Route Origin Validation Lab

Part-1: Installing RPKI Validator (Routinator)

Login Details

- Username `apnic` and password `training`.

Preinstalled packages

To save time, the following dependencies have been preinstalled:

- `GCC (GNU C toolchain)`, `rsync`

Lab Setup

For this lab, we will use Routinator from NLnetLabs as the RPKI validator.

1. Login to your server (SSH from the jumphost to your container using the `username` and `password` given above), where `X` is your VM number:

   ```
   ssh apnic@192.168.30.X
   ```

2. Update the repository

   ```
   sudo apt update && sudo apt upgrade
   ```

   Note: Since Routinator is written in `rust`, first install `rust` using `rustup` (which is a rust installer and version management tool) from the official release channels.

3. Run the following `curl` command which will download a script that downloads `rustup` and installs `rust`

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

   # -f: fail silently (HTTP)
   # -sS: show errors if it fails
   ```

4. Follow the onscreen instructions to install rust:

   - we will go with the default installation option `1`

5. Make sure to set the PATH environment variable as shown in the onscreen instruction:

```
source $HOME/.cargo/env
```

6. Now you can use `cargo` (the rust package manager) to install Routinator:

    - ***NOTE-1***:

        - v0.7.1 ensured RIR TALs were fetched over HTTPS (RRDP).
        - Some more updates have since been included in v0.8.0-rc2

        ```
        cargo install -f --version 0.8.0-rc2 routinator
        ```

        - You can verify your routinator version with:

        ```
        routinator --version
        ```

    - ***NOTE-2***: To update an exisiting installation, do:

        - Rust

        ```
        rustup update
        ```

        - Routinator

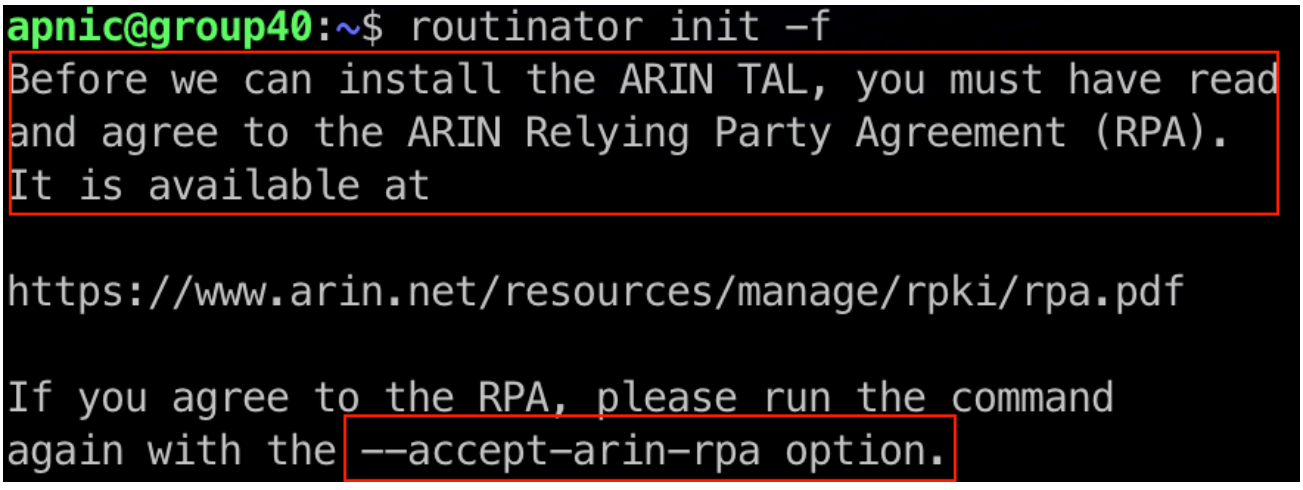
        ```
        cargo install -f routinator
        ```

7. Before running Routinator for the first time, we need to prepare its working environment (directory for the local RPKI cache as well as Trust Anchor Locator - TAL).

    ```
    routinator init -f
    ```

    - `-f` : Forces installation of the TALs even if the TAL directory already exists.

    - Since this is the first time we are using Routinator, it will complain that ARIN's TAL is missing as shown below:

8. If we agree to Arin's relying party agreement, reissue the command with the `--accept-arin-rpa` option as shown below:

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

