SIDR
Internet-Draft
Intended status: Informational
Expires: September 22, 2016

O. Muravskiy
T. Bruijnzeels
RIPE NCC

March 21, 2016

RPKI Certificate Tree Validation by a Relying Party Tool
draft-ietf-sidr-rpki-tree-validation-00

Abstract

This document currently describes the approach to validate the content of the RPKI certificate tree, as used by the RIPE NCC RPKI Validator. This approach is independent of a particular object retrieval mechanism. This allows it to be used with repositories available over the rsync protocol, the RPKI Repository Delta Protocol, and repositories that use a mix of both.

This algorithm does not rely on content of repository directories, but uses the Authority Key Identifier (AKI) field of a manifest and a certificate revocation list (CRL) objects to discover manifest and CRL objects issued by a particular Certificate Authority (CA). It further uses the hashes of manifest entries to discover other objects issued by the CA.

If the working group finds that algorithm outlined here is useful for other implementations, we may either update future revisions of this document to be less specific to the RIPE NCC RPKI Validator implementation, or we may use this document as a starting point of a generic validation document and keep this as a detailed description of the actual RIPE NCC RPKI Validator implementation.

Table of Contents

1. Introduction

In order to use information published in RPKI repositories, Relying Parties (RP) need to retrieve and validate the content of certificates, CRLs, and other RPKI signed objects. To validate a particular object, one must ensure that all certificates in the certificate chain up to the Trust Anchor (TA) are valid. Therefore the validation of a certificate tree is usually performed top-down, starting from the TA certificate and descending down the certificate chain, validating every encountered certificate and its products. The result of this process is a list of all encountered RPKI objects with a validity status attached to each of them. These results may later be used by a Relying Party in taking routing decisions, etc.

Traditionally RPKI data is made available to RPs through the repositories [RFC6481] accessible over rsync protocol. Relying parties are advised to keep a local copy of repository data, and perform regular updates of this copy from the repository (Section 5 of[RFC6481]). The RPKI Repository Delta Protocol [I-D.ietf-sidr-delta-protocol] introduces another method to fetch repository data and keep the local copy up to date with the repository.

This document describes how a Relying Party tool could discover RPKI objects to download, build certificate path, and validate RPKI objects, independently from what repository access protocol is used. To achieve this, it puts downloaded RPKI objects in an object store, where objects could be found by their URI, hash of their content, value of the object's AKI field, or combination of these. It also keeps track of download and validation time for every object, to perform cleanups of the local copy.

2. Top-down Validation of a Single Trust Anchor Certificate Tree

The validation of a Trust Anchor (TA) certificate tree starts from its TA certificate. To retrieve the TA certificate, a Trust Anchor

Locator (TAL) object is used, as described in Section 2.1.

If the TA certificate is retrieved, it is validated according to the Section 7 of [RFC6487] and Section 2.2 of [RFC7730].

Then the TA certificate and all its subordinate objects are validated as described in Section 2.2.

For all repository objects that were validated during this validation run, their validation timestamp is updated in an object store (see Section 4.1.8).

Outdated objects are removed from the store as described in Section 2.3. This completes the validation of the TA certificate tree.

2.1. Fetching the Trust Anchor Certificate Using the Trust Anchor Locator

The following steps are performed in order to fetch the Trust Anchor Certificate:

- (Optional) If the Trust Anchor Locator contains a "prefetch.uris" field, pass the URIs contained there to the fetcher (see Section 3.1.1). (This field is a non-standard extension to the TAL format supported by the RIPE NCC Validator. It helps fetching non-hierarchical rsync repositories more efficiently.)

- Extract the TA certificate URI from the TAL's URI section (see Section 2.1 of[RFC7730]) and pass to the object fetcher (Section 3.1.2).

- Retrieve from the object store (see Section 4.1.4) all certificate objects, for which the URI matches the URI extracted from the TAL in the previous step, and the public key matches the subjectPublicKeyInfo field of the TAL (see Section 2.1 of [RFC7730]).

- If no, or more than one such objects are found, issue an error and stop validation process. Otherwise, use that object as the Trust Anchor certificate.

2.2. Resource Certificate Validation

The following steps describe the validation of a single resource certificate:

- If both the caRepository (Section 4.8.8.1 of [RFC6487]), and the id-ad-rpkiNotify (Section 3.5 of [I-D.ietf-sidr-delta-protocol]) SIA pointers are present in the given resource certificate, use a local policy to determine which pointer to use. Extract the URI from the selected pointer and pass it to the object fetcher (see Section 3.1.1).

- For a given resource certificate, find its manifest and certificate revocation list (CRL), using the procedure described in Section 2.2.1. If no such manifest and CRL could be found, issue an error and stop processing current certificate.

