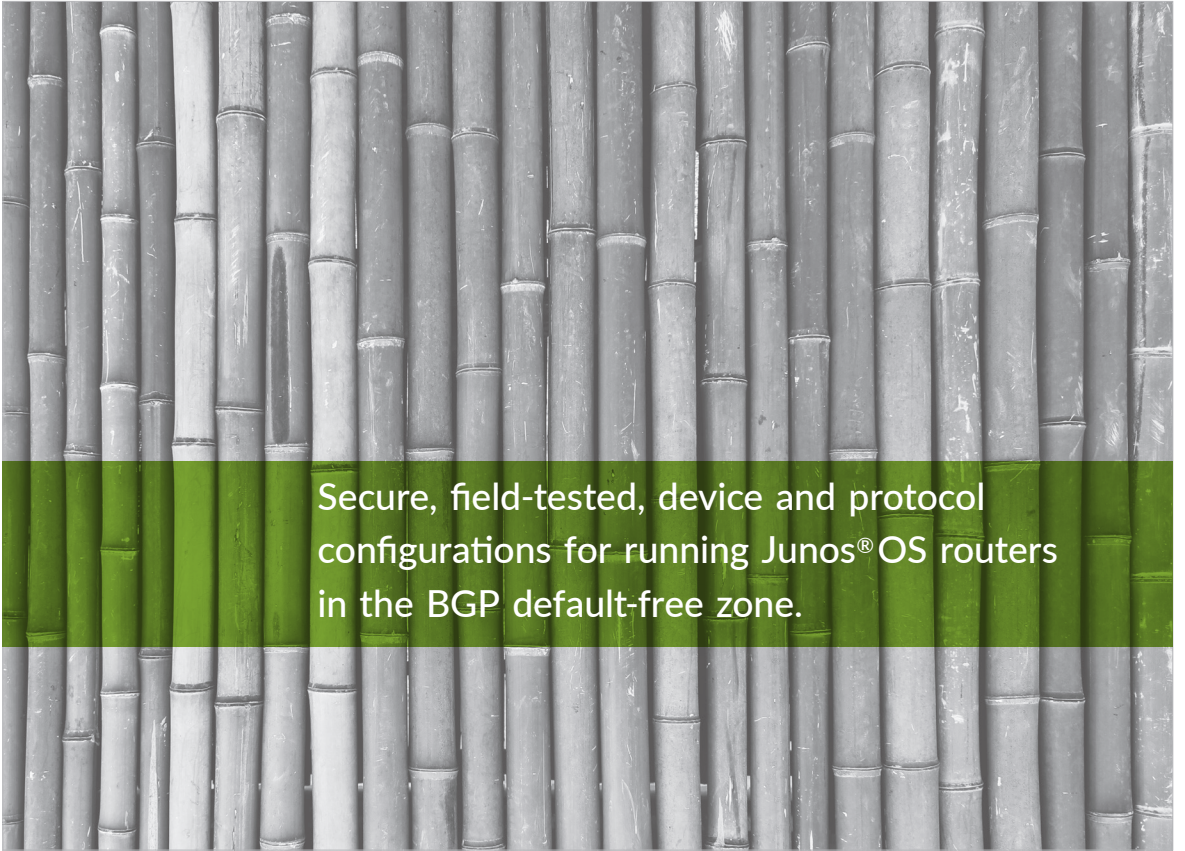


DAY ONE: DEPLOYING BGP ROUTING SECURITY



Secure, field-tested, device and protocol
configurations for running Junos® OS routers
in the BGP default-free zone.

By Melchior Aelmans & Niels Raijer

DAY ONE: DEPLOYING BGP ROUTING SECURITY

This book is intended for network administrators running Junos OS routers in the BGP default-free zone. It provides field-tested device and protocol configurations for creating a secure and stable network, as well as brief background information needed to understand and deploy these solutions in your own environment. While many network administrators may find the contents of this book interesting, its real value is to those running a BGP network without having a default route present in their network (or accepting such a route from their upstream provider) – the default-free zone.

“High-quality Internet services require a global routing table of equally high quality and in this very practical book the authors show how you can improve that global table using the tools available today. It’s easy to read, with detailed tutorials, and even includes copy and paste Junos configuration examples. Whether you have just started working with Internet routing or have done so for many years, this book will show you how to do it better.”

Torunn Narvestad, Senior IP Network Architect, Telenor Norway

“Melchior and Niels have done a fantastic job consolidating a ton of tribal knowledge and disparate information sources into an easy-to-read RPKI Origin Validation deployment guide. This book will help lower the barrier to run a secure and robust network!”

Job Snijders, Internet Architect, NTT Communications

IT'S DAY ONE AND YOU HAVE A JOB TO DO, SO LEARN HOW TO:

- Understand the relevance of filtering routes as you learn them from your customers, peers, and transits.
- Understand what portion of via BGP received routes should be rejected for securing your routing table.
- Implement routing policies that reject invalid routing information.
- Understand and implement redundant Resource Public Key Infrastructure (RPKI) validators.
- Verify your configuration and support your network using basic troubleshooting commands.
- How to use RIR tools to make sure your routes and prefixes are accepted by other ISPs who filter and/or have deployed RPKI.



Juniper Networks Books are focused on network reliability and efficiency. Peruse the complete library at www.juniper.net/books.

JUNIPER
NETWORKS

Day One: Deploying BGP Routing Security

by Melchior Aelmans and Niels Raijer

<i>Foreword by Job Snijders</i>	viii
<i>Chapter 1: Introducing Routing Security</i>	10
<i>Chapter 2: Accepting and Announcing Routes</i>	16
<i>Chapter 3: Configuring RPKI: Resource Public Key Infrastructure</i>	22
<i>Chapter 4: Configuring Routing Policies</i>	31
<i>Chapter 5: Troubleshooting</i>	58
<i>Appendix: How To Automatically Update Prefix Lists</i>	63

© 2019 by Juniper Networks, Inc.

All rights reserved. Juniper Networks and Junos are registered trademarks of Juniper Networks, Inc. in the United States and other countries. The Juniper Networks Logo and the Junos logo, are trademarks of Juniper Networks, Inc. All other trademarks, service marks, registered trademarks, or registered service marks are the property of their respective owners. Juniper Networks assumes no responsibility for any inaccuracies in this document. Juniper Networks reserves the right to change, modify, transfer, or otherwise revise this publication without notice.

Published by Juniper Networks Books

Authors: Melchior Aelmans, Niels Raijer
 Technical Reviewers: Jeff Haas, Colby Barth, Teun Vink,
 Job Snijders, Torunn Narvestad
 Editor in Chief: Patrick Ames
 Copyeditor: Nancy Koerbel

ISBN: 978-1-941441-86-2 (print)

Printed in the USA by Vervante Corporation.

ISBN: 978-1-941441-85-5 (ebook)

Version History: v1, February 2019

2 3 4 5 6 7 8 9 10

<http://www.juniper.net/dayone>

About the Authors

Melchior Aelmans is a Senior Systems Engineer at Juniper Networks, where he has been working with many operators on the design, security, and evolution of their networks. He has over 10 years of experience in various operations, engineering, and sales engineering positions with enterprises, data centers and Service Provider. Before joining Juniper Networks, he worked with eBay, LGL, KPN, etc. Melchior enjoys evangelizing and discussing topics like BGP, peering, routing security, and Internet routing. He also participates in IETF and is a board member of the NLNOG foundation. In his spare time, he enjoys spending time outdoors hiking with his girlfriend and dog and climbing mountains.

Niels Raijer was introduced to e-mail, Gopher and USENET in 1993 as part of his Chemical Engineering education at the University of Amsterdam, and decided that they were what all businesses in the world would need. After graduation he founded Fusix Networks in 1997. Having worked for Demon Internet and other ISPs since then, he is now CTO of Fusix Networks, responsible for providing network consultancy and connectivity services where the keywords are security, stability, and speed. Niels is also the founder of both Coloclue and NLNOG. He is married, has two children, and likes to pretend he is still a fairly decent competitive swimmer.

Authors' Acknowledgments

The authors would like to thank, in random order: Nathalie Trenaman (RIPE NCC); Job Snijders (NTT Communications); Teun Vink (BIT); Torunn Narvestad (Telenor Norway); Jeff Haas, Colby Barth, Kireeti Kompella, Wim Tavernier and Nico Siebelink (Juniper Networks); Chi Ho Kwok and Siebe Schaap (Digibites Technology); Yulia Makhlin (Fusix Networks); Alex Band (NLnet Labs); Klaartje van Leeuwen, and, finally: the NLNOG community, NLnet Labs, the RIPE community, and the RIPE NCC for their valuable contributions, content, input, software, comments, and mental support.

Feedback? Comments? Error reports? Email them to dayone@juniper.net.

Welcome to Day One

This book is part of the *Day One* library, produced and published by Juniper Networks Books.

Day One books cover the Junos OS and Juniper Networks networking essentials with straightforward explanations, step-by-step instructions, and practical examples that are easy to follow. You can obtain the books from various sources:

- Download a free PDF edition at <http://www.juniper.net/dayone>.
- Many of the library's books are available on the Juniper app: [Junos Genius](#).
- Get the ebook edition for iPhones and iPads from the iBooks Store. Search for *Juniper Networks Books* or the title of this book.
- Get the ebook edition for any device that runs the Kindle app (Android, Kindle, iPad, PC, or Mac) by opening your device's Kindle app and going to the Amazon Kindle Store. Search for *Juniper Networks Books* or the title of this book.
- Purchase the paper edition at Vervante Corporation (www.vervante.com) for between \$15-\$40, depending on page length.
- Note that most mobile devices can also view PDF files.

Target Audience

This book is intended for network administrators running BGP on Juniper Networks routers in the default-free zone (DFZ). It provides field-tested device and protocol configurations for creating a secure and stable network, as well as the brief background information needed to understand and deploy these solutions in your own environment. While many network administrators may find the contents of this book interesting, its real value is to those running a BGP network without having a default route present in their network (or accepting such a route from their upstream provider): the DFZ.

IMPORTANT Most techniques described in this book *do not apply* to networks that accept a default route. Why? Well, if you use default routes in your network, most routing security methods like resource public key infrastructure (RPKI — we'll get to that) simply won't work.

What You Need to Know Before Reading This Book

You should be familiar with the basic administrative functions of Junos OS, including the ability to work with operational commands, and to read, understand, and change configurations. There are several books in the *Day One* library on learning Junos, found at <http://www.juniper.net/dayone>.

This book assumes that you, the reader, have intermediate level knowledge of:

- Junos OS and its command-line interface (CLI).
- General BGP protocol usage in Internet service provider (ISP) networks.
- General troubleshooting techniques for ISP networks running the Junos OS.
- The configuration of basic BGP connectivity in the Junos OS, including configuring neighbors and routing policy.
- Basic Junos OS network and system operation.
- Basic (Regional Internet Registry) RIR working knowledge (https://en.wikipedia.org/wiki/Regional_Internet_registry).

What You Will Learn by Reading This Book

This book will help you to:

- Understand the relevance of filtering routes as you learn them from your customers, peers, and transits.
- Understand what portion of BGP-received routes should be rejected for securing your routing table.
- Implement routing policies that reject invalid routing information.
- Understand and implement redundant RPKI validators and use them to filter RPKI invalid routes.
- Verify your configuration and support your network using basic troubleshooting commands.
- Use RIR tools to make sure your routes and prefixes are accepted by other ISPs who filter and/or have deployed RPKI.

Foreword

The Internet is critical to our lives and businesses. The Internet relies on a complex routing eco-system that is made out of the interconnections between our Autonomous Systems (AS). The global routing system is decentralised in nature and technically mostly based on trust. These are two beneficial properties: because of BGP-4's permissionlessness, we can easily interconnect with each other for peering or use multiple transit providers; all the while maintaining a degree of autonomy. However there are downsides too, a misconfiguration in one AS can negatively impact other ASNs! Malicious actors can abuse the trust system.

Perhaps an analogy can be drawn between the rise and fall of open SMTP mail relays and networks without RPKI Origin Validation. Decades ago open mail relays served an important function as they enabled free communication between various parts of the internet. But as the internet grew, so did the appetite for easy-to-exploit spam cannons. Nowadays, every sensible mail operator keeps a tight lid on who can use their mail servers and whom they can reach through those mail servers. It simply has become unattainable to operate in a promiscuous mode. Looking modern day Internet routing challenges, I can see Best Practises for BGP-4 follow a similar evolutionary path.

With that in mind, I'd like to urge everyone to ensure that any routes propagated by their BGP speakers are in fact correct announcements, verified to the best of their abilities. Routing policy statements designed with security in mind are in the best interest of all Internet participants. In other words: as a network's trustworthiness increases, so does its value to the global Internet.

Keep in mind that routing security is not like herd immunity: the benefits of a more secure routing perimeter are immediate and unilateral. There is no requirement for a certain amount of Internet Autonomous Systems to have deployed Origin Validation before the effects positively impact business. When you protect the borders of your administrative domain, you immediately enjoy the benefits and have a competitive advantage compared to those networks who inadvertently accepted a misconfiguration or BGP hijack.

Looking at RPKI Origin Validation and other BGP routing security best practises, it appears the Internet Industry's thinking has shifted from "routing security is a nuisance" to "we have only ourselves to blame if we didn't deploy the bare minimum of BGP filters". Organizations are seeing a measurable positive impact on business operations after developing a stronger routing security posture. Only you can protect your network!

I'd like to thank the authors and Juniper Networks for making *Day One: Deploying BGP Routing Security* available to the operational community. Melchior and Niels have done a fantastic job consolidating a ton of tribal knowledge and disparate information sources into an easy-to-read RPKI Origin Validation deployment guide. This book will help lower the barrier to run a secure and robust network!

Job Snijders, Internet Architect, NTT Communications

February 2019

Chapter 1

Introducing Routing Security

ISPs connect their networks to each other and exchange routing information using the BGP protocol. As the importance of the Internet has grown, the quality and security of Internet routing have become critical.

This *Day One* Book began as a description of how to implement route validation with RPKI, in order to make it easier for you to deploy by providing tested recipes. But once we started writing about RPKI we found that routing policy in ISP networks is far more extensive than RPKI alone. Thus the scope of this book goes beyond RPKI; it can also help you set up routing policies for all routes that you may accept into your network.

NOTE If you are reading this book only because you wish to implement RPKI – and you’re happy with the rest of your routing policies – then you can skip Chapters 2 and 4. To fully secure your routing table, though, please follow through and read the whole book. You may find one or two usable suggestions hidden away in the non-RPKI chapters, too.

