nft_beginners_guide Documentation

Release 0.1

Aljoscha Lautenbach

Apr 06, 2020

CONTENTS

Table of contents

CHAPTER

ONE

INTRODUCTION

1.1 Why this guide?

While there are a number of beginner's guides for iptables and netfilter, I could not find one that uses `nftables`. Every `nftables` tutorial I have found assumes familiarity with `iptables`. So if you want to teach students basic firewall concepts with Linux, you either have to accept that there are no introductory texts or start with `iptables` before moving on to `nftables`. This simple guide attempts to bridge this gap. In many ways this is simply a filtered collection of information from various manpages and other sources in a format that should be beginner friendly.

Disclaimer

I am in no way an expert on either nftables, Netfilter or any of the affiliated projects, so there is a chance that mistakes have crept in. If you find any mistakes, please let me know and I will fix them as soon as possible.

I would also like to point out that the `nft` manpage is an excellent source of information, and much of the basic concepts presented here are more or less direct reproductions from it. If you are interested in more advanced use cases, I would recommend you start with the manpage.

Since I borrow heavily from various sources, I have done my best to add appropriate attribution wherever possible.

1.2 What is a packet filter?

A packet filter, colloquially also known as a firewall, allows an administrator to match network packets with a set of rules that specify how to handle matching packets. For instance you can drop packets, count and log packets, or allow a certain subset of packets.

A simple rule could be to block all traffic to TCP port 22 except for a specific IP address. Similarly, an administrator could block all outgoing packets to the network 10.0.0.0/8, or if the machine routes packets between networks A and B, the administrator could add a rule that allows all HTTP traffic from network A to B but blocks all other traffic from A to B.

The above are just some simple examples and Netfilter in particular has many more capabilities and features. We will demonstrate several more as we go along.

1.3 What are nftables, nf_tables and Netfilter?

To quote from the `nft` manpage, "*nft is the command line tool used to set up, maintain and inspect packet filtering [. . .] rules in the Linux kernel, in the nftables framework. The Linux kernel subsystem is known as nf_tables, and 'nf'*

stands for Netfilter."

Netfilter has been the kernel's packet filtering subsystem since Linux 2.4.x, and for a long time `iptables`, `ip6tables`, `arptables` and `ebtables` have been the main user space utilities to administer the various packet filtering rules. However, their interfaces are less flexible than some people would like, and maintaining larger rulesets can be somewhat cumbersome. Therefore, `nftables` was developed and saw its first official release in Linux 3.13.

`nft` has a completely different syntax than `iptables`, which might take some getting used to for people accustomed to `iptables`, but `nft` has some nice features that make it worth your while. For instance, `nft` can handle all use cases that previously required four separate tools (`iptables`, `ip6tables`, `arptables` and `ebtables`). So instead of having to write two separate rules for a content match in IPv4 and IPv6, `nft` can achieve the same match with a single rule.

For more examples why `nftables` makes your life easier, see: Why you will love nftables

My main two complaints about `nftables` in its current state are (1) that the documentation is severely lacking (one of the reasons why I am writing this tutorial), and (2) that the online help of `nft` could be improved a lot, meaning the positional arguments to `nft` are not always obvious.

CHAPTER

TWO

HOW TO ENABLE OR RESET NFTABLES WITH SYSTEMD

I am assuming that you already have nftables installed, otherwise install it with your method of choice.

Most GNU/Linux distributions use systemd as their init system now, so I will only cover systemd here. In all likelihood, nftables is already enabled, but if it is not, you can enable it as root with:

```
systemctl enable nftables.service
```

If you have changed the config file and want to reload the ruleset, you can run as root:

```
systemctl restart nftables.service
```

CHAPTER

THREE

SIMPLE EXAMPLE

Before we get into the basic concepts, let us start with a simple example, just to give you an idea how `nftables` works.

