BECOME A PATRON (https://www.patreon.com/bePatron?u=10705728)

LINUX JOURNAL
(/)

Taming the Wild Netfilter

Security (/tag/security)

by David A. Bandel on August 31, 2001

For those of you who have taken the plunge and upgraded from kernel 2.2.X (or even 2.0.X) to 2.4.X, congratulations. If, like a number of folks, you're running some form of firewall using either ipchains or ipfwadm, your scripts may work fine. But sooner or later you're probably going to want to upgrade.

In the 2.4.X kernels, Rusty Russell, the Linux packet-filter guru, and his crew of coders have implemented Netfilter into the kernel. Netfilter is the replacement for ipchains or ipfwadm. Fortunately, Netfilter permits you to keep using ipchains or ipfwadm until you can come to grips with iptables by adding a compatibility layer via a kernel module that permits these older packet filters to run. But Netfilter has so many exciting new additions, you'll want to convert those rules as soon as possible. One word of caution, though, if you load the ipchains or ipfwadm modules, you can't load ip_tables (and vice versa). So it's all or nothing. After reading this article, however, making the change should be easy.

For those new to packet filtering, ignore the ipchains translations and use the iptables examples. While not all ipchains commands and options will be translated to iptables, this text should provide a good idea about how to construct a packet-filter firewall by translating ipchains commands into iptables commands.

The reason you'll want to upgrade to Netfilter is because it, unlike ipchains or ipfwadm, is stateful. What this means is it can track connections and permit incoming responses to outgoing requests without creating gaping holes in the firewall. The connection tracking opens a specific, temporary hole for responses and only from the contacted server. We'll see how this works later. The drawback is that with connection tracking in use, Netfilter will need to use a little more memory because the connections are tracked in RAM. So your 4MB 386-16 may no longer be up to the job, depending on your filtering requirements.

Background

The actual Netfilter implementation is broken into two parts, the kernel portion known as Netfilter and the userland tool that interfaces with Netfilter and creates the rulesets, iptables. Both are required to implement your packet-filtering firewall.

First, we'll concentrate on the kernel portion. Netfilter includes support for IPv4 and IPv6. It does not, however, filter any other protocols, so your firewall should not run IPX, AppleTalk or any other protocol that might be used to circumvent iptables rules. Similarly, you must not enable the kernel fast-switching option. This item is one of the last ones in the network options section of the kernel's configuration menu. The code permits fast switching at a low level in the IP stack. The Netfilter code resides at a much higher level, so fast switching effectively bypasses Netfilter completely.

Kernel Configuration

In order to get started using Netfilter, you'll need to have your kernel compiled for Netfilter support. Most distributions include this support by default, so a quick test is in order. If you can insert the module ip_tables, then you won't need to worry about this section. As root, run the command

```
modprobe ip_tables
```

Then run

```
lsmod | grep ip_tables
```

If ip_tables show up, you're in good shape. If not, don't worry, rebuilding a kernel is extremely easy. This text won't cover the complete kernel rebuild process, but many resources are available to help you through this step. If you find you need to rebuild your kernel, the Sidebar will provide you some guidance on what to include for a complete Netfilter-enabled kernel.

Enabling Netfilter in Your Kernel
(/files/linuxjournal.com/linuxjournal/articles/048/4815/4815s1.html)

Netfilter Modules

If you've built and installed all the modules, all modules will auto-install when a rule is entered except ip_tables, ip_nat_ftp and ip_conntrack_ftp. These can be loaded either manually or as part of your iptables startup script.

A full build and installation of Netfilter produces a large number of modules, but most firewalls will only use a few. The modules that aren't loaded aren't taking up memory, so don't worry about what you don't use.

Getting and Installing iptables

Your distribution may have installed iptables, and it almost certainly did if your kernel has Netfilter support. But if you want the very latest, you'll probably have to get it from the Netfilter site. Netfilter is available at netfilter.filewatcher.org (http://netfilter.filewatcher.org). Download and compile it according to the instructions in the INSTALL file. The following instructions assume the kernel sources are in /usr/src/linux. If not, adjust the following instructions appropriately. If you need to run

```
make pending-patches KERNEL_DIR=/usr/src/linux
```

or

```
make patch-o-matic KERNEL_DIR=/usr/src/linux
```

then you'll need to recompile your kernel before continuing. Otherwise, you can ignore these two commands. In general, the patch-o-matic is for users with special needs and is of interest to the average user.

