Network Configuration Example

Configuring Origin Validation for BGP

Release
NCE0066

Modified: 2017-01-23

Juniper Networks, Inc.
1133 Innovation Way
Sunnyvale, California 94089
USA
408-745-2000
www.juniper.net

Network Configuration Example Configuring Origin Validation for BGP
NCE0066

The information in this document is current as of the date on the title page.

YEAR 2000 NOTICE

Juniper Networks hardware and software products are Year 2000 compliant. Junos OS has no known time-related limitations through the year 2038. However, the NTP application is known to have some difficulty in the year 2036.

END USER LICENSE AGREEMENT

The Juniper Networks product that is the subject of this technical documentation consists of (or is intended for use with) Juniper Networks software. Use of such software is subject to the terms and conditions of the End User License Agreement ("EULA") posted at http://www.juniper.net/support/eula.html. By downloading, installing or using such software, you agree to the terms and conditions of that EULA.

Table of Contents

CHAPTER 1

Configuring Origin Validation for BGP

About This Network Configuration Example

This network configuration example describes the uses for and benefits of enabling route origin validation for BGP. It also provides a step-by-step configuration example for BGP route origin validation that includes steps to validate that the configuration is working.

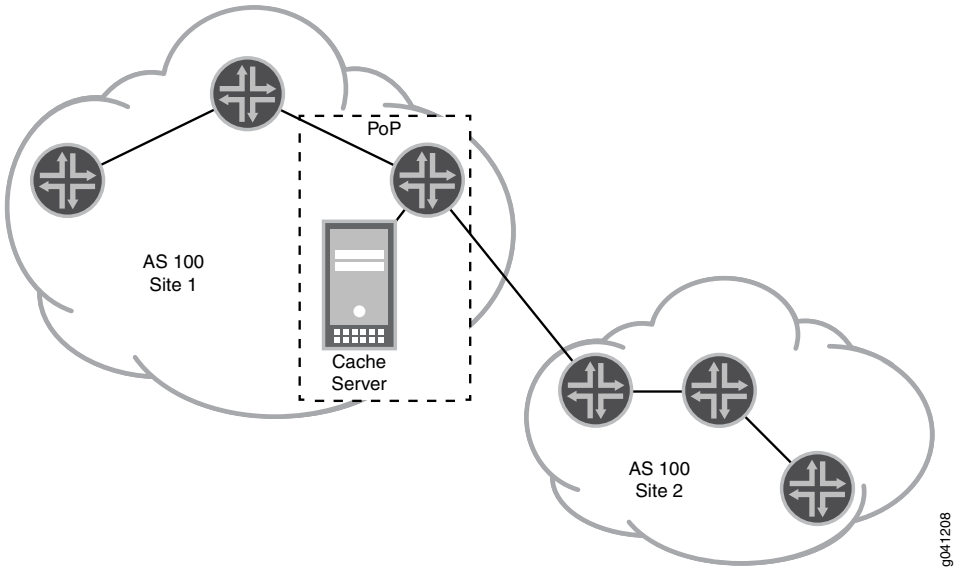
Understanding Origin Validation for BGP

Origin validation helps to prevent the unintentional advertisement of routes. Sometimes network administrators mistakenly advertise routes to networks that they do not control. You can resolve this security issue by configuring origin validation (also known as secure interdomain routing). Origin validation is a mechanism by which route advertisements can be authenticated as originating from an expected autonomous system (AS). Origin validation uses one or more resource public key infrastructure (RPKI) cache servers to perform authentication for specified BGP prefixes. To authenticate a prefix, the router (BGP speaker) queries the database of validated prefix-to-AS mappings, which are downloaded from the cache server, and ensures that the prefix originated from an expected AS.

Junos OS supports origin validation for IPv4 and IPv6 prefixes.

Figure 1 on page 6 shows a sample topology.

Figure 1: Sample Topology for Origin Validation



Supported Standards

The Junos OS implementation of origin validation supports the following RFCs and draft:

- RFC 6810, *The Resource Public Key Infrastructure (RPKI) to Router Protocol*
- RFC 6811, *BGP Prefix Origin Validation*
- Internet draft draft-ietf-sidr-origin-validation-signaling-00, *BGP Prefix Origin Validation State Extended Community* (partial support)

  The extended community (origin validation state) is supported in Junos OS routing policy. The specified change in the route selection procedure is not supported.

How Origin Validation Works

The RPKI and origin validation use X.509 certificates with extensions specified in RFC 3779, *X.509 Extensions for IP Addresses and AS Identifiers*.

The RPKI consists of a distributed collection of information. Each Certification Authority publishes its end-entity (EE) certificates, certificate revocation lists (CRLs), and signed objects at a particular location. All of these repositories form a complete set of information that is available to every RPKI cache server.

Each RPKI cache server maintains a local cache of the entire distributed repository collection by regularly synchronizing each element in the local cache against the original repository publication point.

On the router, the database entries are formatted as route validation (RV) records. An RV record is a (prefix, maximum length, origin AS) triple. It matches any route whose prefix matches the RV prefix, whose prefix length does not exceed the maximum length given in the RV record, and whose origin AS equals the origin AS given in the RV record.

An RV record is a simplified version of a route origin authorization (ROA). An ROA is a digitally signed object that provides a means of verifying that an IP address block holder has authorized an AS to originate routes to one or more prefixes within the address block. ROAs are not directly used in route validation. The cache server exports a simplified version of the ROA to the router as an RV record.

