Linux 101: Configuring and managing iptables for better network security

Version 1.0
December 15, 2005

By Chad Perrin

Takeaway

This document describes how important good iptables management is to tight security in a Linux networking environment.

Table of Contents

Basics of firewalls

Every new Linux user will at some point start wondering about installing a firewall application on this strange new operating system. Eventually everyone also runs across the concept of *iptables*. Early on, one might not know what *iptables* means, or even have heard the term. This is quite a disappointing state of affairs, considering how important good *iptables* management is to tight security in a Linux networking environment.

I'm going to assume you've heard of a firewall and have some vague notion of what that means in relation to computers and networking. As a quick summation, though, a firewall basically provides and enforces rules for allowing or denying network access on specific ports, from or to specific networked computers, and so on. Most Windows users, when they think of a firewall, think of Windows Firewall, ZoneAlarm, Norton Firewall, or what they tend to refer to as a "hardware firewall", such as the many rinky-dink router appliances that can be purchased from Circuit City, Best Buy, and so on.

Windows Firewall and ZoneAlarm (along with a slew of others), sometimes called "software firewalls", are in fact little different in concept from the hardware firewalls. In point of fact, the biggest difference between the concepts is the security that each provides. Because the software firewall is on the local system, it provides a reduced security potential: by the time unauthorized traffic touches the software firewall, it has already touched the system you're trying to protect. That doesn't mean you shouldn't use such a thing, however: it's just an extra layer of security, and used properly it can enhance the overall security of your network. You should never, ever consider it a substitute for a separate (hardware) firewall, however.

Applications such as the Windows Firewall and ZoneAlarm are really pretty low quality, as firewalls go. Even ZoneAlarm Pro (much better than the free version, and staggeringly, incredibly, frighteningly better than the deeply flawed Windows Firewall on Windows XP systems) is not great as a firewall. Norton Firewall is better in some ways than the above, in that it is capable of providing better security, and worse in others, in that it is difficult to configure, hides much of what it's doing even worse than ZoneAlarm (and on a par with Windows Firewall), and in general has the potential to seriously screw things up without ever giving a hint to the user aside from "Oh, it's probably Norton again."

Ultimately, the major problem with all of these popular software firewalls on Windows systems is that they do not operate at a low enough level to provide really significant security. There are a couple of firewall applications for Windows that do provide a more fundamental firewalling capability, making use of Windows kernel socket APIs, but the Windows OS design and driver APIs provide for potential "leakage" that even these Windows socket-layer firewalls (such as the iSafer Winsock Firewall) can be worked around by a clever security cracker, depending on the sort of hardware you're using for network connectivity, what drivers you're using, and so on.

Ultimately, the problem with these Windows-based firewalls is that they're software that sits on top of the OS trying to get the OS to relinquish control of network packet control earlier than it really *wants* to so that the traffic can be filtered effectively.

Free UNIX firewalls

Free versions of UNIX tend to have a much better packet filtering model. Linux, for instance, has the netfilter project, which works on kernel-integrated network traffic filtering. The management system for that, which handles filtering rules for netfilter to apply and enforce, is called *iptables*. The OpenBSD analog to *iptables*, meanwhile, is called *pf*, and there are a number of cited advantages and disadvantages to each in comparison to the other.

In any case, it happens that *iptables* and *pf* both work extremely well as firewalling systems. While I haven't done an exhaustive survey, I'd say that probably at least half of the little hardware firewalls you run across in retail electronics outlets are in fact running a stripped-down embedded Linux kernel with netfilter, some running *iptables* and some running some wacky hybrid thing that replaces *iptables* just to make everything work *differently* somehow—probably to frustrate the efforts of people who would like to have more hands-on control of how the router/firewall appliance is working behind the scenes. Regardless, if you've used a store-bought router/firewall appliance, there's a reasonable chance you've used something running *iptables* for firewalling, even if you've never installed Linux on anything.

