Security: Protect

Martin Meredith teaches you how to manage your ports, deal with vulnerabilities and stop hackers from taking advantage of your server.

Our expert

Martin Meredith is a Debian and Ubuntu developer, and a security expert for a major UK online retailer

Way back in the early days of dial-up, the internet mostly contained library catalogues, military secrets and students' Dungeons and Dragons spec sheets. Now there are websites for people, their pets, their friends and family, and their businesses. However, while most people are happy to use a free hosting provider, or to pay a company to host their websites for them, the more dedicated web master tends to plump up for a dedicated server, or a Virtual Private Server (VPS).

Running your own server means that you have to be aware of the multitude of potential security issues you're exposed to on the internet, though. These days, most home computers have a firewall in place, or connect through a router that can protect them from the dangers lurking on the web. If you own a server, you'll still need a firewall, but there's much more you can do to be safe online and we'll show you how.

How secure are you?

On the internet, every service you connect to has a port that it uses. For example, when you connect to a website, you use port 80 (or port 443 for HTTPS) and when you SSH, you use port 22. FTP uses port 21, IMAP uses port 143, and so on. When a server runs, it opens that port and waits for an incoming connection.

So, how does this affect you? When you run a server, you might have a few different services running – maybe you have a basic LAMP stack, or an mail server. These services normally open up their ports for anyone on the internet to see, which isn't always what we want.

By way of example, let's have a look at a recently set up server using a program called *nmap*. This is normally available from your distribution's package manager, or at **http://nmap.org**. Once you have *nmap* installed, supply it with your server's address to get an output that looks like:

```
mez@lazy: % nmap torpor

Starting Nmap 4.76 ( http://nmap.org ) at 2009-05-04 11:56 BST
Interesting ports on torpor:
Not shown: 984 closed ports
PORT      STATE    SERVICE
22/tcp    open     ssh
25/tcp    open     smtp
53/tcp    open     domain
80/tcp    open     http
110/tcp   open     pop3
143/tcp   open     imap
993/tcp   open     imaps
1234/tcp  open     hotline
3306/tcp  open     mysql
10000/tcp open     snet-sensor-mgmt

Nmap done: 1 IP address (1 host up) scanned in 3.46 seconds
```

The results of *nmap* show 10 ports are open for anyone on the internet to connect to. Most of these are normal things that you'd want to see available on the server, such as SSH and email. Some of them, however, we don't want to have open to the internet (such as *MySQL*) and some of them are just plain confusing, such as port 1234.

Is anybody listening?

So, as we've seen above, we can't always tell what's listening in on a certain port. A quick Google search tells us that port 1234 is normally used by various trojan viruses, which means we should probably do a little more investigation to find out what's going on and why that port would need to be open.

The easiest way to find out what program is listening on a specific port is to use the **netstat** command. Running this without any options will give you a list of the currently open connections. We, however, want to find out what is listening to a specific port.

To find out what program is running on port 1234, we'll need to run **netstat -pnl** as root. This will give us a list that might look a little confusing at first, but we're only interested in two of the columns in the output. Since we already know

» **Last month** We built and configured a home telephone network with Asterisk.

your server

the port number we're looking for, all we need to look at is the column called Local Address for an entry ending with **:1234** (the colon separates the IP address and the port number). The line we're interested in is:

```
tcp        0      0 0.0.0.0:1234            0.0.0.0:*               LISTEN
24481/php-cgi
```

This tells us that the program *php-cgi* is listening on all IP addresses on the server (0.0.0.0 means any IP address) for a connection from anywhere. PHP is a scripting language, and here we've set it to listen in Fast CGI mode to port 1234.

Anyone being able to run scripts on our server is not a good thing, so it's time to enlist the help of *iptables*.

Laying down the law

Available by default on nearly all Linux distributions, *iptables* is Linux's answer to all your firewalling needs. In essence, a firewall sits between your computer and the internet, either denying or allowing traffic that's going to and from your server, based on a set of rules.

iptables is relatively easy to set up, but learning how to write the rules can take time. Below is the rules file that we'll be using on this server:

```
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m state --state RELATED,ESTABLISHED -j
ACCEPT
-A INPUT -s 127.0.0.1 -d 127.0.0.1 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 22 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 25 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 53 -j ACCEPT
-A INPUT -p udp -m udp --dport 53 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 110 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 143 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 993 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 10000 -j ACCEPT
-A INPUT -s 192.168.1.3 -p tcp -m tcp –dport 3306 -j
ACCEPT
-A INPUT -p icmp -m icmp --icmp-type 0 -j ACCEPT
-A INPUT -p icmp -m icmp --icmp-type 8 -j ACCEPT
COMMIT
```

Here, ***filter** tells us which table these rules apply to. We'll only be covering the filter table here, but other tables are available for setting up processes such as NAT (Network Address Translation), routing and so on.

