HAW HAMBURG

Bachelorarbeit

Marcel Röthke

A Test Framework for RPKI Prefix Validation in BGP Implementations

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Marcel Röthke

A Test Framework for RPKI Prefix Validation in BGP Implementations

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Thomas Schmidt
Zweitgutachter: Prof. Dr. Klaus-Peter Kossakowski

Eingereicht am: 04. Januar 2020

Marcel Röthke

Thema der Arbeit

Ein Test-Framework für RPKI Präfix Validierung in BGP Implementierungen

Stichworte

BGP, Präfix-Hijacking, RPKI, RTR, framework, testen

Kurzzusammenfassung

Im Internet wird mit Border Gateway Protocol (BGP) geroutet. Durch die vertrauensbasierte Natur des Protokolls ist es aber anfällig für Angriffe. Um diese Angriffe zukünftig zu verhindern, wurde die Resource Public Key Infrastructure (RPKI) entwickelt. Sie erlaubt es IP-Präfix Besitzern, Betreibern von Autonomen Systemen (ASes) explizit die Benutzung ihrer Präfixe zu gestatten, und bietet damit eine Möglichkeit, die Annoncierung von Präfixen zu validieren. Diese Validierung in Border Gateway Protocol (BGP) Implementierungen zu integrieren ist allerdings nicht trivial. In dieser Bachelorarbeit wird ein Framework zum Testen der Korrektheit dieser Implementierungen entwickelt. Das Framework wird evaluiert, indem es für eine solche Implementierung angepasst und einige konkrete Tests auf Basis des Frameworks implementiert werden. Die Ausführung dieser Tests half beim Finden mehrerer Fehler in der getesteten Implementierung.

Marcel Röthke

Title of Thesis

A Test Framework for RPKI Prefix Validation in BGP Implementations

Keywords

BGP, Prefix-Hijacking, RPKI, RTR, framework, testing

Abstract

Routing on the internet happens via the Border Gateway Protocol (BGP). Due to its trust based nature it is susceptible to attacks. To mitigate these attacks the Resource

Public Key Infrastructure (RPKI) has been developed. This allows IP-Prefix owners to explicitly allow certain Autonomous Systms (ASes) to announce it and thereby provide a way to validate prefix announcements. Integrating this validation into Border Gateway Protocol (BGP) implementations is not trivial though. In this bachelor thesis a framework to test the correctness of the validation results produced by these integrations is developed. The framework is evaluated by adapting it to a specific BGP implementation and implementing a set of concrete tests with the framework. Running these tests helped to find several bugs in the tested implementation.

Contents

List of Figures

List of Tables

Abbreviations

Adj-RIB-In Adjacent-Routing-Information-Base-Incoming.

Adj-RIB-Out Adjacent-Routing-Information-Base-Outgoing.

AS Autonomous System.

ASN Autonomous System Number.

BGP Border Gateway Protocol.

eBGP Exterior Border Gateway Protocol.

EGP Exterior Gateway Protocol.

FIB Forwarding Information Base.

IANA Internet Assigned Numbers Authority.

iBGP Interior Border Gateway Protocol.

IETF Internet Engineering Task Force.

IGP Interior Gateway Protocol.

LIR Local Internet Registry.

Loc-RIB Local Routing Information Base.

NRO Number Resource Organization.

PID Process Identifier.

RIR Regional Internet Registry.

ROA Route Origin Authorization.

RPKI Resource Public Key Infrastructure.

RTR protocol RPKI to Router Protocol.

1 Introduction

The Internet is a vast and complex network of networks. It is controlled not by a single party, but by many with a multitude of interests. Coordinating all involved parties to accomplish good reachability throughout the Internet is—even in the absence of malicious intent—a highly non trivial task. Unfortunately and unsurprisingly not everyone *is* acting with good intentions and additional complexity is added to defend against malicious actors.

The networks making up the Internet are called Autonomous System (AS). One AS consists of at least one IP-Prefix that it can advertise to its neighbors, who must assume that the prefixes advertised to them are coming from their actual owner. This reliance on trust opens up several possible attack vectors. In 2008 [37] one of these vectors was inadvertently used. An Internet service provider tried to block access to a popular video hosting platform in their own network and announced that they own the IP-Prefix of the hosting platform to their neighbors, making it globally inaccessible for several hours. Incidents like this are not rare [27] but few have enough impact to be widely noticeable. Still, there are enough [36, 7, 32, 33] widely known cases for this to be worrisome.

Whether intentional or not, defenses against this kind of attack are required. Therefore, Resource Public Key Infrastructure (RPKI) was introduced. It allows to verify the origin of a prefix announcement and to make appropriate decisions if an AS announces a prefix it is not allowed to. Depending on the policies used incorrect validation could lead to similar reachability problems that an attack would. It is therefore paramount that proper testing is conducted.

2 Background

2.1 IP Address Allocation

IP addresses are allocated in a hierarchical manner as blocks of multiple addresses called IP-Prefix. The global oversight lies with the Internet Assigned Numbers Authority (IANA). Subordinate to the IANA are the Regional Internet Registry (RIR). The five RIR (Table 2.1) are responsible for different geographical regions, and they are united in the Number Resource Organization (NRO) for joint activities, projects and policy coordination. The IANA hands out large prefixes to the RIRs for further distribution to the Local Internet Registries (LIRs). The LIRs are companies and academic Institutions allocating IP-Addresses for their customers, users, members, or own use.