After running

```
make KERNEL_DIR=/usr/src/linux
```

run

```
make install KERNEL_DIR=/usr/src/linux
```

You're now ready to use iptables.

The iptables Command Line

The iptables command line is broken down into as many as six parts. The first part is the iptables command, which will not be discussed further. The second part is the table specification, and the chain name is the third part. The fourth part is the rule specification, which is the part of the command to match against the IP or ICMP header, but also could be a rule number in some incantations. The fifth part is the target, and the sixth part is the target option. The general command line, then, looks like

```
iptables [-t table] -ACDI CHAIN rule-specification å -j TARGET [tar
```

The above does not hold true for all incantations but is the most common. You will also find the -L command useful. It will be covered in this article, along with several other command-line variants.

Tables and Chains

Netfilter has three tables you need to concern yourself with: filter, nat and mangle. This article, whenever it shows an iptables incantation, will always have a table specified. However, if no table is specified, the filter table is assumed. Therefore, if you do not specify a table and your rule fails, try putting a table specification in and try again.

Each table has certain chains available to it. User-created chains will belong to one and only one table. You will see that some built-in chains belong to more than one table, but this is only true for built-in chains. You cannot mix chains from other tables within user-created chains.

The filter table is the basic packet-filter table and has the built-in chains INPUT, FORWARD and OUTPUT. Rules added to user-created chains created in the filter table can only contain targets valid in the INPUT, FORWARD or OUTPUT chains. Packets traversing the filter table will pass through one and only one of INPUT, FORWARD or OUTPUT. The INPUT chain will only be traversed if the packet's destination is the local system. The FORWARD chain will only be traversed if the packet is passing through the local system and bound for another system. And the OUTPUT chain is traversed only by packets originating on the local system with an external destination. Only one chain will be traversed by any given packet. This is in contrast to ipchains that always used the input and output chains and also used the forward chain if the packet was passing through.

Note that in the previous paragraph, the iptables chain names were capitalized, while the ipchains chain names were not. This was deliberate and reflects a change in syntax.

The nat table performs network address translation. Built-in chains for nat are PREROUTING, POSTROUTING and OUTPUT. Each chain permits a particular target. The PREROUTING accepts the DNAT target, and the remaining chains accept the SNAT target. More on these targets shortly.

The mangle table is used to change (mangle) information other than the IP address in the header. It can be used to mark the packets, change type of service (TOS) or change time-to-live (ttl) information.

Rules

The rule-specification portion of each iptables command is the heart of the command. By properly crafting your rule, you can select exactly which packets to which a rule should apply. This selection criteria can be as general or as specific as you need. In most cases, you'll want to make sure specific criteria come before more general criteria.

That said, I'll not belabor rules too much here other than to add that multiple options, some of which carry their own options, can be strung together. But you do need to ensure that rules make sense. For example, don't specify an output interface in an input chain or that rule will never match anything. The syntax may allow you to construct impossible rules, but it's not a good idea. If you're in doubt about a particular rule, you can always make the target a log target then send traffic that matches that rule to see if it does in fact trigger. If you need a tool to test your rules, take a look at SendIP (www.earth.li/projectpurple/progs/sendip.html (http://www.earth.li/projectpurple/progs/sendip.html)).

Targets

Netfilter has four built-in targets: ACCEPT, DROP, QUEUE and RETURN. The DROP target replaces ipchain's DENY target. All other targets used are based on modules that load as a target. These include REJECT, LOG, MARK, MASQUERADE, MIRROR, REDIRECT and TCPMSS. Terminal targets, such as ACCEPT, DROP, REJECT, MASQUERADE, MIRROR and REDIRECT, terminate a chain. A LOG target does not terminate a chain. LOG also neither ACCEPTS, REJECTS nor DROPS a packet, so the chain continues to be traversed. Thus the ipchains -l option is now another target but a nonterminating one. The rest of the chain will be traversed until it hits the policy rule.

The policy rule is the overall rule for the chain. If your FORWARD chain contains a policy of DROP, and no rules above match in the chain, the packet will terminate when it hits the policy rule. Your policy rule can only be one of the built-in targets. You cannot have REJECT as a policy rule.

