Dr Brown's Administeria

Dr Chris Brown
The Doctor provides Linux training, authoring and consultancy. He finds his PhD in particle physics to be of no help in this work at all.

Esoteric system administration goodness from the impenetrable bowels of the server room.

Bring back das blinkenlights

The first computers I worked on were much prettier than they are today. They had rows of little lights and if you really wanted, you could single-step the processor and read out the contents of each register cycle-by-cycle. They were the days of das blinkenlights, and many a computer room carried a faded photocopy of a heavy gothic script:

> "Das machine is nicht fur gerfinger-poken und mittengrabben. Oderwise is easy schnappen der springenwerk, blowen fuse, und poppencorken mit spitzensparken. Der machine is diggen by experten only. Is nicht fur geverken by das dummkopfen. Das rubbernecken sightseenen keepen das cotten picken hands in das pockets, so relaxen und watchen das blinkenlights."

Even the early hobby computers like the Altair 8800 (1975) had them, but by the time of the Commodore PET (1977) they had vanished, never to return.

Disappearing lights

In modern times, less is more has become the mantra for blinkenlights. My new server is so ashamed of the lights on its front panel that they're completely undetectable when it's switched off. My BT Home Hub has four very pretty tri-colour lights, but sadly they don't blinken. So when the thing is working normally (which is 99.9% of the time, thankfully) they carry no useful information whatever.

The lights on my laptop are so subtly understated, and so near the blue end of the visible spectrum, that I have to wait until dusk to find out if I have the caps lock on. To check for any network activity I have to put my left ear on the desk and squint at the tiny LED buried in the RJ45. And nowhere in my entire office is there a light that reports activity on the wireless LAN.

So *bitte, meinen herren*, bringen-sie back a few blinkenlights? Please?

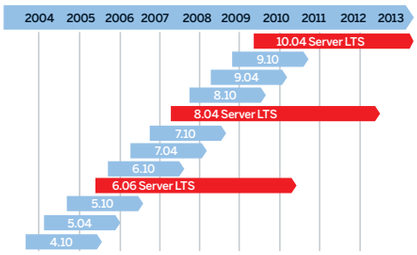
Lucid Lynx server

Ubuntu 10.04 Is the new long-term support release of Ubuntu a giant leap for mankind?

Most of us think of Ubuntu as a friendly Linux desktop, but there's a team at Canonical working hard to bring it into competition with Red Hat in the server market. Ubuntu 10.04 LTS Lucid Lynx will be on the **LXFDVD** next issue and is one of Canonical's 'long term support' releases, which it takes very seriously. For the server product, 'long term' means it'll provide updates and support for five years. Its training courses are based on the LTS releases (so expect new versions of Certified Professional, Server, and cloud courses soon). It provides an upgrade path from the previous 'normal' release (9.10) and the last LTS release (8.04). So 10.04 is an important milestone and, as a server product, it's come a long way since Hardy Heron two years ago. Here are some of the new features.

Cloud computing is written all over 10.04. You get a hint at the first installation menu, with the opportunity to install an Ubuntu Enterprise Cloud (UEC) node. The partitioning screens also show significant differences from 8.04: the default is now to build an ext4 filesystem on top of LVM. You also have the option to encrypt your **home** directory and/or the underlying partition. There are also new installation profiles that make it easy to install servers that fulfil various roles in a Eucalyptus cloud.

Some changes are less obvious. The kernel now supports address space layout randomisation, a security technique that randomises the position of key data areas within a process, making it much harder for an attacker to predict target addresses. The 2.6.32 kernel also supports Kernel SamePage Merging (KSM) memory aggregation, which enables multiple processes with identical memory pages to share a single copy of the page. This is great when *KVM* is running many virtual machines with identical kernels.



› Canonical supports upgrades between 'normal' or 'long term' Ubuntu releases.

Other new stuff

Major maintained packages expected to be added or updated since 8.04 LTS include *Django*, *CouchDB*, *RabbitMQ* (a messaging system based on AMQP), Eucalyptus, *etckeeper* (a revision control system for **/etc** which hooks into the *APT* tools), *Tomcat 6*, *OpenJDK6*, *MySQL 5.1*, *ClamAV*, *SpamAssassin*, *Puppet*, *AutoFS5*, *Samba* 3.4, Ruby, Python 2.6, Erlang, *Nagios 3* and *Corosync* (an open source 'cluster engine').