2.2 IP-Routing

IP-Routing is the process of finding paths through IP Networks. Such routing information may be configured statically by the administrator or dynamically via routing protocols. Networks can be divided into two planes: the control plane and the data plane (also sometimes called forwarding plane). Routing operates on the control plane. The data plane uses the knowledge collected and processed by the control plane to decide what the next hop is when forwarding a package. A routing protocol acting within a network

AFRINIC	Africa
APNIC	East Asia, Oceania, South Asia, and Southeast Asia
ARIN	Antarctica, Canada, parts of the Caribbean, and the United States
LACNIC	most of the Caribbean and all of Latin America
RIPE NCC	Europe, Central Asia, Russia, and West Asia

Table 2.1: The RIR and the area they serve

is called an Interior Gateway Protocol (IGP). A protocol routing between networks is called an Exterior Gateway Protocol (EGP).

Apart from their routing scope routing protocols can also be categorized by how they work. Two general categories are common: vector based and link state routing.

2.2.1 Link State Routing

Link state protocols exchange information about every adjacent link with every other node in the Network. Based on that, every router builds a graph representing the entire network with itself at the center. The best path is than calculated by every node individually to fill the routing table.

2.2.2 Vector Based Routing

Unlike link state protocols, vector based protocols do not have a full view of the entire topology. They only receive the information from their neighbors about the targets that are reachable through them.

Vector based routing protocols can be further subdivided into distance vector and path vector protocols. Their primary difference is that path vector protocols also provide information about the Path an announcement took through the network.

2.3 Autonomous System (AS)

The term AS has been formally defined by the Internet Engineering Task Force (IETF) as " …connected group of one or more IP prefixes run by one or more network operators which has a SINGLE and CLEARLY DEFINED routing policy. " [14, p. 3] and they are identified with their Autonomous System Number (ASN). ASN where originally two Byte integer but have later been extended to four byte, because the two byte ASN pool threatened to reach exhaustion. They are allocated similarly to IP Addresses. With the IANA allocating blocks to the RIRs which in turn allocate to their members. Typical operators of ASes are Internet Service Providers or Companies with a large Internet presence.

2.4 Border Gateway Protocol (BGP)

BGP [24] is a Path Vector Routing Protocol and the dominant exterior gateway routing protocol. It is the only exterior gateway routing protocol on the public Internet, where it is used to route between ASes. Being initially developed for IPv4 routing, it has since been extended [3] to route different protocols, including IPv6. BGP can be used for interior gateway routing as well. That variant is called Interior Border Gateway Protocol (iBGP) in contrast to Exterior Border Gateway Protocol (eBGP).

The exchange of reachability information in BGP is heavily governed by policy not just by connectivity, which is an expression of the commercial contracts between peers [13].

2.4.1 Route Selection

When a new route is received the degree of preference is calculated and the route is installed into the Adjacent-Routing-Information-Base-Incoming (Adj-RIB-In) for the sending peer. The preference is calculated according to local policies or in case of iBGP might be taken from the update itself. It is usually a positive integer, a negative one means that the route is infeasible or has been rejected for other reasons. It will not be considered during later stages of the selection process.

Once the preference has been determined feasible routes from all Adj-RIBs-In are then installed into the Local Routing Information Base (Loc-RIB). Routes with identical destinations are compared by their preference, the route with the highest preference is selected. If necessary, additional rules are applied to break ties.

One of the additional rules used for tie breaking is based on the length of the AS-Path. Unless custom policies say otherwise a route with a shorter AS-Path will be preferred over alternatives with a longer AS-Path.

After all selected routes are installed into the Loc-RIB, the Forwarding Information Base (FIB) is populated from the Loc-RIB and the route dissemination process is started. During the process every route in the Loc-RIB is processed and according to policy installed into the Adjacent-Routing-Information-Base-Outgoing (Adj-RIB-Out) for the corresponding peers. The content of the individual Adj-RIBs-Out is then advertised to the corresponding peers.

NO_EXPORT	Destinations in this community must not be advertised
NO_ADVERTISE	Destinations in this community must not be advertised to other BGP peers.
BLACKHOLE	Traffic to destinations in this community should be dropped.

Table 2.2: Examples for well-known Communities

2.4.2 Communities

BGP communities [8, 30, 15] are a way to communicate additional information to neighboring and remote peers. This allows to group announcements sharing a common property in a distributed way. AS administrators can freely choose which community an announcement belongs to. That choice is often based on the destination, but could also be based on other attributes of the announcement like the AS-Path. Any announcement that is not explicitly assigned to a specific community belongs to the general Internet community.

It is in general unspecified what it means for a destination to be in a certain community, the interpretation is up to the administrator. But there are well known communities (Table 2.2) with specific meanings. It is also possible to register communities with the IANA and assign specific meaning to them.

Later [30] extensions for BGP introduced a way to specify if a community is transitive across ASes.

2.5 BGP Prefix Hijacking

BGP does not prevent unauthorized parties from crafting invalid announcements. If such an invalid announcement is used to direct traffic for parts or even all of the Internet to a certain AS it is called Prefix Hijacking [2]. Here a AS claims to originate a prefix it does not actually originate or claims to have a better route to the prefix that it actually has. This prefix could be unassigned or belong to another AS. If this attack is successful a set of ASes direct traffic to the wrong AS. How large this set is depends on the exact method that is used to execute the attack and on the routing policies of other ASes. There are several ways to hijack a prefix. One is to modify the AS path attribute. Either by shortening the AS-Path attribute to make it more attractive or even truncating it completely and putting oneself at the beginning and thereby claiming ownership of the

Prefix	Maximal Length	ASN
10.10.0.0/16	20	64496

Table 2.3: An example ROA

prefix. The latter method has more potential, but is also easier to detect because it creates a multiple origin AS situation. Another method is to announce a more specific prefix than the owner does which would hijack all traffic to the specific prefix. This method has the downside that—unlike other methods—can not be used to intercept traffic. All methods have in common that they can be used to make a target unreachable by dropping all traffic or to impersonate the service provided by the legitimate destination.