Because of the open, modular design of Linux (and other free versions of UNIX, for that matter), kernel-integrated network packet filtering can be easily implemented and has improved over the years. This allows for a very close marriage of the firewalling capability of such OSes with the network interface itself, providing a basically impenetrable security model, in theory.

In theory, of course, theory and practice are the same; in practice, they are not. The security you can get from this security model, in practice, depends on your ability to effectively define firewall rules and the flexibility and functionality of the filtering rules management system—in this case, *iptables*.

There was a predecessor to *iptables* called *ipchains*. From what I've seen thus far, it looks like *ipchains* differed from *iptables* mostly in that it was a little more difficult to configure and manage, and in that it was stateless, whereas *iptables* is stateful. That means is that *iptables* can actually apply firewall rules based on the current state of network traffic: rules can exist that depend upon the amount of traffic you're receiving on a specific port, for instance, rather than simply blocking or opening that port across the board. This makes *iptables* much, much more useful for ensuring system security than *ipchains*. Interestingly enough, ZoneAlarm is also stateful in a very limited fashion, but its statefulness is largely unconfigurable and the benefits of its stateful operation can be circumvented by automated scripting, if the person writing the scripts knows what he or she is doing.

Basics of iptables

It's always a good idea to have a decent *iptables* configuration in place on a given machine, regardless of outside firewalls you may have. Each individual machine, depending on its uses, might have different packet filtering needs. As a result of this, the external firewall device should have a configuration that permits everything the most permissive of your local systems is going to need to do: each individual machine, then, can deny as much of that as it can get away with, without sacrificing its critical functionality.

When you first set up a Linux machine, it should have an *iptables* configuration of some sort already in place. That might consist, in some distributions of Linux, of a pages-long set of complex rules designed to allow you to make use of hundreds of applications and services that you probably won't ever touch: this is the approach that "kitchen sink" distributions like Mandrake have taken over the years. Minimal systems have a tendency to just set everything to "allow", giving an extremely simple, but essentially pointless, configuration with the assumption that the user will do something to change that. This is the equivalent of installing the security software without configuring it to do anything at all. In practice, the kitchen sink approach tends to be no more effective a security measure than the unconfigured approach.

There are user interface tools that can be used to manage your system's security profile in a higher-level, more abstracted, more "user friendly" manner than hacking *iptables* configuration yourself from the shell with individual table definition and rules management commands. For a comprehensive treatment of how *iptables* can be configured directly, you can have a read through the manpage for it simply by entering "man *iptables*" at the command line.

These user interface firewall management tools, on the other hand, include both CLI-capable tools like Bastille, and GUI tools with pretty colors and clicky buttons like KDE's Guarddog. The browser-interface service management system known as Webmin is capable of *iptables* management as well. There are even Linux distributions whose whole purpose is to provide a GUI front-end to *iptables* with a fair amount of configurability, reasonably sane default configurations, and integration of the configuration interface with that for routing services and other functionality commonly implemented on a network firewall box.

I find that it tends to provide a greater understanding of, and thus better ability to manage, network and system security to work with *iptables* directly instead. It is to the task of making CLI *iptables* management simple and easy that I devote this document.

Simple iptables management

When you're managing a single computer for personal use, and you're about to do something you've never done before, it helps to have a good step-by-step system for getting things going easily and simply. It's good to have a system that makes it easy to improve upon the initial setup as you learn more, too. Perhaps most importantly, it's good to know how to undo the damage if you screw something up.

For those reasons, among others, this brief tutorial will focus on *iptables* management by way of three very convenient and useful commands and a text editor of your choosing. Those commands are *iptables*, *iptables*-save, and *iptables*-restore. As for the text editor, my favorite is vim: your mileage may vary.

When you're managing a network of many users (and by "many" I mean pretty much anything more than five, really), you start needing to be able to reduce workload significantly by automation and reducing repetitive actions across multiple computers. A standard *iptables* configuration that can be deployed as-is across all workstations on your network becomes an important part of the process of network administration, to reduce the amount of time you have to spend on getting every system on the network within operating requirement specifications for security and manageability.

