Routinator User Manual

Release 0.13.1-dev

NLnet Labs

Jan 24, 2024

GETTING STARTED

1 Installation **3**
- 1.1 System Requirements . . . 3
- 1.2 Binary Packages . . . 3
- 1.3 Updating . . . 7
- 1.4 Installing Specific Versions . . . 8

2 Building From Source **11**
- 2.1 Dependencies . . . 11
- 2.2 Building and Updating . . . 12
- 2.3 Statically Linked Routinator . . . 13
- 2.4 Platform Specific Instructions . . . 14

3 Configuration **19**
- 3.1 Setup When Installed From a Package . . . 19
- 3.2 Setup When Built with Cargo . . . 20
- 3.3 Trust Anchor Locators . . . 21
- 3.4 Using Tmpfs for the RPKI Cache . . . 22
- 3.5 Verifying Configuration . . . 22

4 Data Processing **25**
- 4.1 Fetching . . . 25
- 4.2 Validating . . . 26
- 4.3 Storing . . . 27

5 VRP Output Formats **29**

6 Local Exceptions **37**
- 6.1 Limiting Prefix Length . . . 38

7 Logging **39**
- 7.1 Interactive Mode . . . 39
- 7.2 Detached Server Mode . . . 40

8 Running as a Daemon **41**

9 RTR Service **43**
- 9.1 Secure Transports . . . 43
- 9.2 Configuring Routers . . . 45

10 HTTP Service **47**
- 10.1 Query Parameters . . . 47

10.2 TLS Transport . . . 49
10.3 Using a Reverse Proxy . . . 49

11 User Interface **51**

12 API Endpoints **55**

13 Monitoring **57**
13.1 Metrics . . . 57
13.2 Grafana . . . 58

14 Running Interactively **61**
14.1 Query Options . . . 61

15 Validity Checker **65**
15.1 Reading Input From a File . . . 66

16 Dumping Stored Data **69**

17 Manual Page **71**
17.1 Synopsis . . . 71
17.2 Description . . . 71
17.3 Options . . . 71
17.4 Commands . . . 76
17.5 Configuration File . . . 82
17.6 HTTP Service . . . 87
17.7 Logging . . . 88
17.8 Validation . . . 88
17.9 Relaxed Decoding . . . 89
17.10 Signals . . . 90
17.11 Exit Status . . . 90

18 JSON Metrics **91**
18.1 Publication Metrics . . . 92
18.2 Rsync Update Metrics . . . 93
18.3 RRDP Update Metrics . . . 94
18.4 RTR Server Metrics . . . 94
18.5 HTTP Server Metrics . . . 95

19 Prometheus Metrics **97**
19.1 Publication Metrics . . . 97
19.2 Rsync Update Metrics . . . 98
19.3 RRDP Update Metrics . . . 99
19.4 RTR Server Metrics . . . 99
19.5 HTTP Server Metrics . . . 100

20 Unsafe VRPs **101**

21 Advanced Features **103**
21.1 ASPA . . . 103
21.2 BGPsec . . . 104
21.3 Resource Tagged Attestations . . . 106

22 Glossary **107**

Index **109**

Routinator 3000 is free, open-source RPKI (Resource Public Key Infrastructure) Relying Party software made by NLnet Labs. The project is written in Rust, a programming language designed for performance and memory safety.

Lightweight and portable
: Routinator has minimal system requirements and it can run on almost any hardware and platform, with packages available for most. You can also easily run with Docker or Cargo, the Rust package manager.

Full-featured and secure
: Routinator runs as a service that periodically downloads and verifies RPKI data. The built-in HTTPS server offers a user interface, API endpoints for various file formats, as well as logging, status and Prometheus metrics.

Flexible RPKI-to-Router (RTR) support
: Routinator has a built-in RTR server to let routers fetch verified RPKI data. You can also run RTR as a separate daemon using our RPKI data proxy RTRTR, letting you centralise validation and securely distribute processed data to various locations.

Open-source with professional support services
: NLnet Labs offers professional support and consultancy services with a service-level agreement. Community support is available on Discord, Twitter and our mailing list. Routinator is liberally licensed under the BSD 3-Clause license.

CHAPTER

ONE

INSTALLATION

1.1 System Requirements

Routinator has minimal system requirements. When choosing a system, a powerful CPU is not required. Make sure you have 1GB of available memory and 4GB of disk space for the application.

1.1.1 Inode Usage

Please keep in mind that the RPKI consists of a great number of small files. As a result, Routinator will use a large amount of inodes. You should accommodate for at least two million inodes. This will give you ample margin for the RPKI repositories to grow over time, as adoption increases.

Alternatively, you could opt to use a file system such as ZFS, which doesn't use inodes, or btrfs, where inodes are allocated dynamically as needed.

Tip: The `df -i` command shows the amount of inodes available, used, and free.

1.1.2 Firewall Configuration

As new RPKI repositories can emerge in any IP address range and on any domain name, outbound traffic must not be blocked based on IP or DNS in any way. Routinator only needs to establish outbound connections via HTTPS and rsync, on ports 443 and 873, respectively.

1.2 Binary Packages

Getting started with Routinator is really easy by installing a binary package for either Debian and Ubuntu or for Red Hat Enterprise Linux (RHEL) and compatible systems such as Rocky Linux. Alternatively, you can run with Docker.

You can also build Routinator from the source code using Cargo, Rust's build system and package manager. Cargo lets you to run Routinator on almost any operating system and CPU architecture. Refer to the *Building From Source* section to get started.

Debian

To install a Routinator package, you need the 64-bit version of one of these Debian versions:

- Debian Bookworm 12
- Debian Bullseye 11

- Debian Buster 10
- Debian Stretch 9

Packages for the `amd64/x86_64` architecture are available for all listed versions. In addition, we offer `armhf` architecture packages for Debian/Raspbian Bullseye, and `arm64` for Buster.

First update the **apt** package index:

```
sudo apt update
```

Then install packages to allow **apt** to use a repository over HTTPS:

```
sudo apt install \
  ca-certificates \
  curl \
  gnupg \
  lsb-release
```

Add the GPG key from NLnet Labs:

```
curl -fsSL https://packages.nlnetlabs.nl/aptkey.asc | sudo gpg --dearmor -o /usr/share/
↪keyrings/nlnetlabs-archive-keyring.gpg
```

Now, use the following command to set up the *main* repository:

```
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/nlnetlabs-archive-
↪keyring.gpg] https://packages.nlnetlabs.nl/linux/debian \
$(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/nlnetlabs.list > /dev/null
```

Update the **apt** package index once more:

```
sudo apt update
```

You can now install Routinator with:

```
sudo apt install routinator
```

After installation Routinator will run immediately as the user *routinator* and be configured to start at boot. By default, it will run 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
```

You can view the logs with:

```
sudo journalctl --unit=routinator
```

Ubuntu

To install a Routinator package, you need the 64-bit version of one of these Ubuntu versions:

- Ubuntu Jammy 22.04 (LTS)
- Ubuntu Focal 20.04 (LTS)
- Ubuntu Bionic 18.04 (LTS)

- Ubuntu Xenial 16.04 (LTS)

Packages are available for the `amd64/x86_64` architecture only.

First update the **apt** package index:

```
sudo apt update
```

Then install packages to allow **apt** to use a repository over HTTPS:

```
sudo apt install \
  ca-certificates \
  curl \
  gnupg \
  lsb-release
```

Add the GPG key from NLnet Labs:

```
curl -fsSL https://packages.nlnetlabs.nl/aptkey.asc | sudo gpg --dearmor -o /usr/share/
↪keyrings/nlnetlabs-archive-keyring.gpg
```

Now, use the following command to set up the *main* repository:

```
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/nlnetlabs-archive-
↪keyring.gpg] https://packages.nlnetlabs.nl/linux/ubuntu \
$(lsb_release -cs) main" | sudo tee /etc/apt/sources.list.d/nlnetlabs.list > /dev/null
```

Update the **apt** package index once more:

```
sudo apt update
```

You can now install Routinator with:

```
sudo apt install routinator
```

After installation Routinator will run immediately as the user *routinator* and be configured to start at boot. By default, it will run 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
```

You can view the logs with:

```
sudo journalctl --unit=routinator
```

RHEL/CentOS

To install a Routinator package, you need Red Hat Enterprise Linux (RHEL) 7, 8 or 9, or compatible operating system such as Rocky Linux. Packages are available for the `amd64/x86_64` architecture only.

First create a file named `/etc/yum.repos.d/nlnetlabs.repo`, enter this configuration and save it:

```
[nlnetlabs]
name=NLnet Labs
```

(continues on next page)

(continued from previous page)

```
baseurl=https://packages.nlnetlabs.nl/linux/centos/$releasever/main/$basearch
enabled=1
```

Add the GPG key from NLnet Labs:

```
sudo rpm --import https://packages.nlnetlabs.nl/aptkey.asc
```

You can now install Routinator with:

```
sudo yum install -y routinator
```

After installation Routinator will run immediately as the user *routinator* and be configured to start at boot. By default, it will run 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
```

You can view the logs with:

```
sudo journalctl --unit=routinator
```

Docker

Routinator Docker images are built with Alpine Linux. The supported CPU architectures are shown on the Docker Hub Routinator page per Routinator version (aka Docker "tag") in the `OS/ARCH` column.

To run Routinator as a background daemon with the default settings (RTR server on port 3323 and HTTP server on port 8323) can be done like so:

```
sudo docker run -d --restart=unless-stopped --name routinator \
    -p 3323:3323 \
    -p 8323:8323 \
    nlnetlabs/routinator
```

Tip: If no arguments are supplied the Routinator Docker image configures Routinator to run in *server* mode, with *--rtr* 3323 and *--http* 8323.

For backward compatibility with earlier releases it also configures Routinator with *--http* 9556, the port number allocated by the Prometheus project for Routinator metric publication.

The Routinator container is known to run successfully run under gVisor for additional isolation.

To adjust the configuration you can pass command line arguments to Routinator (try *--help* for more information) and/or supply your own Routinator configuration file (by mapping it from the host into the container using `-v host/path/to/routinator.conf:/etc/routinator.conf` and passing `--config /etc/routinator.conf` when running the container).

For example in an IPv6 only network you could invoke Routinator like so to have it listen on IPv6 as well as IPv4:

```
sudo docker run <your usual arguments> \
    server --rtr [::]:3323 --http [::]:8323
```

Note the `server` command passed to Routinator. When you override the default arguments passed to Routinator by the Docker image you must provide all of the arguments required by Routinator. See the *Manual Page* for more information.

To persist the RPKI cache data you can create a separate Docker volume and mount it into the container like so:

```
sudo docker volume create rpki-cache
sudo docker run <your usual arguments> \
    -v rpki-cache:/home/routinator/.rpki-cache \
    nlnetlabs/routinator
```

New in version 0.9.0: RPM packages

New in version 0.11.0: Debian packages for `armhf` and `arm64` architecture

New in version 0.11.2: Ubuntu packages for Jammy 22.04 (LTS)

Deprecated since version 0.12.0: `routinator-init` and `--accept-arin-rpa`

New in version 0.13.0: Packages for Debian Bookworm 12 and RHEL 9

1.3 Updating

Debian

To update an existing Routinator installation, first update the repository using:

```
sudo apt update
```

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

```
sudo apt policy routinator
```

You can upgrade an existing Routinator installation to the latest version using:

```
sudo apt --only-upgrade install routinator
```

Ubuntu

To update an existing Routinator installation, first update the repository using:

```
sudo apt update
```

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

```
sudo apt policy routinator
```

You can upgrade an existing Routinator installation to the latest version using:

```
sudo apt --only-upgrade install routinator
```

RHEL/CentOS

To update an existing Routinator installation, you can use this command to get an overview of the available versions:

```
sudo yum --showduplicates list routinator
```

You can update to the latest version using:

```
sudo yum update -y routinator
```

Docker

Assuming that you run Docker with image *nlnetlabs/routinator*, upgrading to the latest version can be done by running the following commands:

```
sudo docker pull nlnetlabs/routinator
sudo docker rm --force routinator
sudo docker run <your usual arguments> nlnetlabs/routinator
```

1.4 Installing Specific Versions

Before every new release of Routinator, one or more release candidates are provided for testing through every installation method. You can also install a specific version, if needed.

Debian

If you would like to try out release candidates of Routinator you can add the *proposed* repository to the existing *main* repository described earlier.

Assuming you already have followed the steps to install regular releases, run this command to add the additional repository:

```
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/nlnetlabs-archive-
↪keyring.gpg] https://packages.nlnetlabs.nl/linux/debian \
$(lsb_release -cs)-proposed main" | sudo tee /etc/apt/sources.list.d/nlnetlabs-proposed.
↪list > /dev/null
```

Make sure to update the **apt** package index:

```
sudo apt update
```

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

```
sudo apt policy routinator
```

You can install a specific version using <package name>=<version>, e.g.:

```
sudo apt install routinator=0.9.0~rc2-1buster
```

Ubuntu

If you would like to try out release candidates of Routinator you can add the *proposed* repository to the existing *main* repository described earlier.

Assuming you already have followed the steps to install regular releases, run this command to add the additional repository:

```
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/nlnetlabs-archive-
↪keyring.gpg] https://packages.nlnetlabs.nl/linux/ubuntu \
$(lsb_release -cs)-proposed main" | sudo tee /etc/apt/sources.list.d/nlnetlabs-proposed.
↪list > /dev/null
```

Make sure to update the **apt** package index:

```
sudo apt update
```

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

```
sudo apt policy routinator
```

You can install a specific version using <package name>=<version>, e.g.:

```
sudo apt install routinator=0.9.0~rc2-1bionic
```

RHEL/CentOS

To install release candidates of Routinator, create an additional repo file named `/etc/yum.repos.d/nlnetlabs-testing.repo`, enter this configuration and save it:

```
[nlnetlabs-testing]
name=NLnet Labs Testing
baseurl=https://packages.nlnetlabs.nl/linux/centos/$releasever/proposed/$basearch
enabled=1
```

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

```
sudo yum --showduplicates list routinator
```

You can install a specific version using <package name>-<version info>, e.g.:

```
sudo yum install -y routinator-0.9.0~rc2
```

Docker

All release versions of Routinator, as well as release candidates and builds based on the latest main branch are available on Docker Hub.

For example, installing Routinator 0.9.0 RC2 is as simple as:

```
sudo docker run <your usual arguments> nlnetlabs/routinator:v0.9.0-rc2
```

CHAPTER

TWO

BUILDING FROM SOURCE

In addition to meeting the *system requirements*, there are three things you need to build Routinator: rsync, a C toolchain and Rust. You can run Routinator on any operating system and CPU architecture where you can fulfil these requirements.

2.1 Dependencies

To get started you need rsync because some RPKI repositories still 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.

2.1.1 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, as well as macOS should be fine. Alternatively, you can download rsync from the Samba website.

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

2.1.2 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, 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.

2.1.3 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 1.70 or newer. We therefore suggest to use the canonical Rust installation via a tool called **`rustup`**.

Assuming you already have **`curl`** installed, you can install **`rustup`** and Rust by simply entering:

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

Alternatively, visit the Rust website for other installation methods.

2.2 Building and Updating

In Rust, a library or executable program such as Routinator is called a *crate*. Crates are published on crates.io, the Rust package registry. Cargo is the Rust package manager. It is a tool that allows Rust packages to declare their various dependencies and ensure that you'll always get a repeatable build.

Cargo fetches and builds Routinator's dependencies into an executable binary for your platform. By default you install from crates.io, but you can for example also install from a specific Git URL, as explained below.

Installing the latest Routinator release from crates.io is as simple as running:

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

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.

2.2.1 Updating

If you want to update to the latest version of Routinator, it's recommended to update Rust itself as well, using:

```
rustup update
```

Use the `--force` option to overwrite an existing version with the latest Routinator release:

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

2.2.2 Installing Specific Versions

If you want to install 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 routinator --version 0.9.0-rc2
```

All new features of Routinator are built on a branch and merged via a pull request, allowing you to easily try them out using Cargo. If you want to try a specific branch from the repository you can use the `--git` and `--branch` options:

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

See also:

For more installation options refer to the Cargo book.

2.2.3 Enabling or Disabling Features

When you build Routinator, "features" provide a mechanism to express conditional compilation and optional dependencies. The Routinator package defines a set of named features in the `[features]` table of Cargo.toml. The table also defines if a feature is enabled or disabled by default.

Routinator currently has the following features:

`socks` — *Enabled* by default
: Allow the configuration of a SOCKS proxy.

`ui` — *Enabled* by default
: Download and build the the routinator-ui crate to run the *user interface*.

`native-tls` — *Disabled* by default
: Use the native TLS implementation of your system instead of rustls.

`rta` — *Disabled* by default
: Let Routinator validate *Resource Tagged Attestations*.

`aspa` — *Disabled* by default
: Let Routinator validate *ASPA* objects.

Note: ASPA support is temporarily behind a feature flag while the draft is under discussion in the IETF. This way operators can gain operational experience without unintended side effects.

To disable the features that are enabled by default, use the `--no-default-features` option. You can then choose which features you want using the `--features` option, listing each feature separated by commas.

For example, if you want to build Routinator without the user interface, make sure SOCKS support is retained and use the native TLS implementation, enter the following command:

```
cargo install --locked --no-default-features --features socks,native-tls routinator
```

2.3 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
```

2.4 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.

2.4.1 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
```

2.4.2 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
```

2.4.3 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
```

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 --locked 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. 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
```

15. 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
```

16. 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
```

17. 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'
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␣
```

(continues on next page)

(continued from previous page)

```
↪accept'
sudo firewall-cmd --reload
```

18. Navigate to `https://<IP-address>/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.

Important: With Routinator 0.12.0 and newer, initialisation to accept the ARIN Relying Party Agreement (RPA) is no longer required. The RPA has been updated to allow the ARIN TAL to be embedded in Relying Party software. By default, Routinator is now set up to fetch and validate all RPKI data needed for production environments.