Internet detective work

IP sleuth Dr Brown goes in search of a cyber-criminal.

Look in the logs of any computer with a public network connection and you'll see lots of failed login attempts from automated script-driven attacks. If port 22 is open, most will be SSH logins. Looking in the daily security log of a webserver, mostly I saw two types of failure reported:

```
Mar 22 15:09:53 ns sshd[9446]: Failed password for invalid user ns from 174.121.23.114 port 50963 ssh2
```

Here the attacker is fishing for valid usernames. On the day covered by the log file, there were 13,647 such attempts using 959 distinct usernames. The second type was:

```
Mar 22 08:40:45 ns sshd[23270]: Failed password for root from 222.122.163.82 port 56501 ssh2
```

This represents a failed attempt to log in as root, and it occurred 258 times in a 10-minute period. Notice that both these messages contain the IP address of the machine that the attempt came from. In fact, 57% of the 99,647 lines in the log come from the IP address **222.122.163.82**, and we're going to see what we can find out about it. Before you get excited imagining we're going to discover the bad guy's name and address, let's face a couple of things. First, the IP address you've captured may be the address of a NAT gateway, and second, many attacks originate from compromised computers.

> **"You can also query the RIPE database using the whois command."**

Behind the address

Try a reverse DNS lookup. This has been edited for brevity:

```
$ dig -x 222.122.163.82
;; QUESTION SECTION:
;82.163.122.222.in-addr.arpa.   IN      PTR
;; AUTHORITY SECTION:
163.122.222.in-addr.arpa. 600  IN      SOA     ns1.ne.kr.
admin.100dedi.com.
```

No PTR record was found, but we know the nameserver for that address block is in Korea. You can also query the RIPE database at **www.db.ripe.net/whois** using **whois**:

```
$ whois 222.122.163.82
KRNIC is not an ISP but a National Internet Registry similar to APNIC.
The following is organization information that is using the IPv4 address.
IPv4 Address       : 222.122.163.0-222.122.163.255
Network Name       : KORNET-INFRA000001
Connect ISP Name   : KORNET
Registration Date  : 20100126
Publishes          : N
[ Organization Information ]
Organization ID    : ORG1600
Org Name           : Korea Telecom
Address            : Jungja-dong, Bundang-gu, Sungnam-ci
Zip Code           : 463-711
```

So now we know that the attacker's ISP is Korea Telecom. The program *geoiplookup* reports the geographic location:

```
$ geoiplookup 222.122.163.82
GeoIP Country Edition: KR, Korea, Republic of
```

You have to pay a subscription to get updates to the GeoIP database, or you can pay a fee for web-based lookups – see **www.maxmind.com**. Sometimes you can get useful details with *traceroute*. Look at the reported trip times, compare them to the round-trip times reported by *ping*, and you may be able to guess the location of a router near the offending machine.

There are some useful tools for internet sleuthing at **www.robtex.com**. You can discover which sites are hosted at the address, who owns the IP address block, whether the site is blacklisted and much else. (I must stress that there's no reason to suspect the owners of the websites hosted on this machine. They're almost certainly innocent bystanders.)

Finally, we might try an *nmap* scan of the machine:

```
# nmap -O 222.122.163.82
Interesting ports on 222.122.163.82:
Not shown: 985 closed ports
PORT    STATE   SERVICE
21/tcp  open    ftp
22/tcp  open    ssh
53/tcp  open    domain
80/tcp  open    http
 ... output omitted ...
OS details: Linux 2.6.9 - 2.6.15, Linux 2.6.9 - 2.6.15 (x86), Linux 2.6.9 - 2.6.20
```

The open ports here suggest that it's a typical Linux web hosting machine. So... the culprit probably lives in Seoul, uses Linux, walks with a limp, and smokes Gitanes. Elementary! »

