RPKI Hints, Top Tips, and FAQs

On this page I'm collecting how to do various RPKI bits and pieces. Usually because the supplied documentation is incomplete, or just plain useless.

Here is the list (so far):

- NLnetLabs Routinator 3000
- NIC Mexico FORT
- RPKI-client
- GoRTR
- StayRTR

The tips and tricks discussed below all are for Ubuntu 20.04. They should also work just fine on Ubuntu 18.04 (which is supported until April 2023) and I'll note if I experience otherwise. I've not tested anything on the Ubuntu interims since 20.04.

NLnetLabs Routinator

Nothing to say here, the instructions just work, the validator installs sweetly, and just runs. As long as the instructions are followed.

If using Debian/Ubuntu as I do, then just use the supplied package and your favourite package manager. Described in NLnetLabs's Github repo.

You could build from source if you really want to, but why bother. If the link to the supplied package is added to your package manager, for example **apt** on Ubuntu, then create an entry in **/etc/apt/sources.list.d** called **routinator.list** and put this in it (which is for Ubuntu 20.04):

```
deb [arch=amd64] https://packages.nlnetlabs.nl/linux/ubuntu/ focal main
```

Easy!

FORT

FORT is not quite so easy to install, but still relatively simple as long as you follow the instructions closely.

The installation instructions are on Github.

(Actually, Marco d'Itri has created a Debian package which you can use to install from. But the FORT team note that it may be a release or two behind their own packaging.)

First step is to grab the **.deb** file from their archive:

```
wget
```

```
https://github.com/NICMx/FORT-validator/releases/download/1.5.3/fort_1.5.3-1_amd64.deb
```

and then install it:

```
sudo apt install ./fort_1.5.3-1_amd64.deb
```

Note that the **apt install** installs a **systemd** file and starts FORT running automatically. FORT uses TCP/323 as the listener port - you may want to customise this, and to do that, edit the configuration file **/etc/fort/config.json**. This is the configuration file that I use:

```
{
        "tal": "/etc/fort/tal",
        "local-repository": "/var/lib/fort",
        "slurm": "/etc/fort/slurm/",
        "server": {
                "port": "3323"
        },
        "log": {
                "output": "syslog"
        }
}
```

All I have done is modify the port that the server listens on.

The package ships with 4 of the 5 Trust Anchor Locators, so to get the missing one (ARIN's), you will need to run:

```
sudo fort --init-tals --tal=/etc/fort/tal
```

You will be asked to confirm that you have read the Terms and Conditions regarding ARIN's TAL:

```
...
Jan 26 03:50:46 DBG: Done. Total bytes transferred: 466
Jan 26 03:50:46 DBG: HTTP result code: 200
Successfully fetched '/etc/fort/tal/apnic.tal'!

Attention: ARIN requires you to agree to their Relying Party Agreement (RPA)
before you can download and use their TAL.
Please download and read https://www.arin.net/resources/manage/rpki/rpa.pdf
If you agree to the terms, type 'yes' and hit Enter: yes
Jan 26 03:50:51 DBG: HTTP GET:
https://www.arin.net/resources/manage/rpki/arin.tal
Jan 26 03:50:51 DBG: Done. Total bytes transferred: 487
Jan 26 03:50:51 DBG: HTTP result code: 200
Successfully fetched '/etc/fort/tal/arin.tal'!

Jan 26 03:50:51 DBG: HTTP GET:
https://www.lacnic.net/innovaportal/file/4983/1/lacnic.tal
...
```

One thing that I found is that FORT crashes on start up following the above installation instructions to the letter. The issue is that the **/var/lib/fort** folder is owned by **root**, not by the **fort** user. Easy to fix:

```
sudo chown fort:fort /var/lib/fort
```

Then restart FORT:

```
sudo systemctl restart fort
```

and it should run successfully. You should see something like this when you run **systemctl status fort**:

```
* fort.service - FORT RPKI validator
     Loaded: loaded (/lib/systemd/system/fort.service; enabled; vendor
preset: enabled)
    Drop-In: /run/systemd/system/service.d
             └─zzz-lxc-service.conf
     Active: active (running) since Wed 2022-01-26 03:54:05 UTC; 4s ago
       Docs: man:fort(8)
             https://nicmx.github.io/FORT-validator/
   Main PID: 3100 (fort)
      Tasks: 37 (limit: 28794)
     Memory: 12.0M
     CGroup: /system.slice/fort.service
             └─3100 /usr/bin/fort --configuration-file /etc/fort/config.json
```