2.6 RPKI

RPKI [19] provides parts of an Infrastructure to prevent prefix hijacking. It does this by enabling an entity to assert that it is the owner of one or multiple IP-Prefixes or ASN and allows them to explicitly permit a specific set of ASes to announce its IP-Prefixes.

To accomplish this the RPKI contains a cryptographically verifiable copy of the IP-Prefix allocation hierarchy. This allows everyone to verify if a certain entity owns a given prefix. However, this alone does not help in mitigating attacks, because there is still no mapping from ASN to legal entities. And even if it were, just because an entity does not own a prefix does not mean it is not authorized to announce it. This problem is solved by Route Origin Authorization (ROA).

2.6.1 Route Origin Authorization (ROA)

ROA are part of the RPKI and provide positive attestation for prefix announcements. With a ROA an entity can explicitly allow a certain set of AS to announce their prefix. It (example in Table 2.3) contains the prefix, the maximal length a sub-prefix may be announced with, and the ASN of the AS being allowed to announce it. This allows everyone to verify if a prefix announcement is originating at a AS that is allowed to do so.

2.6.2 Prefix Origin Validation

ROAs allow validating [17] prefix announcements in an automated way. Validation works by first looking for the prefix in all known ROAs. If at least one matching one is found, the maximal length allowed by matching ROAs is compared to the prefix from the route announcement. If a ROA with matching prefix and sufficient maximal length is found, the ASN is compared. The validation has three possible results:

- VALID
  The route announcement matches a ROA.
- INVALID
  The route announcement does not match a ROA but the prefix was found. The match can fail because the announced ASN does not match the ASN in the ROA or because the prefix is longer than the ROA allows.
- NOTFOUND
  The route announcement does not match any known ROA.

Validating prefixes as they are received is not enough though. Because the RPKI is not static, it may change at any time. New ROAs are created or old ones expire. When a new ROA is announced or withdrawn every route matching the ROAs prefix regardless of the length must be revalidated, even those with a longer prefix than the maximal length specified by the ROA. Because in case of a withdrawal there might be prefixes validated as INVALID because of its prefix length that now validate as NOTFOUND.

2.6.3 Deployment

A typical RPKI deployment looks as in figure 2.1. ROAs from the global RPKI are cryptographically validated by trusted local caches. Routers then retrieve the validated data from the caches using the RPKI to Router Protocol (RTR protocol) [5, 6] and use that data to validate received announcements.

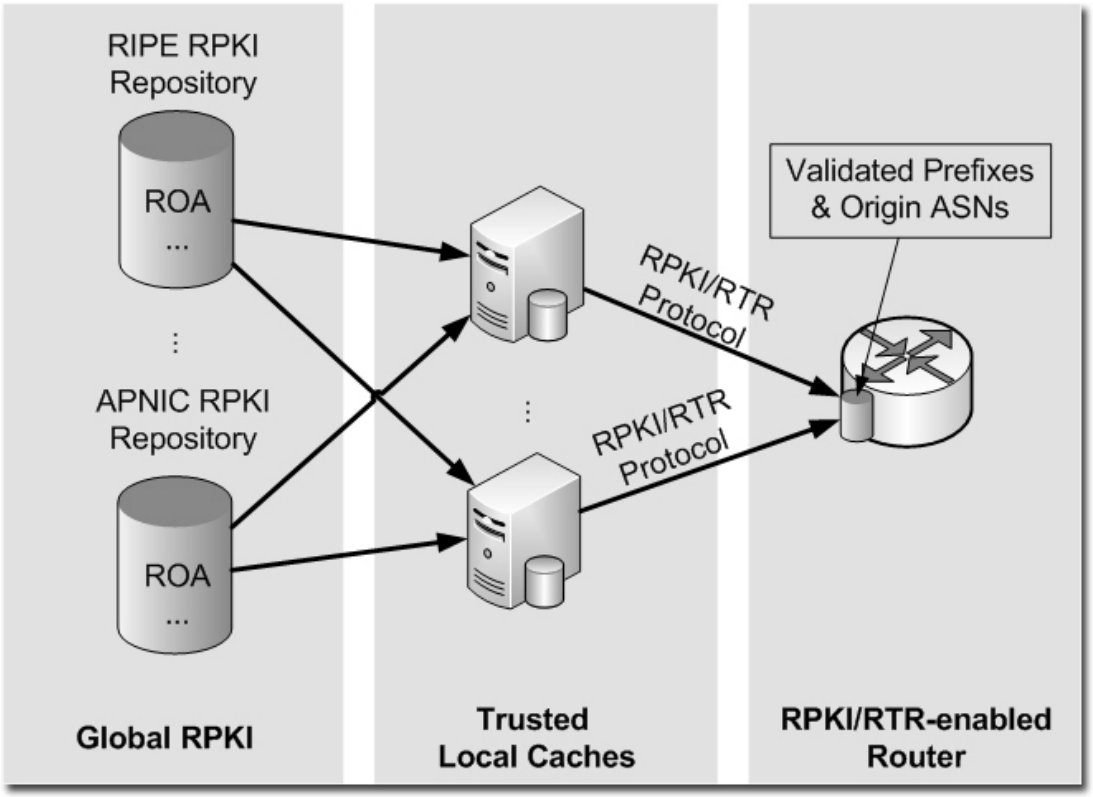

Figure 2.1: RPKI deployment [34]

2.7 RTRlib

RTRlib [35] implements everything a BGP router needs to perform prefix origin validation [20]. The RTR protocol [5, 6] to retrieve ROAs from the trusted cache server, a connection manager to keep connections to multiple cache servers at the same time and handle failover to secondary servers if the primary servers become unavailable, a data structure to efficiently store and retrieve the data, as well as the prefix origin validation. In addition RTRlib provides a set of cli tools to examine or export the ROAs stored in a cache server and perform prefix origin validation.