It is also for reasons related to easy deployment of a standard configuration, easy enforcement and definition of company security policy, and centralized firewall configuration testing, among other reasons, that this brief tutorial will focus on *iptables* management by use of *iptables*, *iptables*-save, and *iptables*-restore commands—and, of course, a text editor of your choosing.

Backup

The first thing to do, of course, is to save your original *iptables* configuration so that you can revert to it easily at a later time if you run into trouble. Doing this is simple and pretty brainless, but it does require you to make some decisions. You can get a look at your current *iptables* configuration by simply entering "*iptables* -L" at the command line, but that doesn't really offer much other than a glimpse into the current configuration. To actually save the configuration in a way that can be reused later, you need to use the *iptables*-save command. Entering "*iptables*-save", with no arguments, produces output that defines the *iptables* rules that are implemented when netfilter is run on your system. This is exactly the data we want saved.

The first decision you'll have to make is where to save your *iptables* configuration files. One option, which is recommended in a number of *howtos*, is to use (or create) a directory at `/var/lib/iptables` and store your configuration files there. Another is, since you're the root user when configuring *iptables*, to save them in the `/root` directory or some subdirectory of that.

In either case, you'll have to make sure that the directory path and the filename you choose will not be so obscure that you'll later forget them, or forget how to find them. If it ends up in a directory called *iptables*, it might be sufficient to save your previous *iptables* configuration file as `saved.cfg`. If not, you might want to save it as *iptables*.saved or, if you're really attached to three letter filename extensions, perhaps `iptables.bak`.

Assuming for the moment that you're saving the file at `/var/lib/iptables/saved.cfg`, the way you'd save the initial configuration is this: first, navigate to that directory (using a command such as `cd /var/lib/iptables`), then enter the command `iptables-save > saved.cfg`.

In case you're not familiar with it, the > character is an extremely useful and widely used shell operator known as a "redirect". It essentially takes the output of the command on the left side and sends it to the file named on the right side of the redirect operator. There is also a left-facing redirect, <, which I will use later. Essentially, it does the opposite (as one might guess): it takes the contents of the file named on the right-hand side of the redirect, and sends it to the command on the left-hand side as its input. By running `iptables-save > saved.cfg`, you are basically just making a file that contains nothing but the output of the `iptables-save` command.

Later, if you feel you need to undo everything you've done to your *iptables* setup, and get it back to the distribution's default (assuming that's the state it was in before you started mucking with it), you can simply enter `iptables-restore < saved.cfg`, or `iptables-restore < /var/lib/iptables/saved.cfg` if your current working directory is not `/var/lib/iptables`.

Armed with this knowledge, you actually already have most of the information necessary to implement the *iptables* management system I'm describing, though I won't be so cruel as to leave you to figure it out on your own. The point of this, after all, is to make the initial implementation, and later self education, as easy as possible.

Flush

The next thing I do, after saving a backup of my original *iptables* configuration on the new system, is flush my *iptables* setup with:

```
iptables -F -t filter
iptables -F -t nat, and
iptables -F -t mangle
```

The first of these three command strings does not technically require the use of the "-t filter" argument, since the filter table is the default target of the *iptables* command, but it doesn't hurt to be explicit. To find out more about the various tables in an *iptables* configuration, you can refer to the manual pages.

You likely won't have to actually perform this task yourself, though doing so won't hurt anything. The reason I do so at times is that, now and then, I essentially need to start building an *iptables* configuration from scratch and find it easier to do this than start with a template configuration like the one I'm going to provide later on.

This, of course, means that my next step would be to use all the *iptables* rules manipulation knowledge I've gleaned from the manpages and other references over the time I've been working with Linux to produce a worthwhile, secure *iptables* configuration. You'll benefit from that by receiving, free of charge and effort, such a reasonably secure configuration template within the next few paragraphs.