The maximum length value must be greater than or equal to the length of the authorized prefix and less than or equal to the length (in bits) of an IP address in the address family (32 for IPv4 and 128 for IPv6). The maximum length defines the IP address prefix that the AS is authorized to advertise.

For example, if the IP address prefix is 200.4.66/24, and the maximum length is 26, the AS is authorized to advertise 200.4.66.0/24, 200.4.66.0/25, 200.4.66.128/25, 200.4.66.0/26, 200.4.66.64/26, 200.4.66.128/26, and 200.4.66.192/26. When the maximum length is not present, the AS is only authorized to advertise exactly the prefix specified in the RV.

As another example, an RV can contain the prefix 200.4.66/24 with a maximum length of 26, as well as the prefix 200.4.66.0/28 with a maximum length of 28. This RV would authorize the AS to advertise any prefix beginning with 200.4.66 with a length of at least 24 and no greater than 26, as well as the specific prefix 200.4.66.0/28.

The origin of a route is represented by the right-most AS number in the AS_PATH attribute. Origin validation operates by comparing the origin AS in a routing update with the authorized source AS published in RV records.

The security provided by origin validation alone is known to be weak against a determined attacker because there is no protection against such an attacker spoofing the source AS. That said, origin validation provides useful protection against accidental announcements.

Although origin validation could be implemented by having each router directly participate in the RPKI, this is seen as too resource intensive (because many public-key cryptography operations are required to validate the RPKI data) as well as operationally intensive to set up and maintain an RPKI configuration on each router. For this reason, a separate RPKI cache server performs public-key validations, and generates a validated database of prefix-to-AS mappings. The validated database is downloaded to a client router over a secure TCP connection. The router thus requires little information about the RPKI infrastructure and has no public-key cryptography requirements, other than the encrypted transport password. The router subsequently uses the downloaded data to validate received route updates.

When you configure server sessions, you can group the sessions together and configure session parameters for each session in the group. The router tries periodically to set up a configurable maximum number of connections to cache servers. If connection setup fails, a new connection attempt is made periodically.

In the meantime, after the validation import policy is applied to the BGP session, route-validation is performed irrespective of cache session state (up or down) and RV database (empty or not empty). If the RV database is empty or none of the cache server sessions are up, the validation state for each route is set to unknown, because no RV record exists to evaluate a received BGP prefix.

The retry-attempt period is configurable. After successfully connecting to a cache server, the router queries for the latest database serial number and requests that the RPKI cache transmits all of the RV entries belonging to that version of the database.

Each inbound message resets a liveliness timer for the RPKI cache server. After all updates are learned, the router performs periodic liveliness checks based on a configurable interval. This is done by sending a serial query protocol data unit (PDU) with the same serial number that the cache server reported in its latest notification PDU. The cache server responds with zero or more updates and an end-of-data (EOD) PDU, which also refreshes the liveliness state of the cache server and resets a record-lifetime timer.

When a prefix is received from an external BGP (EBGP) peer, it is examined by an import policy and marked as Valid, Invalid, or Unknown:

- Valid—Indicates that the prefix and AS pair are found in the database.
- Invalid—Indicates that the prefix is found, but either the corresponding AS received from the EBGP peer is not the AS that appears in the database, or the prefix length in the BGP update message is longer than the maximum length permitted in the database.
- Unknown—Indicates that the prefix is not among the prefixes or prefix ranges in the database.

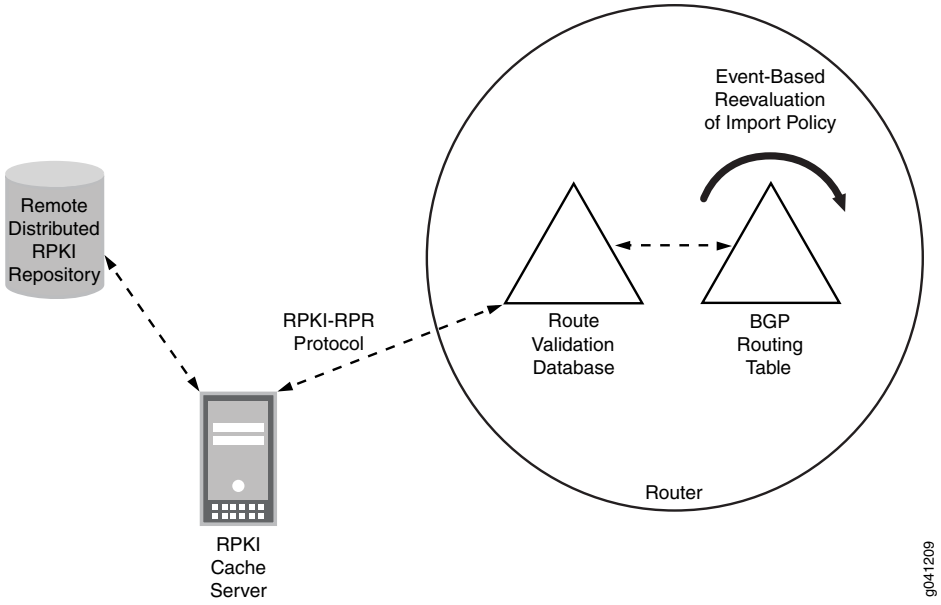
If there are any potential matches for the route in the validation database, the route has to match one of them to be valid. Otherwise, it is invalid. Any match is adequate to make the route valid. It does not need to be a best match. Only if there are no potential matches is the route considered to be unknown. For more information about the prefix-to-AS mapping database logic, see Section 2 of Internet draft draft-ietf-sidr-pfx-validate-01, *BGP Prefix Origin Validation*.