CHAPTER

THREE

CONFIGURATION

Routinator has a large number of configuration options, but in most cases running it with the defaults will work just fine. You can specify options as *command line arguments*, but you can also use a *configuration file*.

Routinator uses the TOML format for specifying options in the configuration 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.

Routinator can run as a daemon but you can also use it interactively from the command line. There are several considerations with regards to how you've installed and how you intend to use Routinator, which we'll cover below.

3.1 Setup When Installed From a Package

The installation script will set up Routinator to run as the user *routinator* and be configured to start at boot. Routinator will use the configuration file `/etc/routinator/routinator.conf` which contains the following pre-configured options:

```
repository-dir = "/var/lib/routinator/rpki-cache"
rtr-listen = ["127.0.0.1:3323"]
http-listen = ["127.0.0.1:8323"]
```

For security reasons the HTTP and RTR server will only listen on localhost, so you will have to change these values to make them accessible to other devices on your network.

The service script that starts Routinator uses the `--config` option to explicitly refer to this configuration file, so any desired changes should be made here. If you would like to know what default settings Routinator runs with in addition to the settings in the config file, you can check with the `config` subcommand:

```
routinator --config /etc/routinator/routinator.conf config
```

This output will also provide you with the correct syntax in case you want to make changes.

Important: Once you have started Routinator as a system service you should not invoke *interactive* validation runs from the command line using `routinator vrps`. If there is specific information you would like to have from Routinator, you should retrieve it via the *user interface* or one of the *HTTP endpoints*.

3.2 Setup When Built with Cargo

If you have built Routinator using Cargo, you have made your own decisions with regards to the user that it runs as and the privileges it has. There is no default configuration file, as it is your choice if you want to use one.

If you run Routinator without referring to a configuration file it will check if the file `$HOME/.routinator.conf` exists and if it does, use it. If no configuration file is available, the default values are used.

You can specify the location of the RPKI cache directory using the *--repository-dir* option. If you don't, one will be created in the default location `$HOME/.rpki-cache/repository`. The *HTTP service* and *RTR service* must be started explicitly using the command line options *--http* and *--rtr*, respectively, or via the configuration file.

You can view the default settings Routinator runs with using:

```
routinator config
```

It will return the list of defaults in the same notation that is used by the *configuration file*, which will be largely similar to this and can serve as a starting point for making your own:

```
allow-dubious-hosts = false
dirty = false
disable-rrdp = false
disable-rsync = false
enable-aspa = false
enable-bgpsec = false
exceptions = []
expire = 7200
history-size = 10
http-listen = []
http-tls-listen = []
log = "default"
log-level = "WARN"
max-ca-depth = 32
max-object-size = 20000000
refresh = 600
repository-dir = "/Users/routinator/.rpki-cache/repository"
retry = 600
rrdp-fallback-time = 3600
rrdp-max-delta-count = 100
rrdp-proxies = []
rrdp-root-certs = []
rrdp-timeout = 300
rsync-command = "rsync"
rsync-timeout = 300
rtr-client-metrics = false
rtr-listen = []
rtr-tcp-keepalive = 60
rtr-tls-listen = []
stale = "reject"
strict = false
syslog-facility = "daemon"
systemd-listen = false
unknown-objects = "warn"
unsafe-vrps = "accept"
validation-threads = 10
```

3.3 Trust Anchor Locators

Fetching data is done by connecting to the *Trust Anchor Locators (TALs)* of the five Regional Internet Registries (RIRs): AFRINIC, APNIC, ARIN, LACNIC and RIPE NCC. TALs provide hints for the trust anchor certificates to be used both to discover and validate all RPKI content. **By default, Routinator will be set up for use in production environments and run with the production TALs of the five RIRs.**

Some RIRs and third parties also provide separate TALs for testing purposes, allowing operators to gain experience with using RPKI in a safe environment. Both the production and testbed TALs are bundled with Routinator and can be enabled and disabled using command line and configuration file options.

Run the following command to list all available TALs:

```
routinator --tal=list
```

This displays the following overview:

```
.---- RIR TALs
|  .- RIR test TALs
V  V

X      afrinic             AFRINIC production TAL
X      apnic               APNIC production TAL
X      arin                ARIN production TAL
X      lacnic              LACNIC production TAL
X      ripe                RIPE production TAL
   X   apnic-testbed       APNIC RPKI Testbed
   X   arin-ote            ARIN Operational Test and Evaluation Environment
   X   ripe-pilot          RIPE NCC RPKI Test Environment
       nlnetlabs-testbed   NLnet Labs RPKI Testbed
```

You can influence which TALs Routinator uses with the `--tal` option, which can be combined with the `--no-rir-tals` option to leave out all RIR production TALs, as well as the `--extra-tals-dir` option to specify a directory containing extra TALs to use.

For example, if you want to add the RIPE NCC RPKI Test Environment to the default TAL set, run:

```
routinator --tal=ripe-pilot
```

If you want to run Routinator without any of the production TALs and only fetch data from the ARIN Operational Test and Evaluation Environment, run:

```
routinator --no-rir-tals --tal=arin-ote
```

Lastly, if you would like to use a TAL that isn't bundled with Routinator you can place it in a directory of your choice, for example `/var/lib/routinator/tals`, and refer to it by running:

```
routinator --extra-tals-dir="/var/lib/routinator/tals"
```

Routinator will use all files in this directory with an extension of *.tal* as TALs. These files need to be in the format described by **RFC 8630**. Note that Routinator will use all TALs provided. That means that if a TAL in this directory is one of the bundled TALs, then these resources will be validated twice.

New in version 0.9.0: `--list-tals`, `--rir-tals`, `--rir-test-tals`, `--tal` and `--skip-tal`

Deprecated since version 0.9.0: `--decline-arin-rpa`, use `--skip-tal` instead

New in version 0.12.0: *--extra-tals-dir*

Deprecated since version 0.12.0: The `init` subcommand, `--list-tals`

3.4 Using Tmpfs for the RPKI Cache

The full RPKI data set consists of hundreds of thousands of small files. This causes a considerable amount of disk I/O with each validation run. If this is undesirable in your setup, you can choose to store the cache in volatile memory using the tmpfs file system.

If you have installed Routinator using a package, by default the RPKI cache directory will be `/var/lib/routinator/rpki-cache`, so we'll use that as an example. Note that the directory you choose must exist before the mount can be done. You should allocate at least 3GB for the cache, but giving it 4GB will allow ample margin for future growth:

```
sudo mount -t tmpfs -o size=4G tmpfs /var/lib/routinator/rpki-cache
```

Tmpfs will behave just like a regular disk, so if it runs out of space Routinator will do a clean crash, stopping validation, the API, HTTP server and most importantly the RTR server, ensuring that no stale data will be served to your routers.

Also keep in mind that every time you restart the machine, the contents of the *tmpfs* file system will be lost. This means that Routinator will have to rebuild its cache from scratch. This is not a problem, other than it having to download several hundred megabytes of data, which usually takes about ten minutes to complete. During this time all services will be unavailable.

Note that your routers should be configured to have a secondary relying party instance available at all times.

3.5 Verifying Configuration

You should verify if Routinator has been configured correctly and your firewall allows the required outbound connections on ports 443 and 873. From a cold start, it will take ten to fifteen minutes to do the first validation run that builds up the validated cache. Subsequent runs will be much faster, because only the changes between the repositories and the validated cache need to be processed.

If you have installed Routinator from a package and run it as a service, you can check the status using:

```
sudo systemctl status routinator
```

And check the logs using:

```
sudo journalctl --unit=routinator
```

Important: Because it is expected that the state of the entire RPKI is not perfect at all times, you may see several warnings about objects that are either stale or failed cryptographic verification, or repositories that are temporarily unavailable.

If you have built Routinator using Cargo 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* twice to increase the *log level* to *debug*:

```
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 VRPs in the default CSV format to standard output.

```
[INFO] Using the following TALs:
[INFO]   * afrinic
[INFO]   * apnic
[INFO]   * arin
[INFO]   * lacnic
[INFO]   * ripe
[DEBUG] Found valid trust anchor https://rpki.ripe.net/ta/ripe-ncc-ta.cer. Processing.
[DEBUG] Found valid trust anchor https://rrdp.lacnic.net/ta/rta-lacnic-rpki.cer.␣
↪Processing.
[DEBUG] Found valid trust anchor https://rpki.afrinic.net/repository/AfriNIC.cer.␣
↪Processing.
[DEBUG] Found valid trust anchor https://rpki.apnic.net/repository/apnic-rpki-root-iana-
↪origin.cer. Processing.
[DEBUG] Found valid trust anchor https://rrdp.arin.net/arin-rpki-ta.cer. Processing.
[DEBUG] RRDP https://rrdp.ripe.net/notification.xml: updating from snapshot.
[DEBUG] RRDP https://rrdp.lacnic.net/rrdp/notification.xml: updating from snapshot.
[DEBUG] RRDP https://rrdp.apnic.net/notification.xml: updating from snapshot.
[DEBUG] RRDP https://rrdp.afrinic.net/notification.xml: updating from snapshot.
[DEBUG] RRDP https://rrdp.arin.net/notification.xml: updating from snapshot.
[DEBUG] RRDP https://rrdp.apnic.net/notification.xml: snapshot update completed.
[DEBUG] RRDP https://rpki-rrdp.us-east-2.amazonaws.com/rrdp/08c2f264-23f9-49fb-9d43-
↪f8b50bec9261/notification.xml: updating from snapshot.
[DEBUG] RRDP https://rpki-rrdp.us-east-2.amazonaws.com/rrdp/08c2f264-23f9-49fb-9d43-
↪f8b50bec9261/notification.xml: snapshot update completed.
[DEBUG] RRDP https://rpki.akrn.net/rrdp/notification.xml: updating from snapshot.
[DEBUG] RRDP https://rpki.akrn.net/rrdp/notification.xml: snapshot update completed.
[DEBUG] RRDP https://rpki.admin.freerangecloud.com/rrdp/notification.xml: updating from␣
↪snapshot.
[DEBUG] RRDP https://rpki.admin.freerangecloud.com/rrdp/notification.xml: snapshot␣
↪update completed.
[DEBUG] RRDP https://rpki.cnnic.cn/rrdp/notify.xml: updating from snapshot.
[DEBUG] RRDP https://rrdp.ripe.net/notification.xml: snapshot update completed.
[DEBUG] RRDP https://0.sb/rrdp/notification.xml: updating from snapshot.
[DEBUG] RRDP https://0.sb/rrdp/notification.xml: snapshot update completed.
[DEBUG] RRDP https://rrdp.sub.apnic.net/notification.xml: updating from snapshot.
[DEBUG] RRDP https://rrdp.sub.apnic.net/notification.xml: snapshot update completed.
[DEBUG] RRDP https://rpki.roa.net/rrdp/notification.xml: updating from snapshot.
[DEBUG] RRDP https://rpki.roa.net/rrdp/notification.xml: snapshot update completed.
[DEBUG] RRDP https://rrdp.rp.ki/notification.xml: updating from snapshot.
[DEBUG] RRDP https://rpki.cnnic.cn/rrdp/notify.xml: snapshot update completed.
...
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
```

(continues on next page)

(continued from previous page)

```
...
```

CHAPTER

FOUR

DATA PROCESSING

4.1 Fetching

There are two protocols in use to transport RPKI data: rsync and the *RPKI Repository Delta Protocol (RRDP)*, which relies on HTTPS. RRDP was designed to be the successor to rsync in the RPKI. As all RPKI repositories currently advertise support for both protocols, Routinator will prefer RRDP if available.

In the RPKI, the certificate hierarchy follows the same structure as the Internet number resource allocation hierarchy. Routinator starts traversing the tree by connecting to the *trust anchors* of the Regional Internet Registries (RIRs). Along the way Routinator will find several pointers to child *publication points*, such as the ones operated by National Internet Registries (NIRs), Local Internet Registries (LIRs) and organisations running delegated RPKI. Each pointer explicitly states if RRDP is offered in addition to rsync.

Fig. 1: The RPKI hierarchy

As a precaution, Routinator will not accept rsync and HTTPS URIs from *RPKI repositories* with dubious hostnames. In particular, it will reject the name *localhost*, URIs that consist of IP addresses, and hostnames that contain an explicit port. You can change this behaviour with the `--allow-dubious-hosts` option.

4.1.1 RRDP Fallback

If an RRDP endpoint is unavailable but it has worked in the past, Routinator will assume this is a transient problem. What action is taken is determined by the `--rrdp-fallback` option. The default policy is *stale*. This means Routinator will retry using RRDP for up to 60 minutes since the last successful update, during which it will rely on the locally cached data for this repository. After this time, Routinator will try to use rsync to fetch the data instead. To spread out load on the rsync server, the exact moment fallback happens is picked randomly between the refresh time and the `--rrdp-fallback-time` value. If rsync communication is unsuccessful too, the local cache is used until the objects go stale and ultimately expire.

Another policy for `--rrdp-fallback` is *never*. This means that rsync is never tried for a CA that advertises the availability of RRDP. Lastly, the policy *new* means that rsync is tried if an update via RRDP fails and there is no local copy of the RRDP repository at all. In other words, an update via RRDP has never succeeded for the repository. Choosing this policy allows a repository operator some leeway when first enabling RRDP support.

New in version 0.9.0.

Changed in version 0.12.0: The `--rrdp-fallback` option

4.1.2 Update Interval

Routinator will fetch new RPKI data ten minutes after the last successful update has finished. The interval can be changed using the `--refresh` option. It is possible that it takes very long to update a repository due to temporary network problems. To ensure a slow repository doesn't stop the entire update process from completing, Routinator has a timeout for stalled connections. For RRDP, this timeout is implemented as an HTTP request timeout. For rsync, the timeout is around the spawned rsync process. The default is five minutes for both and can be changed via the `--rsync-timeout` and `--rrdp-timeout` options.

4.2 Validating

The validation process determines if all certificates, Route Origin Attestations (ROAs) and other signed objects that may appear in the RPKI have the correct signatures. It will also verify if the hashes are correct, no objects have expired and the entire data set is complete. If any of the objects do not pass these checks, the data will be discarded.

Currently, only certificates (.cer), certificate revocation lists (.crl), manifests (.mft), ROAs (.roa), and Ghostbuster Records (.gbr) are allowed to appear the RPKI. If another type of object is encountered Routinator will *warn* by default, but this can be changed with the `--unknown-objects` option.

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 Certificate Authority (CA) and all its objects are still rejected.

4.2.1 Stale Objects

During the validation process, Routinator may encounter objects that are *stale*. In RPKI, manifests and CRLs (Certificate Revocation Lists) 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.

Ongoing standards efforts and operational experiences suggest that stale objects should be rejected, which is the default policy set by the `--stale` option since Routinator 0.8.0. As a result, all material published by the CA issuing this manifest and CRL is considered invalid, including all material of any child CA.

4.2.2 ROAs and VRPs

ROAs are *cryptographic* objects that contain a statement authorising a *single* Autonomous System Number (ASN) to originate *one or more* IP prefixes, along with their maximum prefix length. ROAs can only be created by the legitimate holder of the IP prefixes contained within it, but they can authorise any ASN.

If the ROA passes validation, Routinator will produce one or more *plain text* validated ROA payloads (VRPs) for each ROA, depending on how many IP prefixes are contained within it. Each VRP is a tuple of an ASN, a single prefix and its maximum prefix length. The complete collection of VRPs can be expressed in formats such as CSV or JSON, or exposed via the RPKI-to-Router (RTR) protocol so that they can be compared to all BGP origins seen by your routers. For each route origin it can be determined if they are RPKI *"Valid"*, *"Invalid"* or *"NotFound"*.

4.2.3 Unsafe VRPs

If the address prefix of a VRP overlaps with any resources assigned to a CA that has been rejected because it failed to validate completely, the VRP is said to be *unsafe* since using it may lead to legitimate routes being flagged as RPKI Invalid.

Routinator has an `--unsafe-vrps` option that specifies how to deal with these types of 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.

You can learn more about this topic in the *Unsafe VRPs* section.

4.3 Storing

To be resistant against accidental or malicious errors in the data published by repositories, Routinator retains two separate data sets: one that keeps the data of all publication points as it was received from their remote repository, and another – which we call the *store* – keeps the most recent data of a given RPKI publication point that was found to be correctly published.

Data is only transferred into the store if a manifest was found to be valid and if all files mentioned on the manifest are present and have the correct hash. Otherwise the data for the publication point already present in the store will be used for validation.

If you ever want or need to clear all stored data, you can use the `--fresh` option. This will be like starting Routinator for the very first time:

```
routinator --fresh vrps
```

New in version 0.9.0.

CHAPTER

FIVE

VRP OUTPUT FORMATS

Routinator can perform RPKI validation as a one-time operation or run as a daemon. In both operating modes validated ROA payloads (VRPs) can be generated in a wide range of output formats for various use cases.

Tip: In many of the output formats, the name of the trust anchor from where the VRP originated is provided. This name is derived from the file name of the TAL, without the *.tal* extension. If you would like a different name, the *tal-label* option in the *configuration file* lets you create a mapping between the file name and your desired label.

csv

The list is formatted as lines of comma-separated values of the following items:

- The prefix in slash notation,
- the maximum prefix length,
- the Autonomous System Number, and
- the name of the trust anchor the entry is derived from.

```
ASN,IP Prefix,Max Length,Trust Anchor
AS196615,2001:7fb:fd03::/48,48,ripe
AS196615,2001:7fb:fd04::/48,48,ripe
AS196615,93.175.147.0/24,24,ripe
```

csvcompat

This is the same as the *csv* format 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 format produced by the RIPE NCC RPKI Validator.

```
"ASN","IP Prefix","Max Length","Trust Anchor"
"196615","2001:7fb:fd03::/48","48","ripe"
"196615","2001:7fb:fd04::/48","48","ripe"
"196615","93.175.147.0/24","24","ripe"
```

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

- the not-before and not-after date of the validity of the ROA.

Note: This format is available for backwards compatibility reasons only. One particular limitation is that it does not consider duplicate ROAs. Please use *jsonext* as a comprehensive output format.