Let us start by listing all existing rules:

```
nft list ruleset
```

On a Debian-based system, this might be the output you are seeing:

```
table inet filter {
      chain input {
              type filter hook input priority filter; policy accept;
      }

      chain forward {
              type filter hook forward priority filter; policy accept;
      }

      chain output {
              type filter hook output priority filter; policy accept;
      }
}
```

We will discuss this output in detail later, for now note that you have a table called `filter` that uses a chain called `input`.

If you get no output, that means you have no default configuration set up, and you have to add the table and chain first, before you can do anything with it:

```
nft add table inet filter
nft add chain inet filter input { type filter hook input priority 0 \; policy accept\;
↪}
```

Let us start by adding a rule to block all traffic to port 21 to the input chain:

```
nft add rule inet filter input tcp dport 21 counter drop
```

If you are familiar with `iptables`, note that no interface has been specified, so this rule applies to every network interface on the system. I originally used port 22 here, but I did not want anyone to accidentally lose access to their server if by an off-chance they happened to try this on a remote server with only ssh access...

You could simply test that this rule works by opening a local socket on port 21 (assuming you have no ftp server running), and try to connect:

```
# open a listening socket on port 21
nc -l 21
# in a new terminal, try to connect to port 21 on localhost
nc localhost 21
```

If everything is set up correctly, the client will not be able to connect to the server, so anything you type on the client will not reach the server. If you are not familiar with `netcat`, you might want to try it without any firewall rules first to see what is supposed to happen.

This example was inspired by theurbanpenguin and quidsup on youtube, who each have a nice, short `nft` video tutorial:

- RHCSA 8 - Native Nftables Firewalls on Red Hat Enterprise Linux 8
- Getting Started with nftables Firewall in Debian

CHAPTER

FOUR

BASIC CONCEPTS

4.1 Address families

Nftables has support for six address families (reproduced from the manpage):

ip IPv4 address family.

ip6 IPv6 address family.

inet Internet (IPv4/IPv6) address family.

arp ARP address family, handling IPv4 ARP packets.

bridge Bridge address family, handling packets which traverse a bridge device (switch).

netdev Netdev address family, handling packets from ingress.

In this guide, we mostly focus on *ip* and *inet*.

Note: For some commands, such as chain creation, the address family is optional, but if it is not explicitly specified the address family defaults to **ip**!

4.2 Hooks

Hooks allow you to specify at which stage during the packet processing in the network stack you would like to match your rules. The different address families have different hooks, but in this guide we will focus only on IPv4/IPv6/Inet and Netdev.

4.2.1 IPv4/IPv6/Inet Hooks

There are five hooks you can use for the IPv4, IPv6 and inet address families:

Hook	Description
prerouting	All packets entering the system are processed by the prerouting hook. It is invoked before the routing process and is used for early filtering or changing packet attributes that affect routing.
input	Packets delivered to the local system are processed by the input hook.
forward	Packets forwarded to a different host are processed by the forward hook.
output	Packets sent by local processes are processed by the output hook.
postrouting	All packets leaving the system are processed by the postrouting hook.

On a regular workstation, you can cover the most common use cases with just the *input* and *output* hooks, and if you use VMs or containers potentially the *forward* hook.

4.2.2 Netdev Hooks

For the netdev address family, there is a single hook:

Hook	Description
ingress	All packets entering the system are processed by this hook. It is invoked before layer 3 protocol handlers and it can be used for early filtering and policing.

4.3 Tables and chains

Tables are containers for chains, and chains are containers for rules, as you have seen in the *simple example*.

4.3.1 Tables

Every table is identified by its name and its address family, so if you have the same table name for different address families, those are separate tables. Try for instance:

```
nft add table ip foo
nft add table ip6 foo
nft add table arp foo
nft list ruleset
```

You should now have two new tables, one for IPv4 and one for ARP, both named *foo*.

You can also list all existing tables with:

```
nft list tables
```

If you do not want the ARP table anymore, you can simply reverse the command with *delete*:

```
nft delete table arp foo
```

4.3.2 Chains

There are two types of chains:

1. **base chains**, which are hooked into the network stack using the *hooks* described earlier, and
2. **regular chains**, which can be used as jump targets from base chains, allowing for better rule organization.

