Network Configuration Example

BGP Origin Validation Using Resource Public Key Infrastructure

Published
2020-06-25
# Table of Contents

BGP Origin Validation Using RPKI

About This Network Configuration Example | 5

Use Case Overview | 5

Technical Overview | 6

- Public Key Infrastructure Overview | 6
- Certificate Distribution | 6
- RPKI Details | 7
- How Routers Process RPKI Data | 9
- RPKI Cache Server | 10

BGP Origin Validation Using RPKI | 11

- Configure RPKI | 11
- Configure the RPKI Validator | 14
- RPKI Validator Verification and Operation | 17
- Origin Validation Using RPKI Configured on a Junos OS Router | 19

Conclusion | 24
CHAPTER

BGP Origin Validation Using RPKI

About This Network Configuration Example | 5
Use Case Overview | 5
Technical Overview | 6
BGP Origin Validation Using RPKI | 11
Conclusion | 24
About This Network Configuration Example

This network configuration example (NCE) helps you to perform BGP Origin Validation using Resource Public Key Infrastructure (RPKI). The goal of Origin Validation is to reduce the negative impact of routes with incorrect origins caused by misconfigured routers or intentional hijacks. This document forms a starting point for anyone implementing Origin Validation.

Use Case Overview

BGP in the Internet core relies on a transitive trust model. In a transitive trust, a certain level of trust is inherited by an entity through an intermediary, rather than directly from the source. In a data center, a new process might inherit access rights not only from a parent process but also from the parent process's parent. Transitive trusts are not the only trust model. For example, a bank might trust checks drawn on an account in a different branch, but not on an account from a bank that does not exist. There are also hierarchical trust models where trust passes from a top authority down to others.

On the Internet, the transitive trust nature of BGP routing information can lead to outages when a device incorrectly announces routes it does not own. By design, routers choose a more specific route in favor of a less specific route. An incorrect route announcement that has a longer and more specific network mask can result in the hijacking of a prefix throughout the Internet. This is a common event on the Internet and it usually occurs because of an accidental misconfiguration.

In response to this, the IETF has defined solutions that help to reduce the impact of this inherent vulnerability; Origin Validation.

RPKI is a specialized PKI framework designed to secure the Internet's routing infrastructure. RPKI performs the following:

- Provides a way to validate whether a rightful resource holder authorizes the originating AS number and announces the IPv4 and IPv6 prefixes.
- Makes sure that the prefix lengths for the routes are within the limits defined by their owners.
- Links the route information (a resource) to a trust anchor (a root certificate authority). This linking allows legitimate holders of these route resources to control the operation of routing protocols, in BGP, regarding their route information.
- Uses X.509 certificates, which allow local Internet registries (LIRs) to obtain a resource certificate listing the AS numbers and IP address resources they hold.
Technical Overview

Public Key Infrastructure Overview

PKI uses asymmetric keys called public and private keys to encrypt and decrypt information. RPKI uses one key to encrypt the private key and the other key is used to decrypt the public key. A Certificate Authority (CA) validates the holder of these resources. RPKI uses the hierarchical trust model where the trust runs from the End Entity (EE) Certificates back up to the Root CA Certificate. RPKI uses Digital Signatures to validate they are the legitimate holders of the routing resources.

Certificate Distribution

The certificate structure mirrors the way IP addresses and AS numbers are distributed. The IANA is a department within the Internet Corporation for Assigned Names and Numbers (ICANN). The ICANN distributes names and addresses to the Regional Internet Registries (RIRs). The RIRs distribute names and addresses to their members, the LIRs, and in some cases to end customers.

Figure 1 on page 7 shows a map of the RIRs and their coverage area of responsibility.
RPKI Details

Overall RPKI operation is defined by several RFCs. RPKI uses X.509 certificates (described in RFC 5280 and RFC 6818), extensions for IP addresses and AS identifiers (RFC 3779), BGP prefix origin validation (RFC 6811 and 8481), and Resource Public Key Infrastructure (RFC 6810, 8210, and RFCs 6480-6493). This allows the LIRs to obtain a resource certificate listing the AS numbers and IP addresses resources they hold; a certificate that validates their proof of ownership.

The LIRs create cryptographic attestations using the resource certificate. A cryptographic attestation shows a valid claim to the route announcements they make with the prefixes they hold. Figure 2 on page 8 shows the general content of the X.509 certificate.
RPKI calls these attestations Route Origination Authorizations (ROAs). ROAs describe the holder’s intentions including originating AS number, an IPv4 address or IPv6 prefix, and optionally, the maximum prefix length. If the ROA does not include a maximum length, then the ASN is authorized only to advertise the exact prefix: anything more specific is considered invalid. Using the maximum prefix length is a way to prevent route hijacking through a more specific announcement.