BGP Interaction with the Route Validation Database

The route validation (RV) database contains a collection of RV records that the router downloads from the RPKI cache server. After the RV database is populated with RV records, the RV database scans the RIB-Local routing table to determine if there are any prefixes in RIB-Local that might be affected by the RV records in the database. (RIB-Local contains the IPv4 and IPv6 routes shown in the output of the **show route protocol bgp** command.)

This process triggers a BGP reevaluation of BGP import policies (not export policies).

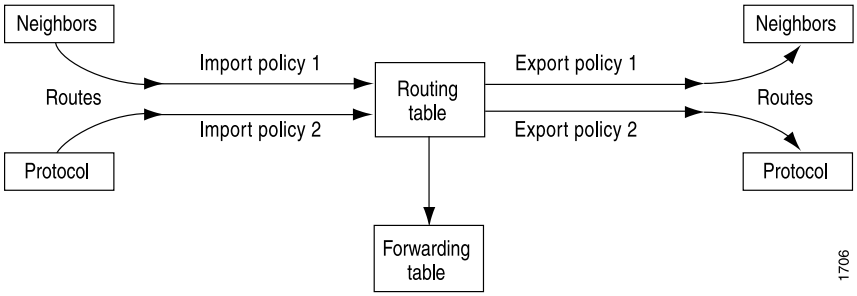
Figure 2 on page 9 shows the process.



Import policies are applied to RIB-In. Another way to understand this is that Import policies are applied to the routes that are shown in the output of the **show route receive-protocol bgp** command, while export policies are applied to routes that are shown by the **show route advertising-protocol bgp** command.

As shown in Figure 3 on page 9, you use import routing policies to control which routes BGP places in the routing table, and export routing policies to control which routes BGP advertises from the routing table to its neighbors.

Figure 3: Importing and Exporting Routing Policies



When you configure a route-validation import policy, the policy configuration uses a **validation-database** match condition. This match condition triggers a query in the RV database for the validation state of a prefix in a given routing instance. The default operation is to query the validation database matching the routing instance. If no route validation instance is found, the master instance is queried.

In the following BGP import policy, the **from validation-database** condition triggers a lookup in the router's RV database. An action is taken if the validation state is valid. The action is to accept the route and set the **validation-state** in the routing table to valid.

```
[edit protocols bgp]
import validation;

[edit policy-options]
policy-statement validation-1 {
  term valid {
    from {
      protocol bgp;
      validation-database valid;
    }
    then {
      validation-state valid;
      accept;
    }
  }
}
```

Community Attribute to Announce RPKI Validation State to IBGP Neighbors

Prefix validation is done only for external BGP (EBGP) updates. Within an AS, you likely do not want to have an RPKI session running on every internal BGP (IBGP) router. Instead, you need a way to carry the validation state across the IBGP mesh so that all IBGP speakers have consistent information. This is accomplished by carrying the validation state in a non-transitive extended community. The community attribute announces and receives the validation state of a prefix between IBGP neighbors.

Junos OS supports the following well-known extended communities for route validation:

- origin-validation-state-valid
- origin-validation-state-invalid
- origin-validation-state-unknown

The following sample BGP import policy is configured on the router that has a session with an RPKI server.

Router With RPKI Session

```
policy-statement validation-1 {
  term valid {
    from {
      protocol bgp;
      validation-database valid;
    }
    then {
      validation-state valid;
      community add origin-validation-state-valid;
      accept;
    }
  }
}
```

The following sample BGP import policy is configured on an IBGP peer router that does not have a session with an RPKI server.

IBGP Peer Router Without RPKI Session

```
policy-statement validation-2 {
  term valid {
    from community origin-validation-state-valid;
```

```
        then validation-state valid;
    }
}
```

Nonstop Active Routing and Origin Validation

When you configure origin validation on a router that has dual Routing Engines and nonstop active routing is enabled, both the master and the standby Routing Engines have a copy of the RV database. These two RV databases remain synchronized with each other.

The router does not maintain two identical sessions with the RPKI server. The RPKI-RTR protocol runs on the master Routing Engine only. On the standby Routing Engine, the RPKI cache server session is always down.

The RV database is actively maintained by the master Routing Engine through its session with the RPKI server. This database is replicated on the standby Routing Engine. Though the session is down on the standby Routing Engine , the replicated RV database does contain RV records. When the standby Routing Engine switches over and becomes the master Routing Engine, it already has a fully populated RV database.

To view the contents of the two databases, use the **show validation database** and **show validation replication database** commands.

Marking a Prefix Range as Never Allowed

The route validation model has one major shortcoming: It only provides positive updates. It can declare which AS is the legitimate owner of a prefix. However, it cannot explicitly convey a negative update, as in: This prefix is never originated by a given AS. This functionality can be provided to some extent using an AS 0 workaround.

The Junos OS implementation does not attempt to restrict its inputs from the cache. For example, an RV record with origin AS 0 is installed and matched upon just like any other. This enables a workaround to mark a prefix range as never allowed to be announced because AS 0 is not a valid AS. The AS in the RV record never matches the AS received from the EBGP peer. Thus, any matching prefix is marked invalid.

Related Documentation

- Example: Configuring Origin Validation for BGP on page 12
- Use Case and Benefit of Origin Validation on page 11

Use Case and Benefit of Origin Validation

If an administrator of an autonomous system (AS) begins advertising all or part of another company's assigned network, BGP has no built-in method to recognize the error and respond in a way that would avoid service interruptions.