- Compare the URI found in the given resource certificate's id-ad-rpkiManifest field (Section 4.8.8.1 of [RFC6487]) with the URI of the manifest found in the previous step. If they are different, issue a warning.

- Perform manifest entries validation as described in Section 2.2.2.

- Validate all resource certificate objects found on the manifest, using the CRL object found on the manifest, according to Section 7 of [RFC6487].

- Validate all ROA objects found on the manifest, using the CRL object found on the manifest, according to the Section 4 of [RFC6482].

- Validate all Ghostbusters Record objects found on the manifest, using the CRL object found on the manifest, according to the Section 7 of [RFC6493].

- For every valid resource certificate object found on the manifest, apply the procedure described in this section (Section 2.2), recursively, provided that this resource certificate (identified

by its SKI) has not yet been validated during current repository validation run.

2.2.1. Finding most recent valid manifest and CRL

Fetch from the store (see Section 4.1.5) all objects of type manifest, whose certificate's AKI field matches the SKI of the current CA certificate.

Find the manifest object with the highest manifestNumber field (Section 4.2.1 of [RFC6486]), for which all following conditions are met:

- There is only one entry in the manifest for which the store contains exactly one object of type CRL, whose hash matches the hash of the entry.
- The manifest's certificate AKI equals the above CRL's AKI
- The above CRL is a valid object according to Section 6.3 of [RFC5280]
- The manifest is a valid object according to Section 4.4 of [RFC6486], using the CRL found above

Report an error for every invalid manifest with the number higher than the number of the valid manifest.

2.2.2. Manifest entries validation

For every entry in the manifest object:

- Construct an entry's URI by appending the entry name to the current CA's publication point URI.
- Get all objects from the store whose hash attribute equals entry's hash (see Section 4.1.3).
- If no such objects found, issue an error.

- For every found object, compare its URI with the URI of the manifest entry. If they do not match, issue a warning.
- If no objects with matching URI found, issue a warning.
- If some objects with non-matching URI found, issue a warning.

2.3. Object Store Cleanup

At the end of the TA tree validation the store cleanup is performed:

- Given all objects that were validated during the current validation run, remove from the store (Section 4.1.7) all objects whose URI attribute matches the URI of one of the validated objects, but the content's hash is different.
- Remove from the store all objects that were last validated more than 7 days ago.
- Remove from the store all objects that were downloaded more than 2 hours ago and have never been used in a validation process.

The time intervals used in the steps above are a matter of local policy.

3. Remote Objects Fetcher

The fetcher is responsible for downloading objects from remote repositories (described in Section 3 of [RFC6481]) using rsync protocol ([rsync]), or RPKI Repository Delta Protocol (RRDP) ([I-D.ietf-sidr-delta-protocol]).

3.1. Fetcher Operations

3.1.1. Fetch repository objects

This operation receives one parameter - a URI. For rsync protocol this URI points to a directory in a remote repository. For RRDP repository it points to the repository's notification file.

The fetcher performs following steps:

- If the given URI has been downloaded recently (as specified by the local policy), skip all following steps.
- Download the remote objects using the URI provided (for an rsync repository use a recursive mode).
- For every new object that is downloaded, try to parse it as an object of specific RPKI type (certificate, manifest, CRL, ROA, Ghostbusters record), based on the object's filename extension (.cer, .mft, .crl, .roa, and .gbr, respectively), and perform basic RPKI object validation, as specified in [RFC6487] and [RFC6488].
- For every downloaded valid object, record it in the object store (Section 4.1.1), and set its last fetch time to the time it was downloaded (Section 4.1.2).

3.1.2. Fetch single repository object

This operation receives one parameter - a URI that points to an object in a remote repository.

The fetcher performs following operations:

- If the given URI has been downloaded recently (as specified by the local policy), skip all following steps.
- Download the remote object using the URI provided.
- Try to parse the downloaded object as an object of a specific RPKI type (certificate, manifest, CRL, ROA, Ghostbusters record), based on the object's filename extension (.cer, .mft, .crl, .roa, and .gbr, respectively), and perform basic RPKI object validation, as specified in [RFC6487] and [RFC6488].
- If the downloaded object is not valid, issue an error and skip further steps.

- Delete objects from the object store (Section 4.1.6) whose URI

matches the URI given.

- Put validated object in the object store (Section 4.1.1), and set its last fetch time to the time it was downloaded (Section 4.1.2).

4. Local Object Store

4.1. Store Operations

4.1.1. Store Repository Object

Put given object in the store, along with its type, URI, hash, and AKI, if there is no record with the same hash and URI fields.

