Introduction to Firewalling on Linux

by Robby Workman
http://slackware.com/~rworkman/
rworkman@slackware.com
http://rlworkman.net/

Presented at:
Southeast Linux Fest
Clemson University
Hendrix Student Center
http://www.southeastlinuxfest.org/

June 13, 2009

General Overview of Firewalls

Q. Just what is a firewall?
Traditionally, a firewall is a machine that inspects the network traffic attempting to go through it and either allows or denies that traffic based on preconfigured policies

- It could be an application-level filter
  - Checks whether/when specific applications and/or users are allowed to connect to other networks
  - May intercept everything coming into and going out of a network
  - Examples — proxy servers (squid), various "software" firewalls on Windows machines (such as ZoneAlarm).
- It could be a "simple" packet filter
  - Inspects each packet entering or leaving a network and determines whether it is allowed based on a set of rules
  - Iptables on linux, pf on OpenBSD are examples
  - Modern packet filters have some more advanced capabilities, such as network address translation, (NAT), altering packet headers, and deeper inspection of packet contents

Overview of the Linux packet filter framework

The packet filter framework on Linux is divided into two parts:

- Netfilter/Xtables — the kernel-space portion
- iptables — the user-space portion

Generally speaking, we tend to refer to them collectively as just “iptables”.

Kernel configuration

Use your distribution's kernel

Seriously.
If you insist upon building a custom kernel, please do not report "bugs" in your Linux distribution until you have reproduced them using the provided kernel. Ignoring this advice is *not* a good way to create a good impression with your distribution's maintainer and/or support team.

If you do build a custom kernel, you will almost surely want to configure all of the Netfilter options as modules (instead of statically compiling it into the kernel), and select **everything** except for the things marked as "deprecated" or "obsolete". Loading or unloading modules is much easier than rebooting when you are troubleshooting or needing additional functionality.

Tables

- ***filter* table**

  for doing the actual packet filtering. This is the default table if you do not specify one when entering rules.

- ***nat* table**

  for rewriting packet source and/or destination

- ***mangle* table**

  for altering packet headers and/or contents

- ***raw* table**

  for avoiding connection tracking, the **NOTRACK** target can be used

Built-in chains

- **INPUT chain**
  present in the *mangle* and *filter* tables. Only packets terminating on localhost traverse this chain.
- **OUTPUT chain**
  present in the *raw*, *nat*, *mangle* and *filter* tables. Only packets originating on localhost traverse this chain.
- **FORWARD chain**
  present in the *mangle* and *filter* tables. Only packets that neither originate nor terminate at the local host traverse this chain.

Built-in chains (cont'd)

- **PREROUTING chain**
  present in the *raw*, *nat* and *mangle* tables. Packets traverse this chain **before** a routing decision is made by the kernel.
- **POSTROUTING chain**
  present in the *nat* and *mangle* tables. Packets traverse this chain **after** a routing decision is made by the kernel.

Basic iptables syntax

Add or delete a rule

```
iptables [-t table] -[AD] chain rule-spec [options]
```

Examples:

```
iptables -t filter -A INPUT -p tcp --dport 22 -j ACCEPT
iptables -D INPUT -p tcp --dport 22 -j ACCEPT
```

Note the `-A` option means “append” — the rule is added to the **end** of the chain.

Basic iptables syntax (cont'd)

Insert a rule into a chain

```
iptables [-t table] -I chain [rulenum] rule-specs [options]
```

Example:

```
iptables -I INPUT 2 -p tcp --dport 110 -j ACCEPT
```

This inserts a rule to accept incoming TCP traffic on port 110 directly before the existing rule number 2.

Basic iptables syntax (cont'd)

Delete a rule from a chain by rule number

```
iptables [-t table] -D chain [rulenum] [options]
```

Example:

```
iptables -D INPUT 2
```

This deletes the rule number 2. Note that you would need to use `iptables --line-numbers -L` to get the number.

Basic iptables syntax (cont'd)

Flush (delete) all rules from a chain

```
iptables [-t table] -F chain [options]
```

Examples:

```
iptables -t filter -F INPUT
iptables -t nat -F POSTROUTING
```

You can also add the -Z switch to zero the packet counters as well. Note that all chains in the specified table will be flushed if you do not specify a chain, and remember that the default chain is **filter** if one is not specified.

Basic iptables syntax (cont'd)

Set the default chain policy

```
iptables [-t table] -P chain target [options]
```

Example:

```
iptables -t filter -P INPUT DROP
```

The chain policy sets the default action to take on the packet if it does not match any of the rules in the chain it traverses.