Suppose, for example, that an administrator in a customer network mistakenly advertises a route (let's say 10.65.153.0/24) directing traffic to the customer's service provider AS 1. This /24 route is a more specific route than the one used by the actual content provider

(10.65.152.0/22) which directs traffic to AS 2. Because of the way routers work, most routers select the more specific route and send traffic to AS 1 instead of AS 2.

The hijacked prefix is seen widely across the Internet as transit routers propagate the updated path information. The invalid routes can be distributed broadly across the Internet as the routers in the default free zone (DFZ) carry the hijacked route. Eventually the correct AS path is restored to BGP peers, but in the meantime service interruptions are to be expected.

Because BGP relies on a transitive trust model, validation between customer and provider is important. In the example above, the service provider AS 1 did not validate the faulty advertisement for 10.65.153.0/24. By accepting this advertisement and readvertising it to its peers and providers, AS 1 was propagating the wrong route. The routers that received this route from AS 1 selected it because it was a more specific route. The actual content provider was advertising 10.65.152.0/22 before the mistake occurred. The /24 was a smaller (and more specific) advertisement. According to the usual BGP route selection process, the /24 was then chosen, effectively completing the hijack.

Even with fast detection and reaction of the content provider and cooperation with other providers, service for their prefix can be interrupted for many minutes up to several hours. The exact duration of the outage depends on your vantage point on the Internet. When these sorts of events occur, there is renewed interest in solutions to this vulnerability. BGP is fundamental to provider relationships and will not be going away anytime soon. This example demonstrates a solution that uses origin validation. This solution relies on cryptographic extensions to BGP and a distributed client-server model that avoids overtaxing router CPUs.

Origin validation helps to overcome the vulnerability of transitive trust by enabling a provider to limit the advertisements it accepts from a customer. The mechanics involve the communication of routing policies based on an extended BGP community attribute.

Related Documentation

- Understanding Origin Validation for BGP on page 5
- Example: Configuring Origin Validation for BGP on page 12

Example: Configuring Origin Validation for BGP

This example shows how to configure origin validation between BGP peers by ensuring that received route advertisements are sent (originated) from the expected autonomous system (AS). If the origin AS is validated, a policy can specify that the prefixes are, in turn, advertised.

Requirements

This example has the following hardware and software requirements:

- Resource public key infrastructure (RPKI) cache server, using third-party software to authenticate BGP prefixes.
- Junos OS Release 12.2 or later running on the routing device that communicates with the cache server over a TCP connection.

NOTE: This configuration example has been tested using the software release listed and is assumed to work on all later releases.

Overview

Sometimes routes are unintentionally advertised due to operator error. To prevent this security issue, you can configure BGP to validate the originating AS. This feature uses a cache server to authenticate prefixes or prefix ranges.

The following configuration statements enable origin AS validation:

```
[edit routing-options]
validation {
  group group-name {
    max-sessions number;
    session address {
      hold-time seconds;
      local-address local-ip-address;
      port port-number;
      preference number;
      record-lifetime seconds;
      refresh-time seconds;
    }
  }
  static {
    record destination {
      maximum-length prefix-length {
        origin-autonomous-system as-number {
          validation-state (invalid | valid);
        }
      }
    }
  }
  traceoptions {
    file filename <files number> <size size> <world-readable | no-world-readable>;
    flag flag;
  }
}
```

This example uses default settings for the validation parameters.

Most of the available configuration statements are optional. The required settings are as follows:

```
validation {
  group group-name {
    session address {
    }
  }
}
```

The **[edit routing-options validation static]** hierarchy level enables you to configure static records on a routing device, thus overwriting records received from an RPKI cache server.

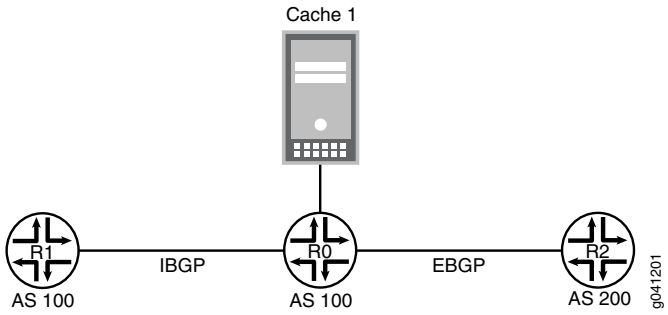
For example:

```
[edit routing-options validation]
user@R0# set static record 10.0.0.0/16 maximum-length 24 origin-autonomous-system
  200 validation-state valid
```

You can configure a routing policy that operates based on the validation state of a route prefix. You can use a community attribute to announce and receive the validation state of a prefix between external BGP (EBGP) and internal BGP (IBGP) peers. Using a routing policy might be more convenient on some routers than configuring a session with an RPKI server. This example demonstrates the use of the validation-state community attribute between IBGP peers.

Figure 4 on page 14 shows the sample topology.

Figure 4: Topology for Origin Validation



In this example, Device R0 has an IBGP connection to Device R1 and an EBGP connection to Device R2. Device R0 receives route validation (RV) records from the cache server using the protocol defined in Internet draft draft-ietf-sidr-rpki-rtr-19, *The RPKI/Router Protocol* to send the RV records. The RPKI-Router Protocol runs over TCP. The RV records are used by Device R0 to build a local RV database. On Device R1, the validation state is set based on the BGP community called validation-state, which is received with the route.