Whether it is threatened by distributed denial of service (DDoS) attacks, prefix hijacks, or ‘just’ unintentional route leaks, your network needs to deliver stability and reliable reachability to the Internet. Most routing incidents are unintentional and due to configuration mistakes, and unfortunately, all network engineers suffer from the fat finger syndrome every now and then! You should consider implementing routing security not only to protect your own network, but perhaps, even more so, to help others protect theirs.

Traditionally, firewalls and IDS/IPS systems are deployed to secure the network, acting on traffic that arrives from the Internet. This means the ‘bad traffic’ has already reached your network before you can reject it and the ‘bad guys’ are communicating with your servers before you can stop them. Extra investments may be needed to filter the clean traffic from the bad traffic.

When you leverage routing security, your network cannot fully communicate with ‘bad actors’ in the first place, thus making securely serving your customers easier and more financially sustainable. The sooner you stop the threat, the better. By deploying the correct methods you may not stop these threats from reaching your network, but you can at least prevent your network from reaching an invalid prefix, making it impossible to establish full two-way communication. So, implementing routing security will make it harder for threats to propagate.

This book talks a lot about the configuration of the BGP-speaking routers that form the edge of a DFZ network. They accept routes across BGP sessions with customers, transit providers, and peers. They also announce routes originating from your network and your customers’ networks. A route advertisement consists, among other things, of a combination of prefix, prefix length, origin AS (autonomous system), and AS path. Routing security aims to reduce the number of potentially invalid announcements that your network accepts by actively rejecting invalid route announcements, and by ensuring you only announce the correct prefixes yourself.

This book collects some of the best current practices for deploying routing security. You can greatly improve the stability and security of your network, and perhaps more importantly the stability and quality of the Internet, by using them.

Filtering and Rejecting Routes is Good!

One of the first concerns that comes to mind when thinking about filtering and rejecting BGP routes is: *if I reject a route, surely my network will not be able to reach the whole Internet anymore?! While this could be true, rejecting invalid routes is actually a good thing! Don’t think of it as making part of the Internet unreachable. Instead, consider that leaving out clearly invalid routes from your routing table means that you can make your network immune to (probably unintentional) incorrect route announcements, as well as from (intentional) IP address hijacks.*

Another way to think about it is that many of the invalid routes you will reject are accidental announcements (mostly typos!) and rejecting them actually keeps the intended destination reachable, instead of making the unintended destination unreachable. In this case, rejecting the route actually guarantees reachability! And as for the intentional prefix hijacks – it is obvious that you do not wish to accept these in the first place. Therefore, the only option is to reject invalid routes and make your routing table stable and secure.

As an example, say you received a route for a /16 from your transit provider, and a peer is sending you a single /24 that falls in that /16. If this is a legitimate route advertisement and your policies would accept it, you will send the traffic to that peer instead of to the transit, which may be your intention. However, if the route is a mistake or a hijack, then accepting it means that you actually make the correct /24 unreachable by accepting that route! So, rejecting routes does not automatically mean making parts of the Internet unreachable at all.

When deploying RPKI, many network administrators decide to accept RPKI invalid routes regardless, but with a lower local preference. This does absolutely nothing to make the routing table more secure: if the invalid route is more specific to the correct route it will always win. Such is the nature of BGP. The only way to correctly implement RPKI is by rejecting invalid routes.

Deploying routing security isn't a single checkbox solution. Instead, it is a collection of routing policies, filters, and external sources such as ROA and RPKI validation tools. Routing policies are a method of influencing the default behavior of a routing protocol. In this case you are trying to influence the default behavior of BGP.

Beside the policies, which contain static or semi-static filters, routing security relies on external sources of information, for example those offered by the Regional Internet Registries (RIRs). In cases where RIR tools are mentioned, this book uses RIPE (www.ripe.net) as an example. Obviously, there are other RIRs (https://en.wikipedia.org/wiki/Regional_Internet_registry) who all have more or less the same databases and tools available. ROA and RPKI validation sources are typical examples of external sources that are leveraged in this book.

Internet Routing Registries

Routing information is stored in Internet Routing Registries (IRRs). Each of the RIRs offers their own IRR. In addition, there are third-party IRRs, some are commercial, and others are available free of charge. As you receive an IP address allocation from your RIR, you can create a route object in an IRR in order to 'connect' your IP address allocation to your AS number.

But, you guessed it, there is a problem. Even though the IRRs are pretty reliable, not all of the data they contain is guaranteed to be correct. In some IRRs you can create objects without checks of any kind to make sure the information is correct.

Unfortunately, in many cases the IRRs still provide the best data, even though that data may not be totally correct. In the majority of cases the data is good enough and it is better than nothing. Therefore, we still use IRR data as needed, but will use more reliable information when and if it is available.

Manually filtering a few downstream customers or peers with strict filters may be trivial, but filtering hundreds or thousands of customers whose prefix list changes by a handful of routes twice a week isn't. Especially if you are using IRR sources to build filters, it becomes very time consuming. So we will not touch on manual IRR filtering in the next chapters, but you will find some tips and tricks on how it's done using scripts in the Appendix.

RPKI

There are hundreds of thousands of route announcements on the Internet today and the *magic million* is not far away. Many of these announcements are invalid. The most common routing error seen is the accidental route leak or the *mis-origination* of a prefix, meaning someone unintentionally announces an IP prefix that they are not the holder of, or they advertise a more specific route without a valid routing object in a RIR database. These errors would cause BGP routers to choose a route that is not intended as it does not lead to the network of the rightful owner of the IP space. As a (partial) answer to this problem, RPKI offers BGP origin validation. The question it tries to answer is: “*Is this particular route announcement authorized by the legitimate holder of the address space?*”

It should be noted that RPKI does not validate the entire AS path. It will only tell you if the originating AS is allowed to advertise that specific prefix. As of this writing, there is no good way to validate the entire AS path. Work is being done in IETF to get a mechanism in place to solve this issue, and those drafts can be found here: <https://datatracker.ietf.org/doc/draft-ietf-grow-rpki-as-cones/> and here: <https://tools.ietf.org/html/draft-azimov-sidrops-aspa-verification-01>.

RPKI allows network operators to create statements about the route announcements they authorize that can be cryptographically validated with the prefixes they hold. These statements are called *route origin authorizations* (ROAs). A ROA states which AS is authorized to originate a certain IP address prefix. In addition, it can determine the maximum length of the prefix the AS is authorized to advertise. Based on this information, other network operators can make routing decisions.

Using the RPKI system fully requires action on two parts:

1. The legitimate holder of an IP prefix creates a certificate, or ROA, stating which ASs their prefixes will be advertised (originated) from and the maximum allowed prefix length.

NOTE Only the owner of the IP address allocation can create a ROA for this allocation. This is more reliable than IRR data.

2. Other network operators can set their routing policies based on the RPKI validity of route announcements when comparing them to the ROAs that were created. These are the routing policies that would ultimately reject invalid announcements.

When it comes to Step 1 above (create ROAs for your own IP allocations), you turn to the web portal of your RIR. For purposes of this book that is the RIPE NCC LIR Portal, where you can easily create your ROAs, and which offers an easy-to-use method of showing your current BGP announcements and confirming whether these are the exact announcements for which you wish to create an ROA. Confirming this choice, and publishing the resulting ROAs, is all you have to do.

If you have not yet created ROAs that cover your IP space, then do so now. Creating ROAs for your IP space means that other networks will not be able to hijack your IP prefixes any more – or more precisely, that those networks which have implemented RPKI will reject these hijacks.

For Step 2 (installing and running a validator that will check the ROAs of other networks), just keep reading and we'll pull you through!

The RPKI is a public key infrastructure, and any party can choose to set up a Certificate Authority (CA) and host it on their own servers. In order to make it easy to start out with RPKI, the RIRs offer readily hosted CAs for IP space that belongs to their members. The ROAs are stored on the CAs, *trust anchors*, run by the RIRs. A ROA is a plain text file containing encrypted information and is downloaded from the trust anchor into a validator, which runs in your own network. The validator then talks to your routers so they can make routing policy decisions based on the RPKI information received from the validator.

When a *route is RPKI invalid*, it means that the advertised route (the IP prefix, the maximum length of the prefix, and the autonomous system number it is originated from) does not correspond with the certificate created. For instance, AS123 created a certificate allowing itself to announce 1.1.0.0/22 but instead you receive 1.1.0.0/22 from AS456. Since there is no certificate, the route from AS456 is invalid and should not be accepted into the routing table (even if the AS path is shorter or the local preference is higher than on the route from AS123). This is known as *BGP Origin Validation*.

Participate In Your Local NOG

In this book, you'll find tested use cases to make your network more secure. The authors use these in production networks and have verified that they work at the time of this writing. However, everything does not always stay the same. New developments lead to new features in our routers' software. Existing software may have bugs that need to be worked around. In the fast-moving world of BGP networking, it is important to stay informed about changes. Our advice in this regard is to get in touch – and stay in touch – with your local Network Operators' Group (NOG). Some NOGs have been around for years and years (like www.nanog.org or www.nlnog.net) and have regular conferences and other events. Other NOGs may be smaller and consist of only a mailing list or IRC channel. Whatever the NOG in your area can offer, it pays to participate.

Some use cases in this book have been taken from the NLNOG BGP filtering guide, which can be found on <http://bgpfilterguide.nlnog.net>. This website is regularly updated as new ways to secure the Internet are found. We recommend checking there every now and then and implementing any new recommendations found there.

Conclusion

Networks need a secure routing table in order to keep bad traffic away. A secure routing table does not contain invalid or bad routes. In order to achieve this as closely as possible, there are multiple ways of filtering route advertisements, such as implementing policies, or by using RPKI.

Filtering and rejecting routes is not scary and does not generally cause reachability issues: in cases of obvious BGP hijacks, rejecting the routes is undeniably good, and in case someone has made a typo in their route announcement, rejecting the route will actually help reachability instead of hindering it.

Now let's get started securing your routing table!

Chapter 2

Accepting and Announcing Routes

Your network accepts and advertises routes to and from various sources via BGP. For example, your transit providers send you a full table (all routes that they know), peers send you their own routes and their customers' routes, and you probably have customers who send you just their own routes (which may be only be a single route). You also want to make sure you are sending the right set of routes to the right peers or transits. All of these need their own filtering style and in this chapter we look at the theory behind that. The actual configuration is in Chapter 3 (for RPKI) and Chapter 4 (for all other routing policy decisions).

Receiving and Advertising Routes to and from Customers

Providing connectivity to your customers is what you have built your network for! If you provide statically-routed services to your customers (for example, leased lines or co-location) and your customers generally use your IP space, then your priority is to turn your own routing table into a *secure routing table*. If you also provide BGP service to your customers (you act as their transit provider and your customers announce their own IP space to you), then you will also want to configure some additional policies in your network to make life easier for your customers and for yourself.

Accepting Routes from Customers

As you set up connections to your customers, BGP will be used to announce their routes into your network. Your customers' announcements cause your network to forward them traffic. By sending your customers' announcements to your peers and transits (your routers will automatically prepend your own autonomous system number onto theirs); your network keeps your customers online. Since

accepting routes from your customers is often directly connected to revenue, it's important to accept as many routes from them as possible.

However, you need to perform checks on the announcements they send and not let them announce just any old routes that could severely harm the stability and security of your network.

These checks are:

- Only accept prefixes that have a valid route object in an IRR database.
- Only accept autonomous system numbers downstream of your customer that have a valid registration in your customers' AS-SET.
- Only accept prefix lengths up to /24 IPv4 and /48 IPv6.
- Only accept RPKI valid and unknown routes. Reject RPKI invalid.
- Reject routes containing a bogon autonomous system number, bogon prefix, prefix longer than /24, or AS path of unreasonable length.
- Reject routes you originate yourself; you should never receive your own routes from a customer.

Since your customers typically send you a few routes, it may seem feasible to implement route filters manually. But in the long run this will not scale, and you're better off using automation to implement the checks on the routes your customers send from the start. An example automation system is shown in the Appendix. This will take care of the route object and AS-SET checks referred to above.

Thanks to these checks, mistakes (they happen!) do not affect your network or the rest of the Internet. For instance, your customer could accidentally misconfigure their router and start announcing a full table to you instead of just their own routes. Without the correct filters on your side of the session you would accept and forward their announcements to the rest of the Internet, potentially disrupting business (and cat videos) worldwide.

Announcing Routes to Customers

Your customers expect to get full Internet connectivity from you. This means you will typically announce to them:

- A default route
- A full table (all routes known in the global routing table)
- Or a combination of a full table and a default route (not covered in detail)

Following the guidelines in this book means you will have a secure routing table to announce to your customers. They may receive fewer routes from you than from your competition (if your competition does not implement a secure routing table).

But in Chapter 1 we showed that this is a good thing! We also believe that providing a default route from this secure routing table is a better default route than one from a non-secure routing table. Why offer a default route to networks that intentionally, or unintentionally, announce the wrong IP address space?

Receiving and Advertising Routes to and from Peers

Now that we've looked at how to filter customer BGP sessions, the next step is peering sessions. In this context, a peer is another BGP-speaking network, roughly the same size, or at least the same type (tier) of network, as yours, that you are exchanging routes (and therefore traffic) with. Whether those peers are private peers or peers on an Internet Exchange Point (IXP) is not important for the setup of your secure routing table.

Typically, you would receive between one and several thousand routes from a peer; the peer would send you their routes and those of their customers, and you would send them your routes and those of your customers. Note that you will not be announcing your peer's routes to your transits – if someone is paying you to distribute their routes further upstream (to your other peers and transits) they are considered a customer, not a peer. Although you do, of course, announce routes learned from a peer to your customers.

Accepting Routes from Peers

A peer's network is a network like yours - operated by humans with fat fingers. Just as you did for customer routes, you will also need to filter the advertisements you accept from peers. In most cases you will receive a substantial number of routes via peers, so filtering on a per prefix basis, or by manually adding policies, doesn't scale.

MORE? The Appendix presents an idea to automate prefix filtering for peers.

Here are some basics you should consider when it comes to accepting routes from peers:

- Obviously you will need to check for RPKI invalids and actively reject those advertisements.
- Received routes from a peer should never contain the AS number of a known transit provider. For example, if your transit provider is AS123, then you should never see AS123 in a path received from a peer. If you do see AS123 in a route advertisement received from a peer, this means the peer is leaking routes. Don't 'graylist' those route leaks by giving them a lower preference "in case you need them" when the transit fails; simply filter them out! A route leak is never to be trusted; your peer will not have enough capacity to handle the traffic anyway.

- Peers also make mistakes. One day the administrator on ‘the other side’ will forget to apply an export policy and announce a full table by accident – so filter what and where you can.
- Reject routes containing a bogon autonomous system number, bogon prefix, prefix longer than /24, or AS path of unreasonable length.
- Reject routes you originate yourself; you should never receive your own routes from a peer.