```
URI,ASN,IP Prefix,Max Length,Not Before,Not After
rsync://rpki.ripe.net/repository/DEFAULT/73/fe2d72-c2dd-46c1-9429-e66369649411/1/
↪49sMtcwyAuAW2lVDSQBGhOHd9og.roa,AS196615,2001:7fb:fd03::/48,48,2021-05-03␣
↪14:51:30,2022-07-01 00:00:00
rsync://rpki.ripe.net/repository/DEFAULT/73/fe2d72-c2dd-46c1-9429-e66369649411/1/
↪49sMtcwyAuAW2lVDSQBGhOHd9og.roa,AS196615,2001:7fb:fd04::/48,48,2021-05-03␣
↪14:51:30,2022-07-01 00:00:00
rsync://rpki.ripe.net/repository/DEFAULT/73/fe2d72-c2dd-46c1-9429-e66369649411/1/
↪49sMtcwyAuAW2lVDSQBGhOHd9og.roa,AS196615,93.175.147.0/24,24,2021-05-03 14:51:30,
↪2022-07-01 00:00:00
```

json

The list is placed into a JSON object with up to four members:

- *roas* contains the validated route origin authorisations,
- *routerKeys* contains the validated *BGPsec* router keys,
- *aspas* contains the validated *ASPA* payload, and
- *metadata* contains some information about the validation run itself.

Of the first three, only those members are present that have not been disabled or excluded.

The *roas* member contains an array of objects with four elements each:

- *asn* lists the Autonomous System Number of the network authorised to originate a prefix,
- *prefix* has the prefix in slash notation,
- *maxLength* states the maximum prefix length of the announced route, and
- *ta* has the trust anchor from which the authorisation was derived.

The *routerKeys* member contains an array of objects with four elements each:

- *asn* contains the autonomous system using the router key,
- *SKI* lists the key identifier as a string of hexadecimal digits,
- *routerPublicKey* contains the actual public key as a Base 64 encoded string, and
- *ta* has the trust anchor from which the authorisation was derived.

The *aspa* member contains an array of objects with four members each:

- *customer* contains the customer ASN,
- *afi* lists the address family as either "ipv4" or "ipv6",
- *providers* contains the provider ASN set as an array, and
- *ta* has the trust anchor from which the authorisation was derived.

The output object also includes a member named *metadata* which provides additional information. Currently, this is a member *generated* which provides the time the list was generated as a Unix timestamp, and a member *generatedTime* which provides the same time but in the standard ISO date format.

```
{
  "metadata": {
    "generated": 1685455841,
    "generatedTime": "2023-05-30T14:10:41Z"
  },
  "roas": [{
    "asn": "AS196615",
    "prefix": "93.175.147.0/24",
    "maxLength": 24,
    "ta": "ripe"
    }
  ],
  "routerKeys": [{
    "asn": "AS211321",
    "SKI": "17316903F0671229E8808BA8E8AB0105FA915A07",
    "routerPublicKey": "MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAET10FMBxP6P3r6aG_
↪ICpfsktp7X6ylJIY8Kye6zkQhNOt0y-cRzYngH8MGzY3cXNvZ64z4CpZ22gf4teybGq8ow",
    "ta": "ripe"
  }],
  "aspas": [{
    "customer": "AS64496",
    "afi": "ipv6",
    "providers": ["AS64499", "AS64511", "AS65551"],
    "ta": "ripe"
  }]
}
```


Changed in version 0.10.0: Add the *metadata* member

Changed in version 0.13.0: Add the *routerKeys* and *aspas* members

jsonext

The list is placed into a JSON object with up to four members:

- *roas* contains the validated route origin authorisations,
- *routerKeys* contains the validated *BGPsec* router keys,
- *aspas* contains the validated *ASPA* objects, and
- *metadata* contains some information about the validation run itself.

Of the first three, only those members are present that have not been disabled or excluded.

The *roas* member contains an array of objects with four elements each:

- *asn* lists the Autonomous System Number of the network authorised to originate a prefix,
- *prefix* has the prefix in slash notation,
- *maxLength* states the maximum prefix length of the announced route, and
- *source* contains information about the source of the authorisation.

The *routerKeys* member contains an array of objects with four elements each:

- *asn* lists the autonomous system using the router key,
- *SKI* has the key identifier as a string of hexadecimal digits,
- *routerPublicKey* has the actual public key as a Base 64 encoded string, and

- *source* contains extended information about the source of the key.

The *aspas* member contains an array of objects with four elements each:

- *customer* contains the customer ASN,
- *afi* specifies the address family as either "ipv4" or "ipv6",
- *providers* contains the provider ASN set as an array, and
- *source* contains information about the source of the authorisation.

This source information the same for route origins, router keys and aspas. It consists of an array. Each item in that array is an object providing details of a source. The object will have a *type* of *roa* if it was derived from a valid ROA object, *cer* if it was derived from a published router certificate, *aspa* if it was derived from an ASPA object, or *exception* if it was an assertion in a local exception file.

For RPKI objects, *tal* provides the name of the trust anchor locator the object was published under, *uri* provides the rsync URI of the ROA or router certificate, *validity* provides the validity of the ROA itself, and *chainValidity* the validity considering the validity of the certificates along the validation chain.

For assertions from local exceptions, *path* will provide the path of the local exceptions file and, optionally, *comment* will provide the comment if given for the assertion.

The output object also includes a member named *metadata* which provides additional information. Currently, this is a member *generated* which provides the time the list was generated as a Unix timestamp, and a member *generatedTime* which provides the same time but in the standard ISO date format.

Please note that because of this additional information, output in jsonext format will be quite large.


```
{
  "metadata": {
    "generated": 1658818561,
    "generatedTime": "2022-07-26T06:56:01Z"
  },
  "roas": [{
      "asn": "AS211321",
      "prefix": "185.49.142.0/24",
      "maxLength": 24,
      "source": [{
        "type": "roa",
        "tal": "ripe",
        "uri": "rsync://testbed.krill.cloud/repo/local-testbed-child/0/
↪3138352e34392e3134322e302f32342d3234203d3e20323131333231.roa",
        "validity": {
          "notBefore": "2022-07-25T20:47:37Z",
          "notAfter": "2023-07-24T20:52:37Z"
        },
        "chainValidity": {
          "notBefore": "2022-07-25T20:47:37Z",
          "notAfter": "2023-02-24T12:31:01Z"
        }
      }]
    }
  ],
  "routerKeys": [{
    "asn": "AS211321",
    "SKI": "17316903F0671229E8808BA8E8AB0105FA915A07",
```


(continues on next page)

(continued from previous page)


```
    "routerPublicKey": "MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAET10FMBxP6P3r6aG_
↪ICpfsktp7X6ylJIY8Kye6zkQhNOt0y-cRzYngH8MGzY3cXNvZ64z4CpZ22gf4teybGq8ow",
    "source": [{
      "type": "cer",
        "tal": "ripe",
      "uri": "rsync://testbed.krill.cloud/repo/local-testbed-child/0/ROUTER-
↪00033979-17316903F0671229E8808BA8E8AB0105FA915A07.cer",
      "validity": {
        "notBefore": "2022-07-25T20:47:37Z",
        "notAfter": "2023-07-24T20:52:37Z"
      },
      "chainValidity": {
        "notBefore": "2022-07-25T20:47:37Z",
        "notAfter": "2023-02-24T12:31:01Z"
      }
    }]
  }],
  "aspas": [{
    "customer": "AS64496",
    "afi": "ipv6",
    "providers": ["AS64499", "AS64511", "AS65551"],
    "source": [{
        "type": "aspa",
        "uri": "rsync://acmecorp.example.net/0/AS64496.asa",
        "tal": "ripe",
        "validity": {
            "notBefore": "2023-04-13T07:21:24Z",
            "notAfter": "2024-04-11T07:26:24Z"
        },
        "chainValidity": {
            "notBefore": "2023-04-18T14:32:13Z",
            "notAfter": "2024-04-11T07:26:24Z"
          }
      }]
  }]
}
```


New in version 0.9.0.

Changed in version 0.10.0: Add metadata

Changed in version 0.11.0: Add *BGPsec* information

Changed in version 0.13.0: Add *ASPA* information

Changed in version 0.13.0: Only include members that have not been disabled or excluded

slurm

The list is formatted as locally added assertions of a *local exceptions* file defined by **RFC 8416** (also known as SLURM). The produced file will have empty validation output filters.


```
{
  "slurmVersion": 1,
  "validationOutputFilters": {
    "prefixFilters": [ ],
```


(continues on next page)

(continued from previous page)


```
    "bgpsecFilters": [ ]
  },
  "locallyAddedAssertions": {
    "prefixAssertions": [
      {
        "asn": 196615,
        "prefix": "93.175.147.0/24",
        "maxPrefixLength": 24,
        "comment": "ripe"
      },
      {
        "asn": 196615,
        "prefix": "2001:7fb:fd03::/48",
        "maxPrefixLength": 48,
        "comment": "ripe"
      },
      {
        "asn": 196615,
        "prefix": "2001:7fb:fd04::/48",
        "maxPrefixLength": 48,
        "comment": "ripe"
      }
    ],
    "bgpsecAssertions": [

    ]
  }
}
```


New in version 0.11.0.

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

```
roa-set {
    2001:7fb:fd03::/48 source-as 196615
    2001:7fb:fd04::/48 source-as 196615
    93.175.147.0/24 source-as 196615
}
```

bird1
Choosing this format causes Routinator to produce a ROA table configuration item for use with BIRD 1.6.

```
roa 2001:7fb:fd03::/48 max 48 as 196615;
roa 2001:7fb:fd04::/48 max 48 as 196615;
roa 93.175.147.0/24 max 24 as 196615;
```

bird2
Choosing this format causes Routinator to produce a route table configuration item for BIRD 2.0 configuration.

```
route 2001:7fb:fd03::/48 max 48 as 196615;
route 2001:7fb:fd04::/48 max 48 as 196615;
route 93.175.147.0/24 max 24 as 196615;
```

rpsl

This format produces a list of RPSL (Routing Policy Specification Language) 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.

```
route: 93.175.147.0/24
origin: AS196615
descr: RPKI attestation
mnt-by: NA
created: 2021-05-07T14:28:17Z
last-modified: 2021-05-07T14:28:17Z
source: ROA-RIPE-RPKI-ROOT
```

summary

This format produces a summary of the content of the RPKI repository. It does not take filters into account and will always provide numbers for the complete repository.

For each trust anchor, it will print the number of verified ROAs and VRPs, router certificates and keys, as well as ASPAs. Note that router keys and ASPAs will only be included in the totals if you have enabled *BGPsec* and *ASPA*, respectively.

```
Summary at 2023-05-30 16:22:27.060940 UTC
afrinic:
            ROAs:    4896 verified;
            VRPs:    6248 verified,    5956 final;
    router certs:       0 verified;
     router keys:       0 verified,       0 final.
           ASPAs:       0 verified,       0 final.
apnic:
            ROAs:   25231 verified;
            VRPs:  109978 verified,  109717 final;
    router certs:       0 verified;
     router keys:       0 verified,       0 final.
           ASPAs:       2 verified,       2 final.
arin:
            ROAs:   63188 verified;
            VRPs:   78064 verified,   76941 final;
    router certs:       1 verified;
     router keys:       1 verified,       1 final.
           ASPAs:       7 verified,       7 final.
lacnic:
            ROAs:   18036 verified;
            VRPs:   32565 verified,   30929 final;
    router certs:       0 verified;
     router keys:       0 verified,       0 final.
           ASPAs:       0 verified,       0 final.
ripe:
            ROAs:   39081 verified;
            VRPs:  211048 verified,  211043 final;
    router certs:       2 verified;
     router keys:       2 verified,       2 final.
           ASPAs:      57 verified,      57 final.
total:
            ROAs:  150432 verified;
```

(continues on next page)

(continued from previous page)

```
           VRPs:  437903 verified,  434586 final;
  router certs:       3 verified;
   router keys:       3 verified,       3 final.
         ASPAs:      66 verified,      66 final.
```

Changed in version 0.11.0: Reformat, sort alphabetically and add *BGPsec* information

New in version 0.13.0: Include *ASPA*

CHAPTER

SIX

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 (Simplified Local Internet Number Resource Management with the RPKI) standard specified in **RFC 8416**.

You can use this example file as a starting point, but you can also build your own exceptions file based on existing VRPs in the global RPKI data set using the *SLURM* output format in combination with the `--select-asn` and `--select-prefix` options.

See also:

- *Running Interactively*

For example, this command will create a SLURM file that always authorises all announcements that are currently done from AS196615:

```
routinator vrps --format slurm --select-asn 196615
```

The output will look like this:

```
{
  "slurmVersion": 1,
  "validationOutputFilters": {
    "prefixFilters": [ ],
    "bgpsecFilters": [ ]
  },
  "locallyAddedAssertions": {
    "prefixAssertions": [
      {
        "asn": 196615,
        "prefix": "93.175.147.0/24",
        "maxPrefixLength": 24,
        "comment": "ripe"
      },
      {
        "asn": 196615,
        "prefix": "2001:7fb:fd03::/48",
        "maxPrefixLength": 48,
        "comment": "ripe"
      },
```

(continues on next page)

(continued from previous page)

```
      {
        "asn": 196615,
        "prefix": "2001:7fb:fd04::/48",
        "maxPrefixLength": 48,
        "comment": "ripe"
      }
    ],
    "bgpsecAssertions": [

    ]
  }
}
```

Use the *--exceptions* option to refer to your file with local exceptions. Routinator verifies that the JSON itself is valid, as well as the specified values. The exceptions file will be re-read on every validation run, so you can simply update the file whenever your exceptions change.

In the metrics Routinator provides, there are counters indicating how many VRPs are added and excluded from the final data set as a result of your exceptions.

6.1 Limiting Prefix Length

It's possible to set the maximum length of IPv4 and IPv6 prefixes that will be included in the VRP data set. You can set this with the *--limit-v4-len* and *--limit-v6-len* options, respectively.

To illustrate this option we'll use an extreme example:

```
routinator --limit-v4-len=8 --limit-v6-len=19 vrps
```

Now, all VRPs will be ignored that have a prefix with a length that is longer than /8 IPv4 and /19 IPv6:

```
ASN,IP Prefix,Max Length,Trust Anchor
AS6253,48.0.0.0/8,24,arin
AS31399,53.0.0.0/8,8,ripe
AS7922,73.0.0.0/8,8,arin
AS3320,2003::/19,19,ripe
AS5511,2a01:c000::/19,48,ripe
```

Note that only the prefix length itself and not the maximum prefix length value of the ROA is considered.

New in version 0.12.0.

CHAPTER

SEVEN

LOGGING

To let you analyse the validated ROA payload (VRP) data set as well as its overall health, Routinator logs an extensive amount of information. The log levels used by syslog are utilised 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 debugging.

7.1 Interactive Mode

When running *interactively* logging information will be printed to standard error by default. You can redirect logging to syslog using the `--syslog` option, or to a file with the `--logfile` option. You can influence the amount of information returned with these options:

`-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 twice 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*. Specifying `-q` twice turns logging off completely.

7.2 Detached Server Mode

When running Routinator detached in *server mode* logging to syslog is implied. Using the `--syslog-facility` option you can specify the syslog facility to use, which is *daemon* by default. You also redirect logging output to a file using the `--logfile` option.

Tip: Though almost all settings are available as command line options, you would likely want to configure logging options in the *configuration file*.

When you run the HTTP service logging information is also available at the `/log` path. This will produce 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.

CHAPTER

EIGHT

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 and RPKI-to-Router (RTR) servers.

If you have installed Routinator through our software package repository, the HTTP and RTR servers are enabled by default via the pre-installed configuration file. However, they are only available on localhost for security reasons. You will have to explicitly change these options to make the services available to other network devices.

If you have built Routinator using Cargo, no servers are enabled by default at all. From the command line you can start Routinator as a daemon using the `server` subcommand. Use the `--http` command line option or the *http-listen* configuration file option to start the HTTP server. To enable the RTR server, use the `--rtr` command line option or the *rtr-listen* option in the configuration file. Of course you also start both.

HTTPS and secure transports for RTR are supported as well. Please read the *HTTP Service* and *RTR Service* sections for details.

Note: Both servers will only start serving data once the first validation run has completed. Routinator will not reread the trust anchor locators after it has started the service. Thus, if you add or change a TAL you must restart Routinator or send it a *SIGUSR1*.

Using 192.0.2.13 as an example IPv4 address, enter the following command to start Routinator with the HTTP server listening on port 8323 and the RTR server on port 3323:

```
routinator server --http 192.0.2.13:8323 --rtr 192.0.2.13:3323
```

Make sure IPv6 addresses are in square brackets, e.g.:

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

By default Routinator will stay attached to your terminal and log to standard error. You can provide the `--detach` option to run it in the background instead, in which case logging information is written to syslog. To learn more about what kind of information is returned and how to influence what is logged and where, refer to the *Logging* section.

Attention: On Linux systems there is an overlap between IPv4 and IPv6. You can't bind to all interfaces on both address families, i.e. `0.0.0.0` and `[::]`, as it will result in a *'address already in use'* error. Instead, to listen to both IPv4 and IPv6 you can simply enter:

```
routinator server --rtr [::]:3323
```

CHAPTER

NINE

RTR SERVICE

Routinator has a built-in server for the RPKI-to-Router (RTR) protocol, which can be started with the `--rtr` command line option or the *rtr-listen* option in the configuration file.

Routinator supports RTR version 1 described in **RFC 8210**, as well as the older version from **RFC 6810**. After the first validation run has completed, routers with support for route origin validation (ROV) can connect to Routinator to fetch the processed data.

Tip: If you would like to run the RTR server as a separate daemon, for example because you want to centralise validation and distribute processed data to various locations where routers can connect, then NLnet Labs provides RTRTR.

In the examples throughout the documentation we use port 3323 for RTR connections, but please note that this is not the IANA (Internet Assigned Numbers Authority)-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.

9.1 Secure Transports

Although there is no mandatory-to-implement transport that provides authentication and integrity protection, **RFC 6810#section-7** defines a number of secure transports for RPKI-RTR that can be used to secure communications, including TLS, SSH, TCP MD5 and TCP-AO.

Routinator has native support for TLS connections, and can be configured to use *SSH Transport* with some additional tooling.

9.1.1 TLS Transport

It's possible to natively use RTR-over-TLS connections with Routinator. There is an IANA-assigned default port for *rpki-rtr-tls* as well, in this case 324.

Currently, very few routers have implemented support for TLS, but it may be especially useful to use secure connections when deploying our RTR data proxy RTRTR, as data may be flowing across the public Internet.

In this example we'll start Routinator's RTR server listening on the IP addresses 192.0.2.13 and 2001:0DB8::13 and use port 3324 to make sure it's not a privileged port.

First, indicate that you want a TLS connection with the `--rtr-tls` option. Then use the `--rtr-tls-cert` option to specify the path to a file containing the server certificates to be used. This file has to contain one or more certificates encoded in PEM format. Lastly, use the `--rtr-tls-key` option to specify the path to a file containing the private key to be used for RTR-over-TLS connections. The file has to contain exactly one private key encoded in PEM format:

```
routinator server --rtr-tls 192.0.2.13:3324 \
                  --rtr-tls [2001:0DB8::13]:3324 \
                  --rtr-tls-cert "/path/to/rtr-tls.crt" \
                  --rtr-tls-key "/path/to/rtr-tls.key"
```

If you want to securely connect to Routinator with RTRTR using the RTR-TLS Unit, a certificate that is trusted by the usual set of web trust anchors will work with no additional configuration. In case you generated a self-signed certificate for Routinator, make sure to copy the certificate to your machine running RTRTR and refer to the path of the file in your unit using the `cacerts` configuration option.

New in version 0.11.0.

9.1.2 SSH Transport

These instructions were contributed by Wild Kat.

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

# Only allow the rpki user to execute this one command
Match User rpki
    ForceCommand /bin/nc localhost 3323
    PasswordAuthentication yes
Match all
```

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 <password>
  transport ssh port 22
```

9.2 Configuring Routers

Route Origin Validation is supported on most hardware and software routers. This documentation does not provide authoritative information on how to configure each router platform, but aims to provide helpful pointers.

9.2.1 Hardware Routers

- Arista EOS
- Cisco IOS and IOS XE
- Cisco IOS-XR
- Extreme Networks SLX-OS
- Huawei VRP
- Juniper Junos
- Nokia SR OS
- MikroTik

9.2.2 Software Routers

- BIRD
- FRRouting
- GoBGP
- OpenBGPD
- VyOS

See also:

Rejecting RPKI Invalid BGP Routes in the NLNOG BGP Filter Guide.

Note: For additions or corrections, please open an issue or submit a pull request.

CHAPTER

TEN

HTTP SERVICE

Routinator has a built-in HTTP server, which can be started with the `--http` command line option or the *http-listen* option in the configuration file. Routinator natively supports *TLS Transport* and the endpoints are set up in such a way that it's easy to configure a *reverse proxy* as well.

In addition to the various *VRP output formats*, Routinator's HTTP server also provides a *user interface*, an *API*, *monitoring* and *logging* endpoints.

After fetching and verifying all RPKI data for the first time, paths are available for each *VRP output format*. For example, at the `/json` path you can fetch a list of all VRPs in JSON format.

```
curl http://192.0.2.13:8323/json
```

10.1 Query Parameters

All paths accept selector expressions to limit the VRPs returned in the form of a query parameter. You can use `select-asn` to select ASNs and `select-prefix` to select prefixes. These expressions can be repeated multiple times. The output for each additional parameter will be added to the results.

For example, to only show the VRPs in JSON format authorising AS196615, use:

```
curl http://192.0.2.13:8323/json?select-asn=196615
```

This will produce the following output:

```
{
   "metadata": {
     "generated": 1626853335,
     "generatedTime": "2021-07-21T07:42:15Z"
  },
   "roas": [
      { "asn": "AS196615", "prefix": "2001:7fb:fd03::/48", "maxLength": 48, "ta": "ripe"␣
↪},
      { "asn": "AS196615", "prefix": "2001:7fb:fd04::/48", "maxLength": 48, "ta": "ripe"␣
↪},
      { "asn": "AS196615", "prefix": "93.175.147.0/24", "maxLength": 24, "ta": "ripe" }
   ]
 }
```

The query parameter `exclude` can be used to exclude certain payload types from the response. The values `routeOrigins`, `routerKeys`, and `aspas` disable inclusion of route origins, router keys, and ASPAs, respectively. The values can either be given in separate `exclude` parameters or included in one separated by commas.

New in version 0.13.0: Allow excluding specific data from the output

10.1.1 More Specific Prefixes

When you query for a prefix, by default Routinator will return the exact match, as well as less specifics. The reason is that a VRP of an overlapping less specific prefix can also affect the RPKI validity of a BGP announcement, depending on the *Maximum Prefix Length (MaxLength)* that is set.

In some cases you may want more specifics to be displayed as well. For this the `more-specifics` query string can be used. For example, when querying for 82.221.32.0/20:

```
curl http://192.0.2.13:8323/json?select-prefix=82.221.32.0/20
```

Routinator will return the exact match and the VRP for the less specific /17 prefix:


```
{
   "metadata": {
      "generated": 1644266267,
      "generatedTime": "2022-02-07T20:37:47Z"
   },
   "roas": [
      { "asn": "AS30818", "prefix": "82.221.32.0/20", "maxLength": 20, "ta": "ripe" },
      { "asn": "AS44515", "prefix": "82.221.0.0/17", "maxLength": 17, "ta": "ripe" }
   ]
}
```


When including the `more-specifics` parameter in the same query:

```
curl http://192.0.2.13:8323/json?select-prefix=82.221.32.0/20&include=more-specifics
```

You will now see that a more specific /23 prefix is returned as well:


```
{
   "metadata": {
      "generated": 1644266267,
      "generatedTime": "2022-02-07T20:37:47Z"
   },
   "roas": [
      { "asn": "AS44515", "prefix": "82.221.46.0/23", "maxLength": 23, "ta": "ripe" },
      { "asn": "AS30818", "prefix": "82.221.32.0/20", "maxLength": 20, "ta": "ripe" },
      { "asn": "AS44515", "prefix": "82.221.0.0/17", "maxLength": 17, "ta": "ripe" }
   ]
}
```

Tip: The `more-specifics` parameter will also work if there is no exactly matching or less specific prefix. In that case you will get a list of all more specific VRPs covered by the prefix you supplied in the query.

Changed in version 0.11.0: `more-specifics` query parameter

10.2 TLS Transport

Routinator offers native TLS support for both HTTP and *RTR connections*. In this example we'll start Routinator's HTTPS server listening on the IP addresses 192.0.2.13 and 2001:0DB8::13 and use port 8324.

First, indicate that you want a TLS connection with the `--http-tls` option. Then use the `--http-tls-cert` option to specify the path to a file containing the server certificates to be used. This file has to contain one or more certificates encoded in PEM format. Lastly, use the `--http-tls-key` option to specify the path to a file containing the private key to be used for HTTPS connections. The file has to contain exactly one private key encoded in PEM format:

```
routinator server --http-tls 192.0.2.13:8324 \
                  --http-tls [2001:0DB8::13]:8324 \
                  --http-tls-cert "/path/to/http-tls.crt" \
                  --http-tls-key "/path/to/http-tls.key"
```

New in version 0.11.0.

10.3 Using a Reverse Proxy

Though TLS is natively supported, it may be more convenient to set up a reverse proxy to serve HTTPS data. This way you'll be using a production grade web server that for example allows automation of certificate renewal.

For convenience, all the files and folders for the *user interface* are hosted under the `/ui` path and the *API endpoints* are under `/api`. For example, this allows you to just expose the UI and not any of the other paths, such as those serving the various *VRP output formats*.

In this example we'll use NGINX, but other web servers will allow a similar, simple configuration. To only expose the user interface, this is what your configuration needs at a minimum when running it on the same server as Routinator runs on, using port 8323.

Using the = modifier, the first entry only forwards if the path is *exactly* `/` so that paths not explicitly mentioned, such as `/json`, are not forwarded. For more information, please refer to the NGINX documentation.

```
location = / {
  proxy_pass http://127.0.0.1:8323/;
}
location /ui {
  proxy_pass http://127.0.0.1:8323/ui;
}
location /api {
  proxy_pass http://127.0.0.1:8323/api;
}
```

CHAPTER

ELEVEN

USER INTERFACE

Routinator's HTTP service offers a web based user interface on the `/ui` path. In addition to displaying detailed statistics from the last validation run Routinator has performed, as well as HTTP and RTR connection metrics, the most prominent functionality is the Prefix Check.

By default, you only need to provide an IP address or prefix. When clicking *Validate*, Routinator will look up from which Autonomous System the closest matching prefix is announced in BGP and perform RPKI validation. Alternatively, you can manually provide an ASN.

The returned RPKI validity state will be *Valid*, *Invalid* or *NotFound* and is based on the current set of Validated ROA Payloads (VRPs) in the cache. Routinator will provide an overview of all VRPs that led to the result, along with the reason for the outcome.

Routinator doesn't just retrieve the ASN for a specific prefix, but it also fetches related information. In addition to validating the longest matching prefix (or exact match if this is what you selected), details can be provided on less specific and more specific announcements seen in BGP, as well as other resources allocated to the same organisation.

Routinator does not perform the BGP and allocation lookups itself, but relies on the open-source roto-api service, developed and hosted by NLnet Labs at bgp-api.net. The service uses these data sources:

- BGP information based on RISWhois data, which is part of the RIPE NCC's Routing Information Service (RIS). This data set is currently updated every 8 hours.
- Resource allocations retrieved from statistics hosted by the five Regional Internet Registries. These are updated daily.

New in version 0.8.3.

Changed in version 0.10.0: The Prefix Check

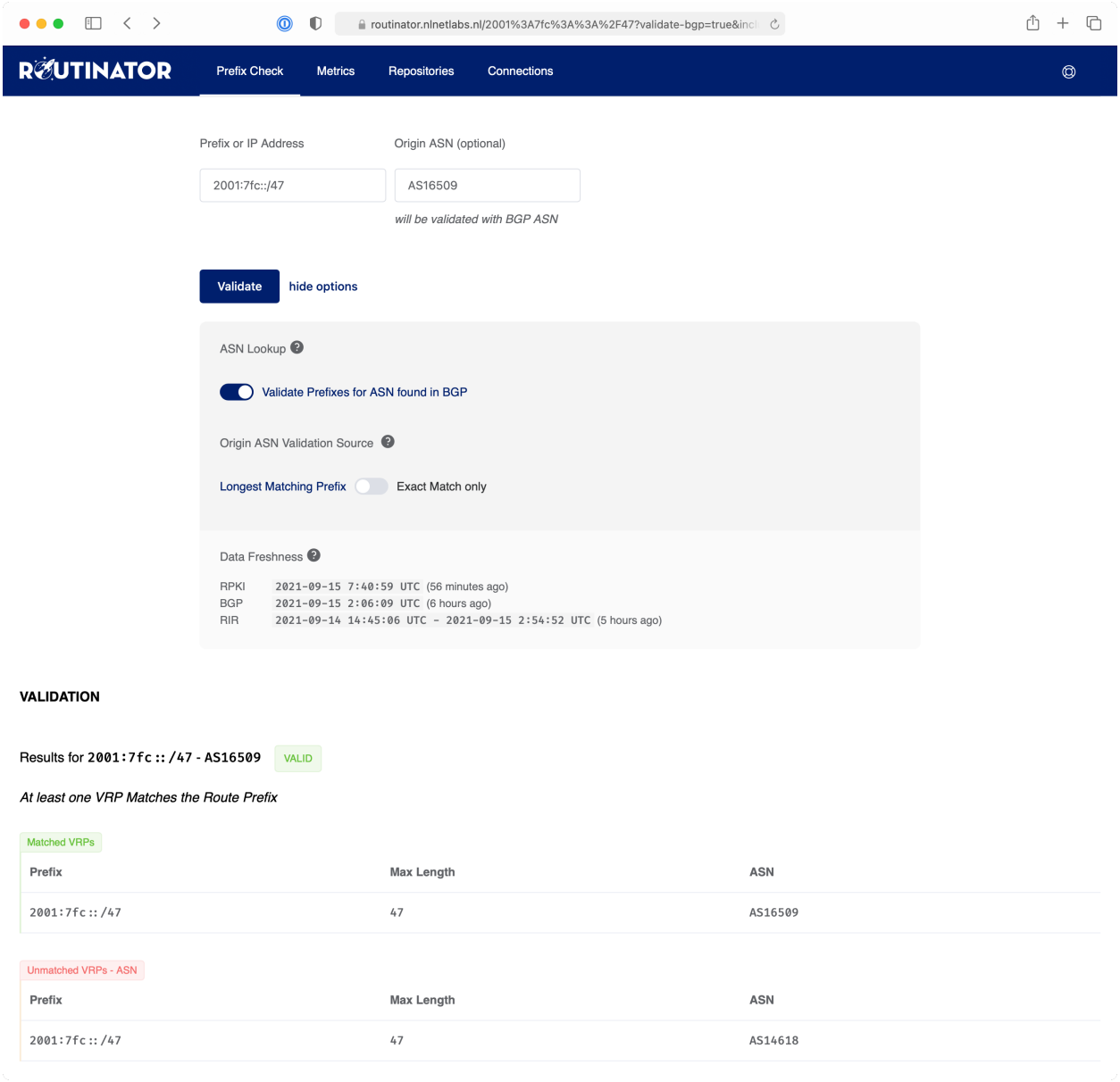

Fig. 1: The Routinator Prefix Check

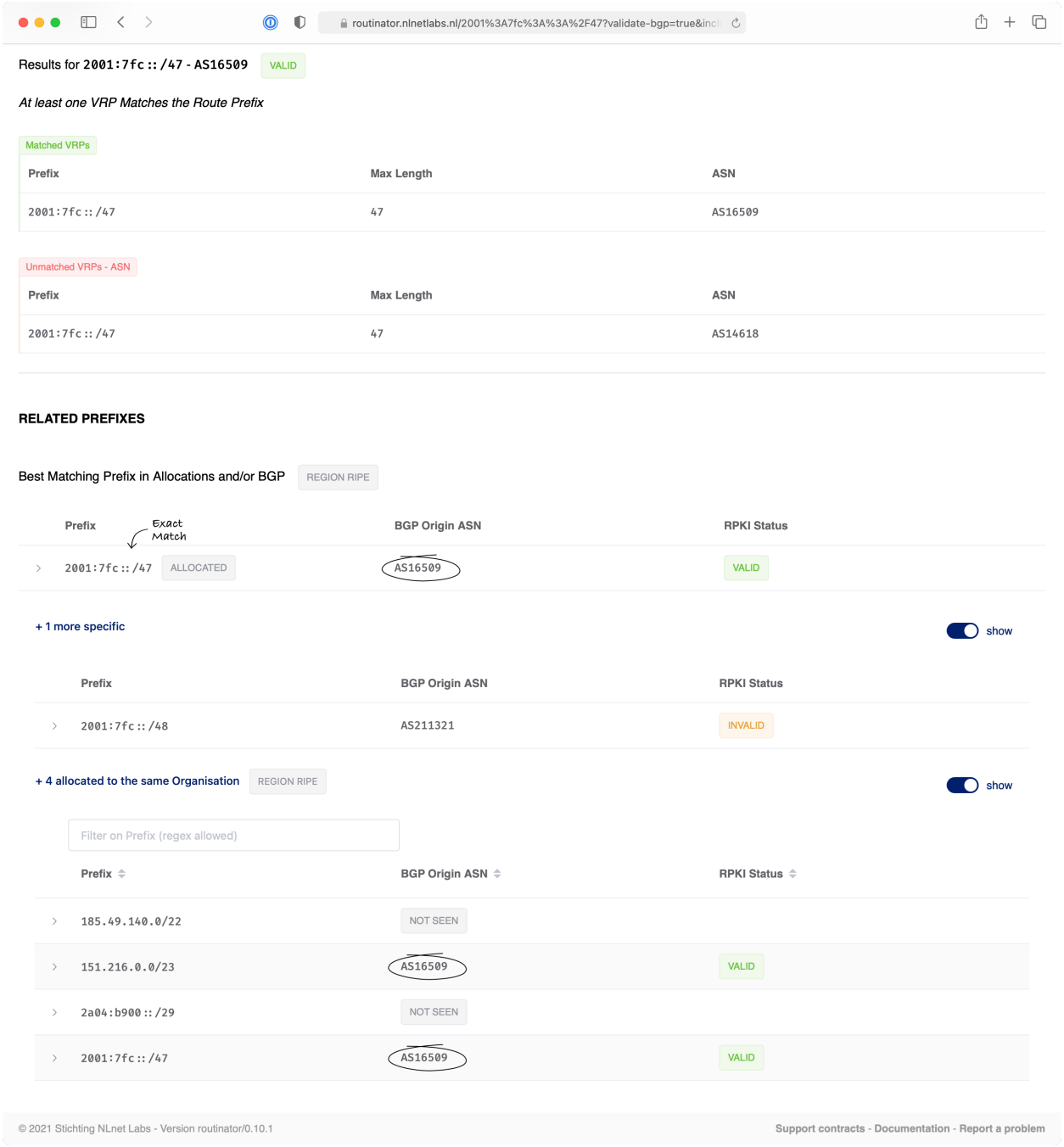

Fig. 2: Prefixes related to your query

CHAPTER

TWELVE

API ENDPOINTS

The HTTP service supports GET requests on the following paths:

`/api/v1/status`
: Returns exhaustive information in JSON format on all trust anchors, repositories, RRDP and rsync connections, as well as RTR and HTTP sessions. This data set provides the source for the Routinator user interface.

`/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. For details about its contents see *validity checker*.

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

`/json-delta, /json-delta?session=session?serial=serial`
: Returns a JSON object with the changes since the dataset version identified by the *session* and *serial* query parameters. If a delta cannot be produced from that version, the full data set is returned and the member *reset* in the object will be set to *true*. In either case, the members *session* and *serial* identify the version of the data set returned and their values should be passed as the query parameters in a future request.

  The members *announced* and *withdrawn* contain arrays with route origins that have been announced and withdrawn, respectively, since the provided session and serial. If *reset* is *true*, the *withdrawn* member is not present.

`/json-delta/notify, /json-delta/notify?session=session&serial=serial`
: Returns a JSON object with two members *session* and *serial* which contain the session ID and serial number of the current data set.

  If the *session* and *serial* query parameters are provided, and the session ID and serial number of the current data set are identical to the provided values, the request will not return until a new data set is available. This can be used as a means to get notified when the data set has been updated.

In addition, the `/log` endpoint returns *logging* information and the `/metrics`, `/status` and `/version` endpoints provide *monitoring* data.

New in version 0.9.0: The `/json-delta` path

Changed in version 0.9.0: The `/api/v1/status` path

New in version 0.13.0: The `/json-delta/notify` path

CHAPTER

THIRTEEN

MONITORING

The HTTP server in Routinator provides endpoints for monitoring the application on the following paths:

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

`/metrics`
: Exposes exhaustive time series data specifically for Prometheus, containing metrics on all trust anchors, repositories, RRDP and rsync connections, as well as RTR and HTTP sessions. If desired, dedicated port 9556 is allocated for the exporter.

`/api/v1/status`
: Returns exhaustive information in JSON format on all trust anchors, repositories, RRDP and rsync connections, as well as RTR and HTTP sessions. This data set provides the source for the Routinator user interface.

`/status`
: Returns a subset of the metrics information in a concise plain text format

13.1 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

- For each cryptographic object that can appear in the RPKI, the number of valid, invalid and stale items per trust anchor and repository
- The number of validated ROA payloads (VRPs) per Trust Anchor and repository
- The number of VRPs added and excluded locally

RTR server

- The current RTR serial number
- The current number of RTR connections
- The total amount of bytes sent and received over the RTR connection
- Metrics for each RTR client is available if the `--rtr-client-metrics` option is provided

HTTP server

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

Refer to the Reference section for a complete overview for all metrics in the *JSON format* and the *Prometheus format*.

13.2 Grafana

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

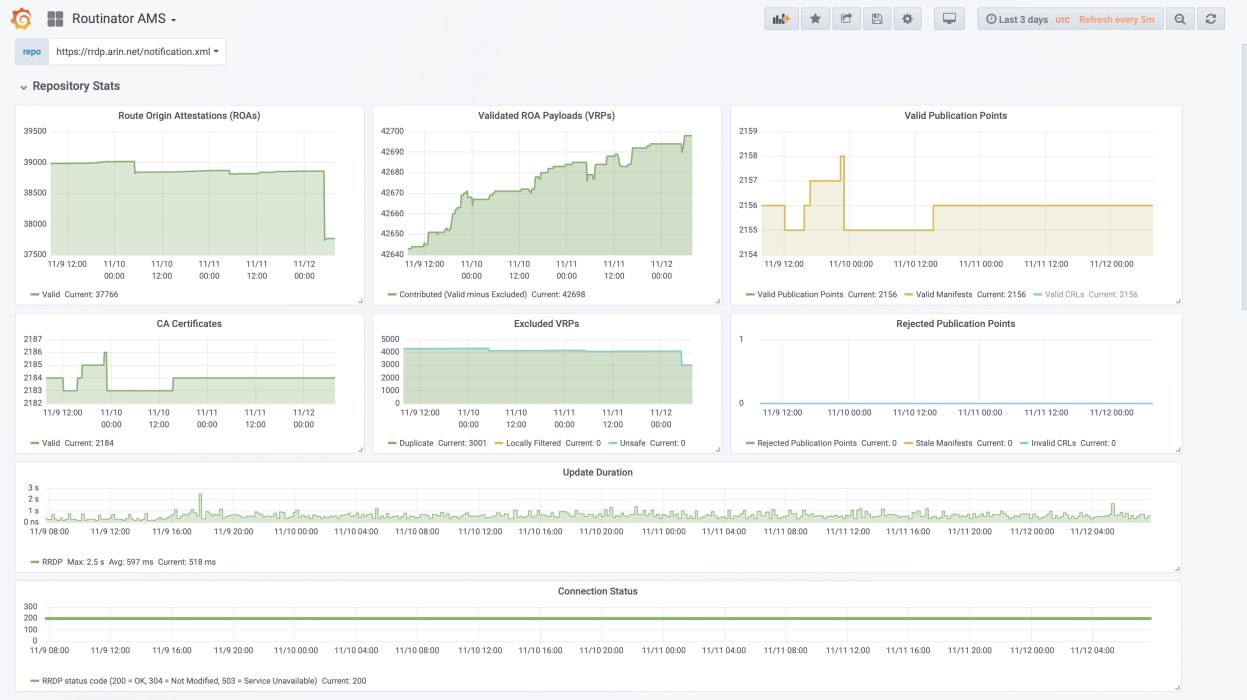

Fig. 1: Time series for each RPKI Repository

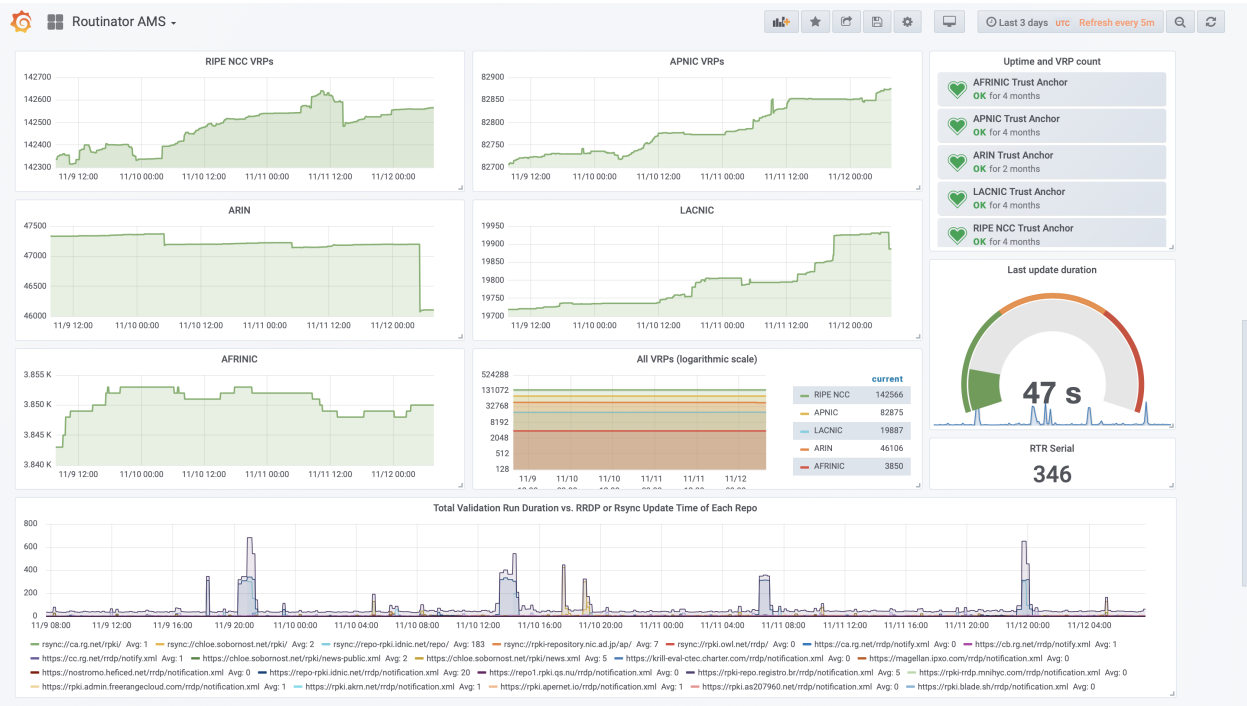

Fig. 2: Time series for each Trust Anchor

CHAPTER

FOURTEEN

RUNNING INTERACTIVELY

Routinator can perform RPKI validation as a one-time operation and print a validated ROA payload (VRP) list in various formats using the *vrps* subcommand and specifying the *desired format*.

Warning: If you have installed Routinator through the NLnet Labs software package repository, the installation script will set up the application to run as a service. You should not run Routinator as a daemon and interactively at the same time on the same machine.

For example, to print the VRPs in CSV format to standard output, run:

```
routinator vrps --format csv
```

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

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

During the validation process, logging information will be printed to standard error. You can influence the amount of details returned with the *--verbose* and *--quiet* options. To learn more about what kind of information returned, refer to the *Logging* section.

If you have enabled *BGPsec* and/or *ASPA* validation, in some output formats the amount of data can be quite overwhelming. You can exclude specific data types for the output with the *--no-route-origins*, *--no-router-keys* and the `--noaspas` options.

Changed in version 0.13.0: Allow excluding specific data from the output.

14.1 Query Options

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

When using *--select-asn*, you can use both `AS64511` and `64511` as the notation. With *--select-prefix*, the result will include VRPs regardless of their ASN and MaxLength. Both selector flags can be combined and used multiple times in a single query. The output for each additional selector will be added to the results.

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* option.

Here is an example selecting VRPs related to a specific ASN, produced in *json* format:

```
routinator vrps --format json --select-asn 196615
```

This results in:


```
{
  "metadata": {
    "generated": 1626853335,
    "generatedTime": "2021-07-21T07:42:15Z"
  },
  "roas": [
    { "asn": "AS196615", "prefix": "2001:7fb:fd03::/48", "maxLength": 48, "ta": "ripe" },
    { "asn": "AS196615", "prefix": "2001:7fb:fd04::/48", "maxLength": 48, "ta": "ripe" },
    { "asn": "AS196615", "prefix": "93.175.147.0/24", "maxLength": 24, "ta": "ripe" }
  ]
}
```


14.1.1 More Specific Prefixes

When you query for a prefix, by default Routinator will return the exact match, as well as less specifics. The reason is that a VRP of an overlapping less specific prefix can also affect the RPKI validity of a BGP announcement, depending on the *Maximum Prefix Length (MaxLength)* that is set.

In some cases you may want more specifics to be displayed as well. For this the `--more-specifics` option can be used. For example, when querying for 82.221.32.0/20:

```
routinator vrps --format json --select-asn 82.221.32.0/20
```

Routinator will return the exact match and the VRP for the less specific /17 prefix:


```
{
   "metadata": {
      "generated": 1644266267,
      "generatedTime": "2022-02-07T20:37:47Z"
   },
   "roas": [
      { "asn": "AS30818", "prefix": "82.221.32.0/20", "maxLength": 20, "ta": "ripe" },
      { "asn": "AS44515", "prefix": "82.221.0.0/17", "maxLength": 17, "ta": "ripe" }
   ]
}
```


When including the `--more-specifics` option in the same query:

```
routinator vrps --format json --select-asn 82.221.32.0/20 --more-specifics
```

You will now see that a more specific /23 prefix is returned as well:


```
{
   "metadata": {
      "generated": 1644266267,
      "generatedTime": "2022-02-07T20:37:47Z"
   },
   "roas": [
      { "asn": "AS44515", "prefix": "82.221.46.0/23", "maxLength": 23, "ta": "ripe" },
      { "asn": "AS30818", "prefix": "82.221.32.0/20", "maxLength": 20, "ta": "ripe" },
      { "asn": "AS44515", "prefix": "82.221.0.0/17", "maxLength": 17, "ta": "ripe" }
```


(continues on next page)

(continued from previous page)

```
    ]
}
```

Tip: The *--more-specifics* option will also work if there is no exactly matching or less specific prefix. In that case you will get a list of all more specific VRPs covered by the prefix you supplied in the query.

14.1.2 Exclude Specific Data Types

If you have enabled *BGPsec* and/or *ASPA* validation, in some output formats the amount of data can be quite overwhelming. You can exclude specific payload types with the *--no-route-origins*, *--no-router-keys* and `--noaspas` options to disable inclusion of route origins, router keys, and ASPAs, respectively.

Deprecated since version 0.9.0: `--filter-asn` and `--filter-prefix`

Changed in version 0.11.0: Add the *--more-specifics* option

New in version 0.13.0: Allow excluding specific data from the output

CHAPTER

FIFTEEN

VALIDITY CHECKER

You can check the RPKI origin validation status of one or more BGP announcements 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* option:

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

This will simply return the RPKI validity state:

```
Invalid
```

You can also add the *--json* option:

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

This will produce a detailed analysis of the reasoning behind the validation outcome printed in JSON format. In case of an *Invalid* state, the `reason` indicates whether this is because the announcement is originated by an unauthorised AS (`"reason": "as"`), or if the length of the announced prefix is more specific than the authorised prefix or, if present, the maximum prefix length allows (`"reason": "length"`). Lastly, a complete list of VRPs that caused the result is included:

```
{
  "validated_routes": [
    {
      "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"
            }
```

(continues on next page)

(continued from previous page)

```
          ],
          "unmatched_length": [
          ]
        }
      }
    }
  ],
  "generatedTime": "2021-07-21T11:36:44Z"
}
```

If you run the HTTP service in daemon mode, validation information is also available via the *user interface* and at the `/validity` API endpoint.

15.1 Reading Input From a File

Routinator can also read input to validate from a file using the *--input* option. If the file is given as a single dash, input is read from standard input. You can also save the results to a file using the *--output* option.

You can provide a simple plain text file with the routes you would like to have verified by Routinator. The input file should have one route announcement per line, provided as a prefix followed by an ASCII-art arrow => surrounded by white space and followed by the AS Number of the originating Autonomous System.

For example, let's provide Routinator with this file, saved as `beacons.txt`:

```
93.175.147.0/24 => 12654
2001:7fb:fd02::/48 => 12654
```

Now we refer to the file with the *--input* option:

```
routinator validate --input beacons.txt
```

Routinator provides the RPKI validity state in the output:

```
93.175.147.0/24 => AS12654: invalid
2001:7fb:fd02::/48 => AS12654: valid
```

With the *--json* option you can provide a file in JSON format. It should consist of a single object with one member *routes* which contains an array of objects. Each object describes one route announcement through its *prefix* and *asn* members which contain a prefix and originating AS number as strings, respectively.

For example, let's provide Routinator with this `beacons.json` file:

```
{
  "routes": [{
      "asn": "AS12654",
      "prefix": "93.175.147.0/24"
    },
    {
      "asn": "AS12654",
      "prefix": "2001:7fb:fd02::/48"
    }
  ]
}
```

Then refer to the file with the *--json* and *--input* options:

```
routinator validate --json --input beacons.json
```

Routinator produces a JSON object that includes the validity state and a detailed analysis of the reasoning behind the outcome of each route:


```
{
  "validated_routes": [
    {
      "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": [
          ]
        }
      }
    },
    {
      "route": {
        "origin_asn": "AS12654",
        "prefix": "2001:7fb:fd02::/48"
      },
      "validity": {
        "state": "valid",
        "description": "At least one VRP Matches the Route Prefix",
        "VRPs": {
          "matched": [
            {
              "asn": "AS12654",
              "prefix": "2001:7fb:fd02::/48",
              "max_length": "48"
            }
          ],
          "unmatched_as": [
          ],
          "unmatched_length": [
          ]
```


(continues on next page)

(continued from previous page)