Configuration

- Configuring Device R0 on page 16
- Configuring Device R1 on page 20
- Configuring Device R2 on page 22

CLI Quick Configuration

To quickly configure this example, copy the following commands, paste them into a text file, remove any line breaks, change any details necessary to match your network configuration, and then copy and paste the commands into the CLI at the **[edit]** hierarchy level.

Device R0

```
set interfaces ge-1/2/0 unit 2 description to-R1
set interfaces ge-1/2/0 unit 2 family inet address 10.0.0.2/30
set interfaces ge-1/2/1 unit 5 description to-R2
set interfaces ge-1/2/1 unit 5 family inet address 10.0.0.5/30
set interfaces ge-1/2/2 unit 9 description to-cache
set interfaces ge-1/2/2 unit 9 family inet address 10.0.0.9/30
set interfaces lo0 unit 1 family inet address 1.0.1.1/32
set protocols bgp group int type internal
set protocols bgp group int local-address 1.0.1.1
set protocols bgp group int export send-direct
set protocols bgp group int neighbor 1.1.1.1
set protocols bgp group ext type external
set protocols bgp group ext import validation
set protocols bgp group ext export send-direct
set protocols bgp group ext peer-as 200
set protocols bgp group ext neighbor 10.0.0.6
set protocols ospf area 0.0.0.0 interface ge-1/2/0.2
set protocols ospf area 0.0.0.0 interface lo0.1 passive
set policy-options policy-statement send-direct from protocol direct
set policy-options policy-statement send-direct then accept
set policy-options policy-statement validation term valid from protocol bgp
set policy-options policy-statement validation term valid from validation-database valid
set policy-options policy-statement validation term valid then local-preference 110
set policy-options policy-statement validation term valid then validation-state valid
set policy-options policy-statement validation term valid then community add
  origin-validation-state-valid
set policy-options policy-statement validation term valid then accept
set policy-options policy-statement validation term invalid from protocol bgp
set policy-options policy-statement validation term invalid from validation-database
  invalid
set policy-options policy-statement validation term invalid then local-preference 90
set policy-options policy-statement validation term invalid then validation-state invalid
set policy-options policy-statement validation term invalid then community add
  origin-validation-state-invalid
set policy-options policy-statement validation term invalid then accept
set policy-options policy-statement validation term unknown from protocol bgp
set policy-options policy-statement validation term unknown then validation-state
  unknown
set policy-options policy-statement validation term unknown then community add
  origin-validation-state-unknown
set policy-options policy-statement validation term unknown then accept
set policy-options community origin-validation-state-invalid members 0x43:100:2
set policy-options community origin-validation-state-unknown members 0x43:100:1
set policy-options community origin-validation-state-valid members 0x43:100:0
set routing-options autonomous-system 100
set routing-options validation group test session 10.0.0.10
```

Device R1

```
set interfaces ge-1/2/0 unit 1 family inet address 10.0.0.1/30
set interfaces lo0 unit 2 family inet address 1.1.1.1/32
set protocols bgp group int type internal
```

```
set protocols bgp group int local-address 1.1.1.1
set protocols bgp group int import validation-ibgp
set protocols bgp group int neighbor 1.0.1.1
set protocols ospf area 0.0.0.0 interface ge-1/2/0.1
set protocols ospf area 0.0.0.0 interface lo0.2 passive
set policy-options policy-statement validation-ibgp term valid from community
  origin-validation-state-valid
set policy-options policy-statement validation-ibgp term valid then validation-state valid
set policy-options policy-statement validation-ibgp term invalid from community
  origin-validation-state-invalid
set policy-options policy-statement validation-ibgp term invalid then validation-state
  invalid
set policy-options policy-statement validation-ibgp term unknown from community
  origin-validation-state-unknown
set policy-options policy-statement validation-ibgp term unknown then validation-state
  unknown
set policy-options community origin-validation-state-invalid members 0x43:100:2
set policy-options community origin-validation-state-unknown members 0x43:100:1
set policy-options community origin-validation-state-valid members 0x43:100:0
set routing-options autonomous-system 100
```

Device R2

```
set interfaces ge-1/2/0 unit 6 family inet address 10.0.0.6/30
set interfaces lo0 unit 5 family inet address 172.16.1.1/32
set interfaces lo0 unit 5 family inet address 192.168.2.3/32
set interfaces lo0 unit 5 family inet address 2.2.0.2/32
set protocols bgp group ext export send-direct
set protocols bgp group ext peer-as 100
set protocols bgp group ext neighbor 10.0.0.5
set policy-options policy-statement send-direct from protocol direct
set policy-options policy-statement send-direct from protocol local
set policy-options policy-statement send-direct then accept
set routing-options autonomous-system 200
```

Configuring Device R0

Step-by-Step Procedure

The following example requires you to navigate various levels in the configuration hierarchy. For information about navigating the CLI, see *Using the CLI Editor in Configuration Mode* in the *CLI User Guide*.

To configure Device R0:

1. Configure the interfaces.

```
[edit interfaces]
user@R0# set ge-1/2/0 unit 2 description to-R1
user@R0# set ge-1/2/0 unit 2 family inet address 10.0.0.2/30

user@R0# set ge-1/2/1 unit 5 description to-R2
user@R0# set ge-1/2/1 unit 5 family inet address 10.0.0.5/30

user@R0# set ge-1/2/2 unit 9 description to-cache
user@R0# set ge-1/2/2 unit 9 family inet address 10.0.0.9/30

user@R0# set lo0 unit 1 family inet address 1.0.1.1/32
```

2. Configure BGP.

   Apply the **send-direct** export policy so that direct routes are exported from the routing table into BGP.