Every chain must belong to a table, so the minimum command to create a new chain looks as follows:

```
nft add chain foo mychain
```

We have now created a new chain called *mychain* in the table *foo*. But wait, if you followed along, you had two tables named *foo*, one for the address family *ip* and one for *ip6*! So which one was it added to? The attentive reader will remember that if no address family is specified, it defaults to *ip*. So the above command added *mychain* only to the *ip foo* table. You can easily verify this with:

```
nft list ruleset
```

In order to avoid any confusion, it is probably best to always add the address family as well:

```
nft add chain ip6 foo mychain
```

This successfully added *mychain* to the *ip6 foo* table.

You may have noted that we specified no hooks, so *mychain* is a **regular chain**, and is currently unused.

In order to create a **base chain** that hooks into the network stack, you have to add the desired **hook**, a specific **type** and a desired **priority**. We already covered *hooks*. The **type** can be one of *filter*, *nat* and *route*. For our purposes in this beginner's guide, the type will always be *filter*. Finally, as the name suggests, **priorities** allow you to specify a priority to determine in which order the chains are traversed.

Note: Since this is a beginner's guide, I will not elaborate any further on types and priorities, but the interested reader is pointed to the manpage, and for additional background reading to the excellent iptables tutorial, specifically Chapter 6 Traversing tables and chains. Just note that the iptables tutorial is outdated, I would not recommend to read it as standalone documentation, make sure to crossreference it with the information in the `nft` manpage.

In the *simple example* you already saw how to create a base chain:

```
nft add chain inet filter myoutputchain { type filter hook output priority 0 \; }
```

This adds a new chain to the table *inet filter*, hooking onto the output hook. In other words, all IP packets that are created on your system and are about to be sent out will traverse this chain.

4.4 Rules

TBD

CHAPTER

FIVE

CREATING A BASIC CONFIGURATION

There are two main ways of using `nft` to configure your firewall:

1. Using the CLI tool `nft` (or its interactive mode `nft -i`) to create tables, chains and rules as you go along.
2. Using a configuration file that you input to `nft -f` (default config typcially at `/etc/nftables.conf` or `/etc/sysconfig/nftables.conf`) (TODO: look up upstream location).

Obviously you can write a basic configuration file, and make dynamic changes with `nft` as you see fit.

CHAPTER

SIX

COMMON USE CASES

6.1 Blocking ports

TBD

6.2 Blocking specific content

TBD

6.3 Allow established connections

TBD

CHAPTER

SEVEN

LIST OF COMMON COMMANDS

TBD

```
# list all rules
nft list ruleset

# list all IPv4 rules
nft list ruleset ip

# remove (flush) all rules
nft flush ruleset

# remove (flush) IPv$ rules
nft flush ruleset ip
```

CHAPTER

EIGHT

ADVANCED USE CASES

8.1 Port knocking

TODO: basic description of port knocking.

TODO: port knocking example.

8.2 DDoS protection

It is difficult to protect against DDoS attacks, but `nftables` allows you to filter packets as soon as they arrive to avoid using unnecessary resources, thus giving you a little bit more breathing room. You can use the address family `netdev` for this. `netdev` includes all traffic directed at your network interface, just after it was passed to the network stack and before any protocol parsing. See also the nftables Wiki.

TODO: example rules here.

CHAPTER

NINE

ACKNOWLEDGMENTS

A special thanks to the Linux kernel community, as well as the netfilter and nftables communities, for their much appreciated work! I am also grateful for the people around Sphinx for having written a great documentation generator.

Many innovations we see today would not be possible without open source software in general, not to mention the important role it plays for democratic societies. Computing would be very dull without it!

Therefore, I would also like to thank everyone involved in the open source ecosystem in general[1]: please keep up the good work! :)

[1] If I were to list all the free and open source software I use daily we would still sit here tomorrow. However, I cannot resist listing a few projects and communities especially dear to my heart: Debian, Emacs, GNU, Linux, Python, LaTeX, i3 and Mozilla. :)