Basic iptables syntax (cont'd)

Create a custom chain

```
iptables [-t table] -N chain
```

Example:

```
iptables -t filter -N State
```

This creates a custom chain called *State* in the **filter** table. You would jump to it with something like this:

```
iptables -t filter -A INPUT -j State
```

Basic iptables syntax (cont'd)

Delete a custom chain

```
iptables [-t table] -X chain
```

Examples:

```
iptables -t filter -X State
```

This deletes the custom *State* chain we just created.

Note that there must not be any other rules that jump to a custom chain in order to remove it.

Planning

- What is the intended purpose of the box?
  (workstation, router/gateway only, multi-purpose router/gateway plus other services, etc.
- If it is going to be a router/gateway, are there any services inside the local network that need to be available to the outside?
- Do you need to do egress filtering (filter packets **leaving** the local network)? Note that this is not as simple as it sounds, and generally speaking, if you have to ask for help, you do not need it yet.

Setting chain policy

- Remember that a chain's policy decides what happens to packets when they "fall off" the chain; that is, if a packet does not match any of the rules that it sees, the chain policy is applied to it.
- Whether you should do a default **ACCEPT** or **DROP** policy depends on your needs, but generally speaking, **DROP** policy is the better option for **filter** table chains (except perhaps **OUTPUT**), and **ACCEPT** policy is better on other tables' chains.

Rule order

- The order of rules is very important. Rules are applied to the packets in the order in which they were added (within the context of each individual table and chain).
- As an example, if you append a rule to the **filter** table's **INPUT** chain to **DROP** packets on port 22, and then append another rule to **ACCEPT** packets on port 22 from a specific IP address, the packets will still be dropped because they will match the **DROP** rule before they match the **ACCEPT** rule. The first matching rule "wins".

Rule order (cont'd)

- The last bit about "first matching rule wins" has an exception though: if the matching rule has a "non-terminating" target, then the packet will continue on to the next rule in the chain.
- Some examples of non-terminating targets are the **LOG**, **ULOG**, and **NOTRACK** targets, as well as **MARK** and any other target suitable for the *mangle* table.
    - Note that the **ULOG** target, while better than the **LOG** target (IMHO) for routine logging of packets, requires the userspace `ulogd` package to be installed.

Sample workstation ruleset

First, let's set our default chain policies:

```
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT ACCEPT
```

Since this is a workstation, you probably do not want to bother with filtering packets that are leaving (hence the **OUTPUT** policy of **ACCEPT**), and since all packets will be terminating on the workstation itself, there won't be any need for using the **FORWARD** chain.

Sample workstation rulset (cont'd)

Now that we have set our chain policies, we need to remove any existing rules from our ruleset.

```
iptables -F
```

Since we did not specify a table or chain, this rule defaults to use the **filter** table and **all** chains in that table. This would do the same thing:

```
iptables -t filter -F INPUT
iptables -t filter -F OUTPUT
iptables -t filter -F FORWARD
```

Sample workstation ruleset (cont'd)

Now, let's allow all traffic on the loopback interface:

```
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT
```

The `-i` flag sets the incoming interface ("**lo**" in this case for loopback), and `-o` sets the outgoing interface. These are only valid in chains where they make sense; for example, there is no incoming interface in the **OUTPUT** chain.

Sample workstation ruleset (cont'd)

We entered these rules in the previous slide:

```
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT
```

Since we set our **OUTPUT** policy to **ACCEPT**, we really do not need the **OUTPUT** rule above, but we add it just to be safe (since we might later change our **OUTPUT** policy).
The loopback interface is how the machine talks to itself, so it should always be allowed or *Bad Things* will happen.

Sample workstation ruleset (cont'd)

Next, we will tell the kernel to always accept incoming traffic that belongs to established connections, and traffic that is related to established connections:

```
iptables -A INPUT -i eth0 -m conntrack \
  --ctstate ESTABLISHED,RELATED -j ACCEPT
```

This assumes that **eth0** is the interface which is connected to the internet.

Sample workstation ruleset (cont'd)

We entered this rule in the previous slide:

```
iptables -A INPUT -i eth0 -m conntrack \
  --ctstate ESTABLISHED,RELATED -j ACCEPT
```

Note that we did not specify a protocol (-p flag) or source/destination address/port. Generally speaking, if something is not specified, then "any" is implied. Be careful with that — you will get an error if you try to pass incompatible options to iptables (example follows).

Sample workstation ruleset (cont'd)

We entered this rule earlier:

```
iptables -A INPUT -i eth0 -m conntrack \
  --ctstate ESTABLISHED,RELATED -j ACCEPT
```

The **RELATED** state is exactly what the name implies — it is for packets that are not part of an established connection, but are related to it. Examples are FTP data transfers and ICMP error packets.