   Apply the **validation** import policy to set the validation-state and BGP community attributes for all the routes imported (or received) from Device R0's EBGP peers.

   Configure an IBGP session with Device R1. Configure an EBGP session with Device R2.

   ```
   [edit protocols bgp]
   user@R0# set group int type internal
   user@R0# set group int local-address 1.0.1.1
   user@R0# set group int export send-direct
   user@R0# set group int neighbor 1.1.1.1

   user@R0# set group ext type external
   user@R0# set group ext import validation
   user@R0# set group ext export send-direct
   user@R0# set group ext peer-as 200
   user@R0# set group ext neighbor 10.0.0.6
   ```

3. Configure OSPF (or another interior gateway protocol [IGP]) on the interface that faces the IBGP peer and on the loopback interface.

   NOTE: If you use the loopback interface address in the IBGP neighbor statement, you must enable an IGP on the loopback interface. Otherwise, the IBGP session is not established.

   ```
   [edit protocols ospf area 0.0.0.0]
   user@R0# set interface ge-1/2/0.2
   user@R0# set interface lo0.1 passive
   ```

4. Configure the routing policy that exports direct routes from the routing table into BGP.

   ```
   [edit policy-options policy-statement send-direct]
   user@R0# set from protocol direct
   user@R0# set then accept
   ```

5. Configure the routing policy that specifies attributes to be modified based on the validation state of each BGP route.

   ```
   [edit policy-options policy-statement validation]
   user@R0# set term valid from protocol bgp
   user@R0# set term valid from validation-database valid
   user@R0# set term valid then local-preference 110
   user@R0# set term valid then validation-state valid
   user@R0# set term valid then community add origin-validation-state-valid
   user@R0# set term valid then accept

   user@R0# set term invalid from protocol bgp
   user@R0# set term invalid from validation-database invalid
   ```

```
user@R0# set term invalid then local-preference 90
user@R0# set term invalid then validation-state invalid
user@R0# set term invalid then community add origin-validation-state-invalid
user@R0# set term invalid then accept

user@R0# set term unknown from protocol bgp
user@R0# set term unknown then validation-state unknown
user@R0# set term unknown then community add origin-validation-state-unknown
user@R0# set term unknown then accept

[edit policy-options]
user@R0# set community origin-validation-state-invalid members 0x43:100:2
user@R0# set community origin-validation-state-unknown members 0x43:100:1
user@R0# set community origin-validation-state-valid members 0x43:100:0
```

6. Configure the session with the RPKI cache server.

```
[edit routing-options validation]
user@R0# set group test session 10.0.0.10
```

7. Configure the autonomous system (AS) number.

```
[edit routing-options]
user@R0# set autonomous-system 100
```

Results From configuration mode, confirm your configuration by entering the **show interfaces**, **show protocols**, **show policy-options**, and **show routing-options** commands. If the output does not display the intended configuration, repeat the instructions in this example to correct the configuration.

```
user@R0# show interfaces
ge-1/2/0 {
  unit 2 {
    description to-R1;
    family inet {
      address 10.0.0.2/30;
    }
  }
}
ge-1/2/1 {
  unit 5 {
    description to-R2;
    family inet {
      address 10.0.0.5/30;
    }
  }
}
ge-1/2/2 {
  unit 9 {
    description to-cache;
    family inet {
      address 10.0.0.9/30;
    }
  }
}
```

```
lo0 {
  unit 1 {
    family inet {
      address 1.0.1.1/32;
    }
  }
}
```

```
user@R0# show protocols
bgp {
  group int {
    type internal;
    local-address 1.0.1.1;
    export send-direct;
    neighbor 1.1.1.1;
  }
  group ext {
    type external;
    import validation;
    export send-direct;
    peer-as 200;
    neighbor 10.0.0.6;
  }
}
ospf {
  area 0.0.0.0 {
    interface ge-1/2/0.2;
    interface lo0.1 {
      passive;
    }
  }
}
```

```
user@R0# show policy-options
policy-statement send-direct {
  from protocol direct;
  then accept;
}
  policy-statement validation {
    term valid {
      from {
        protocol bgp;
        validation-database valid;
      }
      then {
        local-preference 110;
        validation-state valid;
        community add origin-validation-state-valid;
        accept;
      }
    }
    term invalid {
      from {
        protocol bgp;
        validation-database invalid;
      }
      then {
```

```
            local-preference 90;
            validation-state invalid;
            community add origin-validation-state-invalid;
            accept;
          }
        }
        term unknown {
          from protocol bgp;
          then {
            validation-state unknown;
            community add origin-validation-state-unknown;
            accept;
          }
        }
      }
    community origin-validation-state-invalid members 0x43:100:2;
    community origin-validation-state-unknown members 0x43:100:1;
    community origin-validation-state-valid members 0x43:100:0;

    user@R0# show routing-options
    autonomous-system 100;
    validation {
      group test {
        session 10.0.0.10;
      }
    }
```

If you are done configuring the device, enter **commit** from configuration mode.

Configuring Device R1

Step-by-Step Procedure

The following example requires you to navigate various levels in the configuration hierarchy. For information about navigating the CLI, see *Using the CLI Editor in Configuration Mode* in the *CLI User Guide*.

To configure Device R1:

1. Configure the interfaces.

   ```
   [edit interfaces]
   user@R1# set ge-1/2/0 unit 1 family inet address 10.0.0.1/30


   user@R1# set lo0 unit 2 family inet address 1.1.1.1/32
   ```

2. Configure BGP.

   Apply the **validation-ibgp** import policy to set the validation-state and BGP community attributes for all the routes received from Device R1's IBGP peers.

