SIDR
Internet-Draft
Intended status: Informational
Expires: January 9, 2017

O. Muravskiy
T. Bruijnzeels
RIPE NCC

July 8, 2016

RPKI Certificate Tree Validation by a Relying Party Tool
draft-ietf-sidr-rpki-tree-validation-01

Abstract

This document 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.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 9, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

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 the RIPE NCC RPKI Validator discovers RPKI objects to download, builds certificate paths, and validates 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. General Considerations

2.1. Hash comparisons

This algorithm relies on the properties of the file hash algorithm (defined in [RFC6485]) to compute the hash of repository objects. It assumes that any two objects for which the hash value is the same, are identical.

The hash comparison is used when matching objects in the repository with entries on the manifest, and when looking up objects in the object store (Section 5).

2.2. Manifest entries versus repository content

There are several possible ways of discovering products of a CA certificate: one could use all objects located in a repository directory designated as a publication point for a CA, or only objects mentioned on the manifest located at that publication point (see Section 6 of [RFC6486]), or use all objects whose AKI field matches the SKI field of a CA certificate.

Since the current set of RPKI standards requires use of the manifest [RFC6486] to describe the content of a publication point, this implementation requires a consistency between the publication point content and manifest content. Therefore it will not use in the validation process objects that are found in the publication point but do not match any of the entries of that publication point's manifest (see Section 3.2.2). It will also issue warnings for all found mismatches, so that the responsible operators could be made aware of inconsistencies and fix them.

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

1. 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 3.1.

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

3. If the TA certificate is valid, then all its subordinate objects are validated as described in Section 3.2. Otherwise the validation of certificate tree is aborted and an error is issued.

4. For all repository objects that were validated during this validation run, their validation timestamp is updated in an object store (see Section 5.1.7).

5. Outdated objects are removed from the store as described in Section 3.3. This completes the validation of the TA certificate tree.

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

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

1. (Optional) If the Trust Anchor Locator contains a "prefetch.uris" field, pass the URIs contained in that field to the fetcher (see

Section 4.1.1). (This field is a non-standard extension to the TAL format. It helps fetching non-hierarchical rsync repositories more efficiently.)

2. Extract the TA certificate URI from the TAL's URI section (see Section 2.1 of [RFC7730]) and pass it to the object fetcher (Section 4.1.2).

3. Retrieve from the object store (see Section 5.1.3) 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]).

4. If no, or more than one such objects are found, issue an error and abort certificate tree validation process with an error. Otherwise, use the single found object as the Trust Anchor certificate.

3.2. Resource Certificate Validation

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

1. 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 4.1.1).

2. For a given resource certificate, find its manifest and certificate revocation list (CRL), using the procedure described in Section 3.2.1. If no such manifest and CRL could be found, stop validation of this certificate, consider it invalid, and issue an error.

3. 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.

4. Perform manifest entries discovery and validation as described in Section 3.2.2.

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

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

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

8. For every valid resource certificate object found on the manifest, apply the procedure described in this section (Section 3.2), recursively, provided that this resource certificate (identified by its SKI) has not yet been validated during current repository validation run.

3.2.1. Finding the most recent valid manifest and CRL

1. Fetch from the store (see Section 5.1.4) all objects of type manifest, whose certificate's AKI field matches the SKI of the current CA certificate. If no such objects are found, stop processing current resource certificate and issue an error.

2. Find among found objects 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.

3. If there is an object that matches above criteria, consider this object to be the valid manifest, and the CRL found at the previous step - the valid CRL for the current CA certificate's publication point.

4. Report an error for every other manifest with a number higher than the number of the valid manifest.

3.2.2. Manifest entries validation

For every entry in the manifest object:

1. Construct an entry's URI by appending the entry name to the current CA's publication point URI.

2. Get all objects from the store whose hash attribute equals entry's hash (see Section 5.1.2).

3. If no such objects are found, issue an error for this manifest entry and progress to the next entry. This case indicates that the repository does not have an object at the location listed in the manifest, or that the object's hash does not match the hash listed in the manifest.

4. For every found object, compare its URI with the URI of the manifest entry.

   * For every object with non-matching URI issue a warning. This case indicates that the object from the manifest entry is found at a different location in a (possibly different) repository.

   * If no objects with matching URI found, issue a warning. This case indicates that there is no object found in the repository at the location listed in the manifest entry (but there is at least one matching object found at a different location).

5. Use all found objects for further validation.

3.3. Object Store Cleanup

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

1. Given all objects that were encountered during the current validation run, remove from the store (Section 5.1.6) all objects whose URI attribute matches the URI of one of the encountered objects, but the content's hash is different. This removes from the store objects that were replaced in the repository by their newer versions at the same URIs.

2. Remove from the store all objects that were last encountered during validation long time ago (as specified by the local policy). This removes objects that do not appear on any valid manifest anymore (but possibly still published in a repository).

3. Remove from the store all objects that were downloaded recently (as specified by the local policy), but have never been used in a validation process. This removes objects that have never appeared on any valid manifest.

Shortening the time interval used in step 2 will free disk space used by the store, to the expense of downloading removed objects again if they are still published in the repository.

Extending the time interval used in step 3 will prevent repeated downloads of repository objects, with the risk that such objects, if created massively by mistake or adversely, will fill up local disk space, if they are not cleaned up promptly.

4. 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]).

4.1. Fetcher Operations

For every successfully visited URI the fetcher keeps track of the last time it happened.

4.1.1. 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:

1. If data associated with the URI has been downloaded recently (as specified by the local policy), skip following steps.

2. Download the remote objects using the URI provided (for an rsync repository use a recursive mode).

3. 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 (excluding resource certification path validation), as specified in [RFC6487] and [RFC6488].

4. Put every downloaded valid object in the object store (Section 5.1.1).

The time interval used in the step 1 should be chosen based on the acceptable delay in receiving repository updates.

4.1.2. Fetch single repository object

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

The fetcher performs following operations:

1. If data associated with the URI has been downloaded recently (as specified by the local policy), skip all following steps.

2. Download the remote object using the URI provided.

3. 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 (excluding resource certification path validation), as specified in [RFC6487] and [RFC6488].

4. If the downloaded object is not valid, issue an error and skip further steps.

5. Delete all objects from the object store (Section 5.1.5) whose URI matches the URI given.

6. Put the validated object in the object store (Section 5.1.1).

5. Local Object Store

5.1. Store Operations

5.1.1. 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.

5.1.2. Get objects by hash

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

5.1.3. Get certificate objects by URI

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

5.1.4. Get manifest objects by AKI

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

5.1.5. Delete objects for a URI

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

5.1.6. 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.

5.1.7. 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.

6. Acknowledgements

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

7. IANA Considerations

This document has no actions for IANA.

8. Security Considerations

This implementation will not detect possible hash collisions in the hashes of repository objects (calculated using the file hash algorithm specified in [RFC6485]), and