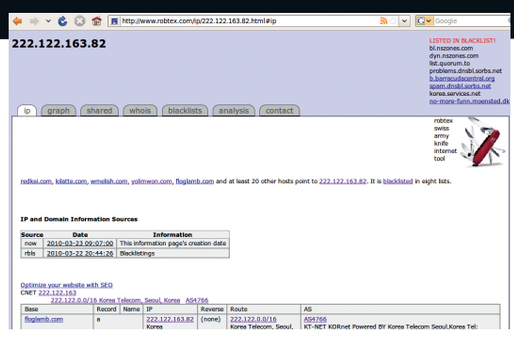
Batten down the hatches

You can reduce the risk of compromise from an SSH login attack by disabling direct root logins: just make sure you have the following line in your **sshd_config** file:

```
PermitRootLogin no
```

Another technique that's particularly effective against brute-force attacks is to analyse the messages logged by the SSHD server actively, and disable access to the IP addresses from which the failed logins originate. *DenyHosts* is one such script – it blocks IP addresses by putting them into **/etc/hosts.deny**.

Fail2ban is also triggered by analysing entries from log files, but it performs its magic by updating *Iptables*' firewall rules.



› From the Robtex site, we can see the attacking machine's ISP and the sites it hosts. Note the blacklist.

Setting up ProFTPD

FTP servers Learn how to set up a professional FTP server and configure anonymous or authenticated logins with the full-featured *ProFTPD*.

Last week I was caught on the hop by a client asking for advice on setting up an FTP server. This is something I haven't done in a long while, and whatever familiarity I may once have had with FTP was largely paged out of my brain. I guess I feel that FTP has perhaps had its day, and that HTTP is more in tune with the times.

However, there's no shortage of FTP servers in the standard repositories; Fedora lists two (*ProFTPD* and *vsftpd*) and Ubuntu lists no less than four. Further extensive research – involving a penny and a call of 'heads' – led me to choose *ProFTP* and this tutorial describes my experience in getting it working. I did this on Fedora 12, but you should have a similar experience on other distributions.

ProFTPD is a robust and full-featured FTP server. It can be run either as a standalone server or from *xinetd*. It supports anonymous and authenticated logins, including logins for 'virtual users' that have accounts defined in LDAP directories or SQL databases. It also allows virtual hosting – the ability to serve several FTP sites with a single instance of the server. We'll look at all those features.

The first thing was to install it and make sure it gets started at boot time:

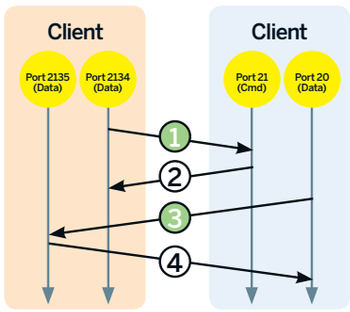
```
# yum install proftpd
# chkconfig proftpd on
```

Once I've installed the software, my instinct is to head for the config file (in this case **/etc/proftpd.conf**). It has a very *Apache*-like feel to its syntax and structure, with XML-style container tags. The default config file that installs with Fedora 12 provides a working configuration, plus the usual range of commented-out serving suggestions. There's a fairly minimal config file shown in **Fig 2**. It's stripped down from the one supplied but is sufficient to support authenticated FTP logins.

Anonymous FTP

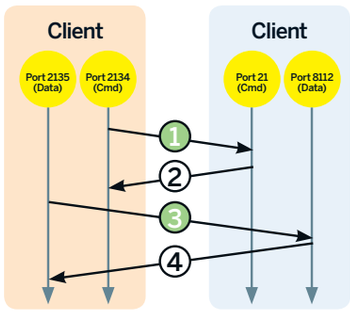
Anonymous FTP enables a user to authenticate to the server with a username of 'ftp' or 'anonymous' and a password which, traditionally, should be your email address but which in practice is not checked. Anonymous FTP is widely used to provide read-only access to internet resources and predates the use of HTTP and websites for this purpose by many years. Anonymous logins short-circuit the security concerns I have about FTP. For one thing, there's no sensitive username and password sent over the network for the bad guys to capture and, for another, the fact that the connection isn't encrypted doesn't matter because the content being transmitted is public anyway!