3 Requirements

The framework should be able to test different implementations with as little modification as possible. Therefore insertion of the test data and extraction of the test result should not rely on implementation specific interfaces if possible.

The framework must be able to assess the correctness of the validation under various circumstances with respect to the ROA, the test subject received from the RPKI cache server. First in a stable situation where neither the routing table nor RPKI changes. Second if validation stays correct when existing routes have to be revalidated because of RPKI changes.

It should furthermore be easy to extend the framework if additional functionality is needed in the future.

4 Design of the Framework

4.1 Design Considerations

This test framework is supposed to test whether a BGP implementation is correctly validating route announcements based on the ROAs provided. The validation shall be tested for static route and ROA configurations, and for cases in which route announcements, ROA-data, or both change. Properly testing this requires that the correct validation result is known to the test.

To accomplish this the route announcements and the ROA the test subject receives should be completely controlled by the test framework. It should also be possible to dynamically announce additional routes, withdraw routes, and add or remove ROAs. It must be possible to retrieve the validation result at a certain point in time to test them for correctness. The validation should not be reimplemented in the framework, but the test should know each correct result.

Since the goal of this test framework is to test as many implementations as possible, it should avoid the use of non-standard protocols for communication with the test subject. This includes the retrieval of validation results. Ideally, the only time implementation specific interfaces are used is at the beginning of the test run to apply the necessary configuration for the framework to communicate with the test subject via standard protocols.

4.2 Components

Several components are necessary to accomplish these goals. First we need the test subject which must be an RPKI enabled BGP implementation. Meaning that it must support prefix origin validation [20] and retrieve ROAs dynamically via the RTR protocol [5, 6].

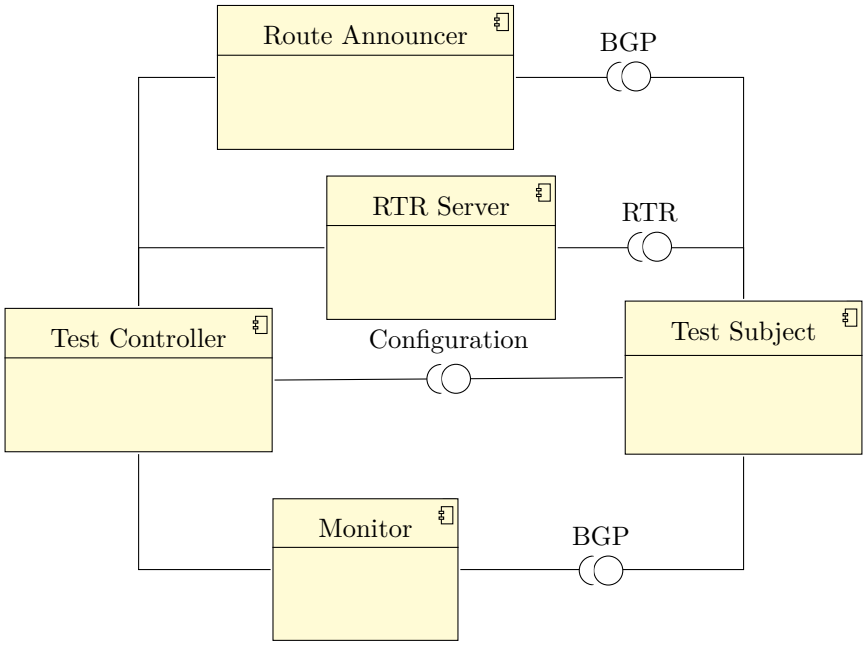

Figure 4.1: Component Diagram for the framework

A component that is capable of announcing and withdrawing routes to the test subject dynamically. A component that can communicate dynamic ROA configurations to the test subject via the RTR protocol. A component that can extract the validation results from the test subject. It does not need to do so continuously, but must be able to do so several times within the same test. Finally, a component orchestrating the tests—the test controller—is needed.

The components and their interfaces, which are described in the next chapter, can be seen in Figure 4.1

4.3 Interfaces

There are several important interfaces that need to be considered. The test subject has four interfaces. It needs a way to receive route announcements, a way to receive RPKI data, a way to retrieve the validation results, and a way to apply the initial configuration.

Route announcer, monitor and ROA installer need to interface with the test subject to fulfill their objectives and have interfaces for the test controller.

Announcing routes to the test subject should happen via BGP, since it is a well defined standard protocol that every BGP implementation speaks by definition.

Retrieval of the validation results on the other hand is more intricate, because BGP does not export detailed RPKI validation results by default. By configuring policies to drop announcements with certain validation results a lot of information could be inferred. But since there are three possible results, one could never be sure why a specific prefix was announced or not announced depending on which of the two states is overloaded. At least not without reconfiguring the test subject during the test. One could fall back to using implementation specific interfaces to retrieve validation results, but that would require implementations specific adaptions. The test subject could instead be configured to attach the validation results to its own BGP announcements in a standard compliant way. Luckily a suitable mechanism exists with BGP communities and a draft standard [18] to communicate exactly the kind of information we want. Said standard describes a non transitive BGP community to signal the results of route origin validation with the announcement.

RPKI data—in particular the ROA—can simply be transferred via the RTR protocol, as it is supported by most BGP implementations that support route origin validation.

4.4 Orchestration

Coordination of the different components is required to conduct the tests. The framework must ensure that, all tests have their dependencies instantiated, and initialized prior to executing the test and properly tear them down after finishing their execution.

5 Implementation

5.1 Choice of Components

Even though routing software is usually written in languages like C, Python was chosen for the test framework. This decision was made partly because of the authors familiarity with the language, the availability of excellent test frameworks and the hope it would ease integration of large software component used in the framework.