```
apnic@group40:~$ routinator init -f --accept-arin-rpa
Created local repository directory /home/apnic/.rpki-cache/repository
Installed 5 TALs in /home/apnic/.rpki-cache/tals
```

   - This will create the local rpki cache directory as well as download the TALs (from the five RIRs) and save it in the relevant directory.

9. Look at the current/default configuration:

```
routinator config
```

```
apnic@group01:~$ routinator config
allow-dubious-hosts = false
dirty = false
disable-rrdp = false
disable-rsync = false
exceptions = []
expire = 7200
history-size = 10
http-listen = []
log = "default"
log-level = "WARN"
refresh = 600
repository-dir = "/home/apnic/.rpki-cache/repository"
retry = 600
rrdp-proxies = []
rrdp-root-certs = []
rsync-command = "rsync"
rsync-timeout = 300
rtr-listen = []
rtr-tcp-keepalive = 60
stale = "reject"
strict = false
syslog-facility = "daemon"
systemd-listen = false
tal-dir = "/home/apnic/.rpki-cache/tals"
unknown-objects = "warn"
unsafe-vrps = "warn"
validation-threads = 8
```

   - To understand the details of the different flags/options or sub-commands, refer the manual page
   - Few of the flags/options are explained below:

     - `expire` : if the RTR session goes down, how long is a client (routers) allowed to retain the ROA cache.

- `refresh` : how long before the validator updates the local repo
- `retry` : an indication of how long an RTR client should wait before trying to refresh again
- `stale` : what to do with stale objects - manifests/CRLs that haven't been updated (*next-update*)
- `strict` : whether we should have strict validation or not

10. Let us now do a test run with the following command to fetch ROAs from the repositories, validate each one, and output the validated ROA payload (`origin ASN, prefix, max prefix-length`).

    Note: It will take a while, so dont worry. Instead of printing to standard output, we can write it to a file with the `--output` option:

    ```
    routinator -v vrps -o routinator.csv
    ```

    - Have a look at the validated ROA payload (VRP) file:

      ```
      more routinator.csv
      ```

11. **[Optional]** If you have two separate Validators installed (for redundancy/code diversity), compare the validated ROA payload outputs for consistency:

    - But before you compare the VRPs, you would need to sort and normalise the outputs. For example, as you can see from the routinator.csv file, it has a extra column for Trust Anchor, which other validators may not output. We can trim it with:

      ```
      cut -d, -f4 --complement routinator.csv > rout_trimmed.csv
      ```

      - Have a look at the trimmped output: `more rout_trimmed.csv`

    - Let us now sort the trimmed outout with a basic `sort` command:

      ```
      sort rout_trimmed.csv > rout_sorted.csv
      ```

      - Have a look at the sorted output: `more rout_sorted.csv`

    - Now your routinator VRP file is ready to be compared with other validator outputs later.

Now your validator is ready to feed the validated cache to BGP speaking routers through the RTR (RPKI-to-Router) protocol.

Part-2: RTR session

Validator side

Routinator can act as an RTR server as specified in RFC8210, that periodically fetches RPKI data, verifies/validates it and allows RPKI enabled routers to connect to it to fetch the validated data (ROA cache).

- ***Note:*** IANA has specified a standard port `323` for RTR, which would require running Routinator as a root.

1. To run Routinator as a RTR server listening on `192.168.30.X` (where X is your VM number) and port `3323` :

   ```
   routinator server --rtr 192.168.30.X:3323 --refresh=300 --detach
   ```

   - If you dont specify the ***refresh*** time, by default the local repo will be updated and re-validated every 600 seconds (10 minutes). The example above uses a `300secs (5 mins)` refresh time.

   Note: If you have IPv6 address configured on routinator, you can listen on both:

   ```
   routinator server --rtr 192.168.30.X:3323 --rtr [2001:0DB8::X]:3323 --refresh=
   300 --detach
   ```