The next few lines, those containing **[0:0]**, set up the default policies for the chains in *iptables*. When we use *iptables*, we generally work with three chains: **INPUT** for all incoming connections, **FORWARD** for connections that are forwarded to another server (we won't be using this here), and **OUTPUT** for all outgoing connections. A policy for these chains can be set to **ACCEPT**, **DROP**, or **REJECT**, which either allows the connection, ignores the connection attempt, or sends back an error code saying that the port is not open.

Here we've set the **ACCEPT** and **FORWARD** chains to **DROP** all incoming connections, and the **OUTPUT** chain to **ACCEPT** by default. However, these defaults are only run when a packet doesn't match any of the rules and even then we'll need to add a few exceptions.

Before we do that, it's worth mentioning that the final line, **COMMIT**, tells *iptables* to apply these rules to the firewall. This is the point where your firewall becomes active and starts defending your server.

Plugging the holes

The lines beginning **-A INPUT** set up our firewall rules. These are written as if you were calling *iptables* directly, but without the **iptables** at the start of the command. You can find out more information about how to write these rules from the *iptables* man page.

Each rule starts with **-A INPUT**, which tells *iptables* to append the rule to the chain for input. If we were adding rules to govern output instead, all we'd need to do is substitute **INPUT** for **OUTPUT** here. After that, we have the matching part of the rule and we end the rule line with **-j ACCEPT**, which tells *iptables* to join the packet to the **ACCEPT** chain – in other words, let the packet through. If we wanted to **REJECT** or **DROP** the packet instead, we could substitute those terms in the place of **ACCEPT** accordingly.

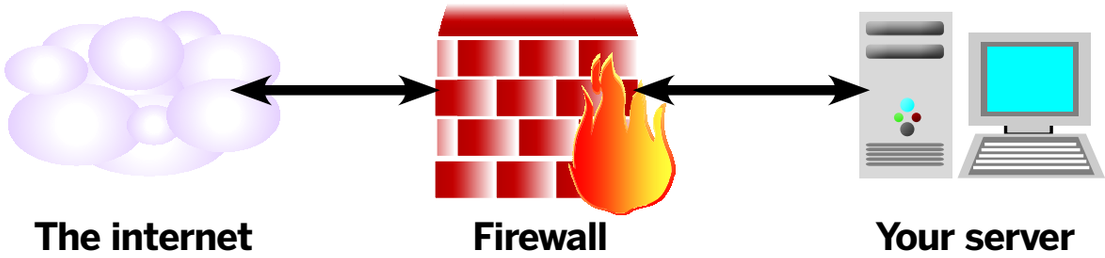
Now let's take a look at the line:

```
-A INPUT -m state --state RELATED,ESTABLISHED -j
ACCEPT
```

which tells *iptables* to allow any connection that's related or established to be allowed through. This is a good rule to have, »

Quick tip

Running **netstat -pnl** as root will tell you which programs are listening to ports.



› You've seen this diagram before I'll wager, and yes, it really is this simple.

since it permits any connection you have already made to continue – for example, the current SSH connection will stay active if you're using SSH to change the firewall rules. It also allows incoming connections that are related to outgoing connections (TCP works both ways and you'll find you can't make outgoing connections from the box without this rule).

The next line down tells *iptables* to accept any connections that originate from 127.0.0.1 (**-s** means source IP address) connecting to 127.0.0.1 (**-d** refers to the destination IP address), which translates to 'accept any connection I make to myself'.

In the next nine lines, we open up the ports that we saw earlier to the outside world. Notice that we don't have any code that allows access to ports 3306 (*MySQL*) and 1234 (the misconfigured PHP installation), because the firewall rule that allows connections locally takes these into account. However, we do want to add an exception to rejecting all outside access to *MySQL*, which comes in the form of:

```
-A INPUT -s 192.168.1.3 -p tcp -m tcp –dport 3306 -j ACCEPT
```

This specifically enables the IP address 192.168.1.3 to connect to port 3306 and it's here because we have another server on our local network that needs to access *MySQL*.

The final two lines above **COMMIT** allow the server to respond to pings, and enable it to receive ping responses.