Since it is not realistic to write a standard compliant BGP and RTR server implementation as part of this work, external components are used to handle these protocols.

5.1.1 General Purpose Test Framework

Since test frameworks have many things in common, general purpose test frameworks exist that handle common task, including running of the individual tests and handling of their dependencies. One such framework is pytest [16]. It is an excellent, well established test framework that is not limited to writing tests for Python software. It has excellent support for dependency handling for individual tests and provides easily readable test reports.

5.1.2 BGP Speaker

There are several open source BGP implementations, the most active right now are:

- BIRD [10]
- FRR [12]
- GoBGP [29]

- ExaBGP [11]

BIRD and GoBGP are well established BGP implementations but lack good interfaces for automation interfaces. FRR has increasing support for instrumentation through a YANG [4] based API but that has not yet been extended to the BGP components.

ExaBGP on the other hand has been explicitly written to be used for automating BGP communication, it is furthermore written in Python which should simplify integration.

5.1.3 RTR Server

There are three open source RTR Server implementations available today.

- RIPE NCC RPKI Validator [25]
- RIPE NCC RPKI Validator 3 [26]
- Routinator [21]

Another is GoRTR [9] but it has not been closely evaluated as part of this work.

None of the evaluated ones provide an API to control the ROAs it serves. This makes sense as they are all intended for use in productions setups and therefore need to cryptographically validate the RPKI. They all download the RPKI directly from the RIRs. However, the RIPE NCC RPKI Validator 3 consist of two independent components: The validator, dealing with the cryptographic validation and providing a web interface and an API to access the validated data, and the RTR Server, which requests the validated ROAs periodically from the validator and serves them via the RTR protocol.

Since the API that provides the ROAs to the RTR Server can easily be emulated and therefore is the only option that does not require a custom RPKI, the RIPE NCC RPKI Validator 3 seems to be the best choice.

5.1.4 Test Subject

As noted, the test subject must be a RPKI enabled BGP implementation. To simplify development it would be good if it were a pure software implementation that works on a home computer. Being open source would also be a plus as it would allow us to analyze any bugs we find with this work. Five implementations are available:

1. OpenBGPD [22]
2. GoBGP [29]
3. BIRD [10]
4. Quagga [23]
5. FRR [12]

OpenBGPD is a stable well working BGP implementation with support for prefix origin validation, unfortunately it does not support the RTR protocol but only statically configured ROA. GoBGP and BIRD are both good candidates as they support prefix origin validation and dynamic ROA retrieval via the RTR protocol. Quagga does not have support for prefix origin validation or the RTR protocol in any official release, but there are patches [1] available based on RTRlib. These patches have almost no chance of being merged though as upstream development has basically ceased. FRR is an actively developed fork of Quagga, support for RPKI has been implemented based on the code written for Quagga. But there has not been systematic testing of the functionality since it was merged. The RPKI support in FRR was in part developed by us.

5.2 Framework

To implement the framework external components needed to be configured and adapters for their use in pytest needed to be written. Pytest already supports everything necessary to implement the actual tests in python with these adapters.

5.2.1 RTR Server

To integrate the RTR Server it was necessary to export ROA data as JSON (example in Listing 5.1) and serve them via HTTP. The RTR Server requests the data in regular intervals. One problem was that by default the interval is 30 seconds. Waiting up to 30 seconds after every ROA change would unnecessarily delay test execution. Unfortunately the RTR Server does not allow to configure the interval. It needs to be changed in the source code. To archive reasonable reaction time the interval was lowered to 1 second.

```
{
    "data": {
        "ready": true,
        "roas": [
            {
                "asn": "44489",
                "maxLength": 24,
                "prefix": "185.131.60.0/22"
            },
            {
                "asn": "44489",
                "maxLength": 25,
                "prefix": "109.164.0.0/17"
            }
        ]
    }
}
```

Listing 5.1: ROAs serialized as JSON for the RTR Server

5.2.2 Route Announcer and Monitor

The route announcer and monitor share most of their adaption code, as both use ExaBGP. They primarily differ in their configuration. Despite it being written in Python it was not straightforward to write the adapter. ExaBGP assumes that the controlling process is a subprocess of itself and that it uses its standard input and output streams to control ExaBGP. But the process running the test framework is the parent. This was solved by writing a simple script that writes its Process Identifier (PID) to a configured location and then sleeps indefinitely. With the PID it is possible to open the standard input and output streams of the process ExaBGP sees as its controller and use them to send commands. This solution was chosen over proxying commands through the controlling process because of its simplicity.

```
process add-remove {
        run ../write_pid.py /tmp/exabgp_sender.pid;
        encoder json;
}
```

```
6  neighbor 127.0.0.1 {
7          router-id 1.2.3.4;
8          local-address 127.0.0.3;
9          local-as 3;
10         peer-as 1;
11         group-updates false;

13         api {
14                 processes [ add-remove ];
15         }
16 }
```

Listing 5.2: ExaBGP configuration for the route announcer

The configuration of the announcer can be seen in listing 5.2. It has two sections: The process section and the neighbor section. The process section describes the command that is used to start the controlling process and configures the desired encoding of the communication. The neighbor section configures the BGP neighbor and how the ExaBGP instance should identify itself with that neighbor. It also configures which process definition should be used for this neighbor.

The notable difference in the monitors configuration is the API section. Line 2 to 5 in Listing 5.3 tell ExaBGP to send every received BGP packet in a JSON based format to the monitor. This allows continuous monitoring of the validation results.

```
1          api {
2                  receive {
3                          parsed;
4                          update;
5                  }
6                  processes [ add-remove ];
7          }
```

Listing 5.3: Excerpt from the ExaBGP configuration for the monitor

5.2.3 Test Subject

The adapter for our example test subject was straightforward to implement, as it only needs to start it with the required parameters and stop it when it is not needed anymore.