Announcing Routes to Peers

You will typically announce the following routes to your peers. In this example it doesn’t really matter if it’s a private peer or if it’s on an IXP:

- Your own routes (the ones that your AS originates)
- Your customer’s routes (the ones that you learn from your BGP customers)

TIP Make sure to tag all advertisements you receive from customers, peers, and transits with communities. This will make it fairly easy to quickly advertise the right set of routes.

For example, if you tag routes received from your customers with a unique community and a general ‘customers community’ you can then advertise specific, or all, customer routes really easily to your peers. Think about the possibilities if you extend this method to peers and transits – and realize that the hard work is actually done in the import policies, not the export policies. Once your import policies filter and tag all received routes correctly, your routing table is clean and exporting routes from it will be a breeze.

Receiving and Advertising Routes from and to Transit Providers

So far, we’ve been talking about running BGP with your customers (who will send you only their own routes) and peers (who will send you their own routes, plus those of their customers).

In contrast, a transit will send you a ‘full table’, meaning that they will send you all the routes they know – a route to every destination on the Internet (hopefully filtered using RPKI, etc.).

A transit provider typically does not send a default route, although they can usually do so on request. In addition to sending you their full routing table, a transit provider will also accept your routes and send them to all of their customers, peers, and other transit providers, making them transport traffic on your behalf and sending it on to your network. So it seems likely that you will always want to accept all routes from your transit providers. But then, not everything is what it seems!

Accepting Routes from Transit Providers

When it comes to accepting routes from your transit provider, you of course need to accept the vast majority of them in order to make your network capable of reaching the entire Internet. However, even transit providers may send you routes that you may want to refuse, such as:

- RPKI invalids;
- Routes containing a bogon autonomous system number, bogon prefix, prefix longer than /24, or AS path of unreasonable length;
- Routes originating from your own network; you should never receive your own routes from a transit.

Announcing Routes to Transits

Typically, you would announce your and your customer's routes to your transit providers—remember the TIP where we used communities to tag those routes? This is the same set of routes that you would announce to your peers. Therefore, the same rules apply as with the 'Announcing routes to peers' instance described earlier in this chapter.

Conclusion

After implementing both the import and export filters and enabling RPKI, both of which are detailed in the next chapters, you should arrive at a situation as illustrated in Figure 2.1, where it is very clear what you accept from whom and to whom you advertise.

Table 2.1 *What and to Whom You Should Advertise*

Classifier (attached in ebgp-in)	ebgp-out to customer	ebgp-out to peer	ebgp-out to upstream
Learned from Customer	accept	accept	accept
Learned from Peer	accept	reject	reject
Learned from Upstream	accept	reject	reject
My own routes	accept	accept	accept
No Classifier	reject	reject	reject

To make it easier remember this simple rule: *you should always tag received routes with communities and advertise to the right parties based on these communities.* This will make your life much easier.

MORE? For more information on communities please see: https://www.juniper.net/documentation/en_US/junos/topics/example/bgp-communities.html, and https://www.juniper.net/documentation/en_US/junos/topics/usage-guidelines/policy-defining-bgp-communities-and-extended-communities-for-use-in-routing-policy-match-conditions.html.

MORE? This chapter was inspired by this presentation: https://ripe77.ripe.net/wp-content/uploads/presentations/59-RIPE77_Snijders_Routing_Policy_Architecture.pdf.

Chapter 3

Configuring RPKI: Resource Public Key Infrastructure

In Chapter 1 you saw how the RPKI provides routing security by validating announcements of prefixes from their AS. You should also have created ROAs for your own IP prefix(es). Now it's time to log in to your router and set up RPKI! This chapter creates a machine running an RPKI validator and configures your network to talk to it, making sure that your routers can use RPKI to validate prefix announcements.

Install the RPKI Validator

Your router does not actually perform the cryptographic checking of the certificates. Instead, you run this on a machine in your own network that downloads the ROAs from the trust anchors and provides a service for your routers to talk to. The story starts by installing a validator somewhere inside your own network.

At the time of this writing, there are two production-ready RPKI validators available. As more and more networks implement RPKI, it is likely that more validators will appear in the future. To help you get started we'll cover one of the validators, the one developed and maintained by the RIPE NCC.

The other production-grade validator has been developed and is maintained by NLnet Labs. Further information on this validator is available at: <https://nlnetlabs.nl/projects/rpki/routinator/>. (It's not the intention of the authors to push you into one of the two directions – we just had to pick one validator for the example.)

The RIPE RPKI Validator is written in Java and it requires a machine (physical or virtual) with at least 2 GB RAM, 1 CPU, and OpenJDK 8 installed. Make sure the machine can reach the Internet (for syncing the ROAs with the trust anchors) and that the machine is reachable from your routers.

TIP For security reasons you should enforce separate and strict security rules in your firewall to make sure only the right parts of the machine are reachable, especially the Internet-facing side, which obviously needs special attention.

The RIPE RPKI Validator project consists of two separately deployable units:

- The RPKI validator itself: this downloads and processes all objects like certificates, manifests, CRLs, ROAs, router certificates, and ghostbuster records; it caches the RPKI data, and it has a (web) user interface for troubleshooting and allow listing.
- The RPKI-RTR server: this is what your routers actually connect to.

The RPKI-RTR server is set up as a separate instance because not everyone needs to run this, and, more importantly, if you do need to run this, then the separate daemon allows you to run more than one instance for redundancy (it keeps state even when the validator is down).

NOTE Running redundant validators in your network is, just as running redundant routers, a good strategy. As there are multiple validators available, running a multi-vendor setup is also possible.

The good thing is that if no validator is available all routes in your routing table will be marked unknown so you will only lose validation state and no actual routing information.

Schematically, this is how communication flows:

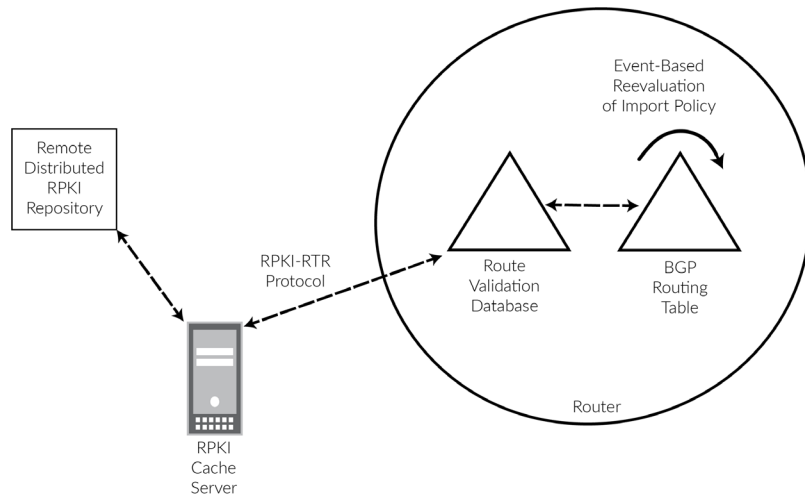


Figure 3.1

Communication Flow for RPKI-RTR

To install the RIPE NCC Validator v3, follow the installation instructions at: <https://github.com/RIPE-NCC/rpki-validator-3/wiki/RIPE-NCC-RPKI-Validator-3-Production>.

By default, the validator will listen on localhost only. In order to be able to reach the web interface (where you can check the status of routes, as well as apply allowlist entries), edit the validator configuration file:

```
# vi /etc/rpki-validator-3/application.properties
```

And make sure to uncomment and change the `server.address` line to:

```
server.address= <ip address>
```

This makes your validator web site globally available on your server's public IP address, which may not be what you want. Therefore, you could use iptables to close down access by making sure you have the following lines in the config:

```
iptables -A INPUT -s <management-prefix> -p tcp -m tcp --dport 8080 -j ACCEPT
iptables -A INPUT -p tcp -m tcp --dport 8080 -j REJECT --reject-with icmp-port-unreachable
```

Replace `<management-prefix>` with a prefix that is allowed to reach the web interface.

The Trust Anchor Locator (TAL) of all RIRs is included in the software distribution, with one exception: ARIN. Unfortunately, some cumbersome extra steps must be taken to include their Trust Anchor in your verification process. In order to obtain the ARIN TAL, go to their web page at <https://www.arin.net/resources/rpki/tal.html>, make sure you agree with the stated terms, and download the TAL in RIPE NCC validator format. Place the TAL file in your home directory on the validator server.

Finally, use the `upload-tal.sh` script to upload the ARIN TAL to the validator:

```
# /usr/bin/upload-tal.sh arin-ripevalidator.tal http://localhost:8080/
```

Then, restart the validator:

```
# systemctl restart rpki-validator-3
```

Now you can browse to the public IP address of your validator, port 8080, and you will see a web page listing the configured Trust Anchors as shown in Figure 3.2.

RPKI Validator			
Trust Anchors ROAs Ignore Filters Whitelist BGP Preview Announcement Preview			
Configured Trust Anchors			
Trust Anchors	Processed Items		
			Last Updated
Afrinic RPKI Root	516	0	0
APNIC RPKI Root	5616	392	0
ARIN	3203	0	0
LACNIC RPKI Root	5020	0	0
RIPE NCC RPKI Root	25049	0	0


 Copyright ©2009-2018 the Réseaux IP Européens Network Coordination Centre RIPE NCC. All rights restricted.

Figure 3.2 *RPKI Validator Trust Anchors*

The same exercise now needs to be done for the RPKI-RTR software. This is the part that talks to your routers and enables them to create their validation database. After installation, edit `/etc/rpki-rtr-server/application.properties` to uncomment the lines that tells the application what IP address to listen on for the (future) web interface and the RTR service:

```
server.address=
rtr.server.address=<your server's public IP>
```

Finally, if you use iptables to limit access to ports 8081 (RTR web interface) and 8323 (RTR service), the access to the web interface should probably be the same as for the validator software just discussed; access to the RTR service should be limited to the IP addresses from which your routers will be accessing the validator (this will usually be the `lo0` address or the address on the interface that is “closest” to the server, and therefore used for outgoing traffic to the server):

```
iptables -A INPUT -s <management-prefix> -p tcp -m tcp --dport 8081 -j ACCEPT
iptables -A INPUT -p tcp -m tcp --dport 8081 -j REJECT --reject-with icmp-port-unreachable
iptables -A INPUT -s <router-prefix> -p tcp -m tcp --dport 8323 -j ACCEPT
iptables -A INPUT -p tcp -m tcp --dport 8323 -j REJECT --reject-with icmp-port-unreachable
```

Restart the RPKI-RTR server:

```
# systemctl restart rpki-rtr-server
```

Connect Your Routers to the Validator

The first step for using origin validation data within your Juniper Networks router is to set up communication with the validator. In this example, the validator has IPv6 address `2001:db8::f00:bba` and the routers address is `2001:db8::1`. This will work using IPv4 as well.

Obviously, both don't have to be on the same subnet. But they must be able to communicate with each other on the RTR port (in the standard configuration, that is, TCP port 8323). Make sure you have opened this port on any firewalls in between, and don't forget to adjust the firewall filter on your router's loopback interface to allow communication with the validator. The terms *cache* and *validator* are used; the cache is part of the validator and holds the ROA information. In this context we use the term validator to identify both, and we will not distinguish between the cache and the validator:

```
routing-options {
  validation {
    group rpki-validator {
      session 2001:db8::f00:baa {
        port 8323;
        local-address 2001:db8::1;
      }
    }
  }
}
```

The commands to achieve this configuration are:

```
set routing-options validation group rpki-validator session 2001:db8::f00:baa port 8323
set routing-options validation group rpki-validator session 2001:db8::f00:baa local-
address 2001:db8::1
```

After committing the candidate configuration, your router will set up a validation session. You can see the session status using:

```
user@router> show validation session
Session                               State   Flaps   Uptime #IPv4/IPv6 records
2001:db8::f00:baa                     Up      0       00:00:47  0/0
```

If the session does not come up, troubleshoot connectivity between the router and the machine running the validator. Remember to adjust the routers loopback firewall filter and/or any iptables or firewall in the path, or a firewall on the validator. Be sure to take the source address used on your router into account.

Configure Your Routers to Tag RPKI Valid Routes

The validation database is a separate entity in your router's memory. Entries in the validation database do not automatically make it into the routing table (let alone into the forwarding table). Making RPKI work for your network means that you have to configure a policy that will look at the state of each prefix and tag the corresponding route in the routing table. Now the RPKI status of the routes in your routing table will be attached to those routes. The final step is to use the RPKI status to accept or reject routes (or take other action on them as required) as shown in Figure 3.3.

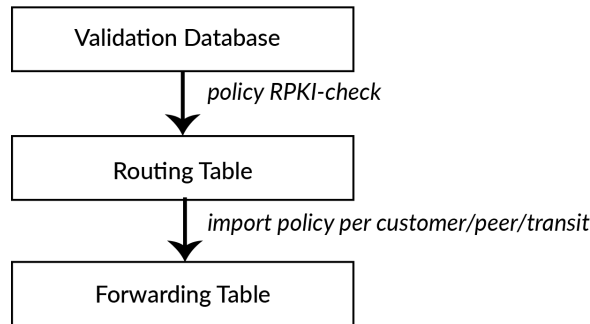


Figure 3.3 Validation Database Sequence

Accepting and Rejecting RPKI Checked Advertisements

As the RPKI validator has been installed and is made available on your routers, nothing has yet happened to your routing table. Now it's time to update your routing policies and actually do something with the RPKI information!

The validation database contains prefixes, prefix lengths, and autonomous system numbers. Your routing policy uses this information to decide which routes to take from the routing information base (RIB) and install into the forwarding information base (FIB).

There are three possible RPKI states in the validation database: valid, invalid, and unknown. As most networks in the world are only in the starting phase of RPKI implementation, most routes will be of unknown state. Your task is to accept the valid and unknown routes, and reject the invalid routes. In addition, add a BGP community – a flag that each route carries with it. This will make troubleshooting easier and will enable your customers to see your RPKI information as well.