You can check by using **ps ax** to get:

```
  195 ?        Ssl   95:13 /usr/bin/fort --configuration-file
/etc/fort/config.json
```

and **netstat -an** (upto Ubuntu 18.04) or **ss -an** (on Ubuntu 20.04 onwards) to get:

```
tcp     LISTEN   0        128                     0.0.0.0:3323
0.0.0.0:*
```

RPKI-client

rpki-client is just a validator - it does not have the functionality to accept connections from a router. We'll come to that later on (we'll need to use Cloudflare's GoRTR or its fork, StayRTR).

rpki-client has no package, although again Marco d'Itri is working on one as you can find from the Debian package tracker.

Before you attempt to download and build it, the **rpki-client** instructions note that you need a few other packages in place. These include **automake**, **autoconf**, **make**, **git** itself, **libtool** and **expat**. This is all quite easy using the Ubuntu package manager.

```
sudo apt install automake autoconf make git libtool libexpat1-dev
```

The other required package noted in the instructions is **tls** from LibreSSL. LibreSSL is a branch of OpenSSL and is used on OpenBSD - not found on Linux, but seems to be appearing in the latest Debian/Ubuntu beta builds. So we need to download the bits we need and install. The **rpki-client** instructions don't say anything about how to do that.

First we go to https://ftp.openbsd.org/pub/OpenBSD/LibreSSL/ and select the latest package, which is libressl-3.4.2.tar.gz at time of writing

```
wget https://ftp.openbsd.org/pub/OpenBSD/LibreSSL/libressl-3.4.2.tar.gz
```

We then unpack it:

```
tar zxf libressl-3.4.2.tar.gz
```

and then build it:

```
cd libressl-3.4.2
./configure --enable-libtls-only
make
sudo make install
```

Note the option to only build **libtls** - we don't need the rest of LibreSSL and it could well interfere with OpenSSL which will already be on the system. Now that **libtls** is built, the **install** action will put the libraries in **/usr/local/lib** like this:

```
-rw-r--r--  1 root root  27392794 Jan 20 15:17 libtls.a
-rw-r--r--  1 root root       933 Jan 20 15:17 libtls.la
lrwxrwxrwx  1 root root        16 Jan 20 15:17 libtls.so -> libtls.so.22.0.0
lrwxrwxrwx  1 root root        16 Jan 20 15:17 libtls.so.22 ->
libtls.so.22.0.0
-rw-r--r--  1 root root  13146784 Jan 20 15:17 libtls.so.22.0.0
```

Run **sudo ldconfig** so that the system knows about the new libraries.

Next we need to get some packages that **rpki-client** needs. These are libssl-dev and rsync.

```
sudo apt install libssl-dev rsync
```

And now we are ready to build **rpki-client**.

Easiest is just to install **rpki-client** from the Github repository:

```
git clone https://github.com/rpki-client/rpki-client-portable.git
```

and then:

```
cd rpki-client-portable
./autogen.sh
```

```
./configure --with-tal-dir=/etc/rpki \
   --with-base-dir=/var/lib/rpki-client \
   --with-output-dir=/var/db/rpki-client
make
```

The **autogen.sh** script fixes the config set up, ready to run **configure** which will produce the **Makefile**. Note that we are specifying where our TALs go, where the temporary files go (following Ubuntu norms), and where the output file storing the VRPs goes (again following Ubuntu norms).