Redirect

After testing that configuration out for a little while, I output it to a file using `iptables-save`. I use a redirect to save the output to a filename such as `saved.std` or `iptables.std`, with "std" being an abbreviation of standard. Thus, I have my standard, generic *iptables* workstation configuration template clearly marked.

Once I have that file saved, I can edit it to contain the *iptables* rules I desire for that specific workstation at my leisure, and use the *iptables*-restore command to load them. By default, *iptables*-restore will flush your *iptables* before rebuilding them, so you don't have to worry about flushing the tables before using that command to ensure that everything works the way you want it to.

Now, if that `save.std` file of mine is something I will want to reuse for a lot of other computers, I should save a copy of it somewhere convenient. Then, to apply it to a new computer, all I have to do is copy it to the new system and, on that system, run `iptables-restore < saved.std` just as I would to restore my backup of the default configuration on the first system. This completely replaces the *iptables* configuration in use with the new configuration, and even restarting the computer doesn't undo the change. It stays put until you change it again.

Here's an example template (**Listing A**) of a *saved.std* file for you, with some explanation:

Listing A

```
*mangle
:PREROUTING ACCEPT [48436:11233990]
:INPUT ACCEPT [48436:11233990]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [29730:6162034]
:POSTROUTING ACCEPT [29730:6162034]
COMMIT

*nat
:PREROUTING ACCEPT [391:49336]
:POSTROUTING ACCEPT [1793:110951]
:OUTPUT ACCEPT [1793:110951]
COMMIT

*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [1418:147349]
-A INPUT -i lo -j ACCEPT
-A INPUT -m state --state ESTABLISHED -j ACCEPT
-A INPUT -p tcp -m tcp --dport 22 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 631 -j ACCEPT
-A INPUT -p all -s 127.0.0.1 -d 127.0.0.1 -j ACCEPT
-A INPUT -j DROP
-A OUTPUT -o lo -j ACCEPT
-A OUTPUT -p tcp -m tcp --sport 22 -j ACCEPT
-A OUTPUT -p tcp -m tcp --sport 631 -j ACCEPT
COMMIT
```

This *iptables* configuration essentially causes your computer to pretend, first off, that any incoming connections that have not been solicited by you just don't exist. It also does the same for forwarding packets. This is a generally good policy, using the *:INPUT DROP* and *:FORWARD DROP* defaults. Exceptions can be created later for specific ports, addresses, and so on.

Meanwhile, for purposes of ensuring you don't forget to allow something that your user can do, you should probably use an *:OUTPUT ACCEPT* default, operating under the assumption that connections initiated by the user are less likely to be a security threat than those initiated from outside the system. A more secure way to configure it is to use *:OUTPUT DROP* with exceptions defined for behaviors you want to allow, but that can get prohibitively difficult with end-user client systems that must perform a wide variety of networking functions.

As such, though it is possible to create this more secure configuration and to manage it well, such is a task too varied in its requirements from one case to another to be included within the scope of this article. Consider this configuration that I have provided to be the "good enough" solution until you've learned enough more about *iptables* to change the configuration to suit your specific, individual needs. It will provide drastically increased security over most default configurations without interfering with your ability to get work done.

-A INPUT -i lo -j ACCEPT allows your system to accept all incoming requests that originate at your own network adapter. This is useful for things like testing your network configuration by pinging yourself, getting local system mail delivered (like when your computer wants to tell you something broke), and so on.

-A INPUT -m state --state ESTABLISHED -j ACCEPT takes advantage of the stateful packet filtering capabilities of *iptables* to allow you to function quite flexibly with default *DROP* policies for incoming packets. This line basically states that any connection that is initiated by you will be allowed to continue, circumventing the *DROP* policies for incoming packets related to already established connections. Otherwise, you might be able to send data to another server, but never know whether it got there because when the server tried to reply your firewall would just drop the packets without comment.