In the configuration snippet below, we created a policy that you can call from other routing policies. It takes the RPKI state in the validation database and then sets the corresponding validation state as the route goes to the RIB. In addition, the policy sets a BGP community as a flag that shows the RPKI status of the prefix. Note that none of the terms in this policy actually accept or reject the route; the policy is purely there to tag the prefixes going to the RIB.

```

policy-options {
  policy-statement RPKI-CHECK {
    term valid {
      from {
        protocol bgp;
        validation-database valid;
      }
      then {
        validation-state valid;
      }
    }
  }
}

```

```

        community add origin-validation-state-valid;
    }
}
term invalid {
    from {
        protocol bgp;
        validation-database invalid;
    }
    then {
        validation-state invalid;
        community add origin-validation-state-invalid;
    }
}
term unknown {
    from {
        protocol bgp;
        validation-database unknown;
    }
    then {
        validation-state unknown;
        community add origin-validation-state-unknown;
    }
}
}
}
}

```

You can enter this policy by cutting and pasting the following lines:

```

set policy-options policy-statement RPKI-CHECK term valid from protocol bgp
set policy-options policy-statement RPKI-CHECK term valid from validation-database valid
set policy-options policy-statement RPKI-CHECK term valid then validation-state valid
set policy-options policy-statement RPKI-CHECK term valid then community add origin-validation-
state-valid
set policy-options policy-statement RPKI-CHECK term invalid from protocol bgp
set policy-options policy-statement RPKI-CHECK term invalid from validation-database invalid
set policy-options policy-statement RPKI-CHECK term invalid then validation-state invalid
set policy-options policy-statement RPKI-CHECK term invalid then community add origin-validation-
state-invalid
set policy-options policy-statement RPKI-CHECK term unknown from protocol bgp
set policy-options policy-statement RPKI-CHECK term unknown from validation-database unknown
set policy-options policy-statement RPKI-CHECK term unknown then validation-state unknown
set policy-options policy-statement RPKI-CHECK term unknown then community add origin-validation-
state-unknown

```

In addition, define the communities used:

```

set policy-options community origin-validation-state-valid members 0x4300:0
set policy-options community origin-validation-state-unknown members 0x4300:1
set policy-options community origin-validation-state-invalid members 0x4300:2

```

These communities are well-known, large communities for tagging routes with their RPKI status as seen from your network. The communities for RPKI valid and RPKI unknown routes are added just for informational purposes. However, you will use the community for RPKI invalid routes later on to actively reject these RPKI invalid routes.

NOTE The policy RPKI-CHECK does not actually accept or reject routes. It just looks at the validation database for each route that passes it, and sets the RPKI status on these routes in the RIB, as well as adding the informational BGP community.

You will now add the policy RPKI-CHECK to the very beginning of *every import policy on your network*, meaning on the BGP sessions with:

- transit providers
- peers
- customers

Best practices are to add the RPKI-CHECK stanza to all your import policies. For example you should add to *the beginning of every import policy on your router*:

```
term RPKI-CHECK {
  from policy RPKI-CHECK;
}
```

Again, note that this does not actually reject RPKI invalid routes. Therefore the last step to an RPKI-secured routing table is to add (somewhere after the term RPKI-CHECK) the term:

```
term RPKI-INVALID {
  from community origin-validation-state-invalid;
  then reject;
}
```

If you commit these changes now, you'll start rejecting RPKI invalid routes. At the time of this writing (early 2019) that's about 6000 invalid prefixes that will not be present in your routing table! You can verify the routes using commands such as:

```
user@router> show route validation-state valid
user@router> show route validation-state unknown
user@router> show route validation-state invalid
```

None of the invalid routes should be active in your routing table.

If you're unsure how the policies all fit together in the grand scheme of things, keep reading. In Chapter 4 we'll present a unified import policy that is the basis of your secure routing table.

Router Vendor's Support for RPKI

This book, being part of Juniper's *Day One* Series, discusses techniques and recipes for deploying RPKI on Juniper routers. However, in order for RPKI validation to work, all your (edge) routers will need to implement it. All routes you learn from other parties (customers, peers, transits) will have to be validated. In case your (edge) routers do not support RPKI, they will still accept invalid routes and install them in your network's routing table.

To make it easier for you to find out how successful your RPKI implementation will be, we compiled a list of routing software that supports RPKI:

- Juniper Networks has supported the routing software listed below since Junos OS version 12.2 (it is advised to take into account PR1309944).
- Cisco:
 - XR 4.2.1 (CRS-x, ASR9000, c12K) / XR 5.1.1 (NCS6000, XRv)
 - XE 3.5 (C7200, c7600, ASR1K, CSR1Kv, ASR9k, ME3600...)
 - IOS15.2(1)S
- Alcatel Lucent has had support since SR-OS 12.0 R4.
- Nokia (R12.0R4):
 - 7210 SAS
 - 7750 SR
 - 7950 XRS
 - VSR
- Quagga has support through BGP-SrX or RTRLib.
- BIRD has support for ROA and supports RPKI-RTR from version 2.0 or via RTRLib
- GoBGP
- FRRouting
- OpenBGPD (support for Origin Validation through *static* configuration)

Conclusion

You have now set up an RPKI validator, and your router is talking to it. Your router now has a validation database and for each prefix, it knows whether it is valid, invalid, or unverified. This information has made it into the routing table and RPKI invalid routes are rejected. In case this is all that you wanted to achieve, congratulations, and you can skip Chapter 4 and turn straight to Chapter 5 where we discuss troubleshooting RPKI. However, we hope you will just keep reading!

In Chapter 4, you'll continue on the security quest and apply routing policies to the routes you've learned from transits, peers, and customers.

Chapter 4

Configuring Routing Policies

If setting up RPKI validation was something new for you, you can now relax because this chapter will probably be more familiar! It will implement policies to filter incoming routes from your customers, peers, and transits, and it will show you how to create scalable filters to advertise routes to your customers, peers, and transits.

You may feel that your import and export policies are already quite good and do not need additional work. And ... we would probably agree! However, even if you are sure that you've already got what it takes to successfully filter routes, please take some time to at least skim through this chapter. There may be one or two ideas in here that are worth your time.

Building Blocks: Basic Filtering Rules

Next we'll address the policies that will be referred to by other policies. They are not unique per customer/peer/transit and need to be defined only once.

In this section, we are not showing you actual policies. You will cut and paste relevant prefix lists and AS path groups, though, that are used in the policies later on in the chapter.

Reject Bogon Autonomous System Numbers

A BGP route announcement contains a field, called *AS path*, consisting of the autonomous system numbers from all networks that propagate the route advertisement in order to reach its destination, the originating autonomous system number.

The AS paths you see in your routing table are automatically created as each network propagating a route advertisement prepends its own autonomous system number to the path. Finally, it is possible to manually prepend an autonomous system number in order to influence routing decisions.

In addition to the public autonomous system numbers assigned by the RIRs, there are private autonomous system numbers that are to be used for different purposes (like private peering with networks that do not need an RIR-assigned autonomous system number).

All autonomous system numbers used to be 16-bit numbers (from 0 up to 65,535), but these days 32-bit autonomous system numbers are universally supported. The different types of autonomous system numbers are:

- 0: reserved
- 1 through 64,495: public autonomous system numbers
- 64,496 through 64,511: reserved to use in documentation
- 64,512 through 65,534: private autonomous system numbers
- 65,535: reserved
- 4,200,000,000 through 4,294,967,294: 32-bit private autonomous system numbers

A popular use of private autonomous system numbers is using such numbers (from a private autonomous system numbers list) for networks that do not have an autonomous system number assigned by an RIR in order to establish a BGP session with an upstream network. In this case, mistakes are easy to make. Private autonomous system numbers can accidentally leak into a publicly visible AS path. Or sometimes a network operator will mistype the autonomous system number they wish to prepend, polluting the path you receive. Therefore you have to reject route advertisements that contain a private or reserved autonomous system number.

The bogon autonomous system numbers are defined in RFCs. The list below shows exactly which RFC describes the bogon autonomous system number, so if you want to learn more about why a certain autonomous system number should

not be in your secure routing table, you can consult the relevant RFC.

Define an as-path group that lists the bogon autonomous system numbers:

```
policy-options {
  as-path-group BOGON-ASNS {
    /* RFC7607 */
    as-path zero ".* 0 .*";
    /* RFC 4893 AS_TRANS */
    as-path as_trans ".* 23456 .*";
    /* RFC 5398 and documentation/example ASNs */ as-path examples1 ".* [64496-64511] .*";
    as-path examples2 ".* [65536-65551] .*";
    /* RFC 6996 Private ASNs*/
    as-path reserved1 ".* [64512-65534] .*";
    as-path reserved2 ".* [4200000000-4294967294] .*"; /* RFC 6996 Last 16 and 32 bit ASNs */
    as-path last16 ".* 65535 .*";
    as-path last32 ".* 4294967295 .*";
    /* RFC IANA reserved ASNs*/
    as-path iana-reserved ".* [65552-131071] .*";
  }
}
```

You do this by entering the following commands:

```
set policy-options as-path-group BOGON-ASNS as-path zero ".* 0 .*"
set policy-options as-path-group BOGON-ASNS as-path as_trans ".* 23456 .*"
set policy-options as-path-group BOGON-ASNS as-path examples1 ".* [64496-64511] .*"
set policy-options as-path-group BOGON-ASNS as-path examples2 ".* [65536-65551] .*"
set policy-options as-path-group BOGON-ASNS as-path reserved1 ".* [64512-65534] .*"
set policy-options as-path-group BOGON-ASNS as-path reserved2 ".* [4200000000-4294967294] .*"
set policy-options as-path-group BOGON-ASNS as-path last16 ".* 65535 .*"
set policy-options as-path-group BOGON-ASNS as-path last32 ".* 4294967295 .*"
set policy-options as-path-group BOGON-ASNS as-path iana-reserved ".* [65552-131071] .*"
```

Remember, this is not the actual policy yet. The policy calling these prefix lists is coming up, so please read through a few more paragraphs!

With this policy your routers will filter routes that you *receive* from outside and that may contain a private autonomous system number. In order to avoid private autonomous system numbers being announced (*sent*) from your network, it is good practice in DFZ operations to enable the `remove-private` option on EBGP sessions.

For example:

```
set protocols bgp group ext type external
set protocols bgp group ext neighbor 192.168.10.1 peer-as 65530
set protocols bgp group ext neighbor 192.168.20.1 remove-private
set protocols bgp group ext neighbor 192.168.20.1 peer-as 200
```

Make sure you have this option set on all your EBGP sessions.

Reject Bogon Prefixes

A bogon prefix should not be visible in the global routing table, as these prefixes are meant for internal use (RFC 1918), test networks (RFC 4737), multicast, and other internal purposes. Thus these prefixes should never be accepted or advertised to the DFZ.

You can create a prefix list as follows:

```
policy-options { prefix-list BOGONS {
    0.0.0.0/8;
    10.0.0.0/8;
    100.64.0.0/10;
    127.0.0.0/8;
    169.254.0.0/16;
    172.16.0.0/12;
    192.0.0.0/24
    192.0.2.0/24;
    192.88.99.0/24;
    192.168.0.0/16;
    198.18.0.0/15;
    198.51.100.0/24;
    203.0.113.0/24;
    224.0.0.0/3;
}
```

You do this by entering the following commands:

```
set policy-options prefix-list BOGONS 0.0.0.0/8
set policy-options prefix-list BOGONS 10.0.0.0/8
set policy-options prefix-list BOGONS 100.64.0.0/10
set policy-options prefix-list BOGONS 127.0.0.0/8
set policy-options prefix-list BOGONS 169.254.0.0/16
set policy-options prefix-list BOGONS 172.16.0.0/12
set policy-options prefix-list BOGONS 192.0.0.0/24
set policy-options prefix-list BOGONS 192.0.2.0/24
set policy-options prefix-list BOGONS 192.88.99.0/24
set policy-options prefix-list BOGONS 192.168.0.0/16
set policy-options prefix-list BOGONS 198.18.0.0/15
set policy-options prefix-list BOGONS 198.51.100.0/24
set policy-options prefix-list BOGONS 203.0.113.0/24
set policy-options prefix-list BOGONS 224.0.0.0/3
```

The same for IPv6 commands:

```
set policy-options prefix-list BOGONS-INET6 0000::/8
set policy-options prefix-list BOGONS-INET6 fe00::/9
set policy-options prefix-list BOGONS-INET6 ff00::/8
set policy-options prefix-list BOGONS-INET6 2001:db8::/32
set policy-options prefix-list BOGONS-INET6 3ffe::/16
set policy-options prefix-list BOGONS-INET6 2001::/32
```

The policy calling these prefix lists is coming up, only a few more paragraphs!

Reject Long Prefixes

The global routing table in the DFZ is expanding daily, as more and more networks start announcing more and more prefixes. In principle, there is no problem announcing routes of any length – technically, even a /32 route (for a single IPv4 address) will work. However, uncontrolled growth of the routing table is not sustainable (just calculate how much bigger the DFZ would be if all IPv4 space would be announced as a /32). Many networks have started filtering on prefix length.

These days you can generally expect your announcement to be dropped if you announce a prefix that is /25 or longer (a subnet of 128 IPv4 addresses or less). The following policy configures your network to do the same, so all these pesky little /28 and /32 routes don't make it to your RIB. The same applies to IPv6 where we put the boundary at /48 subnets.

The policies to filter out routes for these long prefixes appear later in the chapter. They consist of two steps: a generic policy that is called and returns only prefixes of sufficient length, and a policy that takes this list and filters only the relevant routes from it.

Reject Long AS Paths

We've seen before that each network on the route automatically "builds" an AS path, adding its own autonomous system number to the front of the AS path of the advertised route. In practice, since BGP looks for the shortest AS path by default, most active routes your network uses will have an AS path of only 1 - 6 autonomous system numbers. Perhaps sometimes a path can be longer, and then of course there is manual AS path prepending (as a traffic engineering measure), which causes longer paths to be seen in the table. But generally speaking, an AS path longer than a dozen or so autonomous system numbers can be considered useless. If you see such a long AS path, it is probably the work of a network administrator who needs a copy of this book.

The most common reason for extremely long AS paths is prepending. Using two to three times your own autonomous system number in a prepend is, when needed, considered a good practice to influence routes. However, prepending more than that doesn't make sense. Looking at the DFZ, at the time of this writing there are some prefixes with an AS path length of about 40 autonomous system numbers, which we highly doubt is useful. Let's take a safe margin, and consider everything with an AS path of more than 50 to be useless and deserving to be filtered out in our secure routing table.

As always, define what you want to filter first:

```
set policy-options as-path too-many-hops "{50,}"
```

And then, read on to get to the policy where we call this definition! Just one more building block to go.

Reject Routes Containing Known Transit or Very Large Network Autonomous System Numbers

The biggest networks in the world (mainly known as “Tier-1”) never buy transit from each other or from smaller (“Tier-2”) networks. For instance, it would be very strange if your customer or peer started sending you routes that have AS2914 (NTT) or AS1299 (Telia) in their AS path; it is not very likely that NTT or Telia would buy transit from your customer. Therefore, your import policy also contains AS path filters that reject routes that have the AS numbers of the “big names” in them. If you would legitimately receive these from a customer or peer, you are probably not the target audience for this book.

The same goes for some of the other very large networks like Facebook, Google, Microsoft, and Cloudflare. They will probably not buy transit from your peers, so if you receive their routes from your peers, there must be something *fishy* going on. So why not make sure you won’t use any of these routes, which you know are not correct in the first place?

