Routinator User Manual

NLnet Labs

May 11, 2021

CONTENTS

1 Installation **3**
- 1.1 Quick Start with Debian and Ubuntu Packages . . . 3
- 1.2 Quick Start with Docker . . . 4
- 1.3 Quick Start with Cargo . . . 4
- 1.4 System Requirements . . . 4
- 1.5 Getting Started . . . 5
- 1.6 Building . . . 6
- 1.7 Notes . . . 6

2 Installation Notes **7**
- 2.1 Using Native TLS Instead of Rustls . . . 7
- 2.2 Statically Linked Routinator . . . 7
- 2.3 Platform Specific Instructions . . . 8

3 Initialisation **13**
- 3.1 Performing a Test Run . . . 14

4 Running Interactively **15**
- 4.1 Printing a List of VRPs . . . 15
- 4.2 Validity Checker . . . 17

5 Running as a Daemon **19**
- 5.1 The HTTP Service . . . 19
- 5.2 The RTR Service . . . 20

6 Monitoring **23**
- 6.1 Metrics . . . 23
- 6.2 Grafana . . . 24

7 Configuration **25**
- 7.1 Using a Configuration File . . . 26
- 7.2 Applying Local Exceptions . . . 27

8 Manual Page **29**
- 8.1 Synopsis . . . 29
- 8.2 Description . . . 29
- 8.3 Options . . . 29
- 8.4 Commands . . . 32
- 8.5 Trust Anchor Locators . . . 36
- 8.6 Configuration File . . . 36
- 8.7 HTTP Service . . . 39

8.8 Logging . . . 39
8.9 Validation . . . 40
8.10 Relaxed Decoding . . . 40
8.11 Signals . . . 41
8.12 Exit Status . . . 41

Index **43**

Routinator 3000 is free, open source RPKI Relying Party software written by NLnet Labs in the Rust programming language.

The application is designed to be lightweight and have great portability. This means it can run on any Unix-like operating system, but also works on Microsoft Windows. Due to its lean design, it can run effortlessly on minimalist hardware such as a Raspberry Pi.

Routinator connects to the Trust Anchors of the five Regional Internet Registries (RIRs) — APNIC, AFRINIC, ARIN, LACNIC and RIPE NCC — downloads all of the certificates and ROAs in the various repositories, verifies the signatures and makes the result available for use in the BGP workflow.

It is a full featured software package that can perform RPKI validation as a one-time operation and store the result on disk in formats such as CSV, JSON and RPSL, or run as a service that periodically downloads and verifies RPKI data. Routers can connect to Routinator to fetch verified RPKI data via the RTR protocol. The built-in HTTP server offers a user interface and endpoints for the various file formats, as well as logging, status and Prometheus monitoring.

If you run into a problem with Routinator 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.

You can follow the adventures of Routinator on Twitter and listen to its favourite songs on Spotify.

CHAPTER

ONE

INSTALLATION

Getting started with Routinator is really easy either building from Cargo, installing a Debian and Ubuntu package or using Docker.

1.1 Quick Start with Debian and Ubuntu Packages

Assuming you have a machine running a recent Debian or Ubuntu distribution, you can install Routinator 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, initialise, enable and start Routinator by running these commands. Note that `routinator-init` is slightly different than the command used with Cargo.

```
sudo apt install routinator
sudo routinator-init
# Follow instructions provided
sudo systemctl enable --now routinator
```

By default, Routinator will start the RTR server on port 3323 and the HTTP server on port 8323. These, and other values can be changed in the configuration file located in `/etc/routinator/routinator.conf`. You can check the status of Routinator with `sudo systemctl status routinator` and view the logs with `sudo journalctl --unit=routinator`.

1.2 Quick Start with Docker

Due to the impracticality of complying with the ARIN TAL distribution terms in an unsupervised Docker environment, before launching the container it is necessary to first review and agree to the ARIN Relying Party Agreement (RPA). If you agree to the terms, you can let the Routinator Docker image install the TALs into a mounted volume that is later reused for the server:

```
# Create a Docker volume to persist TALs in
sudo docker volume create routinator-tals
# Review the ARIN terms.
# Run a disposable container to install TALs.
sudo docker run --rm -v routinator-tals:/home/routinator/.rpki-cache/tals \
    nlnetlabs/routinator init -f --accept-arin-rpa
# Launch the final detached container named 'routinator' exposing RTR on
# port 3323 and HTTP on port 9556
sudo docker run -d --restart=unless-stopped --name routinator -p 3323:3323 \
     -p 9556:9556 -v routinator-tals:/home/routinator/.rpki-cache/tals \
     nlnetlabs/routinator
```

1.3 Quick Start with Cargo

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 Routinator and start it up as an RTR server listening on 127.0.0.1 port 3323 and HTTP on port 9556:

```
apt install rsync build-essential
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
source ~/.cargo/env
cargo install --locked routinator
routinator init
# Follow instructions provided
routinator server --rtr 192.0.2.13:3323 --http 192.0.2.13:9556
```

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

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

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/routinator.git  --branch main
```

1.4 System Requirements

When choosing a system to run Routinator on, make sure you have 1GB of available memory and 1GB of disk space. This will give you ample margin for the RPKI repositories to grow over time, as adoption increases.

1.5 Getting Started

There are three things you need to install and run Routinator: rsync, a C toolchain and Rust. You can install Routinator on any system where you can fulfil these requirements.

You need rsync because most RPKI repositories currently use it as its main means of distribution. Some of the cryptographic primitives used by Routinator require a C toolchain. Lastly, you need Rust because that's the programming language that Routinator has been written in.

1.5.1 rsync

Currently, Routinator requires the **`rsync`** executable to be in your path. Due to the nature of rsync, it is unclear which particular version you need at the very least, but whatever is being shipped with current Linux and *BSD distributions and macOS should be fine. Alternatively, you can download rsync from its website.

On Windows, Routinator requires the rsync version that comes with Cygwin – make sure to select rsync during the installation phase.

1.5.2 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, **`apt install build-essential`** will install everything you need on Debian/Ubuntu.

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

1.5.3 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 1.42 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
```

1.6 Building

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

```
cargo install --locked routinator
```

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/routinator.git --branch main
```

If you want to update an installed version, you run the same command but add the `-f` flag, a.k.a. force, to approve overwriting the installed version.

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

1.7 Notes

In case you want to build a statically linked Routinator, or you have an Operating System where special care needs to be taken, such as OpenBSD and CentOS, please refer to the *Installation Notes* section.

CHAPTER

TWO

INSTALLATION NOTES

In certain scenarios and on some platforms specific steps are needed in order to get Routinator working as desired.

2.1 Using Native TLS Instead of Rustls

Routinator by default uses Rustls which in most cases is fine. However, if needed you can instead use your system native TLS implementation with Routinator like so:

2.1.1 Cargo

Build Routinator with the `native-tls` feature enabled

```
git clone --branch vX.Y.Z --depth 1 https://github.com/NLnetLabs/routinator.git
cd routinator
cargo build --release --features socks,native-tls
```

2.1.2 Docker

Specify a `native-tls` image tag when running the container

```
sudo docker run -d --restart=unless-stopped --name routinator -p 3323:3323 \
     -p 9556:9556 -v routinator-tals:/home/routinator/.rpki-cache/tals \
     nlnetlabs/routinator:native-tls
```

2.2 Statically Linked Routinator

While Rust binaries are mostly statically linked, they depend on **`libc`** which, as least as **`glibc`** that is standard on Linux systems, is somewhat difficult to link statically. This is why Routinator binaries are actually dynamically linked on **`glibc`** systems and can only be transferred between systems with the same **`glibc`** versions.

However, Rust can build binaries based on the alternative implementation named **`musl`** that can easily be statically linked. Building such binaries is easy with **`rustup`**. You need to install **`musl`** and the correct **`musl`** target such as `x86_64-unknown-linux-musl` for x86_64 Linux systems. Then you can just build Routinator for that target.

On a Debian (and presumably Ubuntu) system, enter the following:

```
sudo apt-get install musl-tools
rustup target add x86_64-unknown-linux-musl
cargo build --target=x86_64-unknown-linux-musl --release
```

2.3 Platform Specific Instructions

Tip: GÉANT has created an Ansible playbook defining a role to deploy Routinator on Ubuntu.

For some platforms, **`rustup`** cannot provide binary releases to install directly. The Rust Platform Support page lists several platforms where official binary releases are not available, but Rust is still guaranteed to build. For these platforms, automated tests are not run so it's not guaranteed to produce a working build, but they often work to quite a good degree.

2.3.1 OpenBSD

On OpenBSD, patches are required to get Rust running correctly, but these are well maintained and offer the latest version of Rust quite quickly.

Rust can be installed on OpenBSD by running:

```
pkg_add rust
```

2.3.2 CentOS 6

The standard installation method does not work when using CentOS 6. Here, you will end up with a long list of error messages about missing assembler instructions. This is because the assembler shipped with CentOS 6 is too old.

You can get the necessary version by installing the Developer Toolset 6 from the Software Collections repository. On a virgin system, you can install Rust using these steps:

```
sudo yum install centos-release-scl
sudo yum install devtoolset-6
scl enable devtoolset-6 bash
curl https://sh.rustup.rs -sSf | sh
source $HOME/.cargo/env
```

2.3.3 SELinux using CentOS 7

This guide, contributed by Rich Compton, describes how to run Routinator on Security Enhanced Linux (SELinux) using CentOS 7.

1. Start by setting the hostname.