Examples

Before we look at examples, let's set some things up. Scripts are nice, and one that runs from rc.local (wherever that is on your system) is always good. So let's write an rc.iptables script as part of our example to implement Netfilter during bootup (see Listing 1). (Note: all iptables rules will start $IPT and should continue unbroken to the end of the line, i.e., the next command. Rules should not be broken on a line.)

Listing 1. rc.iptables Script (/files/linuxjournal.com/linuxjournal/articles/048/4815/4815l1.html)

We've set up some variables to start, stopped forwarding traffic through the system, then inserted some modules. The ip_tables module allows us to start writing rules. The ip_nat_ftp module is necessary if we are using the NAT table to do SNAT or MASQUERADE (covered below) to allow active FTP. The ip_conntrack_ftp allows connection tracking of FTP. This module automatically loads the ip_conntrack module because it is dependent on ip_conntrack. If you don't need or want the ip_conntrack_ftp module, but want to make sure your firewall does IP defragmentation (a good idea), you can substitute ip_conntrack for ip_conntrack_ftp. Let's continue with our script:

```
for i in filter nat mangle
do
$IPT -t $i -F
$IPT -t $i -X
done
```

The above lines will flush (-F) all rules in the chain shown as the argument to -F. Since no chain name was given as an argument, -F will flush all chains. Note that this has to be done for each table, hence the loop. If you aren't using a particular table, you can delete it from the for statement. As each loop starts, you will note that a new module has been loaded: first the iptable_filter module, then the iptable_nat module and finally the iptables_mangle module. If you remove mangle from the loop, the iptables_mangle module will not be loaded. Unused modules can be removed at your leisure.

Under ipchains, you would have used something like

ipchains -F -X

to accomplish the same thing.

Before we continue, let's assume the following setup: a home user and three systems that access the Internet through our firewall/workstation PC. Access is via dial-up (ppp0). If your setup is different, substitute your external interface for ppp0. The systems are not offering any services outside their own network, which is on eth0. We do, however, want all systems inside to be able to surf the Web. For this first example, we'll assume a dynamic IP from an outside ISP [see Listing 2, rc.iptables.dynamic, at ftp.linuxjournal.com/pub/lj/listings/issue89/4815.tgz (ftp://ftp.linuxjournal.com/pub/lj/listings/issue89/4815.tgz)].

Since our firewall PC is also a workstation, it will be originating and terminating its own traffic. While not a good idea for a firewall (these should be dedicated systems) on a home network, we really don't need a dedicated firewall. With this in mind, remember that one difference between iptables and ipchains is in the INPUT, FORWARD and OUTPUT chains. In ipchains, packets traversing the FORWARD chain came from INPUT and went through FORWARD to OUTPUT, so we could locate our rules in the INPUT chain and be safe for packets going to the FORWARD chain. The iptables implementation only uses INPUT for the local system and FORWARD for the other systems. In our case we need identical rules in each of the FORWARD and INPUT chains. To prevent duplicating a lot of rules, let's create a user chain called tcprules and call it from both INPUT and FORWARD. Continuing our script then:

```
$IPT -t filter -N tcprules
```

The ipchains equivalent of this rule would be the same, excluding the -t filter option of the incantation:

```
ipchains -N tcprules
```

Now a little Netfilter magic. We want to prevent someone from connecting to our systems from the outside but permit connections to the outside to be established by our users. The following rules make use of Netfilter's stateful capability:

```
$IPT -t filter -A tcprules -i ppp+ -m state --state ESTABLISHED,REL
$IPT -t filter -A tcprules -i ! ppp+ -m state --state NEW -j ACCEPT
$IPT -t filter -A tcprules -i ppp+ -m state --state NEW,INVALID -j
```

If you want to do something similar in ipchains, the closest you could come is to deny syn packets to the ppp+ interface:

```
ipchains -A input -i ppp+ ! -y -j DENY
```

A couple of quick notes at this point. First, the "!" negates whatever follows it. So **! ppp+** is the same as specifying all other interfaces (in the case of our home user, lo and eth0). The "+" on the end of the ppp tells Netfilter this rule applies to all ppp interfaces. As for the ESTABLISHED, RELATED, NEW and INVALID arguments, they are more than they appear to be. ESTABLISHED permits traffic to continue where it has seen traffic before in both directions. ESTABLISHED obviously applies to TCP connections but also to UDP traffic, such as DNS queries and traceroutes as well as ICMP pings. In fact, packets are first checked to see if the connection exists in the connection tracking table (/proc/net/ip_conntrack). If so, the chains aren't run, the original rule is applied and the packets pass. In some cases, Netfilter is faster than its predecessor because of this check. The RELATED argument covers a multitude of sins. This argument is applied to active FTP, which opens a related connection on port 20, but also applies to ICMP traffic related to the TCP connection. The NEW argument applies to packets with only the SYN bit set (and the ACK bit unset). The INVALID applies to packets that have invalid sets of options, as in an XMAS tree scan.