First, let’s define the AS path group:

```
set policy-options as-path-group BIGNETWORKS as-path 174 ".* 174 .*"
set policy-options as-path-group BIGNETWORKS as-path 209 ".* 209 .*"
set policy-options as-path-group BIGNETWORKS as-path 286 ".* 286 .*"
set policy-options as-path-group BIGNETWORKS as-path 701 ".* 701 .*"
set policy-options as-path-group BIGNETWORKS as-path 702 ".* 702 .*"
set policy-options as-path-group BIGNETWORKS as-path 703 ".* 703 .*"
set policy-options as-path-group BIGNETWORKS as-path 714 ".* 714 .*"
set policy-options as-path-group BIGNETWORKS as-path 1239 ".* 1239 .*"
set policy-options as-path-group BIGNETWORKS as-path 1299 ".* 1299 .*"
set policy-options as-path-group BIGNETWORKS as-path 2828 ".* 2828 .*"
set policy-options as-path-group BIGNETWORKS as-path 2906 ".* 2906 .*"
set policy-options as-path-group BIGNETWORKS as-path 2914 ".* 2914 .*"
set policy-options as-path-group BIGNETWORKS as-path 3209 ".* 3209 .*"
set policy-options as-path-group BIGNETWORKS as-path 3257 ".* 3257 .*"
set policy-options as-path-group BIGNETWORKS as-path 3320 ".* 3320 .*"
set policy-options as-path-group BIGNETWORKS as-path 3356 ".* 3356 .*"
set policy-options as-path-group BIGNETWORKS as-path 3549 ".* 3549 .*"
set policy-options as-path-group BIGNETWORKS as-path 3561 ".* 3561 .*"
set policy-options as-path-group BIGNETWORKS as-path 5511 ".* 5511 .*"
set policy-options as-path-group BIGNETWORKS as-path 6453 ".* 6453 .*"
set policy-options as-path-group BIGNETWORKS as-path 6461 ".* 6461 .*"
set policy-options as-path-group BIGNETWORKS as-path 6762 ".* 6762 .*"
set policy-options as-path-group BIGNETWORKS as-path 7018 ".* 7018 .*"
set policy-options as-path-group BIGNETWORKS as-path 8075 ".* 8075 .*"
set policy-options as-path-group BIGNETWORKS as-path 12956 ".* 12956 .*"
set policy-options as-path-group BIGNETWORKS as-path 13335 ".* 13335 .*"
set policy-options as-path-group BIGNETWORKS as-path 15169 ".* 15169 .*"
set policy-options as-path-group BIGNETWORKS as-path 16509 ".* 16509 .*"
set policy-options as-path-group BIGNETWORKS as-path 32934 ".* 32934 ."
```

Using this AS path group in the policy that’s coming up protects you from accepting routes that your customers or peers are accidentally leaking to you. How this fits into the bigger picture will be shown later on.

Make sure to add this policy only on your customer and peer BGP sessions (and not to your transits) or you will end up with a very small (and a very limited) usable routing table!

MORE? Our list of *big networks* is only a suggestion. If you are looking to improve on the list, this could be a good starting point: <http://as-rank.caida.org/>.

MORE? If you want to do more (going beyond the scope of this book), or if you have downstream customers using BGP sessions to connect to your network, or both, then take a look at two more possible features you can implement:

1. Graceful BGP session shutdown: <https://tools.ietf.org/html/rfc8326>. Graceful Shutdown makes your network respond to an advance warning from a customer, peer, or transit saying that maintenance is coming up. It will minimize downtime that would occur with such maintenance. You can read more on how to implement this at: http://bgpfilterguide.nlnog.net/guides/graceful_shutdown/.
2. Well-Known Blackhole Community: <https://tools.ietf.org/html/rfc7999>. The Well-Known Blackhole Community lets your customers blackhole (null route) traffic to a destination prefix inside of their own prefix. This is useful when a DDoS attack comes in; it would drop all traffic to the victim (making the DDoS attack successful) but at least it keeps the rest of the network online.

Reject the Remainder

By default, the last action in the Junos OS implementation of BGP is to allow everything that is left after policies. The default behavior will become configurable in the future to stay compliant with the BGP-4 specifications and as defined in <https://tools.ietf.org/html/rfc8212>. Though, at the time of writing this book, RFC8212 hasn't been implemented, so you should make sure you have a policy in place to reject the remainder.

Define a policy that simply rejects all routes that it sees:

```
policy-options {
  policy-statement REJECT-ALL {
    then reject;
  }
}
```

Or in cut-and-paste commands:

```
set policy-options policy-statement REJECT-ALL then reject
```

This policy is used later on to create an explicit reject, meaning that you do not rely on whatever the protocol (BGP) does at the end of a policy chain. It is always better to make your configuration clear and explicit (think of your most junior network admin having to troubleshoot a network outage at 4 AM).

Filtering Customer Routes

Finally, it's time to configure some actual routing policy! First, let's look at how to filter routes that you receive from your customers, and how to set up a policy to announce routes to your customers, both for default route and full table scenarios.

Accepting Routes from Customers

Checking the routes that you accept from your customers is the most important bit of configuration in your network! In the end, you need to announce your customer's routes to others. Those announcements need to be clean and contain only correct routing information – so let's start by accepting clean routes only, then you won't have to worry about bringing the Internet to a halt.

The routing policy that you will use to filter the routes received from customers will include a list of the prefixes that the customer is allowed to advertise. In this example you configure the prefix list manually, but in the Appendix you'll find the way to get the prefix lists to auto-update! After finishing this book you should feel comfortable using this configuration for filtering your customer's routes, and don't forget to implement automatic prefix-list updates so your routing table remains secure!

The configuration below assumes that your customer's AS number is 123 and they will be announcing 192.0.2.0/24 and 2001:db8::/32. It also assumes that your own AS number is 456:

```
policy-options {
  prefix-list AS123 {
    192.0.2.0/24;
  }
  prefix-list AS123-INET6 {
    2001:db8::/32; =
  }
}
```

You configure these by entering:

```
set policy-options prefix-list AS123 192.0.2.0/24
set policy-options prefix-list AS123-INET6 2001:db8::/32
```

In addition, let's create two BGP communities that the policies use, as well as one you will need later on:

```
policy-options {
  community RFC-N0-EXPORT members no-export;
  community MYCUSTOMER members 456:9999;
  community MYROUTES members 456:456;
}

set policy-options community RFC-N0-EXPORT members no-export
set policy-options community MYCUSTOMER members 456:9999
set policy-options community MYROUTES members 456:456
```

When creating the discard ‘anchor route’ to originate your prefix, you can tie this community to it:

```
routing-options {
  rib inet.0 {
    static {
      route 192.0.2.0/24 {
        discard;
        community 456:456;
      }
    }
  }
}
```

Now, onto the policies that will actually accept your customer’s routes. It all starts by accepting the exact prefix that they are allowed to announce following the prefix-list:

```
policy-options {
  policy-statement IMPORT-123 {
    term INET {
      from {
        prefix-list-filter AS123 exact;
      }
      then {
        community add MYCUSTOMER;
        accept;
      }
    }
  }
}
```

You configure this by entering:

```
set policy-options policy-statement IMPORT-123 term INET from prefix-list-filter AS123 exact
set policy-options policy-statement IMPORT-123 term INET then community add MYCUSTOMER
set policy-options policy-statement IMPORT-123 term INET then accept
```

This policy is unique to every BGP customer you have, just like every prefix list is unique to each BGP customer. You apply this import policy on the BGP session with your customer in the import policy chain.

We added the BGP community MYCUSTOMER to the routes accepted from your customer. This makes it easier to export these routes later on.

Next, your customer may announce more specific prefixes to you. The prefix list entry may have a /22 defined, for instance, but you want to allow your customer to announce up to a /24. Simply using prefix-list-filter AS123 or longer doesn’t work, because that would allow the customer to announce up to a /32. So first, create a policy that allows routes up to /24, and then combine that with the policy allowing these routes if they are inside the customer prefix list:

```
policy-options {
  policy-statement MORE-SPECIFIC-UPTO-24 {
    term INET {
```

```

        from {
            family inet;
            route-filter 0.0.0.0/0 upto /24;
        }
        then accept;
    }
    term REJECT {
        then reject;
    }
}
policy-statement IMPORT-123 {
    term MORE-SPECIFIC {
        from {
            policy MORE-SPECIFIC-UPTO-24;
            prefix-list-filter AS123 orlonger;
        }
        then {
            community add MYCUSTOMER;
            accept;
        }
    }
}
}
}

```

Note that policy-statement MORE-SPECIFIC-UPTO-24 is generic – it is configured on the router only once and used for all customers. The policy-statement IMPORT-123 is unique to each customer, however, and has to be configured once per customer.

You configure these policies by entering:

```

set policy-options policy-statement MORE-SPECIFIC-UPTO-24 term INET from family inet
set policy-options policy-statement MORE-SPECIFIC-UPTO-24 term INET from route-
filter 0.0.0.0/0 upto /24
set policy-options policy-statement MORE-SPECIFIC-UPTO-24 term INET then accept
set policy-options policy-statement MORE-SPECIFIC-UPTO-24 term REJECT then reject
set policy-options policy-statement IMPORT-123 term MORE-SPECIFIC-INET from policy MORE-SPECIFIC-
UPTO-24
set policy-options policy-statement IMPORT-123 term MORE-SPECIFIC-INET from prefix-list-
filter AS123 orlonger
set policy-options policy-statement IMPORT-123 term MORE-SPECIFIC-INET then community add MYCUSTOMER
set policy-options policy-statement IMPORT-123 term MORE-SPECIFIC-INET then accept

```

The policy MORE-SPECIFIC-UPTO-24 needs to be defined only once. The customer's AS or prefixes are not referenced in this policy. The policy IMPORT-123 has to be defined *once per customer*, though.

The exact same logic applies for the IPv6 policies, where you will accept routes up to /48:

```

policy-options {
    policy-statement IMPORT-123-INET6 {
        term INET6 {
            from {
                prefix-list-filter AS123-INET6 exact;
            }
            then {
                community add MYCUSTOMER;
            }
        }
    }
}

```



```

        accept;
    }
}
policy-statement MORE-SPECIFIC-UPTO-48-INET6 {
    term INET6 {
        from {
            family inet6;
            route-filter ::/0 upto /48;
        }
        then accept;
    }
    term REJECT {
        then reject;
    }
}
policy-statement IMPORT-123-INET6 {
    term MORE-SPECIFIC-INET6 {
        from {
            policy MORE-SPECIFIC-UPTO-48-INET6;
            prefix-list-filter AS123-INET6 orlonger;
        }
        then {
            community add MYCUSTOMER;
            accept;
        }
    }
}
}

```

You configure this by entering:

```

set policy-options policy-statement MORE-SPECIFIC-UPTO-48-INET6 term INET6 from family inet6
set policy-options policy-statement MORE-SPECIFIC-UPTO-48-INET6 term INET6 from route-
filter ::/0 upto /48
set policy-options policy-statement MORE-SPECIFIC-UPTO-48-INET6 term INET6 then accept
set policy-options policy-statement MORE-SPECIFIC-UPTO-48-INET6 term REJECT then reject
set policy-options policy-statement IMPORT-123-INET6 term MORE-SPECIFIC-INET6 from policy MORE-
SPECIFIC-UPTO-48-INET6
set policy-options policy-statement IMPORT-123-INET6 term MORE-SPECIFIC-INET6 from prefix-list-
filter AS123-INET6 orlonger
set policy-options policy-statement IMPORT-123-INET6 term MORE-SPECIFIC-
INET6 then community add MYCUSTOMER
set policy-options policy-statement IMPORT-123-INET6 term MORE-SPECIFIC-INET6 then accept

```

Like with the IPv4 policies, the policy MORE-SPECIFIC-UPTO-48-INET6 needs to be defined only once. The customer's AS or prefixes are not referenced in this policy. The policy IMPORT-123-INET6 has to be defined once per customer, though.

Finally, you may choose to allow your customers to announce even more specific routes to you – more specific than a /24. They could use these routes for traffic engineering, for instance. While you may carry them in your network, you should not announce them to your other customers, peers, or transits. This is why we will accept them here, but add the well-known “NO-EXPORT” BGP community to them:

```

policy-options {
  policy-statement IMPORT-123 {
    term MOST-SPECIFIC-INET {
      from {
        prefix-list-filter AS123 orlonger;
      }
      then {
        community add RFC-NO-EXPORT;
        accept;
      }
    }
  }
}

```

You configure this by entering:

```

set policy-options policy-statement IMPORT-123 term MOST-SPECIFIC-INET from prefix-list-
filter AS123 longer
set policy-options policy-statement IMPORT-123 term MOST-SPECIFIC-INET then community add RFC-NO-
EXPORT
set policy-options policy-statement IMPORT-123 term MOST-SPECIFIC-INET then accept

```

And for IPv6:

```

policy-options {
  policy-statement IMPORT-123-INET6 {
    term MOST-SPECIFIC-INET6 {
      from {
        prefix-list-filter AS123-INET6 orlonger;
      }
      then {
        community add RFC-NO-EXPORT;
        accept;
      }
    }
  }
}

```

And you configure this by entering:

```

set policy-options policy-statement IMPORT-123-INET6 term MOST-SPECIFIC-INET6 from prefix-list-
filter AS123-INET6 longer
set policy-options policy-statement IMPORT-123-INET6 term MOST-SPECIFIC-
INET6 then community add RFC-NO-EXPORT
set policy-options policy-statement IMPORT-123-INET6 term MOST-SPECIFIC-INET6 then accept

```

Now that we've defined the import policy for your customer's routes, you may wonder where the final reject policy that will reject routes that the customer is not allowed to announce is. You're correct of course – here is the entire policy that you will use for importing routes from your customers:

```

policy-options {
  policy-statement TRANSIT-CUSTOMER-GENERIC {
    term DEFAULT-INET {
      from {
        family inet;
        route-filter 0.0.0.0/0 exact;
      }
      then reject;
    }
  }
}

```

```

term DEFAULT-INET6 {
    from {
        family inet6;
        route-filter ::/0 exact;
    }
    then reject;
}
term BOGONS-INET {
    from {
        family inet;
        prefix-list-filter BOGONS orlonger;
    }
    then reject;
}
term BOGONS-INET6 {
    from {
        family inet6;
        prefix-list-filter BOGONS-INET6 orlonger;
    }
    then reject;
}
term BOGON-ASNS {
    from as-path-group BOGON-ASNS;
    then reject;
}
term BIGNETWORKS {
    from as-path-group BIGNETWORKS;
    then reject;
}
term RPKI-CHECK {
    from policy RPKI-CHECK;
}
term RPKI-INVALID {
    from community origin-validation-state-invalid;
    then reject;
}
term NORMALIZE {
    then {
        local-preference 500;
        next term;
    }
}
}
}
policy-options {
    policy-statement IMPORT-CUSTOMER-AS123 {
        term INET {
            from {
                prefix-list-filter AS123 exact;
            }
            then {
                community add MYCUSTOMER;
                accept;
            }
        }
        term INET6 {
            from {
                prefix-list-filter AS123-INET6 exact;
            }
        }
    }
}

