

Introduction to Netfilter and Iptables
Alabama LUG Fest
22 September 2007

Presented by Robby Workman
<http://rlworkman.net/>

The packet filter framework on linux is divided into two parts: netfilter and iptables (though we generally refer to them collectively as iptables)

netfilter: kernel-space

iptables: user-space

Kernel configuration

Use your distribution's kernel. Really. Seriously. If you insist upon building a custom kernel, please don't report "bugs" in your distribution until you've reproduced them with the stock kernel. Ignoring this advice is NOT in the top ten ways to stay on your distribution maintainer's good side... ;-)

If you do build a custom kernel, build the netfilter stuff modular (rather than statically compiled into the kernel), and go ahead and build all of it except for the deprecated and obsolete stuff. When you need new functionality or experience a "gremlin" problem, it's much easier to load/unload a module than to reboot.

Tables: filter (default), nat, mangle, raw

ALL filtering should be done in the filter table

nat table is for rewriting packet source and/or destination

mangle table is for altering packet headers and/or contents

raw table is for avoiding connection tracking; must be used with the NOTRACK target

Chains: PREROUTING, POSTROUTING, INPUT, OUTPUT, FORWARD

The chains listed above are built-in; note that custom chains can be created

All chains are not present in all tables

raw table: PREROUTING, OUTPUT

nat table: PREROUTING, POSTROUTING, OUTPUT

mangle table: PREROUTING, POSTROUTING, INPUT, OUTPUT, FORWARD

filter table: INPUT, OUTPUT, FORWARD

INPUT chain is for packets terminating on the local host

OUTPUT chain is for packets originating on the local host

FORWARD chain is for packets that neither originate nor terminate on the local host

PREROUTING chain is for rules that you want applied **before** a routing decision is made

POSTROUTING chain is for rules that you want applied **after** a routing decision is made

Chain POLICY - Accept all by default and drop known bad stuff, or deny all by default and allow known good stuff? Which one is better is up to you - I recommend a default deny policy, which would be the DROP policy on incoming packets.

Rule order: it matters - rules are applied sequentially in the order in which they were added to the kernel (within the context of each individual table/chain). In other words, if a rule says to accept packets on port 22 (SSH), but an earlier rule says to deny them, the early rule wins.

Note: there is such a thing as a non-terminating target - if a packet matches a rule which contains a non-terminating target (such as the LOG or ULOG targets), then it will continue moving down the chain until it either matches a rule with a terminating target or "falls off" the chain to hit the chain POLICY you defined.

Plan, Plan, Plan - before you start writing any rules, figure out what you want to allow on the box.

What services do you want available to the outside world?

If this is a gateway box, what ports do you want forward to hosts inside the LAN?

Do you want to do egress filtering? ** this is not as simple as it sounds **

Adding/removing firewall rules with iptables

Iptables is NOT a daemon!!! You don't "start" and "stop" it, regardless of what your init scripts might lead you to believe. The iptables userspace tool simply manipulates packet processing rules in kernel space.

With that said, the procedure for "starting" or "stopping" iptables is distribution dependent; some distributions use /etc/init.d/iptables, others use /etc/rc.d/rc.firewall, and there are probably some others out there too.

In Slackware, /etc/rc.d/rc.firewall is used. This script is a series of iptables commands that insert rules into the kernel. This is a good enough approach for relatively small rulesets.

In some others (Debian?), the ruleset is loaded into the kernel via iptables-restore(8) from a file generated by iptables-save(8).

Note that the iptables-{save,restore} combination is preferable, as it is MUCH more efficient than adding the rules to the kernel one-by-one. With iptables-restore(8), the entire ruleset is dropped into the kernel at once, whereas using iptables(8) to add them one by one works something like this:

1. Read entire ruleset from kernel
2. Add new rule to ruleset
3. Load modified ruleset into kernel
4. Repeat #'s 1-3 for each rule added

Syntax:

```
iptables [-t table] -[AD] chain rule-specs [options]
iptables [-t table] -I chain [rulenum] rule-specs [options]
iptables [-t table] -D chain rulenum [options]
iptables [-t table] -[LFZ] [chain] [options]
iptables [-t table] -N chain
iptables [-t table] -X [chain]
iptables [-t table] -P chain target [options]
```

Random Hints

To forward packets from across interfaces, you must have /proc/sys/net/ipv4/ip_forward set to "1"

Generally speaking, if you don't specify something in a rule, that implies "any" -- in other words, if you don't specify a source address, then you have the equivalent of **anywhere**.

Make sure you specify a protocol match (-p tcp) for rules involving source port (--sport) and/or destination port (--dport) matching - if you don't specify a protocol, it implies **any** protocol. Since some protocols don't have a concept of ports, the rule is invalid as written.

If you have a dynamic ip address, use the MASQUERADE target instead of the SNAT target to handle rewriting your packets' source address prior to leaving your network. The MASQUERADE target does exactly the same thing, but it adds a small amount of overhead in monitoring the interface. If you have a static ip address, use the SNAT target.

Try to arrange your rules so that the the earlier rules will catch the large majority of your traffic. The fewer rules that a packet has to hit before the kernel makes a decision what to do with it, the fewer system resources will be used.

Custom chains can be useful to cut down on the number of rules a packet will hit before it reaches a terminating target.

- * TCP, UDP, ICMP chains

Just because something can be done with iptables does not mean that it **should** be done with iptables...

- * Browsing internal http servers

Helpful Resources

Netfilter Homepage -- <http://netfilter.org>

Oskar Andreasson's Iptables Tutorial -- <http://iptables-tutorial.frozentux.net/iptables-tutorial.html>

Daniel de Graaf's page -- <http://danieldegraaf.afraid.org/info/iptables/>

“Linux Firewalls” -- book by Robert Zeigler and Steve Suehring (available on Amazon.com)