```
sudo nmtui-hostname
Hostname will be set
```

2. Set the interface and connect it.

Note: Ensure that "Automatically connect" and "Available to all users" are checked.

```
sudo nmtui-edit
```

3. Install the required packages.

```
sudo yum check-update
sudo yum upgrade -y
sudo yum install -y epel-release
sudo yum install -y vim wget curl net-tools lsof bash-completion yum-utils \
    htop nginx httpd-tools tcpdump rust cargo rsync policycoreutils-python
```

4. Set the timezone to UTC.

```
sudo timedatectl set-timezone UTC
```

5. Remove postfix as it is unneeded.

```
sudo systemctl stop postfix
sudo systemctl disable postfix
```

6. Create a self-signed certificate for NGINX.

```
sudo mkdir /etc/ssl/private
sudo chmod 700 /etc/ssl/private
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
    -keyout /etc/ssl/private/nginx-selfsigned.key \
    -out /etc/ssl/certs/nginx-selfsigned.crt
# Populate the relevant information to generate a self signed certificate
sudo openssl dhparam -out /etc/ssl/certs/dhparam.pem 2048
```

7. Add in the `ssl.conf` file to `/etc/nginx/conf.d/ssl.conf` and edit the `ssl.conf` file to provide the IP of the host in the `server_name` field.
8. Replace `/etc/nginx/nginx.conf` with the `nginx.conf` file.
9. Set the username and password for the web interface authentication.

```
sudo htpasswd -c /etc/nginx/.htpasswd <username>
```

10. Start Nginx and set it up so it starts at boot.

```
sudo systemctl start nginx
sudo systemctl enable nginx
```

11. Add the user "routinator", create the `/opt/routinator` directory and assign it to the "routinator" user and group

```
sudo useradd routinator
sudo mkdir /opt/routinator
sudo chown routinator:routinator /opt/routinator
```

12. Sudo into the routinator user.

```
sudo su - routinator
```

13. Install Routinator and add it to the `$PATH` for user "routinator"

```
cargo install routinator
vi /home/routinator/.bash_profile
Edit the PATH line to include "/home/routinator/.cargo/bin"
PATH=$PATH:$HOME/.local/bin:$HOME/bin:/home/routinator/.cargo/bin
```

14. Initialise Routinator, accept the ARIN TAL and exit back to the user with `sudo`.

```
/home/routinator/.cargo/bin/routinator -b /opt/routinator init -f --accept-arin-rpa
exit
```

15. Create a routinator systemd script using the template below.

```
sudo vi /etc/systemd/system/routinator.service
[Unit]
Description=Routinator RPKI Validator and RTR Server
After=network.target
[Service]
Type=simple
User=routinator
Group=routinator
Restart=on-failure
RestartSec=90
ExecStart=/home/routinator/.cargo/bin/routinator -v -b /opt/routinator server \
    --http 127.0.0.1:8080 --rtr <IPv4 IP>:8323 --rtr [<IPv6 IP>]:8323
TimeoutStartSec=0
[Install]
WantedBy=default.target
```

Note: You must populate the IPv4 and IPv6 addresses. In addition, the IPv6 address needs to have brackets '[ ]' around it. For example:

```
/home/routinator/.cargo/bin/routinator -v -b /opt/routinator server \
--http 127.0.0.1:8080 --rtr 172.16.47.235:8323 --rtr [2001:db8::43]:8323
```

16. Configure SELinux to allow connections to localhost and to allow rsync to write to the `/opt/routinator` directory.

```
sudo setsebool -P httpd_can_network_connect 1
sudo semanage permissive -a rsync_t
```

17. Reload the systemd daemon and set the routinator service to start at boot.

```
sudo systemctl daemon-reload
sudo systemctl enable routinator.service
sudo systemctl start routinator.service
```

18. Set up the firewall to permit ssh, HTTPS and port 8323 for the RTR protocol.

```
sudo firewall-cmd --permanent --remove-service=ssh --zone=public
sudo firewall-cmd --permanent --zone public --add-rich-rule='rule family="ipv4" \
    source address="<IPv4 management subnet>" service name=ssh accept'
sudo firewall-cmd --permanent --zone public --add-rich-rule='rule family="ipv6" \
    source address="<IPv6 management subnet>" service name=ssh accept'
sudo firewall-cmd --permanent --zone public --add-rich-rule='rule family="ipv4" \
    source address="<IPv4 management subnet>" service name=https accept'
```

(continues on next page)

(continued from previous page)

```
sudo firewall-cmd --permanent --zone public --add-rich-rule='rule family="ipv6" \
    source address="<IPv6 management subnet>" service name=https accept'
sudo firewall-cmd --permanent --zone public --add-rich-rule='rule family="ipv4" \
    source address="<peering router IPv4 loopback subnet>" port port=8323␣
↪protocol=tcp accept'
sudo firewall-cmd --permanent --zone public --add-rich-rule='rule family="ipv6" \
    source address="<peering router IPv6 loopback subnet>" port port=8323␣
↪protocol=tcp accept'
sudo firewall-cmd --reload
```

19. Navigate to `https://<IP address of rpki-validator>/metrics` to see if it's working. You should authenticate with the username and password that you provided in step 10 of setting up the RPKI Validation Server.

CHAPTER

THREE

INITIALISATION

Before running Routinator for the first time, you must prepare its working environment. You do this using the *init* command. This will prepare both the directory for the local RPKI cache, as well as the Trust Anchor Locator (TAL) directory.

By default, both directories will be located under `$HOME/.rpki-cache`, but you can change their locations via the command line options *--repository-dir* and *--tal-dir*.

TALs provide hints for the trust anchor certificates to be used both to discover and validate all RPKI content. The five TALs — one for each Regional Internet Registry (RIR) — are bundled with Routinator and installed by the *init* command.

Warning: Using the TAL from ARIN, the RIR for the United States, Canada as well as many Caribbean and North Atlantic islands, requires you to read and accept their Relying Party Agreement before you can use it. Running the *init* command will provide you with instructions.

```
routinator init
Before we can install the ARIN TAL, you must have read
and agree to the ARIN Relying Party Agreement (RPA).
It is available at

https://www.arin.net/resources/manage/rpki/rpa.pdf

If you agree to the RPA, please run the command
again with the --accept-arin-rpa option.
```

Running the *init* command with the *--accept-arin-rpa* option will create the TAL directory and copy the five Trust Anchor Locator files into it.

```
routinator init --accept-arin-rpa
```

If you decide you cannot agree to the ARIN RPA terms, the *--decline-arin-rpa* option will install all TALs except the one for ARIN. If, at a later point, you wish to use the ARIN TAL anyway, you can add it to your current installation using the *--force* option, to force the installation of all TALs.

3.1 Performing a Test Run

To see if Routinator has been initialised correctly and your firewall allows the required connections, it is recommended to perform an initial test run. You can do this by having Routinator print a validated ROA payload (VRP) list with the *vrps* subcommand, and using *-v* to increase the log level so you can verify if Routinator establishes rsync and RRDP connections as expected.

```
routinator -vv vrps
```

Now, you can see how Routinator connects to the RPKI trust anchors, downloads the the contents of the repositories to your machine, verifies it and produces a list of validated ROA payloads in the default CSV format to standard output. Because it is expected that the state of the entire RPKI is not perfect as all times, you may see several warnings during the process about objects that are either stale or failed cryptographic verification. From a cold start, this process will take a couple of minutes.