```

```

        then {
            community add MYCUSTOMER;
            accept;
        }
    }
    term MORE-SPECIFIC-INET {
        from {
            policy MORE-SPECIFIC-UPTO-24;
            prefix-list-filter AS123 orlonger;
        }
        then {
            community add MYCUSTOMER;
            accept;
        }
    }
    term MORE-SPECIFIC-INET6 {
        from {
            policy MORE-SPECIFIC-UPTO-48;
            prefix-list-filter AS123-INET6 orlonger;
        }
        then {
            community add MYCUSTOMER;
            accept;
        }
    }
    term MOST-SPECIFIC-INET {
        from {
            prefix-list-filter AS123 orlonger;
        }
        then {
            community add RFC-NO-EXPORT;
            accept;
        }
    }
    term MOST-SPECIFIC-INET6 {
        from {
            prefix-list-filter AS123-INET6 orlonger;
        }
        then {
            community add RFC-NO-EXPORT;
            accept;
        }
    }
}
policy-options {
    policy-statement REJECT-ALL {
        then reject;
    }
}

```

```

set policy-options policy-statement TRANSIT-CUSTOMER-GENERIC term DEFAULT-INET from family inet
set policy-options policy-statement TRANSIT-CUSTOMER-GENERIC term DEFAULT-INET from route-
filter 0.0.0.0/0 exact
set policy-options policy-statement TRANSIT-CUSTOMER-GENERIC term DEFAULT-INET then reject
set policy-options policy-statement TRANSIT-CUSTOMER-GENERIC term DEFAULT-INET6 from family inet6
set policy-options policy-statement TRANSIT-CUSTOMER-GENERIC term DEFAULT-INET6 from route-
filter ::/0 exact

```

```

set policy-options policy-statement TRANSIT-CUSTOMER-GENERIC term DEFAULT-INET6 then reject
set policy-options policy-statement TRANSIT-CUSTOMER-GENERIC term BOGONS-INET6 from family inet
set policy-options policy-statement TRANSIT-CUSTOMER-GENERIC term BOGONS-INET6 from prefix-list-
filter BOGONS or longer
set policy-options policy-statement TRANSIT-CUSTOMER-GENERIC term BOGONS-INET6 then reject
set policy-options policy-statement TRANSIT-CUSTOMER-GENERIC term BOGONS-INET6 from family inet6
set policy-options policy-statement TRANSIT-CUSTOMER-GENERIC term BOGONS-INET6 from prefix-list-
filter BOGONS-INET6 or longer
set policy-options policy-statement TRANSIT-CUSTOMER-GENERIC term BOGONS-INET6 then reject
set policy-options policy-statement TRANSIT-CUSTOMER-GENERIC term BOGON-ASNS from as-path-
group BOGON-ASNS
set policy-options policy-statement TRANSIT-CUSTOMER-GENERIC term BOGON-ASNS then reject
set policy-options policy-statement TRANSIT-CUSTOMER-GENERIC term BIGNETWORKS from as-path-
group BIGNETWORKS
set policy-options policy-statement TRANSIT-CUSTOMER-GENERIC term BIGNETWORKS then reject
set policy-options policy-statement TRANSIT-CUSTOMER-GENERIC term RPKI-CHECK from policy RPKI-CHECK
set policy-options policy-statement TRANSIT-CUSTOMER-GENERIC term RPKI-
INVALID from community origin-validation-state-invalid
set policy-options policy-statement TRANSIT-CUSTOMER-GENERIC term RPKI-INVALID then reject
set policy-options policy-statement TRANSIT-CUSTOMER-GENERIC term NORMALIZE then local-
preference 500
set policy-options policy-statement TRANSIT-CUSTOMER-GENERIC term NORMALIZE then next term

set policy-options policy-statement IMPORT-CUSTOMER-AS123 term INET from prefix-list-
filter AS123 exact
set policy-options policy-statement IMPORT-CUSTOMER-AS123 term INET then community add MYCUSTOMER
set policy-options policy-statement IMPORT-CUSTOMER-AS123 term INET then accept
set policy-options policy-statement IMPORT-CUSTOMER-AS123 term INET6 from prefix-list-filter AS123-
INET6 exact
set policy-options policy-statement IMPORT-CUSTOMER-AS123 term INET6 then community add MYCUSTOMER
set policy-options policy-statement IMPORT-CUSTOMER-AS123 term INET6 then accept
set policy-options policy-statement IMPORT-CUSTOMER-AS123 term MORE-SPECIFIC-INET from policy MORE-
SPECIFIC-UPTO-25
set policy-options policy-statement IMPORT-CUSTOMER-AS123 term MORE-SPECIFIC-INET from prefix-list-
filter AS123 or longer
set policy-options policy-statement IMPORT-CUSTOMER-AS123 term MORE-SPECIFIC-
INET then community add MYCUSTOMER
set policy-options policy-statement IMPORT-CUSTOMER-AS123 term MORE-SPECIFIC-INET then accept
set policy-options policy-statement IMPORT-CUSTOMER-AS123 term MORE-SPECIFIC-INET6 from policy MORE-
SPECIFIC-UPTO-48
set policy-options policy-statement IMPORT-CUSTOMER-AS123 term MORE-SPECIFIC-INET6 from prefix-list-
filter AS123-INET6 or longer
set policy-options policy-statement IMPORT-CUSTOMER-AS123 term MORE-SPECIFIC-
INET6 then community add MYCUSTOMER
set policy-options policy-statement IMPORT-CUSTOMER-AS123 term MORE-SPECIFIC-INET6 then accept
set policy-options policy-statement IMPORT-CUSTOMER-AS123 term MOST-SPECIFIC-INET from prefix-list-
filter AS123 or longer
set policy-options policy-statement IMPORT-CUSTOMER-AS123 term MOST-SPECIFIC-
INET then community add RFC-NO-EXPORT
set policy-options policy-statement IMPORT-CUSTOMER-AS123 term MOST-SPECIFIC-INET then accept
set policy-options policy-statement IMPORT-CUSTOMER-AS123 term MOST-SPECIFIC-INET6 from prefix-list-
filter AS123-INET6 or longer
set policy-options policy-statement IMPORT-CUSTOMER-AS123 term MOST-SPECIFIC-
INET6 then community add RFC-NO-EXPORT
set policy-options policy-statement IMPORT-CUSTOMER-AS123 term MOST-SPECIFIC-INET6 then accept

set policy-options policy-statement REJECT-ALL then reject

```

The BGP Configuration

The final part of the configuration for importing customer routes is to configure the policies as import policies on the BGP session with the customer. Remember that the first policy, `TRANSIT-CUSTOMER-GENERIC`, is generic and needs to be defined only once; the second policy is AS specific. Each customer has three policies chained together:

1. The generic policy that rejects the unwanted routes;
2. The customer-specific policy that accepts the customer's routes;
3. A final reject policy that rejects everything else.

For instance:

```
group CUST-AS123 {
  type external;
  import [ TRANSIT-CUSTOMER-GENERIC IMPORT-CUSTOMER-AS123 REJECT-ALL ];
  export [ EXPORT-FULL-TABLE REJECT-ALL ];
  remove-private;
  peer-as 123;
  neighbor 1.2.3.4;
}
```

Another customer would be:

```
group CUST-AS456 {
  type external;
  import [ TRANSIT-CUSTOMER-GENERIC IMPORT-CUSTOMER-AS456 REJECT-ALL ];
  export [ EXPORT-FULL-TABLE REJECT-ALL ];
  remove-private;
  peer-as 456;
  neighbor 5.6.7.8;
}
```

Announcing Routes to Customers

Your customers receive your network's routes, which enables them to use your network for sending traffic. This can be either a full table, or a default route. If you wish to send your customers a default route, be sure to generate one by creating a discard default route.

IMPORTANT While we do offer cut-and-paste commands below to announce a default route to customers, you need to have a default route in the first place if you wish to announce it; accepting it from a transit provider may not be what you want, and generating a default route can have adverse effects on your network if done in the wrong place (remember that the default route will make it to all your routers through your IGP). Therefore think this through carefully. We are deliberately not including a cut-and-paste command to generate the default route in your network.

After reading the examples for full table and default route exports, it should be easy for you to create a policy catering to those customers that wish to receive a full table as well as a default route.

Announcing a Full Table to Customers

The policy for sending a full table is more complicated than it would seem. It is necessary to drop routes that have the NO-EXPORT community, for instance. And don't forget to add a term that accepts the routes that your own network originates – a very common mistake is to send your transit customer only the routes that you have learned via EBGp (which is BGP default behavior), and not include your own routes:

```
policy-statement EXPORT-FULL-TABLE {
  term DENY {
    from community RFC-NO-EXPORT;
    then reject;
  }
  term EXPORT-ORIGINATES {
    from community MYROUTES;
    then accept;
  }
}
term EXPORT-BGP {
  from protocol bgp;
  then {
    accept;
  }
}
```

Or in cut-and-paste form:

```
set policy-options policy-statement EXPORT-FULL-TABLE term DENY from community RFC-NO-EXPORT
set policy-options policy-statement EXPORT-FULL-TABLE term DENY then reject
set policy-options policy-statement EXPORT-FULL-TABLE term EXPORT-ORIGINATES from community MYROUTES
set policy-options policy-statement EXPORT-FULL-TABLE term EXPORT-ORIGINATES then accept
set policy-options policy-statement EXPORT-FULL-TABLE term EXPORT-BGP from protocol bgp
set policy-options policy-statement EXPORT-FULL-TABLE term EXPORT-BGP then accept
```

Announcing a Default to Customers

The policies for sending a default route to customers are easy:

```
policy-statement EXPORT-DEFAULT {
  term INET-DEFAULT {
    from {
      family inet;
      route-filter 0.0.0.0/0 exact;
    }
    then {
      accept;
    }
  }
}
term INET6-DEFAULT {
```

```

    from {
        family inet6;
        route-filter ::/0 exact;
    }
    then {
        accept;
    }
}

```

Or in cut-and-paste commands:

```

set policy-options policy-statement EXPORT-DEFAULT term INET-DEFAULT from family inet
set policy-options policy-statement EXPORT-DEFAULT term INET-DEFAULT from route-
filter 0.0.0.0/0 exact
set policy-options policy-statement EXPORT-DEFAULT term INET-DEFAULT then accept
set policy-options policy-statement EXPORT-DEFAULT term INET6-DEFAULT from family inet6
set policy-options policy-statement EXPORT-DEFAULT term INET6-DEFAULT from route-filter ::/0 exact
set policy-options policy-statement EXPORT-DEFAULT term INET6-DEFAULT then accept

```

This will work if you have a default route in your network already. Remember our warning above in case you don't have one.

The BGP Configuration

The final part of configuration for exporting routes to customers is to configure the policies as export policies on the BGP session with the customer. All export policies are generic and need to be defined only once. Each customer has two policies chained together:

1. The generic policy that accepts either default or full table routes.
2. A final reject policy that rejects everything else.

For instance:

```

group CUST-AS123 {
    type external;
    import [ TRANSIT-CUSTOMER-GENERIC IMPORT-CUSTOMER-AS123 REJECT-ALL ];
    export [ EXPORT-FULL-TABLE REJECT-ALL ];
    remove-private;
    peer-as 123;
    neighbor 1.2.3.4;
}

```

Another customer would be:

```

group CUST-AS456 {
    type external;
    import [ TRANSIT-CUSTOMER-GENERIC IMPORT-CUSTOMER-AS456 REJECT-ALL ];
    export [ EXPORT-DEFAULT REJECT-ALL ];
    remove-private;
    peer-as 456;
    neighbor 5.6.7.8;
}

```


Filtering Peering Routes

The policies for peers are a little different from those for customers, but not by much. For peers, you reject the same routes as for customers, and you accept routes inside their AS-SET (if you have automatic prefix filtering implemented). Of course, you reject default routes, RPKI invalids, long AS paths, and bogons. The only major changes are taking a different local preference (lower than customer routes), not adding the BGP community for exporting the peer routes, and not allowing any prefix longer than /24.

Accepting Routes from Peers

This is the entire policy statement for importing routes from peers:

```
policy-options {
  policy-statement PEER-GENERIC {
    term DEFAULT-INET {
      from {
        family inet;
        route-filter 0.0.0.0/0 exact;
      }
      then reject;
    }
    term DEFAULT-INET6 {
      from {
        family inet6;
        route-filter ::/0 exact;
      }
      then reject;
    }
    term BOGONS-INET {
      from {
        family inet;
        prefix-list-filter BOGONS orlonger;
      }
      then reject;
    }
    term BOGONS-INET6 {
      from {
        family inet6;
        prefix-list-filter BOGONS-INET6 orlonger;
      }
      then reject;
    }
    term BOGON-ASNS {
      from as-path-group BOGON-ASNS;
      then reject;
    }
    term BIGNETWORKS {
      from as-path-group BIGNETWORKS;
      then reject;
    }
    term RPKI-CHECK {
      from policy RPKI-CHECK;
    }
  }
}
```

```

    term RPKI-INVALID {
        from community origin-validation-state-invalid;
        then reject;
    }
    term NORMALIZE {
        then {
            local-preference 200;
            next term;
        }
    }
}
}
policy-options {
    policy-statement IMPORT-PEER-AS789 {
        term INET {
            from {
                prefix-list-filter AS789 exact;    # if you have automatic filter generation
            }
            then {
                accept;
            }
        }
        term INET6 {
            from {
                prefix-list-filter AS789-INET6 exact; # if you have automatic filter generation
            }
            then {
                accept;
            }
        }
        term MORE-SPECIFIC-INET {
            from {
                policy MORE-SPECIFIC-UPTO-24;
                prefix-list-filter AS789 orlonger;    # if you have automatic filter generation
            }
            then {
                accept;
            }
        }
        term MORE-SPECIFIC-INET6 {
            from {
                policy MORE-SPECIFIC-UPTO-48;
                prefix-list-filter AS789-INET6 orlonger; # if you have automatic filter generation
            }
            then {
                accept;
            }
        }
    }
}
}
policy-options {
    policy-statement REJECT-ALL {
        then reject;
    }
}
}