   Configure an IBGP session with Device R0.

   ```
   [edit protocols bgp group int]
   user@R1# set type internal
   user@R1# set local-address 1.1.1.1
   user@R1# set import validation-ibgp
   user@R1# set neighbor 1.0.1.1
   ```

3. Configure OSPF.

```
[edit protocols ospf area 0.0.0.0]
user@R1# set interface ge-1/2/0.1
user@R1# set interface lo0.2 passive
```

4. Configure the routing policy that specifies attributes to be modified based on the validation-state BGP community attribute of the BGP routes received from Device R0.

```
[edit policy-options policy-statement validation-ibgp]
user@R1# set term valid from community origin-validation-state-valid
user@R1# set term valid then validation-state valid

user@R1# set term invalid from community origin-validation-state-invalid
user@R1# set term invalid then validation-state invalid

user@R1# set term unknown from community origin-validation-state-unknown
user@R1# set term unknown then validation-state unknown

[edit policy-options]
user@R1# set community origin-validation-state-invalid members 0x43:100:2
user@R1# set community origin-validation-state-unknown members 0x43:100:1
user@R1# set community origin-validation-state-valid members 0x43:100:0
```

5. Configure the autonomous system (AS) number.

```
[edit routing-options]
user@R1# set autonomous-system 100
```

Results From configuration mode, confirm your configuration by entering the **show interfaces**, **show protocols**, **show policy-options**, and **show routing-options** commands. If the output does not display the intended configuration, repeat the instructions in this example to correct the configuration.

```
user@R1# show interfaces
ge-1/2/0 {
    unit 1 {
        family inet {
            address 10.0.0.1/30;
        }
    }
}
lo0 {
    unit 2 {
        family inet {
            address 1.1.1.1/32;
        }
    }
}

user@R1# show protocols
bgp {
    group int {
        type internal;
        local-address 1.1.1.1;
```

```
        import validation-ibgp;
        neighbor 1.0.1.1;
    }
}
ospf {
    area 0.0.0.0 {
        interface ge-1/2/0.1;
        interface lo0.2 {
            passive;
        }
    }
}
```

```
user@R1# show policy-options
policy-statement validation-ibgp {
    term valid {
        from community origin-validation-state-valid;
        then validation-state valid;
    }
    term invalid {
        from community origin-validation-state-invalid;
        then validation-state invalid;
    }
    term unknown {
        from community origin-validation-state-unknown;
        then validation-state unknown;
    }
}
community origin-validation-state-invalid members 0x43:100:2;
community origin-validation-state-unknown members 0x43:100:1;
community origin-validation-state-valid members 0x43:100:0;
}
```

```
user@R1# show routing-options
autonomous-system 100;
```

If you are done configuring the device, enter **commit** from configuration mode.

Configuring Device R2

Step-by-Step Procedure

The following example requires you to navigate various levels in the configuration hierarchy. For information about navigating the CLI, see *Using the CLI Editor in Configuration Mode* in the *CLI User Guide*.

To configure Device R2:

1. Configure the interfaces.

   Several addresses are configured on the loopback interface to serve as routes for demonstration purposes.

```
[edit interfaces]
user@R2# set ge-1/2/0 unit 6 family inet address 10.0.0.6/30

user@R2# set lo0 unit 5 family inet address 172.16.1.1/32
user@R2# set lo0 unit 5 family inet address 192.168.2.3/32
```

```
user@R2# set lo0 unit 5 family inet address 2.2.0.2/32
```

2. Configure BGP.

```
[edit protocols bgp]
user@R2# set group ext export send-direct
user@R2# set group ext peer-as 100
user@R2# set group ext neighbor 10.0.0.5
```

3. Configure the routing policy.

```
[edit policy-options policy-statement send-direct]
user@R2# set from protocol direct
user@R2# set from protocol local
user@R2# set then accept
```

4. Configure the autonomous system (AS) number.

```
[edit routing-options]
user@R2# set autonomous-system 200
```

Results From configuration mode, confirm your configuration by entering the **show interfaces**, **show protocols**, **show policy-options**, and **show routing-options** commands. If the output does not display the intended configuration, repeat the instructions in this example to correct the configuration.

```
user@R2# show interfaces
ge-1/2/0 {
    unit 6 {
        family inet {
            address 10.0.0.6/30;
        }
    }
}
lo0 {
    unit 5 {
        family inet {
            address 172.16.1.1/32;
            address 192.168.2.3/32;
            address 2.2.0.2/32;
        }
    }
}
```

```
user@R2# show protocols
bgp {
    group ext {
        export send-direct;
        peer-as 100;
        neighbor 10.0.0.5;
    }
}
```

```
user@R2# show policy-options
policy-statement send-direct {
    from protocol [ direct local ];
    then accept;
}
```

```
user@R2# show routing-options
autonomous-system 200;
```

If you are done configuring the device, enter **commit** from configuration mode.

Verification

Confirm that the configuration is working properly.

Verifying That the Modified Attributes Are Displayed in the Routing Tables

Purpose Verify that the BGP routes on Device R0 and Device R1 have the expected validation states and the expected local preferences.