Listing 5.4 shows the parameters passed to the test subject at startup. '-Z' disables communication with other components of the routing suite, they are not necessary for these tests to work. '-n' disables route installation into the kernel, as we do not actually want to route anything and do not have enough permissions in the test system to install any routes anyway. '-S' tells the test subject to not check if it has enough permissions, which we do not need due to the prior option. '-M…' loads the rpki module. '-p1179' sets the bgp listening port of the test subject to a port outside the well known range, which we cannot use due to limited privileges.

```
bgpd -Z -n -S -Mbgpd_rpki.so -p1179
```

Listing 5.4: Command line to start the test subject

The configuration, however, needed more care. Listing 5.5 shows the configuration as an example for IPv4 only. The first line is not strictly necessary, but without it the test subject would wait several seconds before starting to evaluate policies—called route maps here—which is not necessary for this test and would needlessly delay it. Lines on 2 and 3 configure the identity of the subject itself, which is ASN 1 and router id 127.0.0.1. Lines 4 to 9 tell the test subject about its neighbors and how to reach them, they also enable passive mode—the test subject only listens for incoming connections but does not try to initiate one—and BGP multihop for both peers—allows to reach peers that are more than one hop away. Lines 11 to 16 set policies for the IPv4 routes and enable soft reconfiguration. Soft reconfiguration ensures that the Adj-RIB-In is not discarded after route selection is over, it is necessary to perform revalidation in case of ROA changes. The policies set earlier are configured for outgoing announcements in line 18 and 19. This ensures that routes are announced with a specific next-hop address. In this case this is done to simplify comparisons of announced and received routes within the test framework. Lines 21 to 31 configure the policies for incoming routes. In this case it simply performs prefix origin validation and puts the announcement in a bgp community. The communities chosen are based on [18] which proposes to use a non-transitive community with the values as shown in Table 5.1. The test subject unfortunately does not support setting arbitrary non-transitive communities from policies. To circumvent this a regular community is used with the same values.

```
1 bgp route-map delay-timer 1
2 router bgp 1
3  bgp router-id 127.0.0.1
4  neighbor 127.0.0.2 remote-as 2
```

Value	Meaning
0	valid
1	not found
2	invalid

Table 5.1: Mapping from prefix origin validation result to BGP community value

```
 neighbor 127.0.0.2 passive
 neighbor 127.0.0.2 ebgp-multihop 64
 neighbor 127.0.0.3 remote-as 3
 neighbor 127.0.0.3 passive
 neighbor 127.0.0.3 ebgp-multihop 64
 !
 address-family ipv4 unicast
  neighbor 127.0.0.2 soft-reconfiguration inbound
  neighbor 127.0.0.2 route-map next-hop-out4 out
  neighbor 127.0.0.3 soft-reconfiguration inbound
  neighbor 127.0.0.3 route-map rpki in
 exit-address-family
!
route-map next-hop-out4 permit 1
 set ip next-hop 10.0.0.1
!
route-map rpki permit 1
 match rpki invalid
 set community 1:0
!
route-map rpki permit 2
 match rpki notfound
 set community 1:1
!
route-map rpki permit 3
 match rpki valid
 set community 1:2
!
rpki
  rpki cache 127.1 8323 preference 5
  exit
```

Listing 5.5: Excerpt from the test subjects configuration

5.3 Adapter Tests

To ensure that the individual components and their adapters work as expected, several tests for the framework itself have been implemented. Especially the adapters for the route announcer, the monitor, and the test subject need to be tested whenever a new test subject is integrated.

The RTR cache server is tested by checking if static as well as dynamic ROA configurations are correctly applied. Since the test framework has no direct way to monitor the available ROAs in the cache server, it was necessary to implement a simple monitor that retrieves the ROAs from the cache server via the RTR protocol. RTRlib with its python binding [28] is used as the client implementation.

Testing the test subject, announcer, and monitor individually would require significant additional work and additional use of implementation specific interfaces. Therefore these components are tested together as one component. The test simply announces and withdraws prefixes via the announcer and checks their presence via the monitor. Missing prefixes or prefix attributes can provide hints to narrow down the cause.

6 Evaluation

The framework has been evaluated by implementing and conducting concrete tests on the test subject that has been integrated during implementation. Every test is conducted twice: once with an initially empty routing table and once with a pre-filled one. The pre-fill is a random selection of 10000 routes from a real world routing table. While these routes might not be representative for every possible routing table, it might be—and as explained later actually is—enough to expose or hide bugs in the used lookup algorithm.

The first set of tests is aimed at verifying if the prefix origin validation works correctly for static ROA and route configurations. To assess this four tests have been implemented, each testing a specific validation result. Invalid results are split into invalid length and invalid AS. The validation does not distinguish between them, but they are tested individually nonetheless because there could be bugs that effect one but not the other. All tests in this first set use the ROAs shown in table 6.1. Each of these tests uses a

Prefix	Maximal Length	ASN
192.168.0.0/16	24	400
2000::/16	32	400

Table 6.1: ROA used in the first set of basic tests

small set of route announcements hand picked to test corner cases with the available ROAs. To test valid announcements two different routes with the extreme prefix lengths are announced per IP version (Table 6.2). The test for checking announcements that

Prefix	ASN
192.168.0.0/16	400
192.168.1.0/24	400

Table 6.2: IPv4 prefixes and their ASN to test validation of valid prefixes

validate as not-found uses prefixes representing the extreme again. First prefixes that are just to short to match one of the provided ROAs are announced—one that is too

long is not announced because it should validate as invalid—and secondly a prefix that is completely unrelated to the provided ROAs is announced. Invalid length is tested by announcing prefixes that exceed the valid length by one bit (Table 6.3). Testing for

Prefix	ASN
192.168.128.0/25	400
2000:124::/33	400

Table 6.3: IP prefixes and their ASN to test validation of invalid length prefixes

correct validation of prefixes with an invalid ASN is similarly simple. The prefixes are otherwise valid but with a different origin ASN (Table 6.4). It might also be interesting

Prefix	ASN
192.168.0.0/16	200
2000:123::/31	600

Table 6.4: IP routes to test validation of invalid as prefixes

to check if the validation result is still correct if multiple problems exist with a given prefix, however, that is left for future work.

