RTRTR User Manual

NLnet Labs

May 26, 2021

CONTENTS

RTRTR is an RPKI data proxy, designed to collect Validated ROA Payloads from one or more sources in multiple formats and dispatch it onwards. It provides the means to implement multiple distribution architectures for RPKI such as centralised RPKI validators that dispatch data to local caching RTR servers.

RTRTR can read RPKI data from multiple RPKI Relying Party packages via RTR and JSON and, in turn, provide an RTR service for routers to connect to. The HTTP server provides the validated data set in JSON format, as well as a monitoring endpoint in plain text and Prometheus format.

If you run into a problem with RTRTR or you have a feature request, please create an issue on Github. We are also happy to accept your pull requests. For general discussion and exchanging operational experiences we provide a mailing list and a Discord server.

CHAPTER

ONE

INTRODUCTION

For larger networks, RTRTR is an ideal companion to Routinator. For example, it is possible to centralise validation performed by Routinator and have RTRTR running in various locations around the world to which routers can connect.

RTRTR can read RPKI data from multiple RPKI Relying Party packages via RTR and JSON and, in turn, provide an RTR service for routers to connect to. The HTTP server provides the validated data set in JSON format, as well as a monitoring endpoint in plain text and Prometheus format.

CHAPTER

TWO

INSTALLATION

Getting started with RTRTR is really easy by either installing a Debian and Ubuntu package, using Docker, or building from CARGO (Rust's build system and package manager).

2.1 Quick Start

Packages

Docker

Cargo

Assuming you have a machine running a recent Debian or Ubuntu distribution, you can install RTRTR from our software package repository. To use this repository, add the line below that corresponds to your operating system to your `/etc/apt/sources.list` or `/etc/apt/sources.list.d/`:

```
deb [arch=amd64] https://packages.nlnetlabs.nl/linux/debian/ stretch main
deb [arch=amd64] https://packages.nlnetlabs.nl/linux/debian/ buster main
deb [arch=amd64] https://packages.nlnetlabs.nl/linux/ubuntu/ xenial main
deb [arch=amd64] https://packages.nlnetlabs.nl/linux/ubuntu/ bionic main
deb [arch=amd64] https://packages.nlnetlabs.nl/linux/ubuntu/ focal main
```

Then run the following commands:

```
sudo apt update && apt-get install -y gnupg2
wget -qO- https://packages.nlnetlabs.nl/aptkey.asc | sudo apt-key add -
sudo apt update
```

You can then install RTRTR using:

```
sudo apt install rtrtr
```

You can now configure RTRTR by editing `/etc/rtrtr.conf` and start it with `sudo systemctl enable --now rtrtr`. You can check the status with the command `sudo systemctl status rtrtr` and view the logs with `sudo journalctl --unit=rtrtr`.

To run RTRTR with Docker you will first need to create an `rtrtr.conf` file somewhere on your host computer and make that available to the Docker container when you run it. For example if your config file is in `/etc/rtrtr.conf` on the host computer:

```
docker run -v /etc/rtrtr.conf:/etc/rtrtr.conf nlnetlabs/rtrtr -c /etc/rtrtr.conf
```

RTRTR will need network access to fetch and publish data according to the configured units and targets respectively. Explaining Docker networking is beyond the scope of this quick start, however below are a couple of examples to get you started.

If you need an RTRTR unit to fetch data from a source port on the host you will also need to give the Docker container access to the host network. For example one way to do this is with `--net=host`, where `...` represents the rest of the arguments to pass to Docker and RTRTR:

```
docker run --net=host ...
```

If you're not using `--net=host` you will need to tell Docker to expoese the RTRTR target ports, either one by one using `-p`, or you can publish the default ports exposed by the Docker container (and at the same time remap them to high numbered ports) using `-P`:

```
docker run -p 8080:8080/tcp -p 9001:9001/tcp ...
```

Or:

```
docker run -P ...
```

Assuming you have a newly installed Debian or Ubuntu machine, you will need to install rsync, the C toolchain and Rust. You can then install RTRTR:

```
apt install curl rsync build-essential
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
source ~/.cargo/env
cargo install --locked rtrtr
```

Once RTRTR is installed, you need to create a *configuration file* that suits your needs. The config file to use needs to be passed to RTRTR via the `-c` option, i.e.:

```
rtrtr -c rtrtr.conf
```

If you have an older version of Rust and RTRTR, you can update via:

```
rustup update
cargo install --locked --force rtrtr
```

If you want to try the main branch from the repository instead of a release version, you can run:

```
cargo install --git https://github.com/NLnetLabs/rtrtr.git --branch main
```

2.2 System Requirements

When choosing a system to run RTRTR on, make sure you have 1GB of available memory and 1GB of disk space.

2.3 Installing From Source

You need a C toolchain and Rust to install and run RTRTR. You can install RTRTR on any system where you can fulfil these requirements.

2.3.1 C Toolchain

Some of the libraries Routinator depends on require a C toolchain to be present. Your system probably has some easy way to install the minimum set of packages to build from C sources. For example, this command will install everything you need on Debian/Ubuntu:

```
apt install build-essential
```

If you are unsure, try to run **`cc`** on a command line. If there is a complaint about missing input files, you are probably good to go.