The default **proftpd.conf** file shipped with the Fedora 12 distro has an **<Anonymous>** stanza that works out of the



Active Mode FTP

› (Fig 1) Using active mode FTP, the server actively creates a data connection back to the client (step 3) using a port number sent by the client in step 1. This will likely be blocked by the client's gateway.



Pasive Mode FTP

› Using passive mode FTP, it's the client that opens the data connection (step 3) using a port number sent by the server in step 2. This mode is more firewall-friendly.

RIP for FTP?

FTP is an ancient protocol. The earliest RFCs date from 1971 and it became formally defined in RFC454 (1973) and RFC959 (1985). Roughly comtemporaneous with Telnet, FTP was designed long before we had firewalls or private IP address spaces or NAT or any of the other dubious delights of the modern internet, and it doesn't play well with them. For a start, FTP uses two separate connections – one for control and one for data. It operates in either 'active' or 'passive' mode. These terms are from the server's point of view and refer to whether the server actively makes a data connection back to the client or whether it passively accepts the data connection from the client. (See **Fig 1**.)

In active mode the server will almost certainly be thwarted in its attempt to connect back to the client because of packet filtering and masquerading performed by the gateway that connects the client to the internet. Passive mode is less troublesome but still needs a separate port (on the server end) for each client, so that potentially a large range of ports need to be opened in the server's firewall.

The FTP protocol has no way to specify the FQDN of the site it's trying to connect to, making name-based virtual hosting impossible. Its authentication mechanisms, which transmit a username and password in cleartext, (and indeed, transmit the files themselves in cleartext) are strongly deprecated in today's security-conscious internet.

It's possible to overcome some of these difficulties by using the local port forwarding feature of SSH to provide an encrypted tunnel for the FTP traffic – and there are fancy connection-tracking tricks that can be used in the firewalls to accommodate FTP's unusual use of ports. But it all reminds me of the Major Oak in Sherwood Forest, which has long been supported by wooden props. Looking at it, I'm seized by a compassionate urge to pull away the props and allow the venerable tree to collapse and die with dignity, so that new growth can emerge the way nature intended.

box. Well, almost. All I had to do was edit the file **/etc/sysconfig/proftpd** and add the line:

```
PROFTPD_OPTIONS="-DANONYMOUS_FTP"
```

then restart the *ProFTPD* server with:

```
service proftpd restart
```

The default configuration defines **~ftp** (which resolves to **/var/ftp**) as the **home** directory for anonymous logins. When you install the *ProFTPD* package it creates the directory **/var/ftp/pub** (which is where to put your content) and **/var/ftp/upload** (which is where anonymous users may upload to).

The default configuration of *ProFTPD* also supports authenticated user logins for users who have regular Linux accounts on the server. After logging in you're automatically placed in your **home** directory on the server. This didn't work for me initially, so I spent some time running *ProFTPD* in non-daemon mode with debugging enabled:

```
proftpd -n -d 5
```

and also running **tail -f** on **/var/log/messages**. Eventually it became clear that SELinux was preventing the daemon from accessing the user's **home** directory, so I did what I usually do in these circumstances – put SELinux into permissive mode:

```
setenforce permissive
```

after which it all worked fine. On a production server you'd probably want to take the SELinux issues more seriously.

A virtual user is one who doesn't have a regular Linux account on the server in **/etc/passwd** or wherever. The term 'virtual user' is misleading; the users are real enough – actual warm, breathing bodies – it's the accounts that are virtual. There are several places where virtual user accounts can be stored, including **AuthUserFiles**, LDAP directories and SQL databases. The simplest virtual user authentication mechanism is the **AuthUserFile**, which is a flat text file in the same format as **/etc/passwd**.

Virtual hosting

What if you want to host several FTP sites on a single system? Unfortunately, you can't do name-based virtual hosting as you can for websites, where multiple sites have entries in DNS that point to the same IP address. The reason this works for HTTP but not for FTP is that the HTTP protocol includes a 'Host' field in the HTTP request header that enables the web server to figure out which site the request is for, even if they all arrive on the same IP address. The FTP protocol has no such field. As a result, virtual hosting of FTP sites requires that each site has a separate IP address so that the server knows which site the request is intended for.