In addition to these simple static tests an additional test that covers the entire range of a ROA has been implemented. This is an extended version of the simple valid test and would of course not be feasible for a ROA with a very long prefix range. Therefore sufficiently small ROAs with only 2^7 possible prefixes are used (Table 6.5).

Prefix	Maximal Length	ASN
192.168.0.0/16	24	400
2000::/32	40	400

Table 6.5: ROA used for testing the full prefix range

After the above described static tests two dynamic tests have been implemented. The purpose of these tests is to check if revalidation is working correctly. Both tests use a set of static routes and dynamic ROAs. The second test is specifically designed for the corner case of prefixes with an invalid length. These are prefixes that are outside the range of the ROA. One could be tempted to optimize the revalidation code to only check prefixes within the range of updated ROAs and thereby missing those with an invalid length.

The first revalidation test starts with the basic set of ROAs and announces a valid pair of prefixes to the test subject. After checking that the validation result is indeed valid, the ROAs are removed. Since there are no ROAs left, now the validation result must be not-found. After establishing that this is indeed the result the test subject came to, the same set of ROAs is announced again but with a different ASN. The prefixes are now expected to validate as invalid.

The final revalidation test again uses the default set of ROAs. It starts by announcing two sets of prefixes. The first set (Table 6.6) contains only initially valid routes, while the second set (Table 6.7) contains initially invalid routes. After confirming that all announced routes have the expected initial validation result, the ROAs are removed again. Both sets should now validate as not-found.

Prefix	ASN
192.168.128.0/23	400
192.168.64.0/24	400
2000:1000::/30	400
2000:123::/32	400

Table 6.6: First set of prefixes for the invalid length revalidation set

Prefix	ASN
192.168.32.0/25	400
192.168.16.0/26	400
2000:1234::/33	400
2000:1234::/34	400

Table 6.7: Second set of prefixes for the invalid length revalidation set

With this set of tests two bugs were found in the revalidation code path of the test subject, both in the lookup that finds the prefixes that must be revalidated. The first bug is due to an implementation detail of the test subjects BGP table and only happens under very specific circumstances that are very unlikely to be reached in a production deployment. This is because it requires a very short prefix to be announced or a very small BGP table. Both conditions are unlikely to be meet in production because RPKI is only relevant for routers that work with the global BGP routing table—which has hundreds of thousands of prefixes—and prefixes smaller than eight practically do not occur [31] on the Internet. Nonetheless, a fix for this bug has been submitted upstream. This also led to the idea of repeating each test with a pre-populated routing table. If this bug can vanish with a populated table, there might also be bugs that vanish with an

empty one. The other bug that was found is exactly the one anticipated above regarding prefixes with an invalid length. A fix for this bug is currently being worked on.

7 Conclusion and Outlook

The goal of this work was to implement a test framework for RPKI prefix validation in BGP implementations. To accomplish this a modular architecture that could easily accommodate external components was created. Then a survey was conducted in order to find software components that could be used to implement the more complex tasks the framework needs to accomplish, since implementing all required functionality within the available time would otherwise not have been possible. Adapting the framework to test different implementations is easily possible as no tight integration into the test subject exists. All communication after the initialization is conducted using standard protocols. To further ease adaption to other implementations a set of self tests has been implemented that can guide troubleshooting of a new adaption.

A set of concrete tests has been implemented in order to show that the framework is capable of the tasks it was designed for. It could be confirmed that it is able to test the prefix origin validation functionality of a BGP speaker for dynamic ROA configurations without relying on implementation specific interfaces.

For now only static prefix configurations have been tested because it is expected that the code path performing the validation is the same for both static and dynamic configurations as both are validated when ingesting Adj-RIB-In. However, it might still be worthwhile to implement a test for dynamic configurations. The developed framework is already capable of supporting such a test.

To profit as much as possible from this framework it should be integrated into the continues integration pipelines of the test subjects. This is planned for the example test subject FRR.

Additionally GoRTR should be properly evaluated as a possible candidate for a faster and easier to install RTR server implementation.

Bibliography

[1] Andreas Reuter and Colin Sames and Michael Mester and Quagga Community. *Quagga Routing Suite with RPKI support*. URL: https://github.com/rtrlib/quagga-rtrlib (visited on 12/17/2019).

[2] Hitesh Ballani, Paul Francis, and Xinyang Zhang. "A Study of Prefix Hijacking and Interception in the Internet". In: *Proc. of SIGCOMM '07*. Kyoto, Japan: ACM, 2007, pp. 265–276.

[3] T. Bates et al. *Multiprotocol Extensions for BGP-4*. RFC 4760. IETF, 2007.

[4] M. Bjorklund. *The YANG 1.1 Data Modeling Language*. RFC 7950. IETF, 2016.

[5] R. Bush and R. Austein. *The Resource Public Key Infrastructure (RPKI) to Router Protocol*. RFC 6810. IETF, 2013.

[6] R. Bush and R. Austein. *The Resource Public Key Infrastructure (RPKI) to Router Protocol, Version 1*. RFC 8210. IETF, 2017.