```

Or in cut-and-paste form:

```

set policy-options policy-statement PEER-GENERIC term DEFAULT-INET from family inet
set policy-options policy-statement PEER-GENERIC term DEFAULT-INET from route-filter 0.0.0.0/0 exact
set policy-options policy-statement PEER-GENERIC term DEFAULT-INET then reject
set policy-options policy-statement PEER-GENERIC term DEFAULT-INET6 from family inet6
set policy-options policy-statement PEER-GENERIC term DEFAULT-INET6 from route-filter ::/0 exact
set policy-options policy-statement PEER-GENERIC term DEFAULT-INET6 then reject
set policy-options policy-statement PEER-GENERIC term BOGONS-INET from family inet
set policy-options policy-statement PEER-GENERIC term BOGONS-INET from prefix-list-
filter BOGONS orlonger
set policy-options policy-statement PEER-GENERIC term BOGONS-INET then reject
set policy-options policy-statement PEER-GENERIC term BOGONS-INET6 from family inet6
set policy-options policy-statement PEER-GENERIC term BOGONS-INET6 from prefix-list-filter BOGONS-
INET6 orlonger
set policy-options policy-statement PEER-GENERIC term BOGONS-INET6 then reject
set policy-options policy-statement PEER-GENERIC term BOGON-ASNS from as-path-group BOGON-ASNS
set policy-options policy-statement PEER-GENERIC term BOGON-ASNS then reject
set policy-options policy-statement PEER-GENERIC term BIGNETWORKS from as-path-group BIGNETWORKS
set policy-options policy-statement PEER-GENERIC term BIGNETWORKS then reject
set policy-options policy-statement PEER-GENERIC term RPKI-CHECK from policy RPKI-CHECK
set policy-options policy-statement PEER-GENERIC term RPKI-INVALID from community origin-validation-
state-invalid
set policy-options policy-statement PEER-GENERIC term RPKI-INVALID then reject
set policy-options policy-statement PEER-GENERIC term NORMALIZE then local-preference 200
set policy-options policy-statement PEER-GENERIC term NORMALIZE then next term

set policy-options policy-statement IMPORT-PEER-AS789 term INET from prefix-list-filter AS789 exact
set policy-options policy-statement IMPORT-PEER-AS789 term INET then accept
set policy-options policy-statement IMPORT-PEER-AS789 term INET6 from prefix-list-filter AS789-
INET6 exact
set policy-options policy-statement IMPORT-PEER-AS789 term INET6 then accept
set policy-options policy-statement IMPORT-PEER-AS789 term MORE-SPECIFIC-INET from policy MORE-
SPECIFIC-UPTO-24
set policy-options policy-statement IMPORT-PEER-AS789 term MORE-SPECIFIC-INET from prefix-list-
filter AS789 orlonger
set policy-options policy-statement IMPORT-PEER-AS789 term MORE-SPECIFIC-INET then accept
set policy-options policy-statement IMPORT-PEER-AS789 term MORE-SPECIFIC-INET6 from policy MORE-
SPECIFIC-UPTO-48
set policy-options policy-statement IMPORT-PEER-AS789 term MORE-SPECIFIC-INET6 from prefix-list-
filter AS789-INET6 orlonger
set policy-options policy-statement IMPORT-PEER-AS789 term MORE-SPECIFIC-INET6 then accept

set policy-options policy-statement REJECT-ALL then reject

```

The BGP Configuration

The final part of configuration for importing peer routes is to configure the policies as import policies on the BGP session with the peer. Remember that the first policy, `PEER-GENERIC`, is generic and needs to be defined only once; the second policy is AS specific. Each peer has three policies chained together:

1. The generic policy that rejects the unwanted routes.
2. The peer-specific policy that accepts the peer's routes.
3. A final reject policy that rejects everything else.

For instance:

```
group PEER-AS789 {
    type external;
    import [ PEER-GENERIC IMPORT-PEER-AS789 REJECT-ALL ];
    export [ EXPORT-PEER REJECT-ALL];
    remove-private;
    peer-as 789;
    neighbor 1.2.3.4;
}
```

Announcing Routes to Peers

Your peers should receive your own routes and the routes of your customers. Fortunately, we have used BGP communities to tag these routes, so it is very simple to create the relevant policy statements:

```
policy-statement EXPORT-PEER {
    term DENY {
        from community RFC-NO-EXPORT;
        then reject;
    }
    term EXPORT-ORIGINATES {
        from community MYROUTES;
        then accept;
    }
}
term EXPORT-CUSTOMER {
    from {
        protocol bgp;
        community MYCUSTOMER;
    }
    then {
        accept;
    }
}
```

Or in cut-and-paste form:

```
set policy-options policy-statement EXPORT-PEER term DENY from community RFC-NO-EXPORT
set policy-options policy-statement EXPORT-PEER term DENY then reject
set policy-options policy-statement EXPORT-PEER term EXPORT-ORIGINATES from community MYROUTES
set policy-options policy-statement EXPORT-PEER term EXPORT-ORIGINATES then accept
set policy-options policy-statement EXPORT-PEER term EXPORT-CUSTOMER from protocol bgp
set policy-options policy-statement EXPORT-PEER term EXPORT-CUSTOMER from community MYCUSTOMER
set policy-options policy-statement EXPORT-PEER term EXPORT-CUSTOMER then accept
```

The BGP Configuration

The final part of the configuration for exporting routes to peers is to configure the policies as export policies on the BGP session with the peer. All export policies are generic and need to be defined only once. Each peer has two policies chained together:

1. The generic policy that accepts your routes and customer routes.

2. A final reject policy that rejects everything else.

For instance:

```
group PEER-AS789 {
  type external;
  import [ PEER-GENERIC IMPORT-PEER-AS789 REJECT-ALL ];
  export [ EXPORT-PEER REJECT-ALL];
  remove-private;
  peer-as 789;
  neighbor 1.2.3.4;
}
```

Filtering Transit Routes

The policies for your transit providers are once again a little different from those for peers, but again not by much. For transits, you reject the same routes as for peers: default routes, RPKI invalids, long AS paths, and bogons. The only major changes are accepting routes from the “big networks” instead of rejecting them, taking a different local preference (lower than peer routes), not adding the BGP community for exporting the transit routes, and accepting all remaining routes instead of rejecting them.

Accepting Routes from Transits

This is the entire policy statement for importing routes from transits:

```
policy-options {
  policy-statement TRANSIT-GENERIC {
    term DEFAULT-INET {
      from {
        family inet;
        route-filter 0.0.0.0/0 exact;
      }
      then reject;
    }
    term DEFAULT-INET6 {
      from {
        family inet6;
        route-filter ::/0 exact;
      }
      then reject;
    }
    term BOGONS-INET {
      from {
        family inet;
        prefix-list-filter BOGONS orlonger;
      }
      then reject;
    }
    term BOGONS-INET6 {
      from {
        family inet6;
        prefix-list-filter BOGONS-INET6 orlonger;
      }
    }
  }
}
```

```

        }
        then reject;
    }
    term BOGON-ASNS {
        from as-path-group BOGON-ASNS;
        then reject;
    }
    term RPKI-CHECK {
        from policy RPKI-CHECK;
    }
    term RPKI-INVALID {
        from community origin-validation-state-invalid;
        then reject;
    }
    term NORMALIZE {
        then {
            local-preference 100;
            next term;
        }
    }
}
}
policy-options {
    policy-statement IMPORT-TRANSIT-AS10 {
        term INET {
            from {
                family inet;
                route-filter 0.0.0.0/0 upto /24;
            }
            then accept;
        }
        term INET6 {
            from {
                family inet6;
                route-filter ::/0 upto /48;
            }
            then accept;
        }
    }
}
}
policy-options {
    policy-statement REJECT-ALL {
        then reject;
    }
}
}

```

Or in cut-and-paste form:

```

set policy-options policy-statement TRANSIT-GENERIC term DEFAULT-INET from family inet
set policy-options policy-statement TRANSIT-GENERIC term DEFAULT-INET from route-
filter 0.0.0.0/0 exact
set policy-options policy-statement TRANSIT-GENERIC term DEFAULT-INET then reject
set policy-options policy-statement TRANSIT-GENERIC term DEFAULT-INET6 from family inet6
set policy-options policy-statement TRANSIT-GENERIC term DEFAULT-INET6 from route-filter ::/0 exact
set policy-options policy-statement TRANSIT-GENERIC term DEFAULT-INET6 then reject
set policy-options policy-statement TRANSIT-GENERIC term BOGONS-INET from family inet
set policy-options policy-statement TRANSIT-GENERIC term BOGONS-INET from prefix-list-
filter BOGONS orlonger

```

```

set policy-options policy-statement TRANSIT-GENERIC term BOGONS-INET then reject
set policy-options policy-statement TRANSIT-GENERIC term BOGONS-INET6 from family inet6
set policy-options policy-statement TRANSIT-GENERIC term BOGONS-INET6 from prefix-list-
filter BOGONS-INET6 orlonger
set policy-options policy-statement TRANSIT-GENERIC term BOGONS-INET6 then reject
set policy-options policy-statement TRANSIT-GENERIC term BOGON-ASNS from as-path-group BOGON-ASNS
set policy-options policy-statement TRANSIT-GENERIC term BOGON-ASNS then reject
set policy-options policy-statement TRANSIT-GENERIC term RPKI-CHECK from policy RPKI-CHECK
set policy-options policy-statement TRANSIT-GENERIC term RPKI-INVALID from community origin-
validation-state-invalid
set policy-options policy-statement TRANSIT-GENERIC term RPKI-INVALID then reject
set policy-options policy-statement TRANSIT-GENERIC term NORMALIZE then local-preference 200
set policy-options policy-statement TRANSIT-GENERIC term NORMALIZE then next term

set policy-options policy-statement IMPORT-TRANSIT-AS10 term INET from family inet
set policy-options policy-statement IMPORT-TRANSIT-AS10 term INET from route-
filter 0.0.0.0/0 upto /24
set policy-options policy-statement IMPORT-TRANSIT-AS10 term INET then accept
set policy-options policy-statement IMPORT-TRANSIT-AS10 term INET6 from family inet6
set policy-options policy-statement IMPORT-TRANSIT-AS10 term INET6 from route-filter ::/0 upto /48
set policy-options policy-statement IMPORT-TRANSIT-AS10 term INET6 then accept

set policy-options policy-statement REJECT-ALL then reject

```

The BGP Configuration

The final part of configuration for importing transit routes is to configure the policies as import policies on the BGP session with the transit provider. Remember that the first policy, `TRANSIT-GENERIC`, is generic and needs to be defined only once; the second policy is AS specific. Each transit has three policies chained together:

1. The generic policy that rejects the unwanted routes.
2. The transit-specific policy that accepts the transit's routes.
3. A final reject policy that rejects everything else.

For instance:

```

group TRANSIT-AS10 {
  type external;
  import [ TRANSIT-GENERIC IMPORT-TRANSIT-AS10 REJECT-ALL ];
  export [ EXPORT-TRANSIT REJECT-ALL ];
  remove-private;
  peer-as 10;
  neighbor 1.2.3.4;
}

```

Announcing Routes to Transits

Your transits should receive your own routes and the routes of your customers. As

with exporting routes to peers, here we use BGP communities to tag these routes, so it is very simple to create the relevant policy statements:

```
policy-statement EXPORT-TRANSIT {
  term DENY {
    from community RFC-NO-EXPORT;
    then reject;
  }
  term EXPORT-ORIGINATES {
    from community MYROUTES;
    then accept;
  }
}
term EXPORT-CUSTOMER {
  from {
    protocol bgp;
    community MYCUSTOMER;
  }
  then {
    accept;
  }
}
```

And in cut-and-paste form:

```
set policy-options policy-statement EXPORT-TRANSIT term DENY from community RFC-NO-EXPORT
set policy-options policy-statement EXPORT-TRANSIT term DENY then reject
set policy-options policy-statement EXPORT-TRANSIT term EXPORT-ORIGINATES from community MYROUTES
set policy-options policy-statement EXPORT-TRANSIT term EXPORT-ORIGINATES then accept
set policy-options policy-statement EXPORT-TRANSIT term EXPORT-CUSTOMER from protocol bgp
set policy-options policy-statement EXPORT-TRANSIT term EXPORT-CUSTOMER from community MYCUSTOMER
set policy-options policy-statement EXPORT-TRANSIT term EXPORT-CUSTOMER then accept
```

The BGP Configuration

The final part of configuration for exporting routes to transits is to configure the policies as export policies on the BGP session with the transit. All export policies are generic and need to be defined only once. Each transit has two policies chained together:

1. The generic policy that accepts your routes and customer routes.
2. A final reject policy that rejects everything else.

For instance:

```
group TRANSIT-AS10 {
  type external;
  import [ TRANSIT-GENERIC IMPORT-TRANSIT-AS10 REJECT-ALL ];
  export [ EXPORT-TRANSIT REJECT-ALL ];
  remove-private;
  peer-as 10;
  neighbor 1.2.3.4;
}
```


Conclusion

From a policy perspective your routers now have what it takes to filter routes, create a secure routing table, and talk BGP to your customers, peers, and transit providers. Your network should be in great shape! Well done!

You might have noticed that we did not configure policies to filter routes you originate on customer, peer, and transit sessions. This was done on purpose, as by nature BGP will filter announcements containing your own AS number in order to prevent loops. If you receive a route advertisement containing your prefix but originating from a different AS number, RPKI will filter those if you have created ROAs. It is however never bad to explicit filter out your own routes on all eBGP sessions.

Next, some ways to check if things are going right and what to do if they aren't.

Chapter 5

Troubleshooting

In life things may not go according to plan – and BGP networking is no exception. As you start using your secure routing table and deploying RPKI in your network, customers may notice that in rare situations some networks or IP addresses they want to access have become unreachable.

This chapter discusses steps you can take to find out whether a reported problem is the result of your routing security implementation. Do not expect a full BGP troubleshooting tutorial, just some tips and tricks and a few pointers on where to find additional information that can come in handy.

MORE? A good starting point for troubleshooting BGP is the Juniper Networks TechLibrary: https://www.juniper.net/documentation/en_US/junos/topics/task/verification/bgp-configuration-process-summary.html.

Check If a Destination is Present In the Routing Table

Junos OS uses two databases (tables) for routing information:

- Routing table: Contains all the routing information learned by all routing protocols [RIB].
- Forwarding table: Contains the routes actually used to forward packets [FIB].

Junos OS installs all active routes from the routing table into the forwarding table. The active routes are routes that are used to forward packets to their destinations. The Junos operating system kernel maintains a master copy of the forwarding table. It copies the forwarding table to the Packet Forwarding Engine, which is the component responsible for forwarding packets.

If a customer suggests that a certain prefix seems unreachable, you need to check the route(s) to that prefix. In order to make sure the route has been installed and the destination is reachable from your network, you should look in both the routing and forwarding table.