```
routinator -vv vrps
rsyncing from rsync://repository.lacnic.net/rpki/.
rsync://repository.lacnic.net/rpki: Running command "rsync" "--timeout=300" "-rltz" "-
↪-delete" "rsync://repository.lacnic.net/rpki/" "/Users/alex/.rpki-cache/repository/
↪rsync/repository.lacnic.net/rpki/"
Found valid trust anchor https://rpki.ripe.net/ta/ripe-ncc-ta.cer. Processing.
RRDP https://rrdp.ripe.net/notification.xml: Updating server
RRDP https://rrdp.ripe.net/notification.xml: updating from snapshot.
Found valid trust anchor https://rpki.afrinic.net/repository/AfriNIC.cer. Processing.
RRDP https://rrdp.afrinic.net/notification.xml: Updating server
RRDP https://rrdp.afrinic.net/notification.xml: updating from snapshot.
Found valid trust anchor https://tal.apnic.net/apnic.cer. Processing.
RRDP https://rrdp.apnic.net/notification.xml: Updating server
RRDP https://rrdp.apnic.net/notification.xml: updating from snapshot.
Found valid trust anchor https://rrdp.arin.net/arin-rpki-ta.cer. Processing.
RRDP https://rrdp.arin.net/notification.xml: Updating server
RRDP https://rrdp.arin.net/notification.xml: updating from snapshot.
rsync://repository.lacnic.net/rpki: successfully completed.
Found valid trust anchor rsync://repository.lacnic.net/rpki/lacnic/rta-lacnic-rpki.
↪cer. Processing.
rsyncing from rsync://rpki-repo.registro.br/repo/.
rsync://rpki-repo.registro.br/repo: Running command "rsync" "--timeout=300" "-rltz" "-
↪-delete" "rsync://rpki-repo.registro.br/repo/" "/Users/alex/.rpki-cache/repository/
↪rsync/rpki-repo.registro.br/repo/"
rsync://rpki-repo.registro.br/repo: successfully completed.
RRDP https://rrdp.rpki.nlnetlabs.nl/rrdp/notification.xml: Updating server
RRDP https://rrdp.rpki.nlnetlabs.nl/rrdp/notification.xml: updating from snapshot.
...
ASN,IP Prefix,Max Length,Trust Anchor
AS137884,103.116.116.0/23,23,apnic
AS9003,91.151.112.0/20,20,ripe
AS38553,120.72.19.0/24,24,apnic
AS58045,37.209.242.0/24,24,ripe
AS9583,202.177.175.0/24,24,apnic
AS50629,2a0f:ba80::/29,29,ripe
AS398085,2602:801:a008::/48,48,arin
AS21050,83.96.22.0/24,24,ripe
AS55577,183.82.223.0/24,24,apnic
AS44444,157.167.73.0/24,24,ripe
AS197695,194.67.97.0/24,24,ripe
...
```

CHAPTER

FOUR

RUNNING INTERACTIVELY

Routinator can perform RPKI validation as a one-time operation and print a Validated ROA Payload (VRP) list in various formats, or it can return the validity of a specific announcement. These functions are accessible on the command line via the following sub-commands:

`vrps` Fetches RPKI data and produces a Validated ROA Payload (VRP) list in the specified format.

`validate` Outputs the RPKI validity for a specific announcement by supplying Routinator with an ASN and a prefix.

4.1 Printing a List of VRPs

Routinator can produce a Validated ROA Payload (VRP) list in many different formats, which are either printed to standard output or saved to a file:

csv The list is formatted as lines of comma-separated values of the prefix in slash notation, the maximum prefix length, the autonomous system number, and an abbreviation for the trust anchor the entry is derived from. The latter is the name of the TAL file without the extension *.tal*. This is the default format used if the `--format` or `-f` option is missing.

csvcompat The same as csv except that all fields are embedded in double quotes and the autonomous system number is given without the prefix AS. This format is pretty much identical to the CSV produced by the RIPE NCC Validator.

csvext This is an extended version of the *csv* format, which was used by the RIPE NCC RPKI Validator 1.x. Each line contains these comma-separated values: the rsync URI of the ROA the line is taken from (or "N/A" if it isn't from a ROA), the autonomous system number, the prefix in slash notation, the maximum prefix length, and lastly the not-before and not-after date of the validity of the ROA.

json The list is placed into a JSON object with a single element *roas* which contains an array of objects with four elements each: The autonomous system number of the network authorised to originate a prefix in *asn*, the prefix in slash notation in *prefix*, the maximum prefix length of the announced route in *maxLength*, and the trust anchor from which the authorisation was derived in *ta*. This format is identical to that produced by the RIPE NCC Validator except for different naming of the trust anchor. Routinator uses the name of the TAL file without the extension *.tal* whereas the RIPE NCC Validator has a dedicated name for each.

openbgpd Choosing this format causes Routinator to produce a *roa-set* configuration item for the OpenBGPD configuration.

bird Choosing this format causes Routinator to produce a roa table configuration item for the BIRD configuration.

bird2 Choosing this format causes Routinator to produce a route table configuration item for the BIRD2 configuration.

rpsl This format produces a list of RPSL objects with the authorisation in the fields *route*, *origin*, and *source*. In addition, the fields *descr*, *mnt-by*, *created*, and *last-modified*, are present with more or less meaningful values.

summary This format produces a summary of the content of the RPKI repository. For each trust anchor, it will print the number of verified ROAs and VRPs. Note that this format does not take filters into account. It will always provide numbers for the complete repository.

For example, to get the validated ROA payloads in CSV format, run:

```
routinator vrps --format csv
ASN,IP Prefix,Max Length,Trust Anchor
AS55803,103.14.64.0/23,23,apnic
AS267868,45.176.192.0/24,24,lacnic
AS41152,82.115.18.0/23,23,ripe
AS28920,185.103.228.0/22,22,ripe
AS11845,209.203.0.0/18,24,afrinic
AS63297,23.179.0.0/24,24,arin
...
```

To generate a file with with the validated ROA payloads in JSON format, run:

```
routinator vrps --format json --output authorisedroutes.json
```

4.1.1 Filtering

In case you are looking for specific information in the output, Routinator allows filtering to see if a prefix or ASN is covered or matched by a VRP. You can do this using the *--filter-asn* and *--filter-prefix* options.

When using *--filter-asn*, you can use both `AS64511` and `64511` as the notation. With *--filter-prefix*, the result will include VRPs regardless of their ASN and MaxLength. Both filter flags can be combined and used multiple times in a single query and will be treated as a logical *"or"*.

A validation run will be started before returning the result, making sure you get the latest information. If you would like a result from the current cache, you can use the *--noupdate* or *-n* option.

Here are some examples filtering for an ASN and prefix in CSV and JSON format:

```
routinator vrps --format csv --filter-asn 196615
ASN,IP Prefix,Max Length,Trust Anchor
AS196615,2001:7fb:fd03::/48,48,ripe
AS196615,93.175.147.0/24,24,ripe
```

```
routinator vrps --format json --filter-prefix 93.175.146.0/24
{
  "roas": [
    { "asn": "AS12654", "prefix": "93.175.146.0/24", "maxLength": 24, "ta": "ripe" }
  ]
}
```

4.2 Validity Checker

You can check the RPKI origin validation status of a specific BGP announcement using the *validate* subcommand and by supplying the ASN and prefix. A validation run will be started before returning the result, making sure you get the latest information. If you would like a result from the current cache, you can use the *--noupdate* or *-n* option.

```
routinator validate --asn 12654 --prefix 93.175.147.0/24
Invalid
```

A detailed analysis of the reasoning behind the validation outcome is printed in JSON format. In case of an Invalid state, whether this because the announcement is originated by an unauthorised AS, or if the prefix is more specific than the maximum prefix length allows. Lastly, a complete list of VRPs that caused the result is included.

```
routinator validate --json --asn 12654 --prefix 93.175.147.0/24
{
  "validated_route": {
   "route": {
     "origin_asn": "AS12654",
     "prefix": "93.175.147.0/24"
   },
   "validity": {
     "state": "Invalid",
     "reason": "as",
     "description": "At least one VRP Covers the Route Prefix, but no VRP ASN matches␣
↪the route origin ASN",
     "VRPs": {
      "matched": [
      ],
      "unmatched_as": [
        {
         "asn": "AS196615",
         "prefix": "93.175.147.0/24",
         "max_length": "24"
        }

      ],
      "unmatched_length": [
      ]       }
   }
  }
}
```

If you run the HTTP service in daemon mode, this information is also available via the `/validity` API endpoint.

CHAPTER

FIVE

RUNNING AS A DAEMON

Routinator can run as a service that periodically fetches RPKI data, verifies it and makes the resulting data set available through the built-in HTTP server and via the RTR protocol. You can start the Routinator service using the *server* subcommand.

5.1 The HTTP Service

In addition to the various VRP output formats, Routinator's HTTP server also provides an API, a user interface and *monitoring endpoints*. The server is not enabled by default for security reasons, nor does it have a default host or port.

Please note that the HTTP server is intended to run on your internal network and doesn't offer HTTPS natively. If this is a requirement, you can for example run Routinator behind an NGINX reverse proxy.

In order to start the HTTP server at 192.0.2.13 and 2001:0DB8::13 on port 8323, run:

```
routinator server --http 192.0.2.13:8323 --http [2001:0DB8::13]:8323
```

The application will stay attached to your terminal unless you provide the *--detach* option.

5.1.1 Output Formats

After fetching and verifying all RPKI data, the following paths are available:

/csv Returns the current set of VRPs in **csv** output format

/csvext Returns the current set of VRPs in **csvext** output format.

/json Returns the current set of VRPs in **json** output format

/openbgpd Returns the current set of VRPs in **OpenBGPD** output format

/bird Returns the current set of VRPs in **bird** output format

/bird2 Returns the current set of VRPs in **bird2** output format

/rpsl Returns the current set of VRPs in **RPSL** output format

5.1.2 API Endpoints

The service supports GET requests with the following paths:

`/metrics` Returns a set of *monitoring* metrics in the format used by Prometheus.

`/status` Returns the current status of the Routinator instance. This is similar to the output of the **`/metrics`** endpoint but in a more human friendly format.

`/log` Returns the logging output of the last validation run. The log level matches that set upon start.

Note that the output is collected after each validation run and is therefore only available after the initial run has concluded.

`/version` Returns the version of the Routinator instance.

`/api/v1/validity/as-number/prefix` Returns a JSON object describing whether the route announcement given by its origin AS number and address prefix is RPKI valid, invalid, or not found. A complete list of VRPs that caused the result is included.

`/validity?asn=as-number&prefix=prefix` Same as above but with a more form-friendly calling convention.

These paths accept filter expressions to limit the VRPs returned in the form of a query string. The field `filter-asn` can be used to filter for ASNs and the field `filter-prefix` can be used to filter for prefixes. The fields can be repeated multiple times.

5.2 The RTR Service

Routinator supports RPKI-RTR as specified in **RFC 8210** as well as the older version described in **RFC 6810**.

When launched as an RTR server, routers with support for route origin validation (ROV) can connect to Routinator to fetch the processed data. This includes hardware routers such as Juniper, Cisco and Nokia, as well as software solutions like BIRD, GoBGP and others.

Like the HTTP server, the RTR server is not started by default, nor does it have a default host or port. Thus, in order to start the RTR server at 192.0.2.13 and 2001:0DB8::13 on port 3323, run Routinator using the *`server`* command:

```
routinator server --rtr 192.0.2.13:3323 --rtr [2001:0DB8::13]:3323
```

Please note that port 3323 is not the IANA-assigned default port for the protocol, which would be 323. But as this is a privileged port, you would need to be running Routinator as root when otherwise there is no reason to do that. The application will stay attached to your terminal unless you provide the *`--detach`* option.

By default, the repository will be updated and verified every 10 minutes. You can change this via the *`--refresh`* option and specify the interval between verification in seconds. That is, if you rather have Routinator validate every 15 minutes, the above command becomes:

```
routinator server --rtr 192.0.2.13:3323 --rtr [2001:0DB8::13]:3323 --refresh=900
```

Communication between Routinator and the router using the RPKI-RTR protocol is done via plain TCP. Below, there is an explanation how to secure the transport using either SSH or TLS.

5.2.1 Secure Transports

These instructions were contributed by wk on Github.

RFC 6810#section-7 defines a number of secure transports for RPKI-RTR that can be used to secure communication between a router and a RPKI relying party.

However, the RPKI Router Implementation Report documented in **RFC 7128#section-5** suggests these secure transports have not been widely implemented. Implementations, however, do exist, and a secure transport could be valuable in situations where the RPKI relying party is provided as a public service, or across a non-trusted network.

SSH Transport

SSH transport for RPKI-RTR can be configured with the help of netcat and OpenSSH.

1. Begin by installing the **`openssh-server`** and **`netcat`** packages.

Make sure Routinator is running as an RTR server on localhost:

```
routinator server --rtr 127.0.0.1:3323
```

2. Create a username and a password for the router to log into the host with, such as `rpki`.
3. Configure OpenSSH to expose an `rpki-rtr` subsystem that acts as a proxy into Routinator by editing the `/etc/ssh/sshd_config` file or equivalent to include the following line:

```
# Define an `rpki-rtr` subsystem which is actually `netcat` used to
# proxy STDIN/STDOUT to a running `routinator server --rtr 127.0.0.1:3323`
Subsystem       rpki-rtr        /bin/nc 127.0.0.1 3323

# Certain routers may use old KEX algos and Ciphers which are no longer enabled by␣
↪default.
# These examples are required in IOS-XR 5.3 but no longer enabled by default in␣
↪OpenSSH 7.3
Ciphers +3des-cbc
KexAlgorithms +diffie-hellman-group1-sha1
```

4. Restart the OpenSSH server daemon.
5. Set up the router running IOS-XR using this example configuration:

```
router bgp 65534
 rpki server 192.168.0.100
  username rpki
  password rpki
  transport ssh port 22
```

TLS Transport

TLS transport for RPKI-RTR can be configured with the help of stunnel.

1. Begin by installing the **`stunnel`** package.
2. Make sure Routinator is running as an RTR server on localhost:

```
routinator server --rtr 127.0.0.1:3323
```

3. Acquire (via for example Let's Encrypt) or generate an SSL certificate. In the example below, an SSL certificate for the domain example.com generated by Let's Encrypt is used.
4. Create an stunnel configuration file by editing `/etc/stunnel/rpki.conf` or equivalent:

```
[rpki]
; Use a letsencrypt certificate for example.com
cert = /etc/letsencrypt/live/example.com/fullchain.pem
key = /etc/letsencrypt/live/example.com/privkey.pem

; Listen for TLS rpki-rtr on port 323 and proxy to port 3323 on localhost
accept = 323
connect = 127.0.0.1:3323
```

5. Restart **`stunnel`** to complete the process.

CHAPTER

SIX

MONITORING

The HTTP server in Routinator provides endpoints for monitoring the application. To launch Routinator in server mode on 192.0.2.13 with RTR running on port 3323 and HTTP on 9556, use the following command:

```
routinator server --rtr 192.0.2.13:3323 --http 192.0.2.13:9556
```

The HTTP service has these monitoring endpoints on the following paths:

`/version` Returns the version of the Routinator instance

`/metrics` Exposes a data format specifically for Prometheus, for which dedicated port 9556 is reserved.

`/status` Returns the information from the `/metrics` endpoint in a more concise format

`/log` Returns the logging output of the last validation run. The log level matches that set upon start.

> Note that the output is collected after each validation run and is therefore only available after the initial run has concluded.

6.1 Metrics

Update metrics

- When the last update started and finished
- The total duration of the last update
- The retrieval duration and exit code for each rsync publication point
- The retrieval duration and HTTP status code for each RRDP publication point

Object metrics

- The number of valid ROAs per Trust Anchor
- The number of Validated ROA Payloads (VRPs) per Trust Anchor
- The number of stale objects found
- The number of VRPs added locally

RTR server

- The current RTR serial number
- The current and total number of RTR connections
- The total amount of bytes sent and received over the RTR connection

HTTP server

- The current and total number of HTTP connections
- The total amount of bytes sent and received over the HTTP connection
- The number of HTTP requests

6.2 Grafana

Using the Prometheus endpoint it's possible to build a detailed dashboard using for example Grafana. We provide a template to get started.

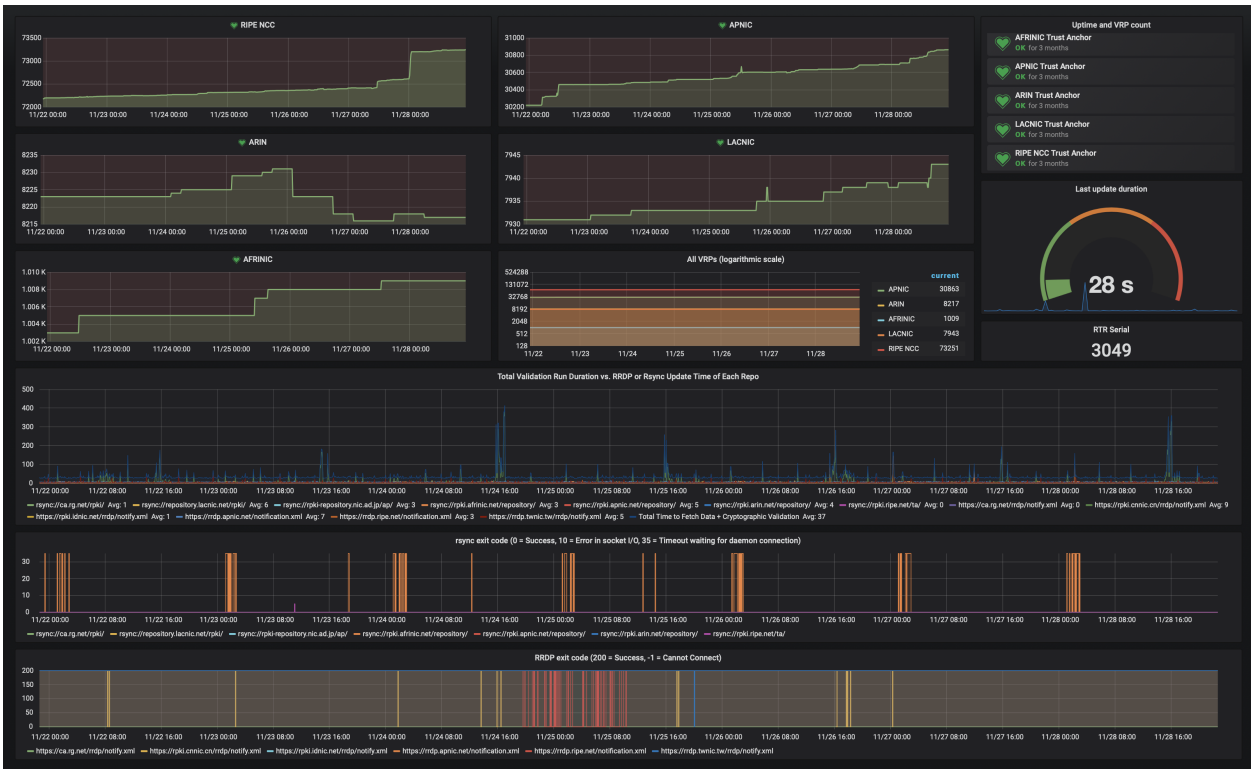

Fig. 1: A sample Grafana dashboard

CHAPTER

SEVEN

CONFIGURATION

Routinator has a number of default settings, such as the location where files are stored, the refresh interval and the log level. You can view these settings by running:

```
routinator config
```

It will return the list of defaults in the same notation that is used by the optional configuration file, which will be largely similar to this:

```
allow-dubious-hosts = false
dirty = false
disable-rrdp = false
disable-rsync = false
exceptions = []
expire = 7200
history-size = 10
http-listen = []
log = "default"
log-level = "WARN"
refresh = 600
repository-dir = "/Users/routinator/.rpki-cache/repository"
retry = 600
rrdp-proxies = []
rrdp-root-certs = []
rsync-command = "rsync"
rsync-timeout = 300
rtr-listen = []
rtr-tcp-keepalive = 60
stale = "reject"
strict = false
syslog-facility = "daemon"
systemd-listen = false
tal-dir = "/Users/routinator/.rpki-cache/tals"
unknown-objects = "warn"
unsafe-vrps = "warn"
validation-threads = 4
```

You can override these defaults, as well as configure a great number of additional options using either command line arguments or via the configuration file.

To get an overview of all available options, please refer to the *configuration file* section of the *Manual Page*, which can be also viewed by running **`routinator man`**.

7.1 Using a Configuration File

Routinator can take its configuration from a file. You can specify such a config file via the `-c` option. If you don't, Routinator will check if there is a `$HOME/.routinator.conf` and if it exists, use it. If it doesn't exist and there is no `-c` option, the default values are used.

For specifying configuration options, Routinator uses a TOML file. Its entries are named similarly to the command line options. A complete sample configuration file showing all the default values can be found in the repository at etc/routinator.conf.example.

Note: One scenario where a configuration file is required is when you would like to pass certain rsync arguments using the `rsync-args` setting, which is not available as a command line option. This is because Routinator will try to find out if your rsync understands the `--contimeout` option and set it if possible.

For example, if you want Routinator to refresh every 15 minutes and run as an RTR server on 192.0.2.13 and 2001:0DB8::13 on port 3323, in addition to providing an HTTP server on port 9556, simply take the output from **`routinator config`** and change the `refresh`, `rtr-listen` and `http-listen` values in your favourite text editor:

```
allow-dubious-hosts = false
dirty = false
disable-rrdp = false
disable-rsync = false
exceptions = []
expire = 7200
history-size = 10
http-listen = ["192.0.2.13:9556", "[2001:0DB8::13]:9556"]
log = "default"
log-level = "WARN"
refresh = 900
repository-dir = "/Users/routinator/.rpki-cache/repository"
retry = 600
rrdp-proxies = []
rrdp-root-certs = []
rsync-command = "rsync"
rsync-timeout = 300
rtr-listen = ["192.0.2.13:3323", "[2001:0DB8::13]:3323"]
rtr-tcp-keepalive = 60
stale = "reject"
strict = false
syslog-facility = "daemon"
systemd-listen = false
tal-dir = "/Users/routinator/.rpki-cache/tals"
unknown-objects = "warn"
unsafe-vrps = "warn"
validation-threads = 4
```

After saving this file as `.routinator.conf` in your home directory, you can start Routinator with:

```
routinator server
```

7.2 Applying Local Exceptions

In some cases, you may want to override the global RPKI data set with your own local exceptions. For example, when a legitimate route announcement is inadvertently flagged as *invalid* due to a misconfigured ROA, you may want to temporarily accept it to give the operators an opportunity to resolve the issue.

You can do this by specifying route origins that should be filtered out of the output, as well as origins that should be added, in a file using JSON notation according to the SLURM standard specified in **RFC 8416**.

A full example file is provided below. This, along with an empty one is available in the repository at /test/slurm.

```
{
  "slurmVersion": 1,
  "validationOutputFilters": {
   "prefixFilters": [
     {
      "prefix": "192.0.2.0/24",
      "comment": "All VRPs encompassed by prefix"
     },
     {
      "asn": 64496,
      "comment": "All VRPs matching ASN"
     },
     {
      "prefix": "198.51.100.0/24",
      "asn": 64497,
      "comment": "All VRPs encompassed by prefix, matching ASN"
     }
   ],
   "bgpsecFilters": [
     {
      "asn": 64496,
      "comment": "All keys for ASN"
     },
     {
      "SKI": "Zm9v",
      "comment": "Key matching Router SKI"
     },
     {
      "asn": 64497,
      "SKI": "YmFy",
      "comment": "Key for ASN 64497 matching Router SKI"
     }
   ]
  },
  "locallyAddedAssertions": {
   "prefixAssertions": [
     {
      "asn": 64496,
      "prefix": "198.51.100.0/24",
      "comment": "My other important route"
     },
     {
      "asn": 64496,
      "prefix": "2001:DB8::/32",
      "maxPrefixLength": 48,
      "comment": "My other important de-aggregated routes"
     }
```

(continues on next page)

(continued from previous page)

