Kernel Korner

The Hidden Treasures of iptables

With these powerful add-ons for iptables you can match strings or port ranges in iptables rules or even create a tar pit for network abusers.

by Chris Lowth

Linux's iptables allows powerful firewalls to be implemented at a minute fraction of the cost of many commercial offerings. Basic iptables firewalls are packet filters, which means they inspect the network communications flowing through them a packet at a time and make choices about how those packets are handled. Simple configurations can be used to drop certain packets and accept others. The choice about which policy to apply to a particular packet commonly is made on the basis of the IP address and port number to which it has been sent and the direction in which it is traveling. iptables also can use state information to make more-informed choices based on the state of the connection to which the packet relates. This is known as connection tracking.

A simple and highly effective firewall configuration blocks inbound TCP/IP connection packets and UDP exchanges initiated from the public Internet while allowing outbound ones over translated addresses. This gives users free access to the outside world while protecting them from unwelcome intrusions. Such configurations are a bit simplistic and may need additional filters to be truly useful, but the basic concept is straightforward.

iptables has a lot more to offer than these simple packet-filtering criteria. Some of the extras are fairly well known and even may make their way into some off-the-shelf Linux distributions, but some lesser-known features are worthy of investigation. These are the hidden treasures I intend to point you toward in this article. It would take a book to describe all the possible features and options associated with them, so all I do here is flag their existence and put you on the path of exploration.

Introducing the POM

Netfilter has two groups of components, the kernel and user-mode pieces. The user-mode group consists of the iptables and related utilities, libraries, manual pages and scripts. The kernel components are patches to existing kernel sources and a number of extra modules.

Applying patches to a system as large and complex as the Linux kernel can be a daunting task to the uninitiated, and the road is littered with traps and potential blind turns. A bad or incompatible patch readily can produce a kernel that doesn't compile, or worse, doesn't boot. The Netfilter team has sought to resolve these difficulties by providing us with a robot guide, POM, or Patch-o-matic. POM is a collection of patches and a script for applying them to your kernel, and it's a joy even for a relative novice to use.

The kernel patches included with POM are classified into a number of groups according to their history and quality. Some of them are base patches needed in every iptables/Netfilter installation. Others are optional or experimental extras that provide interesting features, some of which I describe in this article. These are the promised hidden treasures, what the POM documentation describes as “Maybe broken, Maybe cool extensions.”
Running POM is simple; download the latest Patch-o-matic tarball from the directory /pub/patch-o-matic on ftp.netfilter.org, restore it on your system and run the following command while logged in as root. Make sure to give the correct kernel source directory name as the value of the KERNEL_DIR parameter:

```
KERNEL_DIR=/usr/src/linux-2.4 ./runme extra
```

From there, installation is interactive and more or less self-explanatory.

### Bits of String

The string module probably is the most widely used extra from the POM trove. It allows packets to be matched against strings occurring anywhere in their data payload. This module has all sorts of uses but needs to be applied carefully so as not to be overzealous. One possible use is to block the downloading of ELF executables from the Web. We can set up a filter that identifies Web return traffic by looking for TCP/IP packets coming from the Internet-facing interface with a source port of 80. If we know that an ELF file starts with hex character 7f followed by the letters ELF (which it does), we can use the string match to search for this sequence. Non-ASCII characters can be embedded in the string by using the pipe symbol to enclose them, so we use `|7F|ELF`. Assuming that the Internet-facing network interface is eth0, the command is:

```
iptables -A FORWARD -i eth0 -p tcp --sport 80 -m string --string '|7F|ELF' -j DROP
```

The syntax for embedding hex characters into the string was introduced in iptables 1.2.8. If you are using an earlier version, you need to resort to trickery. For example:

```
--string "`dd if=/bin/ls bs=4 count=1 2>/dev/null`"
```

takes the first four characters of /bin/ls, which is an ELF file that contains the string we want.