The above rules permit the internal systems to pass new connections internally and externally, but don't permit incoming new or invalid packets.

Since we're going to be masquerading our network behind our firewall, we'll need to set up a masquerading rule. This process is very similar to the one used in ipchains. Assuming our internal network is 192.168.0.0/24, we'll use the following rule:

```
$IPT -t nat -A POSTROUTING -o ppp+ -s 192.168.0.0/24 -d 0/0 -j MASQ
```

One difference ipchains users should note is the use of -o ppp+ instead of -i ppp+. This is because with ipchains, we dealt with interfaces as -i. With iptables, -i stands for input interface, and -o stands for output interface. If you mistakenly use -i ppp+ in the line above, no masquerading will take place. In fact, the rule should never match at all.

Let's finish our script by implementing the tcprules above where they are needed, implementing our filter-table policies and turning ip_forwarding back on:

```
$IPT -t filter -A INPUT -j tcprules
$IPT -t filter -A FORWARD -j tcprules
$IPT -t filter -P INPUT DROP
$IPT -t filter -P FORWARD DROP
echo 1 > /proc/sys/net/ipv4/ip_forward
```

By default, the filter-table policies are all ACCEPT. Given the stateful nature of Netfilter, this affords sufficient protection, unlike ipchains. If you think about the stateful rules we've implemented, you'll see that no harm should come with the default policy; in fact, nothing should ever arrive at the default policy. But some believe that it's always better to play it safe, so we can drop anything not addressed already. Note that the policy doesn't have a target per se, so we don't use -j. This is also why only built-in targets can be policies—policies aren't really targets. If you really want REJECT as your policy, you'll need to add something like the following:

```
$IPT -t filter -A tcprules -i ppp+  -j REJECT --reject-with icmp-ho
```

The above rule applies to all packets coming in on your ppp interface. Make sure this rule is your last rule because nothing will pass this rule.
While you could make the rule

```
$IPT -f filter -A tcprules -j REJECT --reject-with icmp-host-unreac
```

you will find that your internal hosts are also blocked because this rule will apply to all interfaces (something you should keep in mind with policies as well).

At this point, I should mention that, unlike ipchains, a target match does not necessarily terminate a chain. For example, if you use the following rule in tcprules:

```
$IPT -t filter -I INPUT -p ICMP -m icmp --icmp-type echo-request -m
```

the next rule (which may or may not also match this packet) will be processed. If a rule match does not drop, reject, accept or queue the packet and is not a return, the chain will continue.

The ipchains crowd will note that under iptables, LOG is a target by itself rather than a simple -l option following the target, as with ipchains. This permits some flexibility, as can be seen from the above rule. The limit match prevents someone malicious from flooding your logs. The LOG target can send a prefix with its message so that it's easy to grep from the logs.

Before we move on, all ipchains users know you can view the chains by using something like

```
ipchains -L -n
```

This will show you the chains. With Netfilter, we need to look at the chains, table by table:

```
iptables -t filter -L -n
iptables -t nat -L -n
iptables -t mangle -L -n
```

You can add a -v to get even more information on each rule:

```
iptables -t filter -L -nv
```

Next is an example to demonstrate the new SNAT and DNAT. If you understand the mangle table and its use, you probably don't need this article and will be e-mailing me about any errors you've noticed.
In this example, we'll assume our home user has a broadband connection and is using eth0 for the internal network and eth1 for the external network. The script begins the same as before, but when we get to the tcprules, we're going to do something a little different. Here, we're going to assume the user has a static IP of 209.127.112.17, which gives us more leeway [see Listing 3, rc.iptables.static, at ftp.linuxjournal.com/pub/lj/listings/issue89/4815.tgz (ftp://ftp.linuxjournal.com/pub/lj/listings/issue89/4815.tgz)]. In fact, we can also assume the user has his own domain name and is running his own e-mail server on port 25, which is located on an internal system with IP 192.168.0.2 (the firewall is 192.168.0.1). His DNS entry points to 209.127.112.17 as his mail server, as shown in Listing 4.