We recommend you be conservative in using maximum length in ROAs. For example, if a prefix has only a few sub-prefix announcements, use multiple ROAs for the announcements, instead of one ROA with a long maximum length.

For example, if, instead of 10.0.0.0/16 and maximum prefix length /24, the resource holder issues 10.0.0.0/16 and 10.0.42.0/24, a forged origin attack cannot succeed against 10.66.6.0/24.

RPKI uses Rsync and RRDP publication protocols to distribute RPKI certificates. This allows you to use the RPKI data to retrieve certificates from the RIRs and store them locally in your cache.

Routers then retrieve the entire RPKI database from the caches using the RPKI-to-router (RPKI-RTR) protocol. The protocol uses a form of serial number logic to check for changes to the database. This means that routers need to download the full database only once, and then check periodically to see if any ROAs have changed.

*Figure 3 on page 9* shows the process of passing these certificates from IANA through a Security Industry Authority (SIA) to the ROA. The ISP adds the IP address, prefix length, and ASN as appropriate.
How Routers Process RPKI Data

It is up to the network operator on how to use RPKI data. The border routers of an AS number should enforce these decisions. The router checks each of received BGP routes against the received ROAs and sets each route’s validation state to one of the following possible states:

- **VALID**—A matching or covering ROA was found with a matching AS number. A covering ROA is one that matches the AS number and falls in between the prefix length and maximum prefix length. For example, the ROA 10.0.0.0/16 with maximum prefix length /24 covers 10.0.0.0/22.

- **INVALID**—A matching or covering ROA was found, but the AS number did not match, or the mask exceeded the defined maximum length.

- **UNKNOWN or NOT FOUND**—No matching or covering ROA was found. This scenario could be because the owner did not create an RPKI statement, or the ROA certificate has expired, but there are other possible causes.

Based on the route’s validation state, each operator decides how to proceed according to their local policy. Junos OS accomplishes this using a routing policy. Ideally, one policy might drop any routes with a state of INVALID, but this action is not mandatory.

Figure 4 on page 10 shows the overall flow from the global RPKI level through the cache to the individual router.
RPKI Cache Server

Origin validation using RPKI requires three components:

- The ROA entries from the RPKI databases
- A router to enforce decisions based on these ROAs and its operator’s policy
- The RPKI cache server (also referred to as the RPKI Validator).

The cache server queries all the distributed (RIRs) RPKI databases to collect all the ROAs and validates each entry’s signature. The cache server then replies to the router’s query by forwarding all ROAs in the validated cache to the router. The router does not decrypt the certificates because this is the task of the validator.
BGP Origin Validation Using RPKI

This example shows how to configure RPKI, configure RPKI validator, RPKI validator verification and operation, and the Origin Validation.

Configure RPKI

This configuration example provides instructions to configure RPKI.

Requirements

This example uses the following hardware and software components:

- Tested with Junos OS 19.2R1.8 (Origin Validation supported since Junos OS Release 12.2).
- Two MX204 routers.
- A virtual machine (VM)
Overview

Figure 5 on page 12 shows a high-level network diagram used for reference throughout this document.

The two MX204 routers, Router1 and Router2, are border routers that control the routes received from upstream ISPs. RPKI is configured on these two routers. The virtual machine (VM) (IP address 172.18.158.39) runs the RIPE NCC RPKI Validator. This VM downloads the RPKI ROA’s from the 5 RIRs.

Figure 5: High-Level Network Diagram for RPKI Origin Validation Testing
Configure RPKI

The BGP import policy accepts both VALID and UNKOWN validation states and rejects routes that have a RPKI INVALID validation state.

The relevant configuration for the Router1 MX204 are highlighted in the following router options:

```plaintext
routing-options {
    router-id 10.104.0.254;
    autonomous-system 64496;
    validation {
        group rpki-validator {
            session 172.18.158.39 { # IP address of the RPKI cache server.
                port 8323; # Other validators may use different ports.
                local-address 172.18.1.127;
            }
        }
    }
}
protocols {
    bgp {
        group isp1-r1 {
            type external;
            # Policy that acts on received BGP routes based on their validation state.
            import route-validation;
            export export-direct;
            neighbor 10.104.0.3 {
                peer-as 64510;
            }
            neighbor 10.104.3.2 {
                peer-as 64500;
            }
            neighbor 10.104.0.1 {
                peer-as 64509;
            }
        }
    }
}
```
Configure the RPKI Validator

RPKI validator configuration uses the RPKI validator from RIPE NCC, and the other validators are available from NLnet Labs, and Cloudflare.