After saving the file (to a location similar to **/etc/iptables.conf**) we can run the command:

```
iptables-restore < /etc/iptables.conf
```

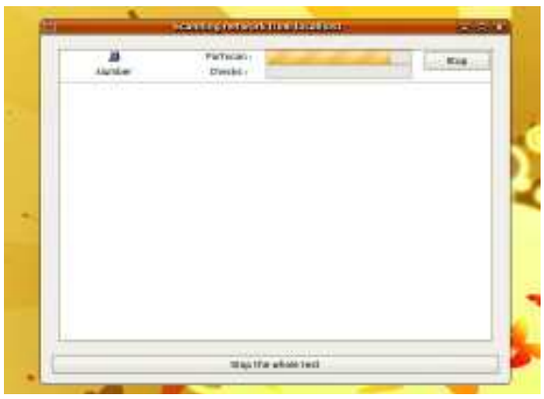
enabling the firewall rules we've written above. If we run *nmap* again, it gives us the following results

```
mez@lazy: % sudo nmap torpor
Starting Nmap 4.76 ( http://nmap.org ) at 2009-05-04 11:56 BST
Interesting ports on torpor:
Not shown: 984 filtered ports
PORT      STATE    SERVICE
22/tcp    open     ssh
25/tcp    open     smtp
53/tcp    open     domain
80/tcp    open     http
110/tcp   open     pop3
143/tcp   open     imap
993/tcp   open     imaps
10000/tcp open     snet-sensor-mgmt
Nmap done: 1 IP address (1 host up) scanned in 10.30 seconds
```

Notice that the output now says that it's not showing filtered ports instead of closed ports. Also, you may have to run *nmap* as root – in our tests, the above firewall configuration worked too well and *nmap* couldn't find all the open ports without being run as root.

Quick tip

You can make your firewall rules run on startup by adding the **iptables-restore** command to **/etc/rc.local**.



› ***Nessus* scans your servers for vulnerabilities in its code that could be exploited by the unscrupulous.**

What's Nessus-ary?

This basic firewall should protect us to some extent, but there are other potential vulnerabilities in every system, especially for servers that don't get updated often. To combat this, there is a tool known as *Nessus*, which comes in two parts – the client and the server. The reason for this is so that the server can be installed in a remote location and used to test the connection to the local service. In this case, the computer that we're testing isn't local to us, so we can install both the server and the client locally.

Once you've installed *Nessus* and *nessusd* from your preferred package manager, you'll need to set up a user. Run the command **nessus-adduser** and follow the instructions on screen. Now, run the *Nessus* client (found under Applications > Internet > Nessus in Ubuntu).

When *Nessus* runs, it works through a list of plugins that enable the testing of various vulnerabilities. Because there are new vulnerabilities discovered every day, it's wise to keep *Nessus* up to date. Tenable Network Security, the application's creator, has two different lists of plugins available – the HomeFeed and the ProfessionalFeed. The ProfessionalFeed provides a more current list, meaning that the newest vulnerabilities can be checked as soon as plugins for them exist. ProfessionalFeed costs $1,200, though, so unless you're working with *Nessus* on a daily basis, or working in a high-security environment, HomeFeed is easily enough.

Tenable requires you to register for its feed downloads, which you can do at **http://linkpot.net/enviably**. After that, run the command **nessus-fetch –register <your registration code>** with the registration code that you'll be supplied via email.

Once that's set up and you load the program, you may find the *Nessus*'s main screen is a bit overwhelming. Instead of trying to walk you through all of it abstractly, we'll show you how to perform a basic scan.

First of all, you need to log in to the server with the credentials that you set up previously. After you've logged in, you'll be presented with the Plugins screen. This is a list of scripts that *Nessus* will try to run. Click the Select All button, and flip to the Scan Options tab. Here, you can find the general scan options for the *Nessus* scan. The most

Et tu, Brutus?

Even with different security measures in place, there's still the possibility that someone can compromise your server simply by guessing the password! There are a number of scripts out there that use a dictionary to try every possible combination of username and password in a relatively short time.

Fail2ban (**www.fail2ban.org**) tries to overcome this by monitoring your log files for failed login attempts. When it detects that someone has tried to login multiple times and failed, it will restrict their access to the server (using the firewall) for a certain amount of time.

This isn't a great solution if you regularly forget your password, but it is good for stopping people in their tracks when they try to use brute force to pry their way into your server.

interesting of these options is the Safe Checks checkbox, because *Nessus* won't perform scans that have the potential to crash your server if it's ticked. However, as long as you have physical access to the server, or some way of resetting it remotely, we'd recommend leaving this box empty, because this enables you to perform a more in-depth scan. Ultimately, we'd rather have *Nessus* find a security hole and crash the server than leave the vulnerability in place.

Finally, all we need to do is put the name or IP address of a server into the Target tab and then getting *Nessus* underway is a simple case of clicking Start The Scan and letting the program do the rest.

Once *Nessus* has finished scanning, it will present you with a report showing a list of items that *Nessus* has found. A lot of the things that *Nessus* returns are informational messages such as 'you have an SSH server running', but anything important will have a red stop icon next to it. If *Nessus* finds vulnerabilities, it will tell you how to fix them (or point you to the right place to find out how to fix them). You should act on any critical points as soon as possible.