-A INPUT -p tcp -m tcp --dport 22 -j ACCEPT allows incoming SSH connections. You might want to change that 22 to another, nonstandard port number, and ensure that anyone that is supposed to have remote SSH access to the computer in question knows to use a different port when making such connections.

For a system to which nobody should ever have remote access, you should remove this line from this file before using it, but it's my experience that remote SSH access is one of the tools that makes secure management of a distributed network possible, and thus most people who aren't on stand-alone systems like a home desktop on a network of one computer will probably want to have some kind of remote SSH access to the system possible. When you're more well versed in *iptables* in the future, you might consider replacing this line with several lines that define what sources are valid for attempts to connect to the computer remotely, so that even on a nonstandard port no system cracker is going to be able to use SSH to get in from the wrong source (such as an incorrect IP address, et cetera), but that's well beyond the scope of this article.

-A INPUT -p tcp -m tcp --dport 631 -j ACCEPT allows for connectivity with network printers using the Common Unix Printing System (CUPS). If that's not something you have to worry about, delete this line. More complex, more secure *iptables* rules than this can be used to do the same thing, but this is a decent starting point.

-A INPUT -p all -s 127.0.0.1 -d 127.0.0.1 -j ACCEPT is another "allow me to talk to myself" line.

Based on the INPUT explanations I've just given, the OUTPUT lines should be fairly self-evident, specifically allowing certain network activities involving outbound connections to occur. A lot more can be done with *iptables* to ensure system security, but I'm only aiming to provide a starting point for managing *iptables* configuration. This very short, simple configuration is superior, security-wise, to every single distro-default *iptables* configuration I've ever laid eyes on, despite the fact they're typically a hundred lines long, give or take---except when they're empty or simply nonexistent.

Summary of implementation

To make it simple, here's what you do with this information:

1. Save your current *iptables* configuration by entering the command *iptables-save > /var/lib/iptables/saved.bak* (or something similar for the filename and path) so that you can undo changes later if you wish.
2. Save the *iptables* configuration file I provided above as */var/lib/iptables/saved.std* (or something similar).
3. Configure your current *iptables* configuration using *saved.std* by entering a command like *iptables-restore < /var/lib/iptables/saved.std*.

Then, if you later decide you want to make changes to one system, but not others, you might copy *saved.std* to *saved.loc*, for instance. You can then edit the *iptables* configuration arguments in the new file to reflect the new configuration requirements, and then use *iptables-restore* to implement the changes on the local system.

For large networks, push scripts or network node cron job pull scripts can be used to place any new versions of the *saved.std* file on those machines, they can even be implemented automatically—though I recommend not allowing local scripts on workstations to alter *iptables* configuration, as this might introduce security vulnerabilities. Rather, a push script run from a central location on your network that places the new configuration file and runs *iptables-restore* might be a better option for cutting down on the amount of administrative overhead.

This is all just the beginning of a serious company-wide security policy relating to firewalls on local systems. More can be done, and in larger networks more really must be done to reduce the amount of work involved. From here, however, you can definitely make a good start on the next best thing to an impenetrable network.

Have at it, and good luck.

Additional resources

- TechRepublic's **Downloads RSS Feed** XML
- Sign up for our Downloads Weekly Update newsletter
- Sign up for TechRepublic's Linux NetNote newsletter
- Check out all of TechRepublic's free newsletters
- Linux 101: How to set up Linux on a personal computer
- 10 things you should do to a new Linux PC before connecting to the Internet
- Lock IT Down: Deploy a robust firewall at minimal cost with IPCop

Version history

Version: 1.0
Published: December 15, 2005

Tell us what you think

TechRepublic downloads are designed to help you get your job done as painlessly and effectively as possible. Because we're continually looking for ways to improve the usefulness of these tools, we need your feedback. Please take a minute to drop us a line and tell us how well this download worked for you and offer your suggestions for improvement.

Thanks!

—The TechRepublic Downloads Team