Listing 4. DNS Entry for Mail Server (/files/linuxjournal.com/linuxjournal/articles/048/4815/4815l4.html)

The first two tcprules are the same as in the first example. But before we drop all other connections, we accept connections on port 25. Then, in the nat table, we take the port 25 connection and forward it to another internal host on the same port. You can do this for all your connections. Remember that if you do this forward for DNS, you need to forward UDP as well as TCP. In fact, unless someone outside is going to do zone transfers, you can drop the TCP part and only pass UDP.

Notice that now, instead of using MASQUERADE as a target for outgoing connections, we're using SNAT. In case you wondered, the S stands for source, which is what is being changed. In the case of DNAT, we changed the destination of the packet. The argument --to-source can take a range of IPs, so your firewall can look like several hosts. If you have five usable IPs from your ISP, you can use all five as outgoing. Then you can point different services to different IPs and have up to five systems behind your firewall answering DNS queries, hosting web sites, accepting mail, etc. Netfilter will also allow you to do rudimentary load balancing. When a range of destinations is used with DNAT, the system with the least number of connections (not necessarily the system with the lightest load) gets the connection.

About the only other thing you may want to know to get started is how to increase the maximum number of connections tracked. This number is arrived at by default depending on the amount of RAM your system has. However, the number is conservative and can be increased. You can find this number this way:

```
cat /proc/sys/net/ipv4/ip_conntrack_max
```

On my system with 256MB of RAM, the number is 16376.
Concluding Remarks
Using Netfilter's stateful rules, you can actually increase the security of your home system with less effort by making judicious use of its connection tracking. Many more options are also available to you. This article scratches only the surface. I recommend you make use of Rusty's Unreliable Guides available on the Netfilter site (mentioned earlier).

For home users with simple needs, keep your firewall simple. I do not recommend most firewall tools and scripts because they layer unnecessary complexity into your firewall. If you don't understand a rule, don't implement it. The first three stateful rules (using the -m state rule) will keep you in good stead. If an attacker has already been in and compromised a system, the rules won't help. They also won't protect you against e-mail-based trojans, but they will protect against direct attacks. I suggest, if you don't use IRC, you log and drop outgoing IRC connections:

```
$IPT -j filter -I tcprules -p tcp --destination-port 6667 -j LOG --
$IPT -j filter -I tcprules 2 -p tcp --destination-port 6667 -j DROP
```

Also, if you don't need anyone entering your network, don't open any ports (as we did in our second example). This article did not discuss how to segregate your network properly to isolate internet-accessible systems from trusted internal systems. If you require this level of complexity, and your risk assessment asks for it, it might be time to call for knowledgeable help.

David A. Bandel (dbandel@pananix.com) is a Linux/UNIX consultant currently living in the Republic of Panama. He is coauthor of Que Special Edition: Using Caldera OpenLinux.

Load Disqus comments (https://www.linuxjournal.com/article/4815#disqus_thread)

You May Like

Extending Landlocked Processes (/content/extending-landlocked-processes)

Zack Brown (/users/zack-brown)

(/content/extending-landlocked-processes)

At Rest Encryption (/content/rest-encryption)

Kyle Rankin (/users/kyle-rankin)

(/content/rest-encryption)

The LJ Password Generator Tool (/content/lj-password-generator-tool)

Dave Taylor (/users/dave-taylor)

(/content/lj-password-generator-tool)

Data Privacy: Why It Matters and How to Protect Yourself (/content/data-privacy-why-it-matters-and-how-protect-yourself)

(/content/data-privacy-why-it-matters-and-how-protect-yourself)

Petros Koutoupis (/users/petros-koutoupis)

Powered By

PIA

private internet access®

xjournal)

of the global Open

S-SERVICE)

S (/RSS_FEEDS)

ERS
TTERS)

DISE
W.LINUXJOURNALSTORE.COM

(/SUBS/CUSTOMER_SERVICE)

(http://www.privateinternetaccess.com/pages/buy-vpn/linuxjournal) (/ABOUTUS)

(/CONTACT)