We can expand this example by declaring that we trust the content from 192.168.0.5 and, therefore, don't want to apply the filter to that server. This is done easily by adding an inverted match on the IP address, like this:

```
iptables -A FORWARD -i eth0 -p tcp ! -s 192.168.0.5 --sport 80 -m string --string '|7F|ELF' -j DROP
```

This example has a couple of problems that highlight the issues with the string match module. First, the rule matches any packet that contains this sequence anywhere in the data, not only at the start of the file. This means the rule could match false positives and block packets we didn't intend. Second, if the string we are looking for actually is split over two adjacent packets, it isn't matched. The module needs the entire string to appear in a single packet.

So, the string module is useful but basic. It doesn't allow for case-insensitive matches or for the location of the string to be specified, nor does it allow strings to be found when split over multiple packets in the data stream. There is plenty of scope for an extended version of this module to be written.
Fewer Rules with mport

The mport extension allows a single rule to specify a number of port numbers and ranges using an extended syntax. Without mport, the iptables command can specify either a single port or a range of adjacent ports in a single command. With mport in place, the syntax allows more complex constructs. For example, we could permit X terminals, Web and mail with a single command, like this:

```
iptables -A INPUT -p tcp -m mport \
--dports 80,110,21,6000:6003 -j ACCEPT
```

Without using mport, this would have to be specified using four separate commands:

```
iptables -A INPUT -p tcp --dports 80 -j ACCEPT
iptables -A INPUT -p tcp --dports 110 -j ACCEPT
iptables -A INPUT -p tcp --dports 21 -j ACCEPT
iptables -A INPUT -p tcp --dports 6000:6003 \
-j ACCEPT
```

Using a single rule in place of four offers a potential performance advantage because packets passing through the system require less processing. It also makes the maintenance of the rules files easier because services requiring identical processing can be grouped together easily. As you probably guessed, mport is short for multiple ports.

Time-Based Rules

The time module allows rules to introduce the time of day and the day of the week into matching logic. Example uses would be to allow access to personal Web sites only during the lunch hour or to divert Web traffic to a secondary server during routine maintenance periods. The following example renders the Web service inaccessible between the hours of 4 and 6:30am on Fridays, presumably for system maintenance:

```
iptables -A INPUT -p tcp -d 80 -m time \
--timestart 04:00 --timestop 06:30 --days Fri \
--syn -j REJECT
```

It is worth noting that the -timestart, -timestop and -days options all must be specified. So if you want a rule
that is not day-of-week dependent, you must specify all seven day names; you can't omit the option.

**Getting Bogged Down—Tar Pits**

You really don't want to wander into a tar pit if you value your life or appreciate changes of scenery. They are nature's equivalent of fly paper; come too close and you won't leave in a hurry. The TARPIT component of iptables is the networking equivalent: if you are unwise enough to establish a TCP/IP connection to a port that is a tar pit, you will find it hard to close the connection and release the used system resources for future use.

To achieve this tar pit state, iptables accepts the incoming TCP/IP connection and then switches to a zero-byte window. This forces the attacker's system to stop sending data, rather like the effect of pressing Ctrl-S on a terminal. Any attempts by the attacker to close the connection are ignored, so the connection remains active and typically times out after only 12–24 minutes. This consumes resources on the attacker's system but not the Linux server or firewall running the tar pit. You could use the following iptables command to pass packets to the pit:

```
iptables -A INPUT -p tcp -j TARPIT
```

You probably don't want to use conntrack and TARPIT on the same system, particularly if you anticipate catching a lot of flies with this particular brand of fly paper. Each stuck connection consumes conntrack resources.

One way to confuse potential attackers is to make your Linux system look like a Microsoft Windows machine by causing the netbios ports to respond to port scans. Then pass any connection requests to the tar pit. This has the effect of wasting attackers' time while they sense a possible opening and try to gain access. They will be frustrated by long timeouts and an apparently buggy target. Rules such as the following produce this result:

```
iptables -A INPUT -p tcp -m tcp -m mport --dports 135,139,1025 -j TARPIT
```

Another possibility is to TARPIT all ports except the ones you genuinely want to use. This again leads outsiders to see every port as open and waste time attempting to gain access. Moreover, a configuration like this prevents tcpdump from correctly determining the operating system running on the server. In this example, we allow Web and e-mail traffic and bog down everything else:

```
iptables -A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
iptables -A INPUT -p tcp -m tcp --dport 25 -j ACCEPT
iptables -A INPUT -p tcp -j TARPIT
```