Hidden in the official instructions is a comment that **rpki-client** runs as a normal user if started as root. So we need to create that normal user:

```
sudo groupadd _rpki-client
sudo useradd –g _rpki-client –s /sbin/nologin –d /nonexistent –c "rpki-
client user" _rpki-client
```

Now we can install RPKI-client:

```
sudo make install
```

which will install the client in **/usr/local/sbin** and the 4 TALs in **/etc/rpki**, as well as create the cache and output directories needed. Note that the ARIN TAL requires users to read the disclaimer first so is not provided by default. So you need to do this manually:

```
wget https://www.arin.net/resources/manage/rpki/arin.tal
sudo mv arin.tal /etc/rpki
```

Now the client can be run. There is no daemon option, it simply runs at the command line, and when it has finished downloading all the VRPs (around 10-15 minutes depending on bandwidth) it exits. But that's okay. Try running the client:

```
sudo /usr/local/sbin/rpki-client
```

You'll see errors about various CAs or files not being accessible - that's their problem, not yours. If you check in the **/var/db/rpki-client** folder you will see an **openbgpd** file once the above run of **rpki-client** completes. This is the configuration you'd need if you run **openbgp**. However, we are going to run a standalone RtR client instead, so we will need JSON output instead.

Once **rpki-client** completes, we can now set it up to run automatically. To do this, we create a file in **/etc/cron.hourly** called **rpki-client**, and in it we put:

```
#!/bin/bash

# run RPKI-client every hour
# - default output location is /var/db/rpki-client
# - -j option means json output, suitable for stayrtr

/usr/local/sbin/rpki-client -j > /tmp/rpki-client.log 2>&1
```

and that's it. Every hour, cron will run **rpki-client** which will produce JSON output of all the VRPs it has collected. As noted above, JSON output is what is used by StayRTR and GoRTR as their input

sources. Make sure that the **/etc/cron.hourly/rpki-client** is executable, otherwise it will not run.

It's a good idea to check the log file about in case **rpki-client** reports issues trying to write local files etc. But mostly what you'll see there are all the transactions with the various CAs, and the problems encountered (there will be lots, unfortunately).

GoRTR

I've included GoRTR here though it is no longer maintained by Cloudflare as the maintainer has moved on to pastures new. All development work is now being carried out on StayRTR which is a hard fork of GoRTR.

Installing Go

First you will need a working Go environment. Full instructions are at https://go.dev/doc/install, and I've reproduced the key pieces here to make it easy for installers.

First off, download the latest Go package (1.17.6 at time of writing):

```
wget https://go.dev/dl/go1.17.6.linux-amd64.tar.gz
```

If you have an existing Go environment, perhaps save it in case something goes wrong with the new version:

```
sudo mv /usr/local/go /usr/local/go.old
```

and then you can unpack the new version:

```
cd /usr/local
sudo chmod 777 .
tar xzf ~/go1.17.6.linux-amd64.tar.gz
sudo chmod 755 .
```

Next add **/usr/local/go/bin** to the **PATH** environment variable. If you use **bash**, this would be in the **.profile** in your home directory, and just add:

```
if [ -d "/usr/local/go/bin" ] ; then
    PATH="$PATH:/usr/local/go/bin"
fi
```

Log off. And then log in again. (Easiest way of activating the updated **PATH**.)

And now check you have a working Go environment:

```
go version
```

If the version shows what you installed, you are set!

Installing GoRTR

Easiest way to do this is to build from the Github repo.

Note1: You could download and use the provided binaries if you wish.

Note2: You could even download the Debian package if you wish, and install that. It needs the **adduser** package, and a **libc** from 2.4 onwards (most modern Ubuntu releases). Bonus with the .deb package is that it comes with a **systemd** configuration.

But we will focus on building from the source.

```
git clone https://github.com/cloudflare/gortr.git
cd gortr
make build-gortr build-rtrmon build-rtrdump
```

which builds **gortr** as well as **rtrmon** and **rtrdump** (the latter used for testing purposes).

Copy the resulting binaries to **/usr/local/bin**:

```
cd dist
sudo -s
cp -p gortr-v0.14.7-1-g2125744-linux-x86_64 /usr/local/bin/gortr
cp -p rtrdump-v0.14.7-1-g2125744-linux-x86_64 /usr/local/bin/rtrdump
cp -p rtrmon-v0.14.7-1-g2125744-linux-x86_64 /usr/local/bin/rtrmon
```

GoRTR has lots of options, but the ones we need are these:

```
 -bind string
       Bind address (default ":8282")
 -cache string
       URL of the cached JSON data (default
"https://rpki.cloudflare.com/rpki.json")
 -checktime
       Check if file is still valid (default true)
 -verify
       Check signature using provided public key (disable by passing -
verify=false)
```

We don't need to use the Cloudflare JSON source, given we have our own from the newly created RPKI-client. RPKI-client doesn't insert a timestamp in the way that GoRTR wants, nor is there a signature on it, so we need to disable that too.

