yourservername/start-slapd
/opt/fedora-ds/start-admin &
```

Directory Structure and Management

Looking at the Administration console, you will see server information on the Default tab (Figure 2) and a search utility on the second tab. On the Servers and Applications tab, expand your server name to display the two server consoles that are accessible: the Administration Server and the Directory Server. Double-click the Directory Server icon, and you are taken to the Tasks tab of the Directory Server (Figure 3). From here, you can start and stop directory services, manage certificates, import/export directory databases and back up your server. The backup feature is one of the highlights of the FDS console. It lets you back up any directory database on your server without any knowledge of the CLI. The only caveat is that you still need to use the CLI to schedule backups.

On the Status tab, you can view the appropriate logs to monitor activity or diagnose problems with FDS (Figure 4). Simply expand one of the logs in the left pane to display its output in the right pane. Use the Continuous Refresh option to view logs in real time.

From the Configuration tab, you can define replicas (copies of the directory) and synchronization options with other servers, edit schema, initialize databases and define options for

Figure 3. Tasks Tab

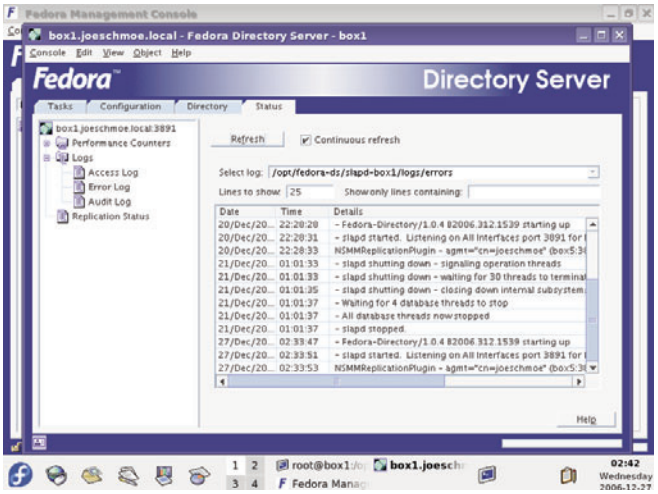


Figure 4. Main Logs

logs and plugins. The Directory tab is laid out by the domains for which the server hosts information. The Netscape folder is created automatically by the installation and stores information about your directory. The config folder is used by FDS to store information about the server's operation. In most cases, it should be left alone, but we will use it when we set up replication.

Before creating your directory structure, you should carefully consider how you want to organize your users in the directory. There is not enough space here for a decent primer on directory planning, but I typically use Organizational Units (OUs) to segment people grouped together by common security needs, such as by department (for example, Accounting). Then, within an OU, I create users and groups for simplifying

In addition to providing network authentication, you easily can extend FDS functionality across other applications, such as NFS, Samba, Sendmail, Postfix and others.

permissions or creating e-mail address lists (for example, Account Managers). FDS also supports roles, which essentially are templates for new users. This strategy is not hard and fast, and you certainly can use the default domain directory structure created during installation for most of your needs (Figure 5). Whatever strategy you choose, you should consult the Red Hat documentation prior to deployment.

To create a new user, right-click an OU under your domain and click on New→User to bring up the New User screen (Figure 5). Fill in the appropriate entries. At minimum, you need a First Name, Last Name and Common Name (cn), which is created from your first and last name. Your login ID is created automatically from your first initial and your last name. You also can enter an e-mail address and a password. From the options in the left pane of the User window, you can add NT User information, Posix Account information and activate/de-activate the account. You can implement a Password Policy from the Directory tab by right-clicking a domain or OU and selecting Manage Password Policy. However, I could not get this feature to work properly.

Replication

Setting up replication in FDS is a relatively painless process. In our scenario, we configure two-way multi-master replication, but Red Hat supports up to four-way. Because we already have one server (server one) in operation, we need another system (server two) configured the same way (Fedora, FDS) with which to replicate. Use the same settings used on server one (other than hostname) to install and configure Fedora 6/FDS. With both servers up, verify time synchronization and name resolution between them. The servers must have synced time or be relatively close in their offset, and they must be able to resolve the other's hostname for replication to work. You can use IP addresses, configure /etc/hosts or use DNS. I recommend having DNS and NTP in place already to make life easier.