There are three ways to check the different tables:

- `show route <destination-prefix>`: This will display the routing table entries.
- `show route forwarding-table destination <destination-prefix>`: This will show the routing engine's version of the destination prefix in the forwarding table.
- `show pfe route ip prefix <destination-prefix>`: This will show the forwarding table entry that is actually installed in each PFE.

In a perfect world the destination prefix will be visible in all three scenarios and should show an RPKI `validation-state: valid` if it has a valid ROA and passed the RPKI validation.

Check RPKI Validation State of a Route

In order to check if a route is used, it is essential to be able to display information about the route validation database when RPKI route validation is configured. You can query all route validation records that match a given prefix or `origin-autonomous-system`. In addition, you can filter the output by a specific RPKI cache session:

```
user@R1> show route
inet.0: 3 destinations, 3 routes (2 active, 0 holddown, 1 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
              Unusable
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
```

Validation states can be any of the states defined in RFC 6811:

- Valid
- Invalid
- Unknown

But also, another state that means “validation was not run against this at all”:

- Unverified

Unverified is different from unknown; a route that is unverified might be any of valid, invalid, or unknown, if validation were attempted. Unverified basically means that Origin Validation (RPKI) simply isn't enabled or isn't running on your router.

MORE? For additional commands visit the Juniper TechLibrary: https://www.juniper.net/documentation/en_US/junos/topics/reference/command-summary/show-validation-database.html.

Check If the RPKI Validator is Reachable and the Database is Up-to-Date

show validation statistics

This command shows statistics about the validation database. Obviously if you have enabled RPKI and the connection with the validator is working, you should see entries in the database:

```
user@host> show validation statistics

Total RV records:          453455
  Total Replication RV records: 453455
    Prefix entries:        35432
    Origin-AS entries:     124400
Memory utilization: 16.31MB
Policy origin-validation requests: 234995
  valid:                   23445
  invalid:                 14666
  unknown:                 34567
BGP import policy reevaluation notifications: 460268
  inet.0:                  435345
  inet6.0:                  3454
```

TechLibrary reference: https://www.juniper.net/documentation/en_US/junos/topics/reference/command-summary/show-validation-statistics.html.

show validation database

This command shows you the actual content of the database:

```
user@host> show validation database
```

RV database for instance master

Prefix	Origin-AS	Session	State	Mismatch
172.16.1.0/24-32		1 10.0.77.1	valid	
172.16.2.0/24-32		2 10.0.77.1	valid	

```

172.16.3.0/24-32          3 10.0.77.1      valid
172.16.4.0/24-32          4 10.0.77.1      valid
172.16.5.0/24-32          5 10.0.77.1      valid
172.16.6.0/24-32          6 10.0.77.1      valid
172.16.7.0/24-32          7 10.0.77.1      valid
172.16.8.0/24-32          8 10.0.77.1      valid
72.9.224.0/19-24        26234 192.168.1.100  valid *
72.9.224.0/19-24        3320 192.168.1.200  invalid *
10.0.0.0/8-32           0 internal      valid

```

IPv4 records: 14

IPv6 records: 0

TechLibrary reference: https://www.juniper.net/documentation/en_US/junos/topics/reference/command-summary/show-validation-database.html.

show validation session

If your validation database is empty or you want to check to see if your validators are still up and running, you can do so with the `show validation session brief` and `show validation session detail` commands:

```
user@host> show validation session brief
```

```

Session              State   Flaps    Uptime #IPv4/IPv6 records
1.3.0.2              up      2      00:01:37 13/0
10.255.255.11        up      3      00:00:01 1/0
10.255.255.12        connect 2      64/68

```

Note that the third session in this output shows `connect` instead of `up`. This is most likely the result of configuring all three sessions under the same RPKI group; if you do so, only two sessions will come up and any additional ones will stay down if you don't change the maximum number of validators per group.

show validation session detail

```
user@host> show validation session detail
```

```

Session 10.0.77.1, State: up
  Group: test, Preference: 100
  Local IPv4 address: 10.0.77.2, Port: 2222
  Refresh time: 300s
  Session flaps: 14, Last Session flap: 5h13m18s ago
  Hold time: 900s
  Record Life time: 3600s
  Serial (Full Update): 0
  Serial (Incremental Update): 0
    Session flaps 2
    Session uptime: 00:48:35
    Last PDU received: 00:03:35
    IPv4 prefix count: 71234
    IPv6 prefix count: 345

```

Re-Run Validation, Optionally, Against Only Specified Routes

When BGP origin validation is configured and for some reason the database gets corrupted, or you would like to refresh it, manually request a route validation policy to be reevaluated. This command causes dependent route validation records to be reevaluated. Dependent route validation records are exactly matching and more specific records:

```
user@host> request validation policy
```

```
Enqueued 1 IPv4 records  
Enqueued 0 IPv6 records
```

TechLibrary reference: https://www.juniper.net/documentation/en_US/junos/topics/reference/command-summary/request-validation-policy.html.

Appendix

How to Automatically Update Prefix Lists

Throughout this book you have worked extensively with prefix lists. A prefix list contains one or more prefixes that can be used in routing policies. When you build your network, you usually start with manually-created and manually-updated prefix lists. Your own set of prefixes (the ones that you will originate) doesn't change very often; you accept everything from your transit providers anyway, and when you do not have a lot of peers or customers there are not many changes to their prefix lists, either. We've all been there.

However, as your network and your customer base grow, it is not feasible to keep manually updating prefix lists. In this Appendix we show you how we use the NETCONF functionality of Junos OS, along with a few Python scripts, to automatically upload prefix list changes.

This Appendix is a bit different from the rest of the book, since we cannot provide ready-made scripts that you can cut-and-paste into your network's management system. Instead we show you how we do it, and invite you to talk to your software development team, encouraging them to be enthusiastic about your new secure routing table, too, and have them implement automatic prefix list changes on your network.

There are three basic steps:

1. Make sure that for each customer, your customer database contains the AS-SET (or AS number) that your customers announce to you.
2. Retrieve the prefix list for that AS-SET from the IRR.
3. Upload the prefix list to your router(s).

Doing this periodically (for instance, twice a day) means that you don't have to worry about customers who wish to change their prefix list. They do not have to create tickets asking you to change their prefix list, and what's more: that means you don't have to handle these tickets any longer.

Customer Database

Your database needs to contain at least two fields for each customer:

- Their AS number (we will use this in the name of the prefix list as it is uploaded to the router).
- Their AS-SET.

Your customer will have created their own AS-SET in the IRR. When connecting your customer to the network, you will check to see if their AS-SET makes sense. It should contain the AS number that they use for their own network (from which they will announce their own prefixes) as well as the AS numbers and/or AS-SETs of their customers. By walking through this “tree”, the `bgpq3` tool will create one single prefix list that collapses all prefixes that your customer may announce.

Retrieve Prefix-list: Use `bgpq3`

The tool to convert AS-SETs to prefix lists is `bgpq3`. The official `bgpq3` web site is at <https://github.com/snar/bgpq3> where you are invited to read, learn, and contribute. After installing `bgpq3` on a machine, you can use it to create prefix lists like the one below:

```
user@server:~$ bgpq3 -h rr.ntt.net -J -l AS123 AS-EXAMPLE
policy-options {
replace:
  prefix-list AS123 {
    192.0.2.0/24;
  }
}
```

And for IPv6:

```
user@server:~$ bgpq3 -6 -h rr.ntt.net -J -l AS123-INET6 AS-EXAMPLE
policy-options {
replace:
  prefix-list AS123-INET6 {
    2001:db8::/32;
  }
}
```

Using `bgpq3`, create a `.txt` file in the directory `prefix-list.d/` for each AS-SET that you wish to auto-update. The `.txt` file should have the name of the prefix list as you want it pushed to the router; and the contents should be a prefix per line. Creating a script that does this periodically is left as an exercise to the reader.

Upload to Your Router

The Junos OS supports multiple ways of uploading external information into the configuration database. Since this is an Appendix, it is not exhaustive and does not offer a ready-made, field-tested configuration. Only an example is provided here. In the example, the NETCONF interface for Junos OS is used to upload information. This example specifically uses the Python 3 language and the Junos PyEZ library created and maintained by Juniper Networks. Note that PyEZ may also be used with Python 2.7 if preferable.

MORE? Check out the *Day One* PyEZ book here: <https://www.juniper.net/us/en/training/jnbooks/day-one/automation-series/junos-pyez-cookbook/>.

In this example, a fictitious Juniper device is used as the configuration target. However, the example should work on any of the Junos OS devices that use prefix lists in the same manner. Our requirement is to update the list of IP prefixes present in the prefix list based on the contents of a text file. The script that will accomplish this is run from a remote host.

Requirements

For the Junos OS device:

- Have the NETCONF SSH subsystem (see: https://www.juniper.net/documentation/en_US/junos/topics/topic-map/netconf-ssh-connection.html) enabled through:
- `# set system services netconf ssh`
- Allow access from remote server on NETCONF port (default: 830)

For the remote server:

- Have Python 3.5 installed
- (alternatively, use Python 2 and adjust syntax accordingly in code below)
- Have Junos PyEZ installed (see: <https://github.com/Juniper/py-junos-eznc>), for example, through:
- `# pip install junos-eznc`

Example Code

The Python 3 code below looks for *.txt files in a directory named `prefix-list.d` inside the parent directory of the script itself. Each such file represents a prefix list to update and should contain prefixes, one per line, that the prefix list should consist of. Empty lines and lines starting with the hash (#) symbol are ignored. The

name of the file (excluding the extension suffix) is used as the name of the prefix list.

Update the configuration variables HOST, USER, and PASS at the top of the script to match your setup. Refer to the comments inside the script for further explanation:

<code>

```
from pathlib import Path

from jnpr import junos
from jnpr.junos.utils.config import Config
from lxml import etree as ET

HOST = 'mx0.example.com'
USER = 'automation'
PASS = 'secret'

# Prefix lists to update.
# Dictionary of {name -> list of prefixes}.
prefix_lists = {}

# Find files matching 'prefix-list.d/*.txt'
# relative to the parent directory of the script.
CONF_DIR = Path(__file__).parent / 'prefix-list.d'

for f in CONF_DIR.glob('*.txt'):
    with f.open('r', encoding='utf-8') as fp:
        # Use the file basename ('stem') as the prefix list name;
        # read, split & trim lines, exclude comments ('#') and empty.
        lines = [x.strip() for x in fp.read().splitlines()]
        prefix_lists[f.stem] = [x for x in lines if x and not x.startswith('#')]

# Connect to configuration target.
# Note: should use agent auth if PASS is None.
print('Connect to [%s]...' % HOST)

with junos.Device(host=HOST, user=USER, passwd=PASS) as device:
    # Open configuration; use private mode to avoid
    # committing existing shared candidate configuration.
    print('Open private configuration...')

    with Config(device, mode='private') as config:
        # Merge XML config for each prefix list.
        for name, prefixes in prefix_lists.items():
            # Generate an XML configuration with the following structure:
            """
            <configuration>
              <policy-options>
                <prefix-list replace="replace">
                  <name>{name}</name>

                  <prefix-list-item>
                    <name>x.x.x.x/n</name>
                  </prefix-list-item>
            """
```

```

        (...)
    </prefix-list>
</policy-options>
</configuration>
"""

patch = ET.Element('configuration')
policy_options = ET.SubElement(patch, 'policy-options')
# Note: replace entire prefix-list to remove extraneous entries.
prefix_list = ET.SubElement(policy_options, 'prefix-list', replace='replace')
ET.SubElement(prefix_list, 'name').text = name

for prefix in prefixes:
    item = ET.SubElement(prefix_list, 'prefix-list-item')
    ET.SubElement(item, 'name').text = prefix

print('\nUpdate prefix list [%s] with [%d] entries:' % (name, len(prefixes)))
print('---\n' + ET.tounicode(patch, pretty_print=True) + '---')

config.load(patch)

print('\nUpdated [%d] prefix lists; config diff:' % len(prefix_lists))
print('---\n' + config.diff() + '---')

while True:
    choice = input('\nDo you want to commit these changes? [y/n] ').lower()
    if choice in ('y', 'yes'):
        print('Committing changes...')
        config.commit()
        print('Commit successful!')
        break
    elif choice in ('n', 'no'):
        print('Changes not committed.')
        break

```

</code>

This code is probably not a cut-and-paste into your own network management system, but it should definitely get you up and running, and ready for automated prefix list changes.

The Next Step: Use the Same Mechanism for Filtering Your Peers

Developers are creatively writing software, and network engineers are creatively using it!

Based on the work described above, not only are you now creating prefix lists to filter your customers, you can use the same mechanism to filter your peers as well; simply create a ‘pseudo customer’ in your network management system, note down the AS-SET that the peer should announce, and then you can refer to the prefix-list in the BGP policy for that peer.

This way you can be sure that your customers, as well as your peers, have a degree of security in the prefixes that they are allowed to announce.

Additional Resources

Now that you have a secure routing table, the next step is to express to customers and partners that you are taking responsibility for a safer, more stable Internet.

One way to express this is to join MANRS (Mutually Agreed Norms for Routing Security). MANRS is a global initiative, supported by the Internet Society, that provides crucial fixes to reduce the most common routing threats. More information is available on their website at: <https://www.manrs.org/>.

In order to get your customers to announce valid, usable information to you, you may have to help them to fix their announcements, IRR registrations, or RPKI ROAs. The more networks that join, the more secure the Internet becomes!

There's great documentation within the Juniper TechLibrary that can help:

- *Enabling BGP to Carry Flow-Specification Routes*: https://www.juniper.net/documentation/en_US/junos/topics/example/routing-bgp-flow-specification-routes.html.
- *BGP Feature Guide for Routing Devices*: https://www.juniper.net/techpubs/en_US/junos15.1/information-products/pathway-pages/config-guide-routing/config-guide-routing-bgp.pdf.

And these frequently asked questions compiled by NLnet Labs are very handy on RPKI specific issues: <https://nlnetlabs.nl/projects/rpki/faq/>.

Conclusion

The authors hope that the information provided in this book will help you get started designing and implementing methods for a secure routing table, and likewise a more stable and reliable network.

It is very likely, and maybe even the wish of the authors, that parts of this book soon become obsolete due to all the work being done to get to a more secure routing table. If parts of this book do become obsolete, we will have achieved our goal!

You might have noticed that implementing routing security can consume quite a bit of time. However, we hope that you will take the time to do so, or even better, help build the tools to do so – there are already many tools and scripts available on sources like Github, which you can use to cut down on time spent. And remember that we all have the same goal, so let us know if you think you can help or have suggestions on how to improve those tools.

If you have any ideas, suggestions, or remarks on how to further develop routing security feel free to reach out to the authors (dayone@juniper.net). By joining forces and helping each other we can make the Internet a safer place for all!