You can find an interesting real-life story of how TARPIT and string helped one particular system administrator (not me) at [www.spinics.net/lists/netfilter/msg17583.html](http://www.spinics.net/lists/netfilter/msg17583.html).
Randomizing

The random match module matches packets based on nothing more than a random choice. You can tune the logic by setting the probability that a packet is matched anywhere between 0% and 100% of the time. Example applications include simulating a faulty connection or server or distributing load across multiple mirrored Web servers. The example below distributes Web traffic among three servers. The first rule sends 33% of the connections to the server at 192.168.0.100. The next 33% is sent to 192.168.0.101 and the last third catches the remainder and passes them to 192.168.0.102:

```bash
iptables -t nat -A PREROUTING -i eth0 -p tcp \
  --dport 80 --syn -m random --average 33 \ 
  -j DNAT --to-destination 192.168.0.100:80
iptables -t nat -A PREROUTING -i eth0 -p tcp \
  --dport 80 --syn -m random --average 50 \ 
  -j DNAT --to-destination 192.168.0.101:80
iptables -t nat -A PREROUTING -i eth0 -p tcp \
  --dport 80 --syn -j DNAT \ 
  --to-destination 192.168.0.102:80
```

As before, this assumes that eth0 is the Internet-facing interface.

A Lot More Where They Came From

Dozens of treasures can be dug up and enjoyed. I have described a small handful here, but there are plenty more. Simply running the runme script and reading the patch descriptions as they are displayed is one way of getting an idea of what is available. Here are a few more examples of what you can find:

- Connection tracking for RSH, MMS (media streaming), PPTP, Quake, RPC and Talk.
- Extended support for configuration and status information access through the /proc filesystem.
- Extended support for IPv6 features.
- Manipulation of options, TTL and more in IP packets.
- Finer control over NATed connections.
- Control over limits on quota and bandwidth usage.
- Anti-OS fingerprinting logic and port-scan detection.
- Connection marking (and mark testing).

**Sources of Wisdom**

The patches added with POM don't add their descriptions to the `iptables` man page, so we need to turn elsewhere for documentation. The basic syntax used to invoke these extensions can be displayed using the `iptables` built-in help facility. For example, `iptables -m random -help` gives the usual help message but with the random module's parameters displayed at the end. The same technique can be applied to the other modules.

You also can refer to the module help files held in the Patch-o-matic directory structure. The file for random is `base/random.patch.help`. Similar files can be found for the other patches.

Finally, make use of the Netfilter Web site, [www.netfilter.org/patch-o-matic](http://www.netfilter.org/patch-o-matic), which contains a description of each of the POM patches.

**Installing New iptables Modules**

The majority of `iptables` extensions have two parts, a patch to the Linux kernel and a configuration helper library for use by the `iptables` user-space program. The detailed procedure for adding a POM module to the kernel and the `iptables` tools is outlined at [www.lowth.com/howto/add-iptables-modules.php](http://www.lowth.com/howto/add-iptables-modules.php). In summary, the steps we need to take are bring your system up to date; download the latest Patch-o-matic sources; patch the kernel using the `runme` script; recompile and install the patched kernel; and recompile and install the `iptables` software.

**Conclusion**

We have seen that Linux's Netfilter provides an excellent set of features for building effective firewalls, but not all of these features are installed by default on many Linux distributions. The Patch-o-matic software allows administrators to extend the base functionalities of their firewalls through an automated approach to patching the Linux kernel.

To finish, take this thought with you: we have seen that `iptables/Netfilter` has a number of exciting possibilities hidden away from initial inspection. The chances are high that the same is true for other packages. This is part of the joy of open-source software; nothing is truly hidden. Everything that exists is waiting there for the skilled seeker to find.

**Acknowledgement**

To Jane Lowth for drawing the figures of Tux.

Chris Lowth works as a consultant for Intercai Mondiale ([www.intercai.co.uk](http://www.intercai.co.uk)), a UK-based telecommunications, IT and business consultancy. He designs security software and network management (OSS) solutions and attempts to play the guitar. Chris can be contacted at chris@lowth.com.