However, if you're fortunate enough to have sufficient spare IP addresses to do things this way, there are two things you'll need to do. First, you'll need to define an IP 'alias' on your network card for each of the IP addresses you want to use. Second, you'll need a **<VirtualHost>** section in the **proftpd.conf** file for each site you want to host.

On a RedHat flavoured system, for each additional IP address you want to assign to your network card you'll need to create a file called something like **/etc/sysconfig/network-scripts/ifcfg-eth0:N** (where **N** is the alias number: 0, 1, 2…). The file will look something like this:

```
DEVICE=eth0:1
BOOTPROTO=static
IPADDR=192.168.1.69
ONBOOT=yes
HWADDR=00:0C:29:E1:E4:23
```

where you should substitute appropriate values for **DEVICE**, **IPADDR** and **HWADDR**. You need a separate file for each IP address. As far as I know, there's no way to say something like

```
# Server Config - config used for anything outside a <VirtualHost> or <Global> context

ServerName                      "ProFTPD server"
ServerIdent                     on "FTP Server ready."
ServerAdmin                     root@localhost
DefaultServer                   on

# Cause every FTP user except adm to be chrooted into their home directory
VRootEngine                     on
DefaultRoot                     ~ !adm
VRootAlias                      etc/security/pam_env.conf /etc/security/pam_env.conf

# Use pam to authenticate (default) and be authoritative
AuthPAMConfig                   proftpd
AuthOrder                       mod_auth_pam.c* mod_auth_unix.c

# Don't do reverse DNS lookups (hangs on DNS problems)
UseReverseDNS                   off

# Set the user and group that the server runs as
User                            nobody
Group                           nobody

# Define the log formats
LogFormat                       default "%h %l %u %t \"%r\" %s %b"
LogFormat                       auth    "%v [%P] %h %t \"%r\" %s"

# Global Config - config common to Server Config and all virtual hosts
<Global>
  Umask                         022

  # Don't do ident queries (hangs when the port is filtered)
  IdentLookups                  off

  # Allow users to overwrite files and change permissions
  AllowOverwrite                yes
  <Limit ALL SITE_CHMOD>
    AllowAll
```

› (Fig 2) This minimal configuration is sufficient to support authenticated FTP logins.

"I'd like to assign all IP addresses in the block 10.1.0.0/24 to this network card". (If I'm wrong, let me know!)

After creating these files, run

```
service network restart
```

to pick up the new IP addresses, then verify that you have network interfaces at all the IP addresses you expect with:

```
ifconfig
```

Each of the FTP sites you want to serve needs a **<VirtualHost>** container within the config file. It will look something like this:

```
<VirtualHost 144.16.44.5>
...
</VirtualHost>
```

or you can use a machine name instead of an IP address:

```
<VirtualHost foo.example.com>
...
</VirtualHost>
```

but don't be fooled by this. The machine name will be resolved to an IP address, and it's the IP address that determines which FTP site will be served. If you have several machine names that resolve to the same IP address in DNS and you have a separate **<VirtualHost>** container for each, the first one will win. If you have stuff that's common to many virtual hosts, you can factor it out and put it into a **<Global>** section within the config file.

As a proof of concept, I assigned a second IP address (192.168.1.69) to my FTP server's network card and added a stanza to **proftpd.conf** – you can find it on the **LXFDVD**. The 'secondary' server provides anonymous FTP access to the directory **/var/ftp/secondary/pub**. I also created **/var/ftp/secondary/welcome.msg**, containing a welcome message.

There's a great deal of flexibility in *ProFTPD* that we haven't explored here, including things like access control, logging and the use of *chroot* jails. LXF

Where to learn more

The *ProFTPD* website can be found at **www.proftpd.org**.

For an alphabetic list of config file directives see: **www.proftpd.org/docs/directives/linked/by-name.html**.

There's a good discussion of active versus passive FTP at **http://slacksite.com/other/ftp.html**.

For more on virtual users, you can find a how-to on using *MySQL* to store user accounts at **www.howtoforge.com/proftpd_mysql_virtual_hosting**. Also have a look at **www.kreci.net/linux/proftpd-virtual-user-quick-howto**, which provides a brutally short explanation of using **AuthUserFile**.