We run GoRTR like this:

```
/usr/local/bin/gortr -bind :3323 -verify=false -cache /var/db/rpki-
client/json -checktime=false
```

which will at least let us test that it works. Run it and see what happens - you should see output at the command line looking like this:

```
INFO[0001] New update (304138 uniques, 304138 total prefixes). 0 bytes.
Updating sha256 hash  ->
0592ddc6e9a82666f8ddc5eda8cad76cb61f22640f17199b1bff06b5928b9718
INFO[0002] Updated added, new serial 0
INFO[0002] GoRTR Server started (sessionID:33094, refresh:3600, retry:600,
expire:7200)
```

And if you check the ports that are listening (**ss -an**) you will see:

```
tcp    LISTEN     0       128          *:3323              *:*
tcp    LISTEN     0       128          *:8080              *:*
```

Port 3323 is the listening port for Router connections. And Port 8080 is the metrics port, for monitoring systems to connect to.

But perhaps this isn't good for long term operations as you'd prefer to have this start running automatically when the system starts. And for that we'd need to set up a suitable **systemd** entry (to be completed).

StayRTR

StayRTR is a hard fork of GoRTR which is no longer maintained by Cloudflare. For this reason, I recommend you use StayRTR rather than GoRTR.

As of writing this, there are no packages you can just download and run. So we'll download the source and build from there.

Note: there is an experimental .deb package by Marco d'Itri, but that needs at least Ubuntu 21.04 to support it, as it requires a libc which is 2.32 or newer (Ubuntu 20.04 uses libc version 2.31).

Given it's parentage in GoRTR, the install process is very similar. First off, make sure you have a working Go environment - consult the instructions in the GoRTR section above.

Easiest way to install StayRTR is build from the Github repo.

```
git clone https://github.com/bgp/stayrtr.git
cd stayrtr
make build-all
```

which builds **stayrtr** as well as **rtrmon** and **rtrdump** (the latter used for testing purposes).

Copy the resulting binaries to **/usr/local/bin** (note if you have GoRTR installed too, as I do, you may want to rename its **rtrdump** and **rtrmon** binaries appropriately):

```
cd dist
sudo cp -p stayrtr-0.1-91-gc4ac625-linux-x86_64 /usr/local/bin/stayrtr
sudo cp -p rtrdump-0.1-91-gc4ac625-linux-x86_64 /usr/local/bin/rtrdump
sudo cp -p rtrmon-0.1-91-gc4ac625-linux-x86_64 /usr/local/bin/rtrmon
```

StayRTR has lots of options, but the ones we need are these:

```
 -bind string
       Bind address (default ":8282")
 -cache string
       URL of the cached JSON data (default
"https://console.rpki-client.org/vrps.json")
```

We don't need to use the public RPKI-client JSON source, given we have our own from the newly created RPKI-client.

We run StayRTR like this:

```
/usr/local/bin/stayrtr -bind :3323 -cache /var/db/rpki-client/json
```

which will at least let us test that it works. Run it and see what happens - you should see output at the command line looking like this:

```
INFO[0000] new cache file: Updating sha256 hash  ->
e0a14ea955e183e2719dcfbee0e9429b34581972c6ad5f6e9e064ee1396caf60
INFO[0001] New update (307007 uniques, 307007 total prefixes).
INFO[0002] Updated added, new serial 0
INFO[0002] StayRTR Server started (sessionID:60037, refresh:3600, retry:600,
expire:7200)
```

And if you check the ports that are listening (**ss -an**) you will see:

```
tcp     LISTEN   0        128          *:9847              *:*
tcp     LISTEN   0        128          *:3323              *:*
```

Port 3323 is the listening port for Router connections. And Port 9847 is the metrics port, for monitoring systems to connect to.

But perhaps this isn't good for long term operations as you'd prefer to have this start running automatically when the system starts. And for that we'd need to set up a suitable **systemd** entry (to be completed).

Back to Home page