The next step is creating a Replication Manager account to bind one server's directory to the other and vice versa. On the Directory tab of the Directory Server console, create a user account under the config folder (off the main root, not your domain), and give it the name Replication Manager, which should create a login ID (uid) of RManager. Use the same password for the RManager on both servers/directories. On server one, click the Configuration tab and then the Replication folder. In the right pane, click Enable Changelog. Click the Use Default button to fill in the default path under your slapd-servername directory. Click Save. Next, expand the Replication folder and click on the userroot database. In the right pane, click on enable replica, and select Multiple Master under the Replica Role section (Figure 6). Enter a unique Replica ID. This number must be different on each server involved in replication. Scroll down to the update section below, and in the Enter a new supplier DN: textbox, enter the following:

```
uid=RManager,cn=config
```

Click Add, and then the Save button at the bottom. On

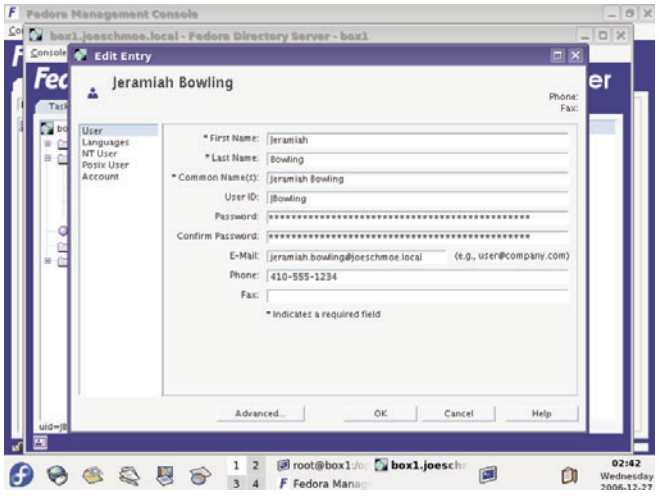


Figure 5. User Screen

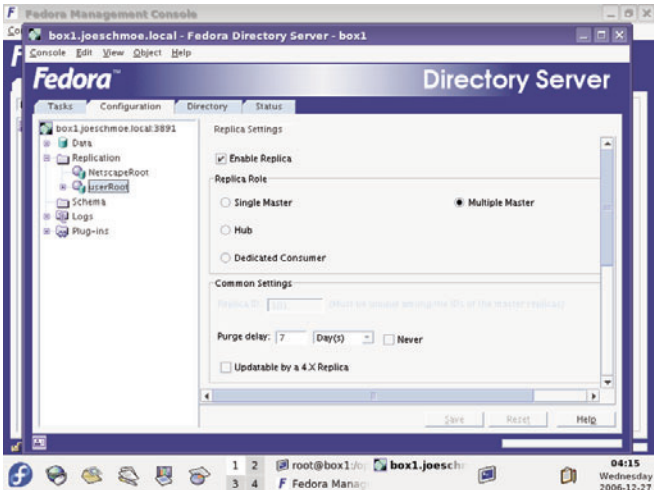


Figure 6. Setting Up Replica

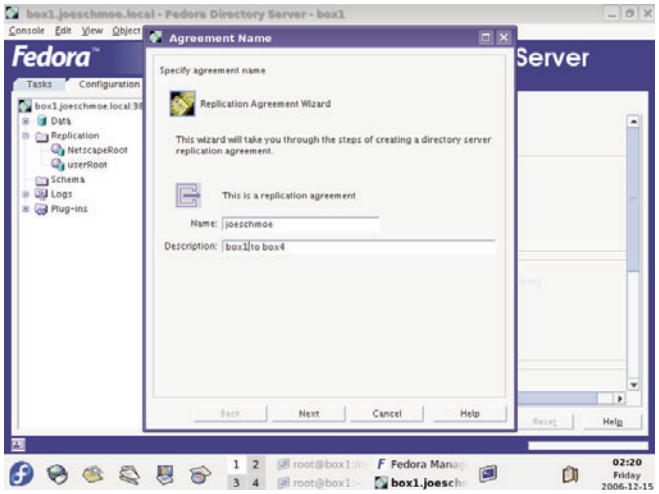


Figure 7. Enter a name and password.