In the case of the RIPE NCC validator, a Linux operating system with OpenJDK 8 or higher and rsync support is required. The validation software is available at the RIPE’s website and it is downloaded to the server. Figure 6 on page 15 shows the RIPE NCC download screen for the RPKI validator software.
This configuration uses a VM running Ubuntu 18.04.4 LTS. This RPKI validator package does not require installation. The package is ready to run when you extract it. The package supports a Web-based interface on port 8080 for monitoring and configuring the validator. Without changes to the configuration, the package listens on port 8323 for RPKI-RTR communications. The configuration is shown below.

```
root@dev01:/var/tmp# wget
```

```bash
--2020-03-31 23:07:38--
Resolving ftp.ripe.net (ftp.ripe.net)... 193.0.6.140, 2001:67c:2e8:22::c100:68c
Connecting to ftp.ripe.net (ftp.ripe.net)|193.0.6.140|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 42677891 (41M) [application/x-gzip]
Saving to: 'rpki-validator-3-latest-dist.tar.gz'

 rpki-validator-3-latest-dist.tar.gz
 100%[====================================] 40.70M 34.9MB/s in 1.2s
By default, the validator listens to the localhost only. If you need to change this behavior, comment the `optserver.address=localhost` or specify a different IP address in the `application.properties` file in the `conf/` directory.

The next step is to start the validator:

```
```

Console shows output and after some time a pattern of retrieving and validating ROAs is noticed.

In summary, the steps include:

- Download TAL from https://www.arin.net/resources/manage/rpki/tal/

Start the RPKI-RTR daemon that handles the connection between the validator and the router.

For a reachable RTR server, besides localhost, you need to change the `application.properties` file in the `conf/` directory for the validator.

---

**RPKI Validator Verification and Operation**

You can use the Web interface to view the state of the RPKI validator process. Figure 7 on page 17 shows how to check the state of downloading and validating ROAs from the preconfigured RIR RPKI repository (Trust Anchors).

**Figure 7: Checking the Configured Trust Anchors**

You can check the connection state from the routers to the RPKI roots as well. Figure 8 on page 18 shows the view of the validated ROAs from various NICs and the RIPE NCC.
The RPKI validator has a very useful feature, the Whitelist, which lets you create your own ROAs locally as seen in Figure 9 on page 19.
Origin Validation Using RPKI Configured on a Junos OS Router

The Junos OS has the following two commands focused on Origin Validation:

1. The `show validation` command displays the state of the router with regard to the RPKI validators.

   ```
   user@Router1> show validation ?
   Possible completions:
   - database: Show contents of route validation database
   - group: Show route validation redundancy groups
   - replication: Show route validation replication information
   - session: Show route validation session information
   - statistics: Show route validation statistics
   ```

   ```
   user@Router1> show validation session
   ```
<table>
<thead>
<tr>
<th>Session records</th>
<th>State</th>
<th>Flaps</th>
<th>Uptime</th>
<th>#IPv4/Ipv6</th>
</tr>
</thead>
<tbody>
<tr>
<td>172.18.158.39</td>
<td>Up</td>
<td>0</td>
<td>06:33:59</td>
<td>3985/668</td>
</tr>
</tbody>
</table>

user@Router1> **show validation database**

**RV database for instance master**

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Origin-AS</th>
<th>Session</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.0/12-16</td>
<td>64508</td>
<td>172.18.158.39</td>
<td>valid</td>
</tr>
<tr>
<td>10.0.0.0/16-16</td>
<td>64508</td>
<td>172.18.158.39</td>
<td>valid</td>
</tr>
<tr>
<td>10.0.0.0/22-24</td>
<td>64500</td>
<td>172.18.158.39</td>
<td>valid</td>
</tr>
<tr>
<td>10.1.0.0/16-16</td>
<td>64508</td>
<td>172.18.158.39</td>
<td>valid</td>
</tr>
<tr>
<td>10.2.0.0/16-16</td>
<td>64508</td>
<td>172.18.158.39</td>
<td>valid</td>
</tr>
<tr>
<td>10.3.0.0/16-16</td>
<td>64508</td>
<td>172.18.158.39</td>
<td>valid</td>
</tr>
<tr>
<td>10.4.0.0/16-16</td>
<td>64508</td>
<td>172.18.158.39</td>
<td>valid</td>
</tr>
<tr>
<td>10.5.0.0/16-16</td>
<td>64508</td>
<td>172.18.158.39</td>
<td>valid</td>
</tr>
<tr>
<td>10.6.0.0/16-16</td>
<td>64508</td>
<td>172.18.158.39</td>
<td>valid</td>
</tr>
<tr>
<td>10.8.0.0/16-16</td>
<td>64508</td>
<td>172.18.158.39</td>
<td>valid</td>
</tr>
<tr>
<td>10.9.0.0/16-16</td>
<td>64508</td>
<td>172.18.158.39</td>
<td>valid</td>
</tr>
<tr>
<td>10.10.0.0/16-16</td>
<td>64508</td>
<td>172.18.158.39</td>
<td>valid</td>
</tr>
<tr>
<td>10.11.0.0/16-16</td>
<td>64508</td>
<td>172.18.158.39</td>
<td>valid</td>
</tr>
<tr>
<td>10.12.0.0/16-16</td>
<td>64508</td>
<td>172.18.158.39</td>
<td>valid</td>
</tr>
<tr>
<td>10.13.0.0/16-16</td>
<td>64508</td>
<td>172.18.158.39</td>
<td>valid</td>
</tr>
<tr>
<td>10.14.0.0/16-16</td>
<td>64508</td>
<td>172.18.158.39</td>
<td>valid</td>
</tr>
<tr>
<td>10.80.0.0/14-14</td>
<td>64507</td>
<td>172.18.158.39</td>
<td>valid</td>
</tr>
</tbody>
</table>
172.16.0.0/16-16  64500  172.18.158.39  valid

user@Router1> **show validation statistics**

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total RV records</td>
<td>4653</td>
</tr>
<tr>
<td>Total Replication RV records</td>
<td>4653</td>
</tr>
<tr>
<td>Prefix entries</td>
<td>4434</td>
</tr>
<tr>
<td>Origin-AS entries</td>
<td>4653</td>
</tr>
<tr>
<td>Memory utilization</td>
<td>907142 bytes</td>
</tr>
<tr>
<td>Policy origin-validation requests</td>
<td>700</td>
</tr>
<tr>
<td>Valid</td>
<td>4</td>
</tr>
<tr>
<td>Invalid</td>
<td>232</td>
</tr>
<tr>
<td>Unknown</td>
<td>464</td>
</tr>
<tr>
<td>BGP import policy reevaluation notifications</td>
<td>100</td>
</tr>
<tr>
<td>inet.0</td>
<td>100</td>
</tr>
<tr>
<td>inet6.0</td>
<td>0</td>
</tr>
</tbody>
</table>

2. The **show route validation-state** command displays routes having a certain validation state.

user@Router1> **show route validation-state ?**

<table>
<thead>
<tr>
<th>Possible completions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invalid</td>
<td>Invalid route validation state</td>
</tr>
<tr>
<td>unknown</td>
<td>Unknown route validation state</td>
</tr>
<tr>
<td>unverified</td>
<td>Unverified route validation state</td>
</tr>
<tr>
<td>valid</td>
<td>Valid route validation state</td>
</tr>
</tbody>
</table>

The following three examples provide received routes and the resultant validation states on Router1.

- **10.0.0/24 thru 10.0.9/24**
  