```
    ],
    "bgpsecAssertions": [
      {
       "asn": 64496,
       "comment" : "My known key for my important ASN",
       "SKI": "<some base64 SKI>",
       "routerPublicKey": "<some base64 public key>"
      }
    ]
  }
}
```

Use the `-x` option to refer to your file with local exceptions. Routinator will re-read that file on every validation run, so you can simply update the file whenever your exceptions change.

CHAPTER

EIGHT

MANUAL PAGE

routinator - RPKI relying party software

Date 2021-12-09

Author Martin Hoffmann

Version 0.8.2

8.1 Synopsis

8.2 Description

Routinator collects and processes Resource Public Key Infrastructure (RPKI) data. It validates the Route Origin Attestations contained in the data and makes them available to your BGP routing workflow.

It can either run in one-shot mode outputting a list of validated route origins in various formats or as a server for the RPKI-to-Router (RTR) protocol that routers often implement to access the data, or via HTTP.

These modes and additional operations can be chosen via commands. For the available commands, see *Commands* below.

8.3 Options

The available options are:

-c `path`, **--config**=`path`
: Provides the path to a file containing basic configuration. If this option is not given, Routinator will try to use `$HOME/.routinator.conf` if that exists. If that doesn't exist, either, default values for the options as described here are used.

  See *Configuration File* below for more information on the format and contents of the configuration file.

-b `dir`, **--base-dir**=`dir`
: Specifies the base directory to keep status information in. Unless overwritten by the `-r` or `-t` options, the local repository will be kept in the sub-directory repository and the TALs will be kept in the sub-directory `tals`.

  If omitted, the base directory defaults to `$HOME/.rpki-cache`.

-r dir, **--repository-dir**=dir
: Specifies the directory to keep the local repository in. This is the place where Routinator stores the RPKI data it has collected and thus is a copy of all the data referenced via the trust anchors.

-t dir, **--tal-dir**=dir
: Specifies the directory containing the trust anchor locators (TALs) to use. Trust anchor locators are the starting points for collecting and validating RPKI data. See *Trust Anchor Locators* for more information on what should be present in this directory.

-x file, **--exceptions**=file
: Provides the path to a local exceptions file. The option can be used multiple times to specify more than one file to use. Each file is a JSON file as described in **RFC 8416**. It lists both route origins that should be filtered out of the output as well as origins that should be added.

--strict
: If this option is present, the repository will be validated in strict mode following the requirements laid out by the standard documents very closely. With the current RPKI repository, using this option will lead to a rather large amount of invalid route origins and should therefore not be used in practice.

  See *Relaxed Decoding* below for more information.

--stale=policy
: This option defines how deal with stale objects. In RPKI, manifests and CRLs can be stale if the time given in their *next-update* field is in the past, indicating that an update to the object was scheduled but didn't happen. This can be because of an operational issue at the issuer or an attacker trying to replay old objects.

  There are three possible policies that define how Routinator should treat stale objects.

  A policy of *reject* instructs Routinator to consider all stale objects invalid. This will result in all material published by the CA issuing this manifest and CRL to be invalid including all material of any child CA.

  The *warn* policy will allow Routinator to consider any stale object to be valid. It will, however, print a warning in the log allowing an operator to follow up on the issue.

  Finally, the *accept* policy will cause Routinator to quietly accept any stale object as valid.

  In Routinator 0.8.0 and newer, *reject* is the default policy if the option is not provided. In all previous versions the default is *warn*.

--unsafe-vrps=policy
: This option defines how to deal with "unsafe VRPs." If the address prefix of a VRP overlaps with any resources assigned to a CA that has been rejected because if failed to validate completely, the VRP is said to be unsafe since using it may lead to legitimate routes being flagged as RPKI invalid.

  There are three options how to deal with unsafe VRPS:

  A policy of *reject* will filter out these VPRs. Warnings will be logged to indicate which VRPs have been filtered

  The *warn* policy will log warnings for unsafe VRPs but will add them to the valid VRPs.

  Finally, the *accept* policy will quietly add unsafe VRPs to the valid VRPs.

  Currently, the default policy is *warn* in order to gain operational experience with the frequency and impact of unsafe VRPs. This default may change in future version.

  For more information on the process of validation implemented in Routinator, see the section VALIDATION below.

--unknown-objects=policy
: Defines how to deal with unknown types of RPKI objects. Currently, only certificates (.cer), CRLs (.crl), manifests (.mft), ROAs (.roa), and Ghostbuster Records (.gbr) are allowed to appear in the RPKI repository.

  There are, once more, three policies for dealing with an object of any other type:

The *reject* policy will reject the object as well as the entire CA. Consequently, an unknown object appearing in a CA will mark all other objects issued by the CA as invalid as well.

The policy of *warn* will log a warning, ignore the object, and accept all known objects issued by the CA.

The similar policy of *accept* will quietly ignore the object and accept all known objects issued by the CA.

The default policy if the option is missing is *warn*.

Note that even if unknown objects are accepted, they must appear in the manifest and the hash over their content must match the one given in the manifest. If the hash does not match, the CA and all its objects are still rejected.

--allow-dubious-hosts
As a precaution, Routinator will reject rsync and HTTPS URIs from RPKI data with dubious host names. In particular, it will reject the name *localhost*, host names that consist of IP addresses, and a host name that contains an explicit port.

This option allows to disable this filtering.

--disable-rsync
If this option is present, rsync is disabled and only RRDP will be used.

--rsync-command=command
Provides the command to run for rsync. This is only the command itself. If you need to provide options to rsync, use the `rsync-args` configuration file setting instead.

If this option is not given, Routinator will simply run rsync and hope that it is in the path.

--rsync-timeout=seconds
Sets the number of seconds an rsync command is allowed to run before it is terminated early. This protects against hanging rsync commands that prevent Routinator from continuing. The default is 300 seconds which should be long enough except for very slow networks.

--disable-rrdp
If this option is present, RRDP is disabled and only rsync will be used.

--rrdp-timeout=seconds
Sets the timeout in seconds for any RRDP-related network operation, i.e., connects, reads, and writes. If this option is omitted, the default timeout of 30 seconds is used. Set the option to 0 to disable the timeout.

--rrdp-connect-timeout=seconds
Sets the timeout in seconds for RRDP connect requests. If omitted, the general timeout will be used.

--rrdp-local-addr=addr
If present, sets the local address that the RRDP client should bind to when doing outgoing requests.

--rrdp-root-cert=path
This option provides a path to a file that contains a certificate in PEM encoding that should be used as a trusted certificate for HTTPS server authentication. The option can be given more than once.

Providing this option does not disable the set of regular HTTPS authentication trust certificates.

--rrdp-proxy=uri
This option provides the URI of a proxy to use for all HTTP connections made by the RRDP client. It can be either an HTTP or a SOCKS URI. The option can be given multiple times in which case proxies are tried in the given order.

--dirty
If this option is present, unused files and directories will not be deleted from the repository directory after each validation run.

--validation-threads=count
Sets the number of threads to distribute work to for validation. Note that the current processing model validates

trust anchors all in one go, so you are likely to see less than that number of threads used throughout the validation run.

-v, --verbose
Print more information. If given twice, even more information is printed.

More specifically, a single `-v` increases the log level from the default of warn to info, specifying it more than once increases it to debug.

-q, --quiet
Print less information. Given twice, print nothing at all.

A single `-q` will drop the log level to error. Repeating `-q` more than once turns logging off completely.

--syslog
Redirect logging output to syslog.

This option is implied if a command is used that causes Routinator to run in daemon mode.

--syslog-facility=facility
If logging to syslog is used, this option can be used to specify the syslog facility to use. The default is daemon.

--logfile=path
Redirect logging output to the given file.

-h, --help
Print some help information.

-V, --version
Print version information.

8.4 Commands

Routinator provides a number of operations around the local RPKI repository. These can be requested by providing different commands on the command line.

init
Prepares the local repository directories and the TAL directory for running Routinator. Specifically, makes sure the local repository directory exists, and creates the TAL directory and fills it with the TALs of the five RIRs.

For more information about TALs, see *Trust Anchor Locators* below.

-f, --force
Forces installation of the TALs even if the TAL directory already exists.

--accept-arin-rpa
Before you can use the ARIN TAL, you need to agree to the ARIN Relying Party Agreement (RPA). You can find it at https://www.arin.net/resources/manage/rpki/rpa.pdf and explicitly agree to it via this option. This explicit agreement is necessary in order to install the ARIN TAL.

--decline-arin-rpa
If, after reading the ARIN Relying Party Agreement, you decide you do not or cannot agree to it, this option allows you to skip installation of the ARIN TAL. Note that this means Routinator will not have access to any information published for resources assigned under ARIN.

vrps
This command requests that Routinator update the local repository and then validate the Route Origin Attestations in the repository and output the valid route origins, which are also known as Validated ROA Payload or VRPs, as a list.

-o file, **--output**=file
Specifies the output file to write the list to. If this option is missing or file is - the list is printed to standard output.

-f format, **--format**=format
The output format to use. Routinator currently supports the following formats:

csv The list is formatted as lines of comma-separated values of the prefix in slash notation, the maximum prefix length, the autonomous system number, and an abbreviation for the trust anchor the entry is derived from. The latter is the name of the TAL file without the extension *.tal*.

This is the default format used if the *-f* option is missing.

csvcompat The same as csv except that all fields are embedded in double quotes and the autonomous system number is given without the prefix AS. This format is pretty much identical to the CSV produced by the RIPE NCC Validator.

csvext An extended version of csv each line contains these comma-separated values: the rsync URI of the ROA the line is taken from (or "N/A" if it isn't from a ROA), the autonomous system number, the prefix in slash notation, the maximum prefix length, the not-before date and not-after date of the validity of the ROA.

This format was used in the RIPE NCC RPKI Validator version 1. That version produces one file per trust anchor. This is not currently supported by Routinator – all entries will be in one single output file.

json The list is placed into a JSON object with a single element *roas* which contains an array of objects with four elements each: The autonomous system number of the network authorized to originate a prefix in *asn*, the prefix in slash notation in *prefix*, the maximum prefix length of the announced route in *maxLength*, and the trust anchor from which the authorization was derived in *ta*. This format is identical to that produced by the RIPE NCC RPKI Validator except for different naming of the trust anchor. Routinator uses the name of the TAL file without the extension *.tal* whereas the RIPE NCC Validator has a dedicated name for each.

openbgpd Choosing this format causes Routinator to produce a roa- set configuration item for the OpenBGPD configuration.

bird Choosing this format causes Routinator to produce a roa table configuration item for the BIRD configuration.

bird2 Choosing this format causes Routinator to produce a roa table configuration item for the BIRD2 configuration.

rpsl This format produces a list of RPSL objects with the authorization in the fields *route*, *origin*, and *source*. In addition, the fields *descr*, *mnt-by*, *created*, and *last-modified*, are present with more or less meaningful values.

summary This format produces a summary of the content of the RPKI repository. For each trust anchor, it will print the number of verified ROAs and VRPs. Note that this format does not take filters into account. It will always provide numbers for the complete repository.

none This format produces no output whatsoever.

-n, --noupdate
The repository will not be updated before producing the list.

--complete
If any of the rsync commands needed to update the repository failed, Routinator completes the operation and exits with status code 2. Normally, it would exit with status code 0 indicating success.

-a asn, **--filter-asn**=asn