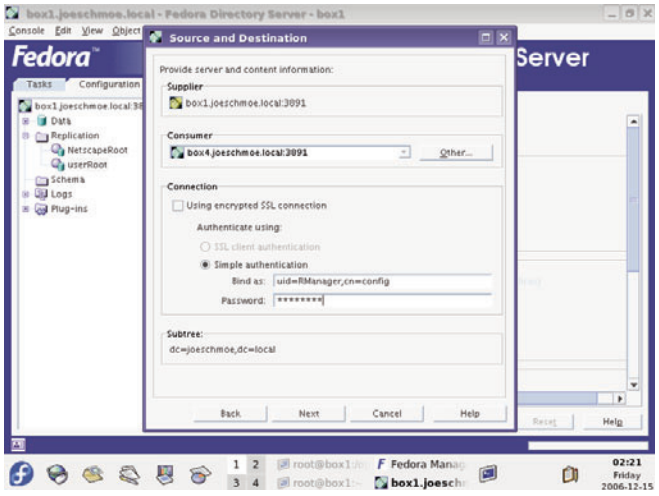


Figure 8. Enter information for the RManager account.

server two, perform the same steps as just completed for creating the RManager account in the config folder, enabling changelogs and creating a Multiple Master Replica (with a different Replica ID).

Now, you need to set up a replication agreement back on server one. From the Configuration tab of the Directory Server, right-click the userroot database, and select New Replication Agreement. A wizard then guides you through the process. Enter a name and description for the agreement (Figure 7). On the Source and Destination screen under Consumer, click the Other button to add server two as a consumer. You must use the FQDN and the correct port (3891 in our case). Use Simple Authentication in the Connection box, and enter the full context (uid=Rmanager,cn=config) of the RManager account and the password for the account (Figure 8). On the next screen, check off the box to enable Fractional Replication to send only deltas rather than the entire directory database when changes occur, which is very useful over WAN connections. On the Schedule screen, select Always keep directories in sync to keep. You also could choose to schedule your updates. On the Initialize Consumer screen, use the Initialize Now option. If you experience any difficulty in initializing a consumer (server two), you can use the Create consumer initialization file option and copy the resulting ldif folder to server two and import and initialize it from the Directory Server Tasks pane. When you click next, you will see the replication taking place. A summary appears at the end of the process. To verify replication took place, check the Replication and Error logs on the first server for success messages. To complete MMR, set up a replication agreement on the server, listing server one as the consumer, but do not initialize at the end of the wizard. Doing so would overwrite the directory on server one with the directory on server two. With our setup complete, we now have a redundant authentication infrastructure in place, and if we choose, we can add another two Read-Write replicas/Masters.

Authenticating Desktop Clients

With our infrastructure in place, we can connect our desktop clients. For our configuration, we use native Fedora 6 clients and Windows XP clients to simulate a mixed environment. Other Linux flavors can connect to FDS, but for space constraints, we won't delve into connecting them. It should be noted that most distributions like Fedora use PAM, the /etc/nsswitch.conf and /etc/ldap.conf files to set up LDAP authentication. Regardless of client type, the user account attempting to log in must contain Posix information in the directory in order to authenticate to the FDS server. To connect Fedora clients, use the built-in Authentication utility available in both GNOME and KDE (Figure 9). The nice thing about the utility is that it does all the work for you. You do not have to edit any of the other files previously mentioned manually. Open the utility and enable LDAP on the User Information tab and the Authentication tabs. Once you click OK to these settings, Fedora updates your nsswitch.conf and /etc/pam.d/system-auth files immediately. Upon reboot, your system now uses PAM instead of your local passwd and shadow files to authenticate users.

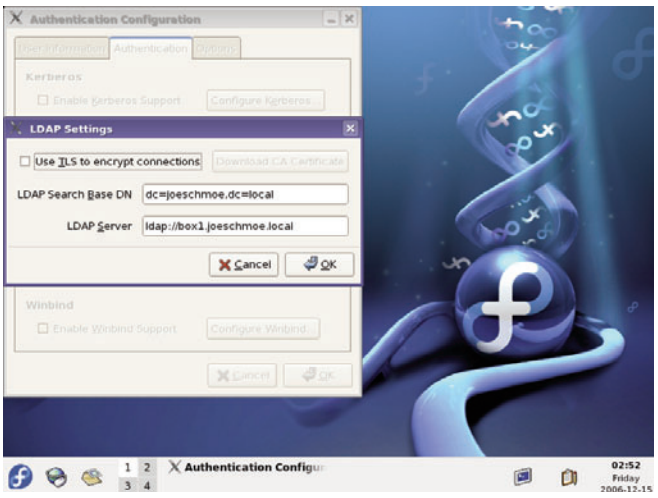


Figure 9. The Built-in Authentication Utility

During login, the local system pulls the LDAP account's Posix information from FDS and sets the system to match the preferences set on the account regarding home directory and shell options. With a little manual work, you also can use automount locally to authenticate and mount network volumes at login time automatically.