2.3.2 Rust

The Rust compiler runs on, and compiles to, a great number of platforms, though not all of them are equally supported. The official Rust Platform Support page provides an overview of the various support levels.

While some system distributions include Rust as system packages, Routinator relies on a relatively new version of Rust, currently 1.45 or newer. We therefore suggest to use the canonical Rust installation via a tool called **`rustup`**.

To install **`rustup`** and Rust, simply do:

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

Alternatively, visit the official Rust website for other installation methods.

You can update your Rust installation later by running:

```
rustup update
```

2.3.3 Building

The easiest way to get Routinator is to leave it to Cargo by saying:

```
cargo install --locked rtrtr
```

The command will build Routinator and install it in the same directory that Cargo itself lives in, likely `$HOME/.cargo/bin`. This means RTRTR will be in your path, too.

2.4 Installing Specific Versions

Release Candidates of RTRTR are also available on our software package repository. To install these as well, add the line below that corresponds to your operating system to your `/etc/apt/sources.list` or `/etc/apt/sources.list.d/`:

```
deb [arch=amd64] https://packages.nlnetlabs.nl/linux/debian/ stretch-proposed main
deb [arch=amd64] https://packages.nlnetlabs.nl/linux/debian/ buster-proposed main
deb [arch=amd64] https://packages.nlnetlabs.nl/linux/ubuntu/ xenial-proposed main
deb [arch=amd64] https://packages.nlnetlabs.nl/linux/ubuntu/ bionic-proposed main
deb [arch=amd64] https://packages.nlnetlabs.nl/linux/ubuntu/ focal-proposed main
```

You can use this command to get an overview of the available versions:

```
apt policy rtrtr
```

If you want to install a Release Candidate or a specific version of Routinator using Cargo, explicitly use the `--version` option. If needed, use the `--force` option to overwrite an existing version:

```
cargo install --locked --force rtrtr --version 0.1.1
```

If you want to try the main branch from the repository instead of a release version, you can run:

```
cargo install --git https://github.com/NLnetLabs/rtrtr.git --branch main
```

CHAPTER

THREE

CONFIGURATION

A configuration file is required for RTRTR to run. It describes which components should be loaded and how they will be connected. The file is in a format call TOML (Tom's Obvious Minimal Language), which is somewhat similar to INI files. You can find more information on the TOML website.

The file's content starts out with a number of optional general parameters:

```
# The minimum log level to consider.
log_level = "debug"

# The target for logging. This can be "syslog", "stderr", "file", or
# "default".
log_target = "stderr"

# If syslog is used, the syslog facility can be given:
log_facility = "daemon"

# If file logging is used, the log file must be given.
log_file = "/var/log/rtrtr.log"
```

RTRTR has a built in HTTP server that provides status information at the **`/status`** path and Prometheus metrics at the **`/metrics`** path:

```
# Where should the HTTP server listen on?
#
# The HTTP server provides access to Prometheus-style metrics under the
# `/metrics` path and plain text status information under `/status` and
# can be used as a target for serving data (see below for more on targets).
http-listen = ["127.0.0.1:8080"]
```

RTRTR uses two classes of components: *units* and *targets*. Units take data from somewhere and produce a single, constantly updated data set. Targets take the data set from exactly one other unit and serve it in some specific way.

Both units and targets have a name — so that we can refer to them — and a type that defines which particular kind of unit or target this is. For each type, additional arguments need to be provided. Which these are and what they mean depends on the type.

At this time, there are only two types of units and one type of target. Each unit and target gets its own section in the config. The name of the section, given in square brackets, describes whether a unit or target is wanted and, after a dot, the name of the unit or target.

Let's start with a unit for an RTR client. We call it `local-3323` because it connects to port 3323 on localhost. You can, of course, choose whatever name you like:

```
[units.local-3323]
```

The type of this unit is `rtr` for an RTR client using plain TCP:

```
type = "rtr"
```

The rtr unit needs one more argument: where to connect to:

```
remote = "localhost:3323"
```

Let's add another RTR unit for another server:

```
[units.local-3324]
type = "rtr"
remote = "localhost:3324"

[units.local-json]
type = "json"
uri = "http://localhost:8323/json"
refresh = 60

[units.cloudflare-json]
type = "json"
uri = "https://rpki.cloudflare.com/rpki.json"
refresh = 60
```

The second unit type is called `any`. It is given any number of other units and picks the data set from one of them. Units can signal that they currently don't have an up-to-date dataset available, so an any unit can skip those and make sure to always have an up-to-date data set.

```
[units.any-rtr]
type = "any"
```

The names of the units the any unit should get its data from:

```
sources = [ "local-3323", "local-3324", "cloudflare-json" ]
```

Whether the unit should pick a unit random every time it needs to switch or rather go through the list in order:

```
random = false
```

Finally, we need to do something with the data: serve it via RTR. This is what the `rtr` target does:

```
[targets.local-9001]
type = "rtr"
```

The `rtr` target can listen on multiple addresses, so the listen argument is a list:

```
listen = [ "127.0.0.1:9001" ]
```

The name of the unit the target should receive its data from:

```
unit = "any-rtr"
```

```
[targets.http-json]
type = "http"
path = "/json"
format = "json"
unit = "any-rtr"
```