: Only output VRPs for the given ASN. The option can be given multiple times, in which case VRPs for all provided ASNs are provided. ASNs can be given with or without the prefix AS.

-p prefix, **--filter-prefix**=prefix
: Only output VRPs with an address prefix that covers the given prefix, i.e., whose prefix is equal to or less specific than the given prefix. This will include VRPs regardless of their ASN and max length. In other words, the output will include all VRPs that need to be considered when deciding whether an announcement for the prefix is RPKI valid or invalid.

  The option can be given multiple times, in which case VRPs for all prefixes are provided. It can also be combined with one or more ASN filters. Then all matching VRPs are included. That is, filters combine as "or" not "and."

validate
: This command can be used to perform RPKI route origin validation for a route announcement. Routinator will determine whether the provided announcement is RPKI valid, invalid, or not found.

  -a asn, **--asn**=asn
  : The AS number of the autonomous system that originated the route announcement. ASNs can be given with or without the prefix AS.

  -p prefix, **--prefix**=prefix
  : The address prefix the route announcement is for.

  -j, --json
  : A detailed analysis on the reasoning behind the validation is printed in JSON format including lists of the VPRs that caused the particular result. If this option is omitted, Routinator will only print the determined state.

  -n, --noupdate
  : The repository will not be updated before performing validation.

  --complete
  : If any of the rsync commands needed to update the repository failed, Routinator completes the operation and exits with status code 2. Normally, it would exit with status code 0 indicating success.

server
: This command causes Routinator to act as a server for the RPKI-to-Router (RTR) and HTTP protocols. In this mode, Routinator will read all the TALs (See *Trust Anchor Locators* below) and will stay attached to the terminal unless the *-d* option is given.

  The server will periodically update the local repository, every ten minutes by default, notify any clients of changes, and let them fetch validated data. It will not, however, reread the trust anchor locators. Thus, if you update them, you will have to restart Routinator.

  You can provide a number of addresses and ports to listen on for RTR and HTTP through command line options or their configuration file equivalent. Currently, Routinator will only start listening on these ports after an initial validation run has finished.

  It will not listen on any sockets unless explicitly specified. It will still run and periodically update the repository. This might be useful for use with *vrps* mode with the *-n* option.

  -d, --detach
  : If present, Routinator will detach from the terminal after a successful start.

  --rtr=addr:port
  : Specifies a local address and port to listen on for incoming RTR connections.

    Routinator supports both protocol version 0 defined in **RFC 6810** and version 1 defined in **RFC 8210**. However, it does not support router keys introduced in version 1. IPv6 addresses must be enclosed in

square brackets. You can provide the option multiple times to let Routinator listen on multiple address-port pairs.

--http=addr:port
Specifies the address and port to listen on for incoming HTTP connections. See *HTTP Service* below for more information on the HTTP service provided by Routinator.

--listen-systemd
The RTR listening socket will be acquired from systemd via socket activation. Use this option together with systemd's socket units to allow a Routinator running as a regular user to bind to the default RTR port 323.

Currently, all TCP listener sockets handed over by systemd will be used for the RTR protocol.

--refresh=seconds
The amount of seconds the server should wait after having finished updating and validating the local repository before starting to update again. The next update will be earlier if objects in the repository expire earlier. The default value is 600 seconds.

--retry=seconds
The amount of seconds to suggest to an RTR client to wait before trying to request data again if that failed. The default value is 600 seconds, as recommended in **RFC 8210**.

--expire=seconds
The amount of seconds to an RTR client can keep using data if it cannot refresh it. After that time, the client should discard the data. Note that this value was introduced in version 1 of the RTR protocol and is thus not relevant for clients that only implement version 0. The default value, as recommended in **RFC 8210**, is 7200 seconds.

--history=count
In RTR, a client can request to only receive the changes that happened since the last version of the data it had seen. This option sets how many change sets the server will at most keep. If a client requests changes from an older version, it will get the current full set.

Note that routers typically stay connected with their RTR server and therefore really only ever need one single change set. Additionally, if RTR server or router are restarted, they will have a new session with new change sets and need to exchange a full data set, too. Thus, increasing the value probably only ever increases memory consumption.

The default value is 10.

--pid-file=path
States a file which will be used in daemon mode to store the processes PID. While the process is running, it will keep the file locked.

--working-dir=path
The working directory for the daemon process. In daemon mode, Routinator will change to this directory while detaching from the terminal.

--chroot=path
The root directory for the daemon process. If this option is provided, the daemon process will change its root directory to the given directory. This will only work if all other paths provided via the configuration or command line options are under this directory.

--user=user-name
The name of the user to change to for the daemon process. It this option is provided, Routinator will run as that user after the listening sockets for HTTP and RTR have been created. The option has no effect unless *--detach* is also used.

--group=group-name
The name of the group to change to for the daemon process. It this option is provided, Routinator will run

as that group after the listening sockets for HTTP and RTR have been created. The option has no effect unless *--detach* is also used.

update
Updates the local repository by resyncing all known publication points. The command will also validate the updated repository to discover any new publication points that appear in the repository and fetch their data.

As such, the command really is a shortcut for running **routinator** *vrps* *-f* `none`.

--complete
If any of the rsync commands needed to update the repository failed, Routinator completes the operation and exits with status code 2. Normally, it would exit with status code 0 indicating success.

man
Displays the manual page, i.e., this page.

-o `file`, **--output**=`file`
If this option is provided, the manual page will be written to the given file instead of displaying it. Use - to output the manual page to standard output.

8.5 Trust Anchor Locators

RPKI uses trust anchor locators, or TALs, to identify the location and public keys of the trusted root CA certificates. Routinator keeps these TALs in files in the TAL directory which can be set by the *-t* option. If the *-b* option is used instead, the TAL directory will be in the subdirectory *tals* under the directory specified in this option. The default location, if no options are used at all is `$HOME/.rpki-cache/tals`.

This directory can be created and populated with the TALs of the five Regional Internet Registries (RIRs) via the **init** command.

If the directory does exist, Routinator will use all files with an extension of *.tal* in this directory. This means that you can add and remove trust anchors by adding and removing files in this directory. If you add files, make sure they are in the format described by **RFC 7730** or the upcoming **RFC 8630**.

8.6 Configuration File

Instead of providing all options on the command line, they can also be provided through a configuration file. Such a file can be selected through the *-c* option. If no configuration file is specified this way but a file named `$HOME/.routinator.conf` is present, this file is used.

The configuration file is a file in TOML format. In short, it consists of a sequence of key-value pairs, each on its own line. Strings are to be enclosed in double quotes. Lists can be given by enclosing a comma-separated list of values in square brackets.

The configuration file can contain the following entries. All path values are interpreted relative to the directory the configuration file is located in. All values can be overridden via the command line options.

repository-dir A string containing the path to the directory to store the local repository in. This entry is mandatory.

tal-dir A string containing the path to the directory that contains the Trust Anchor Locators. This entry is mandatory.

exceptions A list of strings, each containing the path to a file with local exceptions. If missing, no local exception files are used.

strict A boolean specifying whether strict validation should be employed. If missing, strict validation will not be used.

stale A string specifying the policy for dealing with stale objects.

reject Consider all stale objects invalid rendering all material published by the CA issuing the stale object to be invalid including all material of any child CA.

warn Consider stale objects to be valid but print a warning to the log.

accept Quietly consider stale objects valid.

unsafe-vrps A string specifying the policy for dealing with unsafe VRPs.

reject Filter unsafe VPRs and add warning messages to the log.

warn Warn about unsafe VRPs in the log but add them to the final set of VRPs. This is the default policy if the value is missing.

accept Quietly add unsafe VRPs to the final set of VRPs.

unknown-objects A string specifying the policy for dealing with unknown RPKI object types.

reject Reject the object and its issuing CA.

warn Warn about the object but ignore it and accept the issuing CA. This is the default policy if the value is missing.

accept Quietly ignore the object and accept the issuing CA.

allow-dubious-hosts A boolean value that, if present and true, disables Routinator's filtering of dubious host names in rsync and HTTPS URIs from RPKI data.

disable-rsync A boolean value that, if present and true, turns off the use of rsync.

rsync-command A string specifying the command to use for running rsync. The default is simply *rsync*.

rsync-args A list of strings containing the arguments to be passed to the rsync command. Each string is an argument of its own.

If this option is not provided, Routinator will try to find out if your rsync understands the `--contimeout` option and, if so, will set it to 10 thus letting connection attempts time out after ten seconds. If your rsync is too old to support this option, no arguments are used.

rsync-timeout An integer value specifying the number seconds an rsync command is allowed to run before it is being terminated. The default if the value is missing is 300 seconds.

disable-rrdp A boolean value that, if present and true, turns off the use of RRDP.

rrdp-timeout An integer value that provides a timeout in seconds for all individual RRDP-related network operations, i.e., connects, reads, and writes. If the value is missing, a default timeout of 30 seconds will be used. Set the value to 0 to turn the timeout off.

rrdp-connect-timeout An integer value that, if present, sets a separate timeout in seconds for RRDP connect requests only.

rrdp-local-addr A string value that provides the local address to be used by RRDP connections.

rrdp-root-certs A list of strings each providing a path to a file containing a trust anchor certificate for HTTPS authentication of RRDP connections. In addition to the certificates provided via this option, the system's own trust store is used.

rrdp-proxies A list of string each providing the URI for a proxy for outgoing RRDP connections. The proxies are tried in order for each request. HTTP and SOCKS5 proxies are supported.

dirty A boolean value which, if true, specifies that unused files and directories should not be deleted from the repository directory after each validation run. If left out, its value will be false and unused files will be deleted.

validation-threads An integer value specifying the number of threads to be used during validation of the repository. If this value is missing, the number of CPUs in the system is used.

log-level A string value specifying the maximum log level for which log messages should be emitted. The default is warn.

log A string specifying where to send log messages to. This can be one of the following values:

default Log messages will be sent to standard error if Routinator stays attached to the terminal or to syslog if it runs in daemon mode.

stderr Log messages will be sent to standard error.

syslog Log messages will be sent to syslog.

file Log messages will be sent to the file specified through the log-file configuration file entry.

The default if this value is missing is, unsurprisingly, default.

log-file A string value containing the path to a file to which log messages will be appended if the log configuration value is set to file. In this case, the value is mandatory.

syslog-facility A string value specifying the syslog facility to use for logging to syslog. The default value if this entry is missing is daemon.

rtr-listen An array of string values each providing the address and port which the RTR daemon should listen on in TCP mode. Address and port should be separated by a colon. IPv6 address should be enclosed in square brackets.

http-listen An array of string values each providing the address and port which the HTTP service should listen on. Address and port should be separated by a colon. IPv6 address should be enclosed in square brackets.

listen-systemd The RTR TCP listening socket will be acquired from systemd via socket activation. Use this option together with systemd's socket units to allow Routinator running as a regular user to bind to the default RTR port 323.