Trip 'em up

If all goes well with your *Nessus* scan and subsequent vulnerability fixing, you should now have a server that's hard for someone to get into. However, that doesn't mean that it's impossible to breach your security – the only way to make sure that nobody can get into your system it to turn it off.

So, what can we do if we can't guarantee the security of a system unless it's off, or at least disconnected from the internet? At the point where a hacker gets into your system, you can still, at a minimum, make sure you know that they've been there. An Intrusion Detection System (IDS) enables you to do this, although we hope you'll never have to rely on it.

The most widely known IDS at the moment is *Tripwire*, which has been around since 1992. At the moment, three programs called *Tripwire* are available and they all do the same thing. Only one of them, however, is open source. You can find it in your distribution's package manager, or by heading over to **http://tripwire.sf.net**.

Tripwire works by creating a database of all the files on your system and notifying you when something about them has changed. Because of the way the system works, the best time to install *Tripwire* is before you connect the system to the internet. If someone has worked their way into your system before the software has been installed, then *Tripwire* will just end up making sure that any back door the attacker has created stays in place.

Setting up *Tripwire* is a lengthy process, and somewhat beyond the scope of this tutorial, but you can find more information about the process at **www.alwanza.com/howto/linux/tripwire.html** and **www.tripwire.com**.

Keep an eye on your servers

Possibly the most important thing to do when considering security for your servers is to make sure that you keep an eye on what's going on. This process is generally known as auditing and, if done correctly, can ensure that issues are resolved before they have the opportunity to become blown out of proportion.

While we've already used *Nessus* to audit the security from outside the server and mentioned using *Tripwire* to audit the integrity of files on the system, there are many other auditing tools available. However, we've found that the best way of auditing a system is to read the log files – or least get a program to do that for you (more on that in a moment).

Servers tend to generate a lot of information about what's happening. If you have a look in your **/var/log** folder, you'll find a variety of different log files, ranging from the system log to *Apache* access logs. These logs provide plenty of useful information, but figuring out what's worth reading and what's not can be a tedious job.

This is where *Logwatch* comes in – it's a utility that reads your log files, and can send you daily emails about the most interesting parts. These emails may be long, but they will let you know when things aren't going to plan. They can also keep you informed if one of your users starts trying to access things that they shouldn't be, giving you the time to address the issue before they cause a problem.

Under Debian or Ubuntu, the installation and configuration of *Logwatch* is simple. However, to receive the information as emails in HTML format, you'll need to alter the output method, output format, and the email address you're sending the information to. You can change these options by editing **/usr/share/logwatch/default.conf/logwatch.conf** so that:

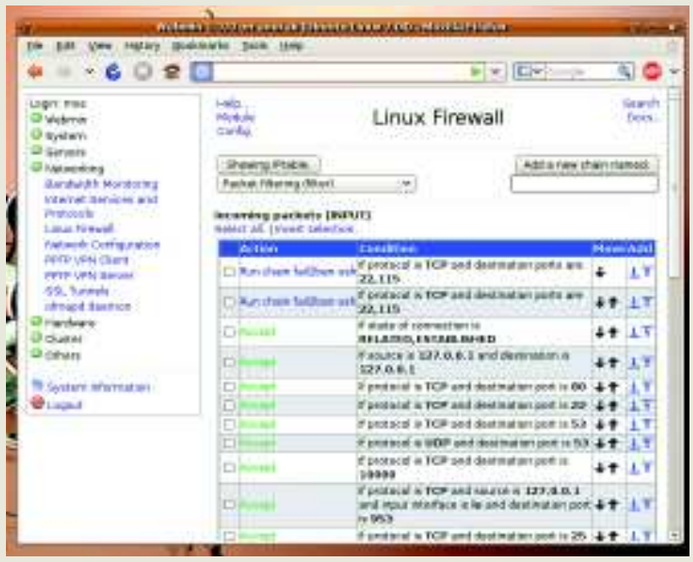
```
Output = mail
Format = html
MailTo = youraddress@yourdomain.net
```

This will send the reports to your email inbox, meaning you can stay up to date with server news wherever you are. LXF

A command line alternative

In this tutorial, we've spent some time teaching you how to write firewall rules on the command line. There are, however, alternatives to this approach. Our favourite (and one that we tend to use on a regular basis) is *Webmin*, a tool that enables you to manage your server through your web browser. If you've installed *Webmin* on your server, you can find the firewall options under Networking > Firewall.

If you've understood the basics of creating rules for a firewall that we've covered here, using *Webmin* should be a walk in the park. To begin with, click Revert Configuration to load the firewall rules that you set up earlier. Then you can modify the rules as appropriate for your new *Webmin* setup. Finally, you should hit Apply Configuration to finalise the modified rules once you've finished changing them.



» **Next month** Get total privacy on the web with the Incognito live CD.