```
        }
      }
    }
  ]
}
```

New in version 0.9.0.

CHAPTER

SIXTEEN

DUMPING STORED DATA

The *dump* subcommand writes the contents of all stored data to the file system. This is primarily intended for debugging but can be used to get access to the view of the RPKI data that Routinator currently sees. This subcommand has only one option, *--output*, which specifies the directory where the output should be written.

Three directories will be created in the output directory:

rrdp
: This directory contains all the files collected via RRDP from the various repositories. Each repository is stored in its own directory. The mapping between `rpkiNotify` URI and path is provided in the `repositories.json` file. For each repository, the files are stored in a directory structure based on the components of the file as rsync URI.

rsync
: This directory contains all the files collected via rsync. The files are stored in a directory structure based on the components of the file's rsync URI.

store
: This directory contains all the files used for validation. Files collected via RRDP or rsync are copied to the store if they are correctly referenced by a valid manifest. This part contains one directory for each RRDP repository similarly structured to the `rrdp` directory and one additional directory `rsync` that contains files collected via rsync.

ta
: This directory contains the trust anchor certificates. Files are stored in a directory structure two levels deep. The first level is the schema portion of the certificate's URI, i.e., `https` or `rsync`, and the second level is the authority portion of the URI, e.g., `tal.apnic.net`. Within this second level, the certificate is stored in a file that has the hexadecimal encoding of the SHA-256 hash of the certificate's URI as the file name with the extension `.cer` appended.

New in version 0.9.0.

Changed in version 0.11.1: Stored trust anchor certificates are dumped into the `ta` directory.

CHAPTER

SEVENTEEN

MANUAL PAGE

17.1 Synopsis

routinator [options] *vrps* [vrps-options] [-o *output-file*] [-f *format*]

routinator [options] *validate* [validate-options] [-a *asn*] [-p *prefix*]

routinator [options] *server* [server-options]

routinator [options] *update* [update-options]

routinator *man* [-o *file*]

routinator -h

routinator -V

17.2 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 run in one-shot mode outputting a list of validated ROA payloads in various formats, as a server for the RPKI-to-Router (RTR) protocol that many routers 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.

17.3 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.

-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.

If omitted, defaults to `$HOME/.rpki-cache/repository`.

--no-rir-tals

If present, Routinator will not use the bundled trust anchor locators (TALs) of the five Regional Internet Registries (RIRs).

Trust anchor locators are the starting points for collecting and validating RPKI data. Each of the five RIRs provides a TAL that adds resources from their area. For normal production installations, these are the only TALs that should be used.

Using this option as well as the *--tal* and *--extra-tals-dir* options you can change which TALs Routinator should use.

--tal=name

Use the bundled TAL with the given name in addition to any other TAL.

Each RIR TAL is available through this option as well as TALs for a few select test environments. If you use this option with the name *list*, Routinator will print a list of all available bundled TALS and exit.

The option can be given more than once.

--extra-tals-dir=dir

Specifies a directory containing additional trust anchor locators (TALs) to use. Routinator will use all files in this directory with an extension of *.tal* as TALs. These files need to be in the format described by **RFC 8630**.

Note that Routinator will use all TALs provided. That means that if a TAL in this directory is one of the bundled TALs, then these resources will be validated twice.

-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 version 0.7.0 the default for this option was *warn*. In all previous versions *warn* was hard-wired.

--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 VRPs. 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. This is the default policy.

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.

--limit-v4-len=length, **--limit-v6-len**=length

If present, defines the maximum length of IPv4 prefixes or IPv6 prefixes, respectively, that will be included in the VRP data set. All VRPs for prefixes with a longer prefix length will be ignored. Note that only the prefix length itself, not the max length is considered.

If either option is missing, VRPs for all prefixes of that particular address family are included.

--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.

--fresh

Delete and re-initialize the local data storage before starting. This option should be used when Routinator fails after reporting corrupt data storage.

--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. Set the option to 0 to disable the timeout.

--disable-rrdp

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

--rrdp-fallback=policy

Defines the circumstance when access via rsync should be tried for a CA that announces it can be updated via RRDP. In general, access via RRDP is less resource intensive and more secure than rsync and will therefore be preferred. This option specifies what to do when access to an RRDP repository fails.

The policy `never` means that rsync is never tried for a CA that announces RRDP.

The policy `stale` means that rsync is tried if an update via RRDP fails and there is no current local copy of the RRDP repository. A local copy is considered current if it was last updated within a time span chosen on a per-repository basis between the *--refresh* time and *--rrdp-fallback-time*.

The policy `new` means that rsync is tried if an update via RRDP fails and there is no local copy of the RRDP repository at all. In other words, an update via RRDP has never succeeded for the repository. Choosing this policy allows a repository operator some leeway when first enabling RRDP support.

The default policy if this option is not given is `stale`.

--rrdp-fallback-time=seconds

Sets the maximum time in seconds since a last successful update of an RRDP repository before Routinator falls back to using rsync. The default is 3600 seconds. If the given value is smaller than twice the refresh time, it is silently increased to that value.

The actual time is chosen at random between the refresh time and this value in order to spread out load on the rsync server.

--rrdp-max-delta-count=count

If the number of deltas necessary to update an RRDP repository is larger than the value provided by this option, the snapshot is used instead. If the option is missing, the default of 100 is 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 300 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-tcp-keepalive=seconds

Sets the value of the TCP keepalive duration in seconds for RRDP connections. The default if this option is omitted is 60 seconds. Set the option to 0 to disable the use of TCP keepalives.

--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.

--rrdp-keep-responses=path

If this option is enabled, the bodies of all HTTPS responses received from RRDP servers will be stored under *path*. The sub-path will be constructed using the components of the requested URI. For the responses to the notification files, the timestamp is appended to the path to make it possible to distinguish the series of requests made over time.

--max-object-size=BYTES

Limits the size of individual objects received via either rsync or RRDP to the given number of bytes. The default value if this option is not present is 20,000,000 (i.e., 20 MBytes). Use a value of 0 to disable the limit.

--max-ca-depth=count

The maximum number of CAs a given CA may be away from a trust anchor certificate before it is rejected. The default value is 32.

--enable-bgpsec

If this option is present, BGPsec router keys will be processed during validation and included in the produced data set.

--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*.

See *LOGGING* below for more information on what information is logged at the different levels.

-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.

17.4 Commands

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

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 Payloads 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 autonomous system number, the prefix in slash notation, the maximum prefix length, 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 can be overwritten with the *tal-labels* config file option.

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 up to four members: *roas* contains the validated route origin authorizations, *routerKeys* contains the validated BGPsec router keys, *aspas* contains the validated ASPA payload, and *metadata* contains some information about the validation run itself. Of the first three, only those members are present that have not been disabled or excluded.

The *roas* member 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*.

The *routerKeys* member contains an array of objects with four elements each: The autonomous system using the router key is given in *asn*, the key identifier as a string of hexadecimal digits in *SKI*, the

actual public key as a Base 64 encoded string in *routerPublicKey*, and the trust anchor from which the authorization was derived in *ta*.

The *aspa* member contains an array of objects with four members each: The *customer* member contains the customer ASN, *afi* the address family as either "ipv4" or "ipv6", *providers* contains the provider ASN set as an array, and the trust anchor from which the authorization was derived in *ta*.

The output object also includes a member named *metadata* which provides additional information. Currently, this is a member *generated* which provides the time the list was generated as a Unix timestamp, and a member *generatedTime* which provides the same time but in the standard ISO date format.

If only route origins are included, 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.

jsonext

The list is placed into a JSON object with up to four members: *roas* contains the validated route origin authorizations, *routerKeys* contains the validated BGPsec router keys, *aspas* contains the validated ASPA payload, and *metadata* contains some information about the validation run itself. Of the first three, only those members are present that have not been disabled or excluded.

The *roas* member 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 extended information about the source of the authorization in *source*.

The *routerKeys* member contains an array of objects with four elements each: The autonomous system using the router key is given in *asn*, the key identifier as a string of hexadecimal digits in *SKI*, the actual public key as a Base 64 encoded string in *routerPublicKey*, and extended information about the source of the key is contained in *source*.

The *aspa* member contains an array of objects with four members each: The *customer* member contains the customer ASN, *afi* the address family as either "ipv4" or "ipv6", *providers* contains the provider ASN set as an array, and information about the source of the data can be found in *source*.

This source information the same for route origins and router keys. It consists of an array. Each item in that array is an object providing details of a source. The object will have a *type* of *roa* if it was derived from a valid ROA object, *cer* if it was derived from a published router certificate, or *exception* if it was an assertion in a local exception file.

For RPKI objects, *tal* provides the name of the trust anchor locator the object was published under, *uri* provides the rsync URI of the ROA or router certificate, *validity* provides the validity of the ROA itself, and *chainValidity* the validity considering the validity of the certificates along the validation chain.

For assertions from local exceptions, *path* will provide the path of the local exceptions file and, optionally, *comment* will provide the comment if given for the assertion.

The output object also includes a member named *metadata* which provides additional information. Currently, this is a member *generated* which provides the time the list was generated as a Unix timestamp, and a member *generatedTime* which provides the same time but in the standard ISO date format.

Please note that because of this additional information, output in `jsonext` format will be quite large.

slurm

The list is formatted as locally added assertions of a local exceptions file defined by RFC 8416 (also known as SLURM). The produced file will have empty validation output filters.

openbgpd

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

bird1
Choosing this format causes Routinator to produce a *roa table* configuration item for the BIRD1 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, complete the operation but provide exit status 2. If this option is not given, the operation will complete with exit status 0 in this case.

`-a asn, --select-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, --select-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 selections. Then all matching VRPs are included. That is, selectors combine as "or" not "and".

`-m, --more-specifics`

Include VRPs with prefixes that are more specifics of those given by the *-p* option. Without this option, only VRPs with prefixes equal or less specific are included.

Note that VRPs with more specific prefixes have no influence on whether a route is RPKI valid or invalid and therefore these VRPs are of an informational nature only.

`--no-route-origins, --no-router-keys, --no-aspas`

These three options can be used to exclude the various payload types from being included in the output.

`validate`

This command can be used to perform RPKI route origin validation for one or more route announcements. Routinator will determine whether the provided announcements are RPKI valid, invalid, or not found.

A single route announcement can be given directly on the command line:

-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 VRPs that caused the particular result. If this option is omitted, Routinator will only print the determined state.

Alternatively, a list of route announcements can be read from a file or standard input.

-i file, **--input**=file

If present, input is read from the given file. If the file is given is a single dash, input is read from standard output.

-j, **--json**

If this option is provided, the input is assumed to be JSON format. It should consist of a single object with one member *routes* which contains an array of objects. Each object describes one route announcement through its *prefix* and *asn* members which contain a prefix and originating AS Number as strings, respectively.

If the option is not provided, the input is assumed to consist of simple plain text with one route announcement per line, provided as a prefix followed by an ASCII-art arrow => surrounded by white space and followed by the AS Number of originating autonomous system.

The following additional options are available independently of the input method.

-o file, **--output**=file

Output is written to the provided file. If the option is omitted or *file* is given as a single dash, output is written to standard output.

-n, **--noupdate**

The repository will not be updated before performing validation.

--complete

If any of the rsync commands needed to update the repository failed, complete the operation but provide exit status 2. If this option is not given, the operation will complete with exit status 0 in this case.

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 Trust Anchor Locators 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.

--rtr-tls=addr:port

Specifies a local address and port to listen for incoming TLS-encrypted RTR connections.

The private key and server certificate given via the *--rtr-tls-key* and *--rtr-tls-cert* or their equivalent config file options will be used for connections.

The option can be given multiple times, but the same key and certificate will be used for all connections.

--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.

--http-tls=addr:port

Specifies a local address and port to listen of for incoming TLS-encrypted HTTP connections.

The private key and server certificate given via the *--http-tls-key* and *--http-tls-cert* or their equivalent config file options will be used for connections.

The option can be given multiple times, but the same key and certificate will be used for all connections.

--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.

--rtr-tcp-keepalive=seconds

The number of seconds to wait before sending a TCP keepalive on an established RTR connection. By default, TCP keepalive is enabled on all RTR connections with an idle time of 60 seconds. Set this option to 0 to disable keepalives.

On some systems, notably OpenBSD, this option only enables TCP keepalives if set to any value other than 0. You will have to use the system's own mechanisms to change the idle times.

--rtr-client-metrics

If provided, the server metrics will include separate metrics for every RTR client. Clients are identified by their RTR source IP address. This is disabled by default to avoid accidentally leaking information about the local network topology.

--rtr-tls-key

Specifies the path to a file containing the private key to be used for RTR-over-TLS connections. The file has to contain exactly one private key encoded in PEM format.

--rtr-tls-cert

Specifies the path to a file containing the server certificates to be used for RTR-over-TLS connections. The file has to contain one or more certificates encoded in PEM format.

--http-tls-key

Specifies the path to a file containing the private key to be used for HTTP-over-TLS connections. The file has to contain exactly one private key encoded in PEM format.

--http-tls-cert

Specifies the path to a file containing the server certificates to be used for HTTP-over-TLS connections. The file has to contain one or more certificates encoded in PEM format.

--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 server mode. It this option is provided, Routinator will run as that user after the listening sockets for HTTP and RTR have been created. This may cause problems, if the user is not allowed to write to the directory given as repository directory or is not allowed to read the TAL directory or local exception files.

--group=group-name

The name of the group to change to for server mode. It this option is provided, Routinator will run as that group after the listening sockets for HTTP and RTR have been created.

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. If this option is not given, the operation will complete with exit status 0 in this case.

dump

Writes the content of all stored data to the file system. This is primarily intended for debugging but can be used to get access to the view of the RPKI data that Routinator currently sees.

-o dir, **--output**=dir

Write the output to the given directory. If the option is omitted, the current directory is used.

Three directories will be created in the output directory:

The *rrdp* directory will contain all the files collected via RRDP from the various repositories. Each repository is stored in its own directory. The mapping between rpkiNotify URI and path is provided in the *repositories.json* file. For each repository, the files are stored in a directory structure based on the components of the file as rsync URI.

The *rsync* directory contains all the files collected via rsync. The files are stored in a directory structure based on the components of the file's rsync URI.

The *store* directory contains all the files used for validation. Files collected via RRDP or rsync are copied to the store if they are correctly referenced by a valid manifest. This part contains one directory for each RRDP repository similarly structured to the *rrdp* directory and one additional directory *rsync* that contains files collected via rsync.

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.

17.5 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.

no-rir-tals
A boolean specifying whether the five RIR Trust Anchor Locators (TALs) should not be added to the set of evaluated TALs. If missing, the RIR TALs will be used.

tals
A list of strings, each containing the name of a bundled TAL to be added to the set of TALs to be evaluated.

extra-tals-dir
A string containing the path to a directory that contains additional TALs.

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. This is the default policy if the value is missing.

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 VRPs and add warning messages to the log.

warn
Warn about unsafe VRPs in the log but add them to the final set of VRPs.

accept
Quietly add unsafe VRPs to the final set of VRPs. This is the default policy if the value is missing.

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.

limit-v4-len
An integer value which, if present, limits the length of IPv4 prefixes for which VPRs are included in the data set to the given value.

limit-v6-len
An integer value which, if present, limits the length of IPv6 prefixes for which VPRs are included in the data set to the given value.

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. Set the value to 0 to turn the timeout off.

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

rrdp-fallback
A string value specifying the circumstances under which an update via rsync is tried if an update via RRDP fails. See *--rrdp-fallback* for details on the available policies.

rrdp-fallback-time
An integer value specifying the maximum number of seconds since a last successful update of an RRDP repository before Routinator falls back to using rsync. The default in case the value is missing is 3600 seconds. If the value provided is smaller than twice the refresh time, it is silently increased to that value.

rrdp-max-delta-count
An integer value that specifies the maximum number of deltas necessary to update an RRDP repository before using the snapshot instead. If the value is missing, the default of 100 is used.

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 300 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-tcp-keepalive
An integer value that provides the duration in seconds for the TCP keepalive option on RRDP connections. If the value is missing, a duration of 60 seconds is used. Set the value to 0 to disable the use of TCP keepalive for RRDP connections.

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.

rrdp-keep-responses
A string containing a path to a directory into which the bodies of all HTTPS responses received from RRDP servers will be stored. The sub-path will be constructed using the components of the requested URI. For the responses to the notification files, the timestamp is appended to the path to make it possible to distinguish the series of requests made over time.

max-object-size
An integer value that provides a limit for the size of individual objects received via either rsync or RRDP to the given number of bytes. The default value if this option is not present is 20,000,000 (i.e., 20 MBytes). A value of 0 disables the limit.

max-ca-depth
An integer value that specifies the maximum number of CAs a given CA may be away from a trust anchor certificate before it is rejected. If the option is missing, a default of 32 will be used.

enable-bgpsec
A boolean value specifying whether BGPsec router keys should be included in the published dataset. If false or missing, no router keys will be included.

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*.

See *LOGGING* below for more information on what information is logged at the different levels.

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 an address and port on which the RTR server should listen in TCP mode. Address and port should be separated by a colon. IPv6 address should be enclosed in square brackets.

rtr-tls-listen
An array of string values each providing an address and port on which the RTR server should listen in TLS 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 an address and port on which the HTTP server should listene. Address and port should be separated by a colon. IPv6 address should be enclosed in square brackets.

http-tls-listen
An array of string values each providing an address and port on which the HTTP server should listen in TLS mode. 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.

rtr-tcp-keepalive
An integer value specifying the number of seconds to wait before sending a TCP keepalive on an established RTR connection. If this option is missing, TCP keepalive will be enabled on all RTR connections with an idle time of 60 seconds. If this option is present and set to zero, TCP keepalives are disabled.

On some systems, notably OpenBSD, this option only enables TCP keepalives if set to any value other than 0. You will have to use the system's own mechanisms to change the idle times.

rtr-client-metrics
A boolean value specifying whether server metrics should include separate metrics for every RTR client. If the value is missing, no RTR client metrics will be provided.

rtr-tls-key
A string value providing the path to a file containing the private key to be used by the RTR server in TLS mode. The file must contain one private key in PEM format.

rtr-tls-cert