[7] Adrian Chadd. *The "AS7007 Incident"*. URL: https://lists.ucc.gu.uwa.edu.au/pipermail/lore/2006-August/000040.html (visited on 01/01/2020).

[8] R. Chandra, P. Traina, and T. Li. *BGP Communities Attribute*. RFC 1997. IETF, 1996.

[9] Cloudflare. *GoRTR*. URL: https://github.com/cloudflare/gortr (visited on 12/13/2019).

[10] CZ.NIC Labs. *BIRD Internet Routing Daemon*. URL: https://bird.network.cz/ (visited on 12/13/2019).

[11] Exa Networks. *ExaBGP*. URL: https://github.com/Exa-Networks/exabgp (visited on 12/17/2019).

[12] FRRouting Project. *FRRouting Protocol Suite*. URL: https://frrouting.org/ (visited on 12/17/2019).

[13] Lixin Gao. "On Inferring Autonomous System Relationships in the Internet". In: *IEEE/ACM Trans. Netw.* 9.6 (2001), pp. 733–745.

[14] J. Hawkinson and T. Bates. *Guidelines for creation, selection, and registration of an Autonomous System (AS)*. RFC 1930. IETF, 1996.

[15] J. Heitz et al. *BGP Large Communities Attribute*. RFC 8092. IETF, 2017.

[16] Holger Krekel and pytest-dev team. *pytest*. URL: https://pytest.readthedocs.io/en/latest/ (visited on 01/01/2019).

[17] G. Huston and G. Michaelson. *Validation of Route Origination Using the Resource Certificate Public Key Infrastructure (PKI) and Route Origin Authorizations (ROAs)*. RFC 6483. IETF, 2012.

[18] Thomas King et al. *Signaling Prefix Origin Validation Results from an RPKI Origin Validating BGP Speaker to BGP Peers*. Internet-Draft – work in progress 03. IETF, 2019.

[19] M. Lepinski and S. Kent. *An Infrastructure to Support Secure Internet Routing*. RFC 6480. IETF, 2012.

[20] P. Mohapatra et al. *BGP Prefix Origin Validation*. RFC 6811. IETF, 2013.

[21] NLnetLabs. *Routinator*. URL: https://github.com/NLnetLabs/routinator (visited on 12/13/2019).

[22] OpenBSD Project. *OpenBGPD*. URL: http://www.openbgpd.org/ (visited on 12/17/2019).

[23] Quagga Community. *Quagga Routing Suite*. URL: https://www.quagga.net/ (visited on 12/17/2019).

[24] Y. Rekhter, T. Li, and S. Hares. *A Border Gateway Protocol 4 (BGP-4)*. RFC 4271. IETF, 2006.

[25] RIPE NCC. *RPKI Validator*. https://github.com/RIPE-NCC/rpki-validator. 2015.

[26] RIPE NCC. *RPKI Validator 3*. URL: https://github.com/RIPE-NCC/rpki-validator-3 (visited on 12/13/2019).

[27] Andrei Robachevsky. *14,000 Incidents: A 2017 Routing Security Year in Review*. Jan. 9, 2018. URL: https://www.internetsociety.org/blog/2018/01/14000-incidents-2017-routing-security-year-review/ (visited on 01/01/2020).

[28] RTRlib maintainers. *RTRlib python binding*. URL: https://github.com/rtrlib/python-binding (visited on 12/19/2019).

[29] Ryu SDN Framework Community. *GoBGP*. 2019. URL: https://osrg.github.io/gobgp/ (visited on 12/17/2019).

[30] S. Sangli, D. Tappan, and Y. Rekhter. *BGP Extended Communities Attribute*. RFC 4360. IETF, 2006.

[31] Stephen Strowes. *Visibility of IPv4 and IPv6 Prefix Lengths in 2019*. 2019. URL: https://labs.ripe.net/Members/stephen_strowes/visibility-of-prefix-lengths-in-ipv4-and-ipv6 (visited on 12/30/2019).

[32] Tao Wan and P. C. van Oorschot. "Analysis of BGP prefix origins during Google's May 2005 outage". In: *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*. 2006, 8 pp.–. DOI: 10.1109/IPDPS.2006.1639679.

[33] Andree Toonk. *BGPstream and The Curious Case of AS12389*. Apr. 27, 2017. URL: https://bgpmon.net/bgpstream-and-the-curious-case-of-as12389/ (visited on 01/01/2020).

[34] Matthias Wählisch. *Beta Version of the RPKI RTR Client C Library Released*. 2011. URL: https://labs.ripe.net/Members/waehlisch/beta-version-of-the-rpki-rtr-client-c-library-released (visited on 12/10/2019).

[35] Matthias Wählisch et al. "RTRlib: An Open-Source Library in C for RPKI-based Prefix Origin Validation". In: *Proc. of USENIX Security Workshop CSET'13*. Berkeley, CA, USA: USENIX Assoc., 2013. URL: https://www.usenix.org/conference/cset13/rtrlib-open-source-library-c-rpki-based-prefix-origin-validation.

[36] *Why Google Went Offline Today and a Bit about How the Internet Works*. http://blog.cloudflare.com/why-google-went-offline-today-and-a-bit-about. Retrieved 2013-08-27. 2012.

[37] *YouTube Hijacking: A RIPE NCC RIS case study*. http://www.ripe.net/internet-coordination/news/industry-developments/youtube-hijacking-a-ripe-ncc-ris-case-study. Retrieved 2013-08-16. 2008.

Glossary

AS-Path Every AS which a route announcement has traversed in the order they were visited.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Ein Test-Framework für RPKI Präfix Validierung in BGP Implementierungen

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	_____
Ort	Datum	Unterschrift im Original