Connecting XP clients is almost as easy. Typically, NT/2000/XP users are forced to use the built-in MSGINA.dll to authenticate to Microsoft networks only. In the past, vendors such as Novell have used their own proprietary clients to work around this, but now the open-source pgina client has solved this problem. To connect 2000/XP clients, download the main pgina zipfile from the project page on SourceForge, and extract the files. For this article, I used version 1.8.4 as I ran into some dll issues with version 1.8.8. You also need to download and extract the Plugin bundle. Run the x86 installer from the extracted files, accepting all default options, but do not start the Configuration Tool at the end. Next, install the LDAPAuth plugin from the extracted Plugin bundle. When done installing, open the Configuration Tool under the Pgina Program Group under the Start menu. On the Plugin tab, browse to your ldapauth_plus.dll in the directory specified during the install. Check off the option to Show authentication method selection box. This gives you the option of logging locally if you run into problems. Without this, the only way to bypass the pgina client is through Safe Mode. Now, click on the Configure button, and enter the LDAP server name, port and context you want pgina to use to search for clients. I suggest using the Search Mode as your LDAP method as it will search the entire directory if it cannot find your user ID. Click OK twice to save your settings. Use the Plugin Tester tool before rebooting to load your client and test connectivity (Figure 10). On the next login, the user will receive the prompt shown in Figure 11.

Setting up replication in FDS is a relatively painless process.

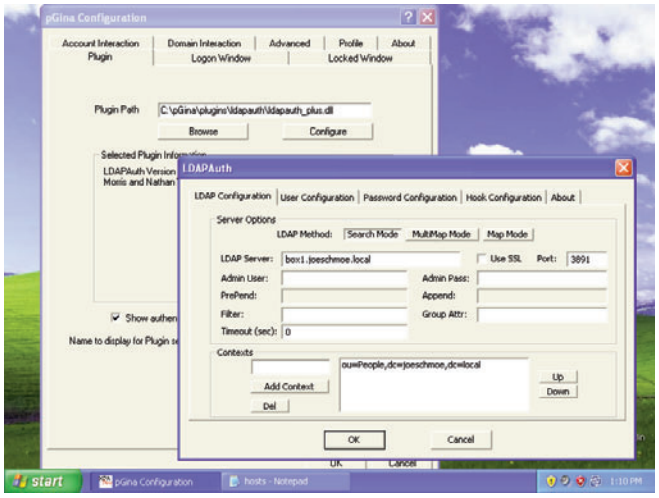


Figure 10. Plugin Tester Tool

Figure 11. Login Prompt

Conclusions

FDS is a powerful platform, and this article has barely scratched the surface. There simply is not room to squeeze all of FDS's other features, such as encryption or AD synchronization, into a single article. If you are interested in these items or want to know how to extend FDS to other applications, check out the wiki and the how-tos on the project's documentation page for further information. Judging from our simple configuration here, FDS seems evolutionary, not revolutionary. It does not change the way in which LDAP operates at a fundamental level. What it does do is take the complex task of administering LDAP and makes it easier while extending normally commercial features, such as MMR, to open source. By adding pgina into the mix, you can tap further into FDS's flexibility and cost savings without needing to deploy an array of services to connect Windows and Linux clients. So, if you are looking for a simple, reliable and cost-saving alternative to other LDAP products, consider FDS.■

Jeramiah Bowling has been a systems administrator and network engineer for more than ten years. He works for a regional accounting and auditing firm in Hunt Valley, Maryland, and holds numerous industry certifications including the CISSP. Your comments are welcome at jb50c@yahoo.com.

Resources

Main Fedora Site: **fedora.redhat.com**

Fedora ISOs: **fedora.redhat.com/Download**

Fedora Directory Server Site: **directory.fedora.redhat.com**

Main pgina Site: **www.pgina.org**

pgina Downloads: **sourceforge.net/project/showfiles.php?group_id=53525**