refresh An integer value specifying the number of seconds Routinator should wait between consecutive validation runs in server mode. The next validation run will happen earlier, if objects expire earlier. The default is 600 seconds.

retry An integer value specifying the number of seconds an RTR client is requested to wait after it failed to receive a data set. The default is 600 seconds.

expire An integer value specifying the number of seconds an RTR client is requested to use a data set if it cannot get an update before throwing it away and continuing with no data at all. The default is 7200 seconds if it cannot get an update before throwing it away and continuing with no data at all. The default is 7200 seconds.

history-size An integer value specifying how many change sets Routinator should keep in RTR server mode. The default is 10.

pid-file A string value containing a path pointing to the PID file to be used in daemon mode.

working-dir A string value containing a path to the working directory for the daemon process.

chroot A string value containing the path any daemon process should use as its root directory.

user A string value containing the user name a daemon process should run as.

group A string value containing the group name a daemon process should run as.

tal-label An array containing arrays of two string values mapping the name of a TAL file (without the path but including the extension) as given by the first string to the name of the TAL to be included where the TAL is referenced in output as given by the second string.

If the options missing or if a TAL isn't mentioned in the option, Routinator will construct a name for the TAL by using its file name (without the path) and dropping the extension.

8.7 HTTP Service

Routinator can provide an HTTP service allowing to fetch the Validated ROA Payload in various formats. The service does not support HTTPS and should only be used within the local network.

The service only supports GET requests with the following paths:

`/metrics` Returns a set of monitoring metrics in the format used by Prometheus.

`/status` Returns the current status of the Routinator instance. This is similar to the output of the **/metrics** endpoint but in a more human friendly format.

`/log` Returns the logging output of the last validation run. The log level matches that set upon start.

Note that the output is collected after each validation run and is therefore only available after the initial run has concluded.

`/version` Returns the version of the Routinator instance.

`/api/v1/validity/as-number/prefix` Returns a JSON object describing whether the route announcement given by its origin AS number and address prefix is RPKI valid, invalid, or not found. The returned object is compatible with that provided by the RIPE NCC RPKI Validator. For more information, see https://ripe.net/support/documentation/developer-documentation/rpki-validator-api

`/validity?asn=as-number&prefix=prefix` Same as above but with a more form-friendly calling convention.

In addition, the current set of VRPs is available for each output format at a path with the same name as the output format. E.g., the CSV output is available at `/csv`.

These paths accept filter expressions to limit the VRPs returned in the form of a query string. The field `filter-asn` can be used to filter for ASNs and the field `filter-prefix` can be used to filter for prefixes. The fields can be repeated multiple times.

This works in the same way as the options of the same name to the *`vrps`* command.

8.8 Logging

In order to allow diagnosis of the VRP data set as well as its overall health, Routinator logs an extensive amount of information. The log levels used by syslog are utilized to allow filtering this information for particular use cases.

The log levels represent the following information:

error Information related to events that prevent Routinator from continuing to operate at all as well as all issues related to local configuration even if Routinator will continue to run.

warn Information about events and data that influences the set of VRPs produced by Routinator. This includes failures to communicate with repository servers, or encountering invalid objects.

info Information about events and data that could be considered abnormal but do not influence the set of VRPs produced. For example, when filtering of unsafe VRPs is disabled, the unsafe VRPs are logged with this level.

debug Information about the internal state of Routinator that may be useful for, well, debugging.

8.9 Validation

In *vrps* and *server* mode, Routinator will produce a set of VRPs from the data published in the RPKI repository. It will walk over all certfication authorities (CAs) starting with those referred to in the configured TALs.

Each CA is checked whether all its published objects are present, correctly encoded, and have been signed by the CA. If any of the objects fail this check, the entire CA will be rejected. If an object of an unknown type is encountered, the behaviour depends on the `unknown-objects` policy. If this policy has a value of *reject* the entire CA will be rejected. In this case, only certificates (.cer), CRLs (.crl), manifestes (.mft), ROAs (.roa), and Ghostbuster records (.gbr) will be accepted.

If a CA is rejected, none of its ROAs will be added to the VRP set but also none of its child CAs will be considered at all; their published data will not be fetched or validated.

If a prefix has its ROAs published by different CAs, this will lead to some of its VRPs being dropped while others are still added. If the VRP for the legitimately announced route is among those having been dropped, the route becomes RPKI invalid. This can happen both by operator error or through an active attack.

In addition, if a VRP for a less specific prefix exists that covers the prefix of the dropped VRP, the route will be invalidated by the less specific VRP.

Because of this risk of accidentally or maliciously invalidating routes, VRPs that have address prefixes overlapping with resources of rejected CAs are called *unsafe VRPs*.

In order to avoid these situations and instead fall back to an RPKI unknown state for such routes, Routinator allows to filter out these unsafe VRPs. This can be enabled via the `--unsafe-vrps=reject` command line option or setting `unsafe-vrps=reject` in the config file.

By default, this filter is currently disabled but warnings are logged about unsafe VPRs. This allows to assess the operation impact of such a filter. Depending on this assessment, the default may change in future version.

One exception from this rule are CAs that have the full address space assigned, i.e., 0.0.0.0/0 and ::/0. Adding these to the filter would wipe out all VRPs. These prefixes are used by the RIR trust anchors to avoid having to update these often. However, each RIR has its own address space so losing all VRPs should something happen to a trust anchor is unnecessary.

8.10 Relaxed Decoding

The documents defining RPKI include a number of very strict rules regarding the formatting of the objects published in the RPKI repository. However, because RPKI reuses existing technology, real-world applications produce objects that do not follow these strict requirements.

As a consequence, a significant portion of the RPKI repository is actually invalid if the rules are followed. We therefore introduce two decoding modes: strict and relaxed. Strict mode rejects any object that does not pass all checks laid out by the relevant RFCs. Relaxed mode ignores a number of these checks.

This memo documents the violations we encountered and are dealing with in relaxed decoding mode.

Resource Certificates (RFC 6487) Resource certificates are defined as a profile on the more general Internet PKI certificates defined in **RFC 5280**.

Subject and Issuer The RFC restricts the type used for CommonName attributes to PrintableString, allowing only a subset of ASCII characters, while **RFC 5280** allows a number of additional string types. At least one CA produces resource certificates with Utf8Strings.

In relaxed mode, we will only check that the general structure of the issuer and subject fields are correct and allow any number and types of attributes. This seems justified since RPKI explicitly does not use these fields.

Signed Objects (RFC 6488) Signed objects are defined as a profile on CMS messages defined in RFC 5652.

DER Encoding RFC 6488 demands all signed objects to be DER encoded while the more general CMS format allows any BER encoding – DER is a stricter subset of the more general BER. At least one CA does indeed produce BER encoded signed objects.

In relaxed mode, we will allow BER encoding.

Note that this isn't just nit-picking. In BER encoding, octet strings can be broken up into a sequence of sub-strings. Since those strings are in some places used to carry encoded content themselves, such an encoding does make parsing significantly more difficult. At least one CA does produce such broken-up strings.

8.11 Signals

SIGUSR1: Reload TALs and restart validation When receiving SIGUSR1, Routinator will attempt to reload the TALs and, if that succeeds, restart validation. If loading the TALs fails, Routinator will exit.

8.12 Exit Status

Upon success, the exit status 0 is returned. If any fatal error happens, the exit status will be 1. Some commands provide a `--complete` option which will cause the exit status to be 2 if any of the rsync commands to update the repository fail.

INDEX

Symbols

-V
    command line option, 32
--accept-arin-rpa
    command line option, 32
--allow-dubious-hosts
    command line option, 31
--asn=asn
    command line option, 34
--base-dir=dir
    command line option, 29
--chroot=path
    command line option, 35
--complete
    command line option, 33, 34, 36
--config=path
    command line option, 29
--decline-arin-rpa
    command line option, 32
--detach
    command line option, 34
--dirty
    command line option, 31
--disable-rrdp
    command line option, 31
--disable-rsync
    command line option, 31
--exceptions=file
    command line option, 30
--expire=seconds
    command line option, 35
--filter-asn=asn
    command line option, 33
--filter-prefix=prefix
    command line option, 34
--force
    command line option, 32
--format=format
    command line option, 33
--group=group-name
    command line option, 35
--help
    command line option, 32
--history=count
    command line option, 35
--http=addr:port
    command line option, 35
--json
    command line option, 34
--listen-systemd
    command line option, 35
--logfile=path
    command line option, 32
--noupdate
    command line option, 33, 34
--output=file
    command line option, 32, 36
--pid-file=path
    command line option, 35
--prefix=prefix
    command line option, 34
--quiet
    command line option, 32
--refresh=seconds
    command line option, 35
--repository-dir=dir
    command line option, 29
--retry=seconds
    command line option, 35
--rrdp-connect-timeout=seconds
    command line option, 31
--rrdp-local-addr=addr
    command line option, 31
--rrdp-proxy=uri
    command line option, 31
--rrdp-root-cert=path
    command line option, 31
--rrdp-timeout=seconds
    command line option, 31
--rsync-command=command
    command line option, 31
--rsync-timeout=seconds
    command line option, 31
--rtr=addr:port

command line option, 34
--stale=policy
command line option, 30
--strict
command line option, 30
--syslog
command line option, 32
--syslog-facility=facility
command line option, 32
--tal-dir=dir
command line option, 30
--unknown-objects=policy
command line option, 30
--unsafe-vrps=policy
command line option, 30
--user=user-name
command line option, 35
--validation-threads=count
command line option, 31
--verbose
command line option, 32
--version
command line option, 32
--working-dir=path
command line option, 35
-a asn
command line option, 33, 34
-b dir
command line option, 29
-c path
command line option, 29
-d
command line option, 34
-f
command line option, 32
-f format
command line option, 33
-h
command line option, 32
-j
command line option, 34
-n
command line option, 33, 34
-o file
command line option, 32, 36
-p prefix
command line option, 34
-q
command line option, 32
-r dir
command line option, 29
-t dir
command line option, 30
-v
command line option, 32
-x file
command line option, 30

C

command line option
-V, 32
--accept-arin-rpa, 32
--allow-dubious-hosts, 31
--asn=asn, 34
--base-dir=dir, 29
--chroot=path, 35
--complete, 33, 34, 36
--config=path, 29
--decline-arin-rpa, 32
--detach, 34
--dirty, 31
--disable-rrdp, 31
--disable-rsync, 31
--exceptions=file, 30
--expire=seconds, 35
--filter-asn=asn, 33
--filter-prefix=prefix, 34
--force, 32
--format=format, 33
--group=group-name, 35
--help, 32
--history=count, 35
--http=addr:port, 35
--json, 34
--listen-systemd, 35
--logfile=path, 32
--noupdate, 33, 34
--output=file, 32, 36
--pid-file=path, 35
--prefix=prefix, 34
--quiet, 32
--refresh=seconds, 35
--repository-dir=dir, 29
--retry=seconds, 35
--rrdp-connect-timeout=seconds, 31
--rrdp-local-addr=addr, 31
--rrdp-proxy=uri, 31
--rrdp-root-cert=path, 31
--rrdp-timeout=seconds, 31
--rsync-command=command, 31
--rsync-timeout=seconds, 31
--rtr=addr:port, 34
--stale=policy, 30
--strict, 30
--syslog, 32
--syslog-facility=facility, 32
--tal-dir=dir, 30
--unknown-objects=policy, 30

--unsafe-vrps=policy, 30
--user=user-name, 35
--validation-threads=count, 31
--verbose, 32
--version, 32
--working-dir=path, 35
-a asn, 33, 34
-b dir, 29
-c path, 29
-d, 34
-f, 32
-f format, 33
-h, 32
-j, 34
-n, 33, 34
-o file, 32, 36
-p prefix, 34
-q, 32
-r dir, 29
-t dir, 30
-v, 32
-x file, 30

I

init
    module sub-command, 32

M

man
    module sub-command, 36
module sub-command
    init, 32
    man, 36
    server, 34
    update, 36
    validate, 34
    vrps, 32

R

RFC
    RFC 5280, 40
    RFC 5652, 41
    RFC 6487, 40
    RFC 6488, 41
    RFC 6810, 20, 34
    RFC 6810#section-7, 21
    RFC 7128#section-5, 21
    RFC 7730, 36
    RFC 8210, 20, 34, 35
    RFC 8416, 27, 30
    RFC 8630, 36

S

server
    module sub-command, 34

U

update
    module sub-command, 36

V

validate
    module sub-command, 34
vrps
    module sub-command, 32