Action From operational mode, enter the **show route** command.

```
user@R0> show route
inet.0: 12 destinations, 13 routes (12 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

1.0.1.1/32          *[Direct/0] 04:53:39
                     >    via lo0.1
1.1.1.1/32          *[OSPF/10] 04:50:53, metric 1
                     > to 10.0.0.1 via lt-1/2/0.2
2.2.0.2/32          *[BGP/170] 01:30:37, localpref 110
                       AS path: 200 I, validation-state: valid
                     > to 10.0.0.6 via lt-1/2/0.5
10.0.0.0/30         *[Direct/0] 04:51:44
                     >    via lt-1/2/0.2
10.0.0.2/32         *[Local/0] 04:51:45
                          Local via lt-1/2/0.2
10.0.0.4/30         *[Direct/0] 04:51:44
                     >    via lt-1/2/0.5
                     [BGP/170] 02:24:57, localpref 110
                       AS path: 200 I, validation-state: valid
                     > to 10.0.0.6 via lt-1/2/0.5
10.0.0.5/32         *[Local/0] 04:51:45
                          Local via lt-1/2/0.5
10.0.0.8/30         *[Direct/0] 03:01:28
                     >    via lt-1/2/0.9
10.0.0.9/32         *[Local/0] 04:51:45
                          Local via lt-1/2/0.9
172.16.1.1/32       *[BGP/170] 02:24:57, localpref 90
                       AS path: 200 I, validation-state: invalid
                     > to 10.0.0.6 via lt-1/2/0.5
192.168.2.3/32      *[BGP/170] 02:24:57, localpref 100
                       AS path: 200 I, validation-state: validation-state: unknown
                     > to 10.0.0.6 via lt-1/2/0.5
224.0.0.5/32        *[OSPF/10] 04:53:46, metric 1
                          MultiRecv

user@R1> show route
```

```
inet.0: 10 destinations, 12 routes (10 active, 0 holddown, 0 hidden)
+ = Active Route, - = Last Active, * = Both

2.2.0.2/32          *[BGP/170] 01:06:58, localpref 110, from 1.0.1.1
                       AS path: 200 I, validation-state: valid
                     > to 10.0.0.2 via lt-1/2/0.1
172.16.1.1/32       *[BGP/170] 00:40:52, localpref 90, from 1.0.1.1
                       AS path: 200 I, validation-state: invalid
                     > to 10.0.0.2 via lt-1/2/0.1
192.168.2.3/32      *[BGP/170] 01:06:58, localpref 100, from 1.0.1.1
                       AS path: 200 I, validation-state: unknown
                     > to 10.0.0.2 via lt-1/2/0.1
224.0.0.5/32        *[OSPF/10] 04:57:09, metric 1
                           MultiRecv
```

Meaning The routes have the expected validation states and local-preference values, based on information received from the RPKI cache server.

Using Trace Operations

Purpose Configure trace operations for origin validation, and monitor the results of a newly advertised route.

Action

- On Device R0, configure tracing.

  ```
  [edit routing-options validation traceoptions]
  user@R0# set file rv-tracing
  user@R0# set flag all

  user@R0# commit
  ```

- On Device R2, add a route by adding another address on the loopback interface.

  ```
  [edit interfaces lo0 unit 5 family inet]
  user@R2# set address 10.4.4.4/32

  user@R2# commit
  ```

- On Device R0, check the trace file.

  ```
  user@R0> file show /var/log/rv-tracing
  Jan 27 11:27:43.804803 rv_get_policy_state: rt 10.4.4.4/32 origin-as 200,
  validation result valid
  Jan 27 11:27:43.944037 task_job_create_background: create prio 7 job
  Route-validation GC for task Route Validation
  Jan 27 11:27:43.986580 background dispatch running job Route-validation GC for
   task Route Validation
  Jan 27 11:27:43.987374 task_job_delete: delete background job Route-validation
   GC for task Route Validation
  Jan 27 11:27:43.987463 background dispatch completed job Route-validation GC
  for task Route Validation
  ```

Meaning Route validation is operating as expected.

Displaying Validation Information

Purpose Run the various validation commands.

Action

```
user@R0> show validation statistics
Total RV records: 3
Total Replication RV records: 3
  Prefix entries: 3
  Origin-AS entries: 3
Memory utilization: 9789 bytes
Policy origin-validation requests: 114
  Valid: 32
  Invalid: 54
  Unknown: 28
BGP import policy reevaluation notifications: 156
  inet.0, 156
  inet6.0, 0

user@R0> show validation database
RV database for instance master

Prefix                Origin-AS Session                              State
 Mismatch
2.0.0.0/8-32                200 10.0.0.10                           valid
10.0.0.0/8-32               200 10.0.0.10                           valid
172.0.0.0/8-12              200 10.0.0.10                           invalid


  IPv4 records: 3
  IPv6 records: 0

user@R0> show validation replication database
RRV replication database for instance master

Prefix                Origin-AS Session                              State
2.0.0.0/8-32                200 10.0.0.10                           valid
10.0.0.0/8-32               200 10.0.0.10                           valid
172.0.0.0/8-12              200 10.0.0.10                           invalid


  IPv4 records: 3
  IPv6 records: 0

user@R0> show validation group
master
  Group: test, Maximum sessions: 2
    Session 10.0.0.10, State: Connect, Preference: 100

user@R0> show validation session
Session                                 State   Flaps     Uptime #IPv4/IPv6
records
10.0.0.10                               Up          0   00:02:28 1/0

user@R0> request validation policy
  Enqueued 3 IPv4 records
  Enqueued 0 IPv6 records
```

Related Documentation

- Use Case and Benefit of Origin Validation on page 11
- Understanding Origin Validation for BGP on page 5