The RPKI validator has ROAs for 10.0.0/22 with a maximum prefix length of 24, which means all of the routes should have a state of VALID; however, there is also an ROA for 10.0.0.0/12-16 (maximum /16) for origin AS of 64508. This marks 10.0.4/24 thru 10.0.9/24 as INVALID.

- **172.16.0/24 thru 172.16.9/24**
  
The RPKI validator has ROAs for 172.16.0.0/16 and a maximum prefix length of 16, which means all of these routes should have a state of INVALID. Even though the origin AS number is correct, they have exceeded the maximum prefix length of /16.

- **172.30.0.0/24 thru 172.30.9.0/24**
The RPKI validator does not have ROAs for any of these routes, which means all of these should have a state of UNKNOWN.

user@Router1> show route validation-state valid

inet.0: 40 destinations, 40 routes (40 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

10.0.0.0/24       *[BGP/170] 00:36:02, localpref 100
   AS path: 64500 ?, validation-state: valid
           > to 10.104.3.2 via ge-1/0/1.3110
10.0.1.0/24       *[BGP/170] 00:36:02, localpref 100
   AS path: 64500 ?, validation-state: valid
           > to 10.104.3.2 via ge-1/0/1.3110
10.0.2.0/24       *[BGP/170] 00:36:02, localpref 100
   AS path: 64500 ?, validation-state: valid
           > to 10.104.3.2 via ge-1/0/1.3110
10.0.3.0/24       *[BGP/170] 00:36:02, localpref 100
   AS path: 64500 ?, validation-state: valid
           > to 10.104.3.2 via ge-1/0/1.3110

user@Router1> show route validation-state unknown

inet.0: 40 destinations, 40 routes (40 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

172.30.0.0/24    *[BGP/170] 00:36:06, localpref 100
   AS path: 64500 ?, validation-state: unknown
           > to 10.104.3.2 via ge-1/0/1.3110
172.30.1.0/24    *[BGP/170] 00:36:06, localpref 100
   AS path: 64500 ?, validation-state: unknown
           > to 10.104.3.2 via ge-1/0/1.3110
172.30.2.0/24    *[BGP/170] 00:36:06, localpref 100
   AS path: 64500 ?, validation-state: unknown
           > to 10.104.3.2 via ge-1/0/1.3110
172.30.3.0/24    *[BGP/170] 00:36:06, localpref 100
   AS path: 64500 ?, validation-state: unknown
           > to 10.104.3.2 via ge-1/0/1.3110
172.30.4.0/24    *[BGP/170] 00:36:06, localpref 100
   AS path: 64500 ?, validation-state: unknown
           > to 10.104.3.2 via ge-1/0/1.3110
172.30.5.0/24    *[BGP/170] 00:36:06, localpref 100
   AS path: 64500 ?, validation-state: unknown
           > to 10.104.3.2 via ge-1/0/1.3110
172.30.6.0/24 *[BGP/170] 00:36:06, localpref 100
  AS path: 64500 ?, validation-state: unknown
    > to 10.104.3.2 via ge-1/0/1.3110
172.30.7.0/24 *[BGP/170] 00:36:06, localpref 100
  AS path: 64500 ?, validation-state: unknown
    > to 10.104.3.2 via ge-1/0/1.3110
172.30.8.0/24 *[BGP/170] 00:36:06, localpref 100
  AS path: 64500 ?, validation-state: unknown
    > to 10.104.3.2 via ge-1/0/1.3110
172.30.9.0/24 *[BGP/170] 00:36:06, localpref 100
  AS path: 64500 ?, validation-state: unknown
    > to 10.104.3.2 via ge-1/0/1.3110

user@Router1> show route validation-state invalid

inet.0: 40 destinations, 40 routes (40 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

  10.0.4.0/24 *[BGP/170] 00:36:09, localpref 100
    AS path: 64500 ?, validation-state: invalid
    > to 10.104.3.2 via ge-1/0/1.3110
  10.0.5.0/24 *[BGP/170] 00:36:09, localpref 100
    AS path: 64500 ?, validation-state: invalid
    > to 10.104.3.2 via ge-1/0/1.3110
  10.0.6.0/24 *[BGP/170] 00:36:09, localpref 100
    AS path: 64500 ?, validation-state: invalid
    > to 10.104.3.2 via ge-1/0/1.3110
  10.0.7.0/24 *[BGP/170] 00:36:09, localpref 100
    AS path: 64500 ?, validation-state: invalid
    > to 10.104.3.2 via ge-1/0/1.3110
  10.0.8.0/24 *[BGP/170] 00:36:09, localpref 100
    AS path: 64500 ?, validation-state: invalid
    > to 10.104.3.2 via ge-1/0/1.3110
  10.0.9.0/24 *[BGP/170] 00:36:09, localpref 100
    AS path: 64500 ?, validation-state: invalid
    > to 10.104.3.2 via ge-1/0/1.3110
172.16.0.0/24 *[BGP/170] 00:36:08, localpref 100
  AS path: 64500 ?, validation-state: invalid
    > to 10.104.3.2 via ge-1/0/1.3110
172.16.1.0/24 *[BGP/170] 00:36:08, localpref 100
  AS path: 64500 ?, validation-state: invalid
    > to 10.104.3.2 via ge-1/0/1.3110
172.16.2.0/24 *[BGP/170] 00:36:08, localpref 100
  AS path: 64500 ?, validation-state: invalid
Conclusion

The goal of Origin Validation is to reduce the negative impact of routes with incorrect origins caused by misconfigured routers or intentional hijacks.

Origin Validation has been in use on the Internet since January of 2011 and continued adoption is ongoing. Large Tier-1 networks NTT, AT&T, Cloudflare, Telia Carrier, and others have already deployed it.

This NCE shows the purpose and configuration for Origin Validation. It also shows the basic operation with the Junos OS router CLI commands used to verify its operation.

This document forms a starting point for anyone implementing Origin Validation. When setting up Origin Validation in a production environment, we recommend you to define the local policy on how to treat each validation state and then define the BGP import policy to match.

Refer to Juniper Network’s technical documentation on the Junos OS routing policies for more information.
See the Day One book *Deploying BGP Routing Security* for a detailed overview of routing security suggestions.