Sample workstation ruleset (cont'd)

As mentioned in a previous slide, if something is not specified, then "any" is implied. One example of incompatible options to iptables is:

```
iptables -A INPUT --dport 22 -j ACCEPT
```

Since no protocol is specified, this is equivalent to `-p all` — however, some protocols (ICMP, for example) do not have ports; therefore, this line is invalid and will result in an error:

```
iptables v1.4.3.2:  Unknown option '--dport'
Try 'iptables -h' or 'iptables --help'
```

Sample workstation ruleset (cont'd)

At this point, we have a functional and secure “firewall” on our workstation. However, we might want to allow ssh connections from remote machines, so we will add this rule:

```
iptables -A INPUT -i eth0 -p tcp --syn --dport 22 \
  -m conntrack --ctstate NEW -j ACCEPT
```

This will allow new packets on port 22 with the SYN flag set.

Loading rulesets at boot

We have a secure and functional ruleset now, but sooner or later, we will have to reboot our workstation. We really do not want to have to type all that in again, so now what?

On Slackware, we place the iptables commands in `/etc/rc.d/rc.firewall` — if that file exists and is executable, it will be run during boot from `/etc/rc.d/rc.inet2`.

Usage hints

- As much as possible, organize your rules so that most of your traffic will be matched by earlier rules in the ruleset. System resource usage is minimized by decreasing the number of rules that a packet hits before it matches.
- Custom chains can be useful to decrease the number of rules a packet has to hit. For example, you can create separate chains for each protocol — there is no need to test UDP traffic against rules that only apply to TCP traffic.

Usage hints (cont'd)

- If you have a dynamic ip address and are building a router/gateway machine, you should use the **MASQUERADE** target instead of the **SNAT** target to rewrite the outgoing packets' source addresses prior to leaving your network. The **MASQUERADE** target does the same thing as the **SNAT** target, but it adds a small amount of overhead in monitoring the interface. If you have a static ip address, use the **SNAT** target.

Advanced configuration

- In order to forward packets across interfaces, the value of `/proc/sys/net/ipv4/ip_forward` must be set to 1.
- iptables is NOT a daemon! It is not something that you "start" and "stop" — regardless of what some distributions' init scripts might imply. The iptables userspace tool simply manipulates packet processing rules in kernelspace.

Advanced configuration (cont'd)

- While Slackware's `/etc/rc.d/rc.firewall` script's format uses actual iptables invocations to load rules into the kernel, some distributions instead use **iptables-restore** to automatically load a complete ruleset generated from **iptables-save**.
- If you have a very large set of rules, you will probably want to consider using the combination of **iptables-save** and **iptables-restore** instead, as it is generally faster — we will discuss why on the next slide.

Advanced configuration (cont'd)

- When using the **iptables** binary to add rules to the kernel, the process works something like this:
  1. read entire ruleset from kernel
  2. add new rule to ruleset
  3. load modified ruleset into kernel
  4. repeat steps 1-3 for each new rule
- with **iptables-restore**, the entire ruleset is loaded into the kernel in one pass, resulting in a much faster load time.

Other resources

- Netfilter Home Page
  `http://netfilter.org/`
- Oskar Andreasson's iptables Tutorial
  `http://iptables-tutorial.frozentux.net/`
  mirror (my site): `http://iptables.rlworkman.net`
- Daniel de Graaf's Home Page
  `http://danieldegraaf.afraid.org/info/iptables/`
- Book: "Linux Firewalls" by Robert Zeigler and Steve Suehring (available on amazon.com)

Credits and acknowledgments

- Thanks to Robert Zeigler, whose second edition of “Linux Firewalls” was invaluable in my learning.
- Thanks to Oskar Andreasson for the iptables tutorial — it was invaluable in closing some gaps in my understanding.
- Thanks to both the Slackware team and the guys on cardinal (`http://cardinal.lizella.net/`) for tolerating my incessant rambling while writing this.
- Finally, BIG thanks to Jan Engelhardt for reviewing this for technical accuracy and rewriting it in latex.

About the author

Robby Workman is a high school Physical Science teacher who lives in a rural area outside Tuscaloosa, Alabama, USA with his wife and daughter. He has been a Linux and Slackware user since July 2004, and a member of the Slackware development team since January 2007. He also is a founding member and current admin of the SlackBuilds.org project, which was created in July of 2006.

Feedback and suggestions are welcome at any of the following addresses:
`rworkman@slackware.com`
`rworkman@slackbuilds.org`
`rw@rlworkman.net`

A copy of this presentation can be found at:
`http://rlworkman.net/SELF-2009/`