4.1.2. Update object's last fetch time

For all objects in the store whose URI matches the given URI, set the last fetch time attribute to the given timestamp.

4.1.3. Get objects by hash

Retrieve all objects from the store whose hash attribute matches the given hash.

4.1.4. Get certificate objects by URI

Retrieve from the store all objects of type certificate, whose URI attribute matches the given URI.

4.1.5. Get manifest objects by AKI

Retrieve from the store all objects of type manifest, whose AKI attribute matches the given AKI.

4.1.6. Delete objects for URI

For a given URI, delete all objects in the store with matching URI attribute.

4.1.7. Delete outdated objects

For a given URI and a list of hashes, delete all objects in the store with matching URI, whose hash attribute is not in the given list of hashes.

4.1.8. Update object's validation time

For all objects in the store whose hash attribute matches the given hash, set the last validation time attribute to the given timestamp.

5. Acknowledgements

This document describes the algorithm as it is implemented by the software development team at the RIPE NCC. The original idea behind it was outlined by Tim Bruijnzeels. The authors would also like to acknowledge contributions by Carlos Martinez, Andy Newton, and Rob Austein.

6. IANA Considerations

This document has no actions for IANA.

7. Security Considerations

This algorithm uses the content of a manifest object to discover other objects issued by a particular CA. It verifies that the manifest is located in the publication point designated in the CA Certificate. However, if there are other (not enlisted in the manifest) objects located in that publication point directory, they will be ignored, even if their content is correct and they are issued by the same CA as the manifest.

In contrast, objects whose content hash matches the hash listed in the manifest, but that are not located in the publication directory listed in their CA certificate, will be used in the validation process (although a warning will be issued in that case).

The store cleanup procedure described in Section 2.3 tries to minimise removal and subsequent re-fetch of objects that are published in some repository but not used in the validation. Once such objects are removed from the remote repository, they will be discarded from the local object store after a period of time specified by a local policy. By generating an excessive amount of syntactically valid RPKI objects, a man-in-the-middle attack rendered between a validating tool and a repository could force an implementation to fetch and store those objects in the object store before they are being validated and discarded, leading to an out-of-memory or out-of-disk-space conditions, and, subsequently, a denial of service.

8. References

8.1. Normative References

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <http://www.rfc-editor.org/info/rfc5280>.

[RFC6481] Huston, G., Loomans, R., and G. Michaelson, "A Profile for Resource Certificate Repository Structure", RFC 6481, DOI 10.17487/RFC6481, February 2012, <http://www.rfc-editor.org/info/rfc6481>.

[RFC6482] Lepinski, M., Kent, S., and D. Kong, "A Profile for Route Origin Authorizations (ROAs)", RFC 6482, DOI 10.17487/RFC6482, February 2012, <http://www.rfc-editor.org/info/rfc6482>.

[RFC6486] Austein, R., Huston, G., Kent, S., and M. Lepinski, "Manifests for the Resource Public Key Infrastructure (RPKI)", RFC 6486, DOI 10.17487/RFC6486, February 2012, <http://www.rfc-editor.org/info/rfc6486>.

[RFC6487] Huston, G., Michaelson, G., and R. Loomans, "A Profile for X.509 PKIX Resource Certificates", RFC 6487, DOI 10.17487/RFC6487, February 2012, <http://www.rfc-editor.org/info/rfc6487>.

[RFC6488] Lepinski, M., Chi, A., and S. Kent, "Signed Object Template for the Resource Public Key Infrastructure (RPKI)", RFC 6488, DOI 10.17487/RFC6488, February 2012, <http://www.rfc-editor.org/info/rfc6488>.

[RFC6493] Bush, R., "The Resource Public Key Infrastructure (RPKI) Ghostbusters Record", RFC 6493, DOI 10.17487/RFC6493, February 2012, <http://www.rfc-editor.org/info/rfc6493>.

[RFC7730] Huston, G., Weiler, S., Michaelson, G., and S. Kent, "Resource Public Key Infrastructure (RPKI) Trust Anchor Locator", RFC 7730, DOI 10.17487/RFC7730, January 2016, <http://www.rfc-editor.org/info/rfc7730>.

8.2. Informative References

[I-D.ietf-sidr-delta-protocol]
Bruijnzeels, T., Muravskiy, O., Weber, B., Austein, R., and D. Mandelberg, "RPKI Repository Delta Protocol", draft-ietf-sidr-delta-protocol-02 (work in progress), March 2016.

[rsync] "Rsync home page", <https://rsync.samba.org>.

Authors' Addresses

Oleg Muravskiy
RIPE NCC

Email: oleg@ripe.net

Tim Bruijnzeels
RIPE NCC

Email: tim@ripe.net