A string value providing the path to a file containing the server certificates to be used by the RTR server in TLS mode. The file must contain one or more certificates in PEM format.

http-tls-key
A string value providing the path to a file containing the private key to be used by the HTTP server in TLS mode. The file must contain one private key in PEM format.

http-tls-cert
A string value providing the path to a file containing the server certificates to be used by the HTTP server in TLS mode. The file must contain one or more certificates in PEM format.

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-labels
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.

17.6 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.

/api/v1/status
Returns the current status in JSON 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.

/json-delta, /json-delta?session=session&serial=serial
Returns a JSON object with the changes since the dataset version identified by the *session* and *serial* query parameters. If a delta cannot be produced from that version, the full data set is returned and the member *reset* in the object will be set to *true*. In either case, the members *session* and *serial* identify the version of the data set returned and their values should be passed as the query parameters in a future request.

The members *announced* and *withdrawn* contain arrays with route origins that have been announced and withdrawn, respectively, since the provided session and serial. If *reset* is *true*, the *withdrawn* member is not present.

/json-delta/notify, /json-delta/notify?session=session&serial=serial
: Returns a JSON object with two members *session* and *serial* which contain the session ID and serial number of the current data set.

  If the *session* and *serial* query parameters are provided, and the session ID and serial number of the current data set are identical to the provided values, the request will not return until a new data set is available. This can be used as a means to get notified when the data set has been updated.

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

In addition, the query parameter `include=more-specifics` will cause the inclusion of VRPs for more specific prefixes of prefixes given via `select-prefix`.

Finally, the query parameter `exclude` can be used to exclude certain payload types from the response. The values `routeOrigins`, `routerKeys`, and `aspas` disable inclusion of route origins, router keys, and ASPAs, respectively. The values can either be given in separate `exclude` parameters or included in one separated by commas.

These parameters work in the same way as the options of the same name to the *vrps* command.

17.7 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.

17.8 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 certification 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), manifests (.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 VRPs. This allows to assess the operation impact of such a filter. Depending on this assessment, the default may change in future versions.

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.

17.9 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.

17.10 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.

SIGUSR2: Re-open log file
When receiving SIGUSR2 and logging to a file is enabled, Routinator will re-open the log file. If this fails, Routinator will exit.

17.11 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.

CHAPTER

EIGHTEEN

JSON METRICS

Routinator's *monitoring service* provides comprehensive metrics in JSON format `/api/v1/status` endpoint. Here you can find an overview of all metrics and their meaning.

The JSON metrics consists of an object with the following members:

`version`
: The version of Routinator.

`serial`
: The current serial number for data served to *RTR* clients.

`now`
: The date and time in UTC when this report was created.

`lastUpdateStart`
: The date and time in UTC when the last validation run started.

`lastUpdateDone`
: The date and time in UTC when the last validation run completed.

`lastUpdateDuration`
: The duration of the last validation run in seconds.

`tals`
: Metrics for each configured trust anchor. In most cases these will be the five Regional Internet Registries, but will include the trust anchors of any configured testbeds as well.

  Each element of this object contains a *publication metrics value* as described below.

`repositories`
: Metrics for each repository encountered during validation. Note that the data given here relates to the repository content used during validation. If the repository failed to update, then these numbers are from the stored old data.

  Each element of this object contains a *publication metrics value* as described below. In addition, there is a member `type` that describes whether the repository is an RRDP or rsync repository.

`vrpsAddedLocally`
: The number of *VRPs* added to the final data set from *local exceptions*.

`rsync`
: Metrics for updates via rsync.

  This is an object with one element for each repository that was updated via rsync during the last validation run. Each element contains an *rsync update metrics value* as described below.

`rrdp`
: Metrics for updates via RRDP.

This is an object with one element for each repository that was updated via rsync during the last validation run. Each element contains an *RRDP update metrics value* as described below.

rtr
Metrics for the built-in RTR server. See *RTR metrics* below.

http
Metrics for the built-in HTTP server. See *HTTP metrics* below.

18.1 Publication Metrics

Publication metrics are provided both for all trust anchors and for each RPKI repository. They contain the following information:

vrpsTotal
The total number of *VRPs* found to be present and valid.

vrpsUnsafe
The number of *VRPs* that are considered *unsafe*. Depending on configuration, these may be included in the final set or dropped from it.

vrpsLocallyFiltered
The number of *VRPs* that are filtered as the result of a *local exception*.

vrpsDuplicate
The number of duplicate *VRPs* resulting from ROAs containing the same authorisation.

Note that if a VRP appears in multiple trust anchors or repositories, which occurrence is considered the duplicate depends on the order of processing which may change between validation runs. Thus, this number may change unexpectedly.

vrpsFinal
The number of *VRPs* that are contributed by this trust anchor or repository to the final set provided to your routers. This is the total number of VRPs, minus the ones that are locally filtered, duplicate, and, if configured to be dropped, unsafe.

validPublicationPoints
The number of valid *publication points*.

rejectedPublicationPoints
The number of rejected *publication points*.

A publication point is rejected if its manifest is invalid or if any objects listed on the manifest are missing or have a different content hash.

validManifests
The number of valid *manifests*.

invalidManifests
The number of invalid *manifests*.

A manifest is invalid if it is not correctly encoded, has expired or is not correctly signed by the issuing CA.

staleManifests
The number of *stale manifests*.

A manifest is stale if the current time is past the time an update to the manifest should have been issued. Whether a stale manifest is valid or invalid depends on configuration. By default it is considered invalid.

missingManifests
The number of missing *manifests*.

`validCRLs`
: The number of valid *certificate revocation lists*.

`invalidCRLs`
: The number of invalid *certificate revocation lists*.

  A CRL is invalid if it is not correctly encoded or is not correctly signed by the issuing CA.

`staleCRLs`
: The number of *stale certificate revocation lists*.

  A CRL is stale if the current time is past the time an update should have been issued. Whether a stale CRL is valid or invalid depends on configuration. By default it is considered invalid.

`strayCRLs`
: The number of stray *certificate revocation lists*.

  Each CA should only issue one CRL. This CRL should both be listed on the manifest and used by the manifest's certificate itself. Any manifest listed on the manifest that is not also the manifest's own CRL is considered a stray.

`validCACerts`
: The number of Certificate Authority (CA) certificates found to be present and valid.

`validEECerts`
: The number of End Entity (EE) certificates found to be present and valid.

  This only refers to such certificates included as stand-alone files which are BGPsec router certificates.

`invalidCerts`
: The number of invalid stand-alone certificates, either CA or EE certificates.

`validROAs`
: The number of valid *Route Origin Attestations*

`invalidROAs`
: The number of invalid *Route Origin Attestations*.

`validGBRs`
: The number of valid *Ghostbusters Records*.

  Note that currently the content of a Ghostbuster Record is not checked.

`InvalidGBRs`
: The number of invalid *Ghostbusters Records*.

`otherObjects`
: The number of objects found that are not certificates (.cer), Certificate Revocation Lists (.crl), manifests (.mft), ROAs (.roa), or Ghostbuster Records (.gbr).

18.2 Rsync Update Metrics

For each repository updated via rsync the following values are given.

`status`
: The status code returned by the rsync process. A value of 0 means the process has finished successfully. The meaning of other values depends on the rsync client used. Please refer to its documentation for further details.

`duration`
: The duration the rsync process was running in seconds.

18.3 RRDP Update Metrics

For each repository updated via RRDP the following values are given.

`status`
: The overall status of the update. This will be 200 if the updated succeeded, 304 if no update was necessary because the data was already current, and any other value for a failed update. If the value is -1, it was not possible to reach the HTTPS server at all.

`notifyStatus`
: The status of retrieving the notification file. This is the first step of an RRDP update. A value of 200 indicates that the file was successfully retrieved. A value of 304 indicates that the file hasn't changed since last update and no actual update is necessary. Any other value represents an error.

`payloadStatus`
: The status of retrieving the actual payload. This is the second step of an RRDP update and may either represent a single HTTPS request for the snapshot file or a series of HTTPS request for the sequence of delta files necessary to update from the last known state.

  A value of 0 means that no payload retrieval was necessary. A value of 200 means that the update was successful. Any other value indicates an error. In case of a sequence of delta updates, this error may have been preceded by one or more successful requests.

`duration`
: The overall duration of the RRDP update in seconds.

`serial`
: The serial number stated by the RRDP server for the current data set. With each update the serial number is increased by one.

`session`
: The identifier of the current session of the RRDP server. Serial numbers are only valid within the same session. If the server needs to restart its sequence for whatever reason, it needs to choose a new session ID and all data will have to be updated through a snapshot.

`delta`
: Whether data was updated via a sequence of deltas (`true`) or a full snapshot had to be retrieved (`false`).

`snapshotReason`
: If this is not `null`, it provides a reason why a snapshot was used instead of a delta as a short explanatory string.

18.4 RTR Server Metrics

A number of metrics are provided describing the state of the included RTR server. These metrics are available whether the RTR server is actually enabled or not.

`currentConnections`
: The number of currently open RTR connections.

`bytesRead`
: The total number of bytes read from RTR connections. In other words, describes how much data has been sent by clients.

`bytesWritten`
: The total number of bytes written to RTR connections. In other words, describes how much data has been sent to clients.

If `rtr-client-metrics` are enabled via configuration or command line, an additional object `clients` will appear that list the IP addresses of clients seen by the RTR server providing the following information for them.

`connections`
: The number of currently open connections from that address. The number should normally be 0 or 1 but can be higher if the address is the public side of a NAT.

`updated`
: Contains the time of the last successful update by the client.

`lastReset`
: Contains the time of the last successful cache reset by the client.

`resetQueries`
: Contains the number of reset queries by the client.

`serialQueries`
: Contains the number of serial queries by the client.

`serial`
: The highest serial of the data provided to a client from that address. This can be used to determine when the client has last updated.

`read` and `written`
: Bytes read from and written to clients from that address.

18.5 HTTP Server Metrics

A number of metrics are provided describing the state of the included HTTP server.

`totalConnections`
: The total number of connections made with the HTTP server.

`currentConnections`
: The number of currently open connections. This should at least be 1 as there is a connection open when requesting the JSON metrics.

`requests`
: The total number of requests received and answered by the HTTP server.

`bytesRead` and `bytesWritten`
: The number of bytes read from and written to HTTP clients.

CHAPTER

NINETEEN

PROMETHEUS METRICS

Routinator's *monitoring service* provides comprehensive metrics in Prometheus format at the `/metrics` endpoint. Here you can find an overview of all metrics and their meaning.

routinator_last_update_start
: Seconds since the start of the last update.

routinator_last_update_duration
: Duration of the last update in seconds.

routinator_last_update_done
: Seconds since the end of the last update.

routinator_serial
: The current serial number for data served to *RTR* clients.

19.1 Publication Metrics

Publication metrics are provided for all trust anchors and for each RPKI repository.

All metrics for trust anchors have a label `name`, named after the Trust Anchor Locator file name without the *.tal* extension, e.g. *arin*. All metrics for repositories have a label `uri` specifying the URI of the notification file of the RRDP repository, or the base URI of the rsync repository.

routinator_{ta,repository}_publication_points_total
: The number of *publication points* per trust anchor. In most cases these will be the five Regional Internet Registries, but will include the trust anchors of any configured testbeds as well.

  This metric has two labels: either `name` or `uri`, followed by the `state` which is *valid* or *rejected*.

routinator_{ta,repository}_objects_total
: Metrics for each configured trust anchor. In most cases these will be the five Regional Internet Registries, but will include the trust anchors of any configured testbeds as well.

  This metric has three labels: either `name` or `uri`, followed by `type` for the type of object, e.g. *crl*, and lastly `state` describing its validity state, such as *valid* or *stale*.

  The types and states of objects can be:

  - `manifest` - The number of *manifests* for each of the states *valid*, *invalid*, *stale* and *missing*. A manifest is *invalid* if it is not correctly encoded, has expired or is not correctly signed by the issuing CA. It is considered *stale* if the current time is past the time an update to the manifest should have been issued. Whether a *stale* manifest is *valid* or *invalid* depends on configuration. By default a *stale* manifest is considered *invalid*.

- `crl` - The number of *certificate revocation lists* for each of the states *valid*, *invalid*, *stale* and *stray*. A CRL is *invalid* if it is not correctly encoded or is not correctly signed by the issuing CA. It is considered *stale* if the current time is past the time an update to the manifest should have been issued. Whether a *stale* manifest is *valid* or *invalid* depends on configuration. By default a *stale* CRL is considered *invalid*. Lastly, each CA should only issue one CRL. This CRL should both be listed on the manifest and used by the manifest's certificate itself. Any manifest listed on the manifest that is not also the manifest's own CRL is considered a *stray*.
- `ca_cert` - The number of Certificate Authority (CA) certificates with the state *valid*.
- `router_cert` - The number of router certificates found to be present and *valid*. This only refers to such certificates included as stand-alone files which are BGPsec router certificates.
- `roa` - The number of *Route Origin Attestations* for each of the states *valid* and *invalid*.
- `gbr` - The number of *Ghostbusters Records* for each of the states *valid* and *invalid*. Note that currently the content of a Ghostbuster Record is not checked.
- `other` - The number of objects found that are not certificates (.cer), Certificate Revocation Lists (.crl), manifests (.mft), ROAs (.roa), or Ghostbuster Records (.gbr) and have the state *invalid*.

The following metrics all have just one label, either `name` in case of a trust anchor or `uri` for repositories:

routinator_{ta,repository}_valid_vrps_total
The number of *VRPs* found to be present and valid.

routinator_{ta,repository}_unsafe_vrps_total
The number of *VRPs* found to be *unsafe*.

routinator_{ta,repository}_locally_filtered_vrps_total
The number of *VRPs* that are filtered as the result of a *local exception*.

routinator_{ta,repository}_duplicate_vrps_total
The number of duplicate *VRPs* resulting from ROAs containing the same authorisation.

Note that if a VRP appears in multiple trust anchors or repositories, which occurrence is considered the duplicate depends on the order of processing which may change between validation runs. Thus, this number may change unexpectedly.

routinator_{ta,repository}_contributed_vrps_total
The number of *VRPs* that are contributed by this trust anchor or repository to the final set provided to your routers. This is the total number of VRPs, minus the ones that are locally filtered, duplicate, and, if configured to be dropped, unsafe.

19.2 Rsync Update Metrics

For each repository updated via rsync the following values are given.

routinator_rsync_status
The status code returned by the rsync process. A value of 0 means the process has finished successfully. The meaning of other values depends on the rsync client used. Please refer to its documentation for further details.

routinator_rsync_duration
The duration the rsync process was running in seconds.

19.3 RRDP Update Metrics

For each repository updated via RRDP the following values are given. All metrics have a label `uri` specifying the URI of the notification file of the RRDP repository.

routinator_rrdp_status
: The overall status of the update. This will be 200 if the updated succeeded, 304 if no update was necessary because the data was already current, and any other value for a failed update. If the value is -1, it was not possible to reach the HTTPS server at all.

routinator_rrdp_notification_status
: The status of retrieving the notification file. This is the first step of an RRDP update. A value of 200 indicates that the file was successfully retrieved. A value of 304 indicates that the file hasn't changed since last update and no actual update is necessary. Any other value represents an error.

routinator_rrdp_payload_status
: The status of retrieving the actual payload. This is the second step of an RRDP update and may either represent a single HTTPS request for the snapshot file or a series of HTTPS request for the sequence of delta files necessary to update from the last known state.

  A value of 0 means that no payload retrieval was necessary. A value of 200 means that the update was successful. Any other value indicates an error. In case of a sequence of delta updates, this error may have been preceded by one or more successful requests.

routinator_rrdp_duration
: The overall duration of the RRDP update in seconds.

routinator_rrdp_serial_info
: The serial number stated by the RRDP server for the current data set. With each update the serial number is increased by one.

19.4 RTR Server Metrics

A number of metrics are provided describing the state of the included RTR server. These metrics are available whether the RTR server is actually enabled or not.

routinator_rtr_current_connections
: The number of currently open RTR connections.

routinator_rtr_bytes_read
: The total number of bytes read from RTR connections. In other words, describes how much data has been sent by clients.

routinator_rtr_bytes_written
: The total number of bytes written to RTR connections. In other words, describes how much data has been sent to clients.

routinator_rtr_client_last_reset_seconds
: The amount of seconds since last cache reset by a client address.

routinator_rtr_client_reset_queries
: The number of reset queries by a client address.

routinator_rtr_client_serial_queries
: The number of of serial queries by a client address.

New in version 0.12.0: `routinator_rtr_client_last_reset_seconds`, `routinator_rtr_client_reset_queries` and `routinator_rtr_client_serial_queries`

19.5 HTTP Server Metrics

A number of metrics are provided describing the state of the included HTTP server.

`routinator_http_connections`
: The total number of connections made with the HTTP server.

`routinator_http_current_connections`
: The number of currently open connections. This should at least be 1 as there is a connection open when requesting the JSON metrics.

`requests`
: The total number of requests received and answered by the HTTP server.

`routinator_http_bytes_read` and `routinator_http_bytes_written`
: The number of bytes read from and written to HTTP clients.

CHAPTER

TWENTY

UNSAFE VRPS

Routinator is unique among relying party software in its ability to alert operators to a subtle condition we refer to as *"Unsafe VRPs"*. We will explore that concept here.

If the address prefix of a Validated ROA Payload (VRP) overlaps with any resources assigned to a Certification Authority (CA) that has been rejected because it failed to validate completely, the VRP is said to be *unsafe* since using it may lead to legitimate routes being flagged as RPKI Invalid.

In the Hosted RPKI systems that the five Regional Internet Registries offer, all certificates and ROAs reside within a single system and all related objects are published in a single repository. In addition, these systems do not allow sub-delegation of resources. As a result, relying party software will normally fetch and validate the entire set of objects, or in case of an outage nothing at all. This makes the occurrence of unsafe VRPs highly unlikely.

When an organisation runs RPKI with their own CA, they can delegate a subset of their resources to another party, such as their customer, who in turn runs their own CA. Both parties can publish in a repository they host themselves, or one that is offered by a third party as a service. Because there are now more variables at play, such as broken CAs or unavailable repositories, there is a possibility of unsafe VRPs emerging.

Unsafe VRPs typically occur when the organisation that holds the superset of resources publishes a ROA for their aggregate prefix, e.g. 2001:db8::/32-32, and the customer publishes a ROA to authorise a more specific, e.g. 2001:db8:abcd:/48-48. Now, when the customer CA is unavailable for any reason and validation fails, the VRP for 2001:db8:abcd:/48 will be marked as *unsafe*. Note that the reason for the unavailability can be that the CA itself is broken, or because the repository that hosts the ROA is unavailable for a prolonged period.

Routinator has an `--unsafe-vrps` option that specifies how to deal with unsafe resources when conditions creating unsafe VRPs exist. Currently, the default policy is to *accept* unsafe VRPs. This means VRPs will not be analysed for being unsafe at all, nor will any metrics be generated. The other options are *warn*, which will report any unsafe VRP that was encountered and *reject*, filtering out VRPs that are marked as unsafe. For the latter two options metrics are made available.

CHAPTER

TWENTYONE

ADVANCED FEATURES

Routinator offers several advanced features to let operators gain operational experience with some of the ongoing work in the Internet Engineering Task Force to improve and augment RPKI.

Note: The Hosted RPKI systems that the RIRs offer currently only support the creation of ROAs. To manage ASPA, BGPsec or other RPKI objects, you can run Delegated RPKI with Krill.

21.1 ASPA

Autonomous System Provider Authorisation (ASPA), currently described in two Internet drafts in the IETF, applies the concepts of authenticated origins we know from ROAs to the propagation of routes. An ASPA is a digitally signed object through which the holder of an Autonomous System (AS) can authorise one or more other ASes as its upstream providers. When validated, an ASPA's content can be used for detection and mitigation of route leaks.

Note: ASPA support is temporarily behind a feature flag while the draft is under discussion in the IETF. This way operators can gain operational experience without unintended side effects. See *Enabling or Disabling Features* for more information.

You can let Routinator process ASPA objects and include them in the published dataset, as well as the metrics, using the `--enable-aspa` option or by setting `enable-aspa` to True in the *configuration file*. ASPA information will be exposed via RTR, as well as in the *json* and *jsonext* output formats, e.g.:


```
{
    "metadata": {
        "generated": 1681829067,
        "generatedTime": "2023-04-18T14:44:27Z"
    },
    "roas": [{
        "asn": "AS196615",
        "prefix": "93.175.147.0/24",
        "maxLength": 24,
        "source": [{
            "type": "roa",
            "uri": "rsync://rpki.ripe.net/repository/DEFAULT/73/fe2d72-c2dd-46c1-9429-
↪e66369649411/1/49sMtcwyAuAW2lVDSQBGhOHd9og.roa",
            "validity": {
                "notBefore": "2023-01-01T08:44:47Z",
```


(continues on next page)

(continued from previous page)


```
                "notAfter": "2024-07-01T00:00:00Z"
            },
            "chainValidity": {
                "notBefore": "2023-04-18T14:32:13Z",
                "notAfter": "2024-07-01T00:00:00Z"
            }
        }]
    }],
    "routerKeys": [],
    "aspas": [{
      "customer": "AS64496",
      "afi": "ipv6",
      "providers": ["AS64499", "AS64511", "AS65551"],
      "source": [{
          "type": "aspa",
          "uri": "rsync://acmecorp.example.net/0/AS64496.asa",
          "tal": "ripe",
          "validity": {
              "notBefore": "2023-04-13T07:21:24Z",
              "notAfter": "2024-04-11T07:26:24Z"
          },
          "chainValidity": {
              "notBefore": "2023-04-18T14:32:13Z",
              "notAfter": "2024-04-11T07:26:24Z"
            }
        }]
    }]
}
```


See also:

- A Profile for Autonomous System Provider Authorization
- BGP AS_PATH Verification Based on Autonomous System Provider Authorization (ASPA) Objects
- Manage ASPA objects with Krill

New in version 0.13.0.

21.2 BGPsec

The goal of BGPsec, as described in **RFC 8209**, is to provide full AS path validation. For this operators will need to publish BGPsec router keys in the RPKI. As there is currently very limited deployment, validating these objects with Routinator is not enabled by default.

You can let Routinator process router keys and include them in the published dataset, as well as the metrics, using the `--enable-bgpsec` option or by setting `enable-bgpsec` to True in the *configuration file*. BGPsec information will be exposed via RTR, as well as in the *SLURM*, *json* and *jsonext* output formats, e.g.:


```
{
    "metadata": {
        "generated": 1626853335,
        "generatedTime": "2021-07-21T07:42:15Z"
```


(continues on next page)

(continued from previous page)


```
    },
    "roas": [{
        "asn": "AS196615",
        "prefix": "93.175.147.0/24",
        "maxLength": 24,
        "source": [{
            "type": "roa",
            "uri": "rsync://rpki.ripe.net/repository/DEFAULT/73/fe2d72-c2dd-46c1-9429-
↪e66369649411/1/49sMtcwyAuAW2lVDSQBGhOHd9og.roa",
            "validity": {
                "notBefore": "2021-01-01T04:39:56Z",
                "notAfter": "2022-07-01T00:00:00Z"
            },
            "chainValidity": {
                "notBefore": "2021-05-06T12:51:30Z",
                "notAfter": "2022-07-01T00:00:00Z"
            }
        }]
    }],
    "routerKeys": [{
        "asn": "AS64496",
        "SKI": "E2F075EC50E9F2EFCED81D44491D25D42A298D89",
        "routerPublicKey": "kwEwYHKoZIzj0CAtig5-QfEKpTtFgiqfiAFQg--LAQerAH2Mpp-
↪GucoDAGBbhIqMFQYIKoZIzj0DAQcDQgAEgFcjQ_D33wNPsXxnAGb-mtZ7XQrVO9DQ6UlASh",
        "source": [{
            "type": "roa",
            "uri": "rsync://acmecorp.example.net/rpki/RIPE-NLACMECORP/R0tgdREopjYdeyeI-
↪wXUJQ4p786.cer",
            "validity": {
                "notBefore": "2021-11-09T17:04:40Z",
                "notAfter": "2022-11-09T17:04:39Z"
            },
            "chainValidity": {
                "notBefore": "2022-01-16T14:45:51Z",
                "notAfter": "2022-08-06T00:00:00Z"
            }
        }]
    }],
    "aspas": []
}
```


See also:

- BGPsec Protocol Specification
- A Profile for BGPsec Router Certificates, Certificate Revocation Lists, and Certification Requests
- Manage BGPSec Router Certificates with Krill

New in version 0.11.0.

21.3 Resource Tagged Attestations

Resource Tagged Attestations (RTAs) allow any arbitrary file to be signed 'with resources' by one or more parties. The RTA object is a separate file that cryptographically connects the document with a set of resources. The receiver of the object can use Routinator to show these resources, and verify that it was created by their rightful holder(s).

One practical example where RTA could be valuable is to authorise a Bring Your Own IP (BYOIP) process, where you bring part or all of your publicly routable IPv4 or IPv6 address range from your on-premises network to a cloud provider. The document authorising BYOIP could be signed using RTA.

RTA objects can be generated using Krill, the RPKI Certificate Authority software from NLnet Labs, and you can use the MyAPNIC hosted service. The objects can be validated using Routinator if it is built with RTA support, using the *features* functionality provided by Cargo:

```
cargo install --locked --features rta routinator
```

You can now interactively validate an RTA signed object. If it is valid, Routinator will report the resources used to sign the object:

```
routinator rta acme-corp-byoip.rta

192.0.2.0/24
203.0.113.0/24
2001:db8::/48
```

See also:

- A profile for Resource Tagged Attestations (RTAs)
- Moving RPKI Beyond Routing Security
- A proof-of-concept for constructing and validating RTAs

New in version 0.8.0.

CHAPTER

TWENTYTWO

GLOSSARY

Certificate Revocation List (CRL)
: A list of digital certificates that have been revoked by the issuing Certificate Authority (CA) before their scheduled expiration date and should no longer be trusted. Each entry in a Certificate Revocation List includes the serial number of the revoked certificate and the revocation date. The CRL file is signed by the CA to prevent tampering. The RPKI CRL profile is defined in **RFC 6487**.

Ghostbusters Record (GBR)
: An RPKI object described in **RFC 6493** that contains human contact information that may be verified (indirectly) by a CA certificate. The data in the record are those of a severely profiled vCard. Note that support for *publication* of GBR records is not widely implemented yet. As a result, Routinator will validate the object, but not produce any output for it.

Manifest
: A manifest is a signed object that contains a listing of all the signed objects in the repository publication point associated with an authority responsible for publishing in the repository. Refer to **RFC 6486** for more information.

Maximum Prefix Length (MaxLength)
: The most specific announcement of an IP prefix an Autonomous System is authorised to do according to the published *ROA*.

Publication Point
: A directory within a *repository* that contains all the objects published by a single CA.

Repository
: The RPKI repository system consists of multiple distributed and delegated repository *publication points*. Each repository publication point is associated with one or more RPKI certificates' publication points.

Resource Public Key Infrastructure (RPKI)
: RPKI proves the association between specific IP address blocks or Autonomous System Numbers (ASNs) and the holders of those Internet number resources. The certificates are proof of the resource holder's right of use of their resources and can be validated cryptographically. RPKI is based on an X.509 certificate profile defined in **RFC 3779**. Using RPKI to support secure Internet routing is described in **RFC 6480**.

Route Origin Attestation (ROA)
: A cryptographically signed object that contains a statement authorising a *single* Autonomous System Number (ASN) to originate one or more IP prefixes, along with their maximum prefix length. A ROA can only be created by the legitimate holder of the IP prefixes contained within it.

Route Origin Validation (ROV)
: A mechanism by which route advertisements can be authenticated as originating from an expected, authorised Autonomous System (AS).

RPKI Relying Parties
: Those who want to use a Public Key Infrastructure (PKI) to validate digitally signed attestations.

RPKI Repository Delta Protocol (RRDP)
: Described in **RFC 8182**, RRDP is a repository access protocol based on Update Notification, Snapshot, and Delta Files that a *Relying Party* can retrieve over the HTTPS protocol.

RPKI-to-Router (RPKI-RTR)
: The RPKI to Router protocol provides a simple but reliable mechanism for routers to receive RPKI prefix origin data from a trusted cache. It is standardised in **RFC 6810** (v0) and **RFC 8210** (v1).

Stale Object
: 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.

Trust Anchor (TA)
: Each of the five Regional Internet Registries (RIRs) publishes a trust anchor that includes all resources (a '0/0' self-signed X.509 CA certificate). They issue a child certificate containing all the resources that are held and managed by the RIR.

Trust Anchor Locator (TAL)
: The Trust Anchor Locator (TAL) is used to retrieve and verify the authenticity of a *trust anchor*. Specified in **RFC 8630**, a TAL contains one or more URIs pointing to the RIR root certificate, as well as the public key of the trust anchor in DER format, encoded in Base64. The TAL is constant so long as the trust anchor's public key and its location do not change.

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.

Validated ROA Payload (VRP)
: RPKI Relying Party software performs cryptographic verification on all published *ROAs*. If everything checks out, the software will produce one or more validated ROA payloads (VRPs) for each ROA, depending on how many IP prefixes are contained with in it. Each VRP is a tuple of an ASN, a single prefix and its maximum prefix length. If verification fails, the ROA is discarded and it'll be like no statement was ever made.

INDEX

Symbols

-V
    command line option, 75
--allow-dubious-hosts
    command line option, 73
--asn
    command line option, 78
--chroot
    command line option, 81
--complete
    command line option, 78, 79, 82
--config
    command line option, 71
--detach
    command line option, 79
--dirty
    command line option, 75
--disable-rrdp
    command line option, 74
--disable-rsync
    command line option, 73
--enable-bgpsec
    command line option, 75
--exceptions
    command line option, 72
--expire
    command line option, 81
--extra-tals-dir
    command line option, 72
--format
    command line option, 76
--fresh
    command line option, 73
--group
    command line option, 81
--help
    command line option, 75
--history
    command line option, 81
--http
    command line option, 80
--http-tls
    command line option, 80
--http-tls-cert
    command line option, 80
--http-tls-key
    command line option, 80
--input
    command line option, 79
--json
    command line option, 79
--limit-v4-len
    command line option, 73
--limit-v6-len
    command line option, 73
--listen-systemd
    command line option, 80
--logfile
    command line option, 75
--max-ca-depth
    command line option, 75
--max-object-size
    command line option, 75
--more-specifics
    command line option, 78
--no-aspas
    command line option, 78
--no-rir-tals
    command line option, 72
--no-route-origins
    command line option, 78
--no-router-keys
    command line option, 78
--noupdate
    command line option, 78, 79
--output
    command line option, 76, 79, 82
--pid-file
    command line option, 81
--prefix
    command line option, 79
--quiet
    command line option, 75
--refresh

command line option, 81
--repository-dir
command line option, 71
--retry
command line option, 81
--rrdp-connect-timeout
command line option, 74
--rrdp-fallback
command line option, 74
--rrdp-fallback-time
command line option, 74
--rrdp-keep-responses
command line option, 75
--rrdp-local-addr
command line option, 74
--rrdp-max-delta-count
command line option, 74
--rrdp-proxy
command line option, 74
--rrdp-root-cert
command line option, 74
--rrdp-tcp-keepalive
command line option, 74
--rrdp-timeout
command line option, 74
--rsync-command
command line option, 73
--rsync-timeout
command line option, 73
--rtr
command line option, 79
--rtr-client-metrics
command line option, 80
--rtr-tcp-keepalive
command line option, 80
--rtr-tls
command line option, 80
--rtr-tls-cert
command line option, 80
--rtr-tls-key
command line option, 80
--select-asn
command line option, 78
--select-prefix
command line option, 78
--stale
command line option, 72
--strict
command line option, 72
--syslog
command line option, 75
--syslog-facility
command line option, 75
--tal
command line option, 72
--unknown-objects
command line option, 73
--unsafe-vrps
command line option, 72
--user
command line option, 81
--validation-threads
command line option, 75
--verbose
command line option, 75
--version
command line option, 75
--working-dir
command line option, 81
-a
command line option, 78
-c
command line option, 71
-d
command line option, 79
-f
command line option, 76
-h
command line option, 75
-i
command line option, 79
-j
command line option, 79
-m
command line option, 78
-n
command line option, 78, 79
-o
command line option, 76, 79, 82
-p
command line option, 78, 79
-q
command line option, 75
-r
command line option, 71
-v
command line option, 75
-x
command line option, 72

A

allow-dubious-hosts, **83**

B

bird1, **34**
bird2, **34**

C

Certificate Revocation List (*CRL*), **107**
chroot, **87**
command line option
    -V, 75
    --allow-dubious-hosts, 73
    --asn, 78
    --chroot, 81
    --complete, 78, 79, 82
    --config, 71
    --detach, 79
    --dirty, 75
    --disable-rrdp, 74
    --disable-rsync, 73
    --enable-bgpsec, 75
    --exceptions, 72
    --expire, 81
    --extra-tals-dir, 72
    --format, 76
    --fresh, 73
    --group, 81
    --help, 75
    --history, 81
    --http, 80
    --http-tls, 80
    --http-tls-cert, 80
    --http-tls-key, 80
    --input, 79
    --json, 79
    --limit-v4-len, 73
    --limit-v6-len, 73
    --listen-systemd, 80
    --logfile, 75
    --max-ca-depth, 75
    --max-object-size, 75
    --more-specifics, 78
    --no-aspas, 78
    --no-rir-tals, 72
    --no-route-origins, 78
    --no-router-keys, 78
    --noupdate, 78, 79
    --output, 76, 79, 82
    --pid-file, 81
    --prefix, 79
    --quiet, 75
    --refresh, 81
    --repository-dir, 71
    --retry, 81
    --rrdp-connect-timeout, 74
    --rrdp-fallback, 74
    --rrdp-fallback-time, 74
    --rrdp-keep-responses, 75
    --rrdp-local-addr, 74
    --rrdp-max-delta-count, 74
    --rrdp-proxy, 74
    --rrdp-root-cert, 74
    --rrdp-tcp-keepalive, 74
    --rrdp-timeout, 74
    --rsync-command, 73
    --rsync-timeout, 73
    --rtr, 79
    --rtr-client-metrics, 80
    --rtr-tcp-keepalive, 80
    --rtr-tls, 80
    --rtr-tls-cert, 80
    --rtr-tls-key, 80
    --select-asn, 78
    --select-prefix, 78
    --stale, 72
    --strict, 72
    --syslog, 75
    --syslog-facility, 75
    --tal, 72
    --unknown-objects, 73
    --unsafe-vrps, 72
    --user, 81
    --validation-threads, 75
    --verbose, 75
    --version, 75
    --working-dir, 81
    -a, 78
    -c, 71
    -d, 79
    -f, 76
    -h, 75
    -i, 79
    -j, 79
    -m, 78
    -n, 78, 79
    -o, 76, 79, 82
    -p, 78, 79
    -q, 75
    -r, 71
    -v, 75
    -x, 72
csv, **29**
csvcompat, **29**
csvext, **29**

D

dirty, **85**
disable-rrdp, **84**
disable-rsync, **83**
dump
    module sub-command, 82

E

enable-bgpsec, **85**

exceptions, **83**
expire, **86**
extra-tals-dir, **83**

G

Ghostbusters Record (*GBR*), **107**
group, **87**

H

history-size, **86**
http-listen, **86**
http-tls-cert, **86**
http-tls-key, **86**
http-tls-listen, **86**

J

json, **30**
jsonext, **31**

L

limit-v4-len, **83**
limit-v6-len, **83**
listen-systemd, **86**
log, **85**
log-file, **85**
log-level, **85**

M

man
    module sub-command, 82
Manifest, **107**
max-ca-depth, **85**
max-object-size, **85**
Maximum Prefix Length (*MaxLength*), **107**
module sub-command
    dump, 82
    man, 82
    server, 79
    update, 81
    validate, 78
    vrps, 76

N

no-rir-tals, **82**

O

openbgpd, **34**

P

pid-file, **86**
Publication Point, **107**

R

refresh, **86**
Repository, **107**
repository-dir, **82**
Resource Public Key Infrastructure (*RPKI*), **107**
retry, **86**
RFC
    RFC 3779, 107
    RFC 5280, 89
    RFC 5652, 89
    RFC 6480, 107
    RFC 6486, 107
    RFC 6487, 89, 107
    RFC 6488, 89
    RFC 6493, 107
    RFC 6810, 43, 80, 108
    RFC 6810#section-7, 43
    RFC 8182, 108
    RFC 8209, 104
    RFC 8210, 43, 80, 81, 108
    RFC 8416, 33, 37, 72
    RFC 8630, 21, 72, 108
Route Origin Attestation (*ROA*), **107**
Route Origin Validation (*ROV*), **107**
RPKI Relying Parties, **107**
RPKI Repository Delta Protocol (*RRDP*), **108**
RPKI-to-Router (*RPKI-RTR*), **108**
rpsl, **35**
rrdp-connect-timeout, **84**
rrdp-fallback, **84**
rrdp-fallback-time, **84**
rrdp-keep-responses, **84**
rrdp-local-addr, **84**
rrdp-max-delta-count, **84**
rrdp-proxies, **84**
rrdp-root-certs, **84**
rrdp-tcp-keepalive, **84**
rrdp-timeout, **84**
rsync-args, **84**
rsync-command, **84**
rsync-timeout, **84**
rtr-client-metrics, **86**
rtr-listen, **85**
rtr-tcp-keepalive, **86**
rtr-tls-cert, **86**
rtr-tls-key, **86**
rtr-tls-listen, **85**

S

server
    module sub-command, 79
slurm, **33**
stale, **83**
Stale Object, **108**

strict, 83
summary, 35
syslog-facility, 85

T

tal-labels, 87
tals, 82
Trust Anchor (*TA*), 108
Trust Anchor Locator (*TAL*), 108

U

unknown-objects, 83
Unsafe VRPs, 108
unsafe-vrps, 83
update
    module sub-command, 81
user, 87

V

validate
    module sub-command, 78
Validated ROA Payload (*VRP*), 108
validation-threads, 85
vrps
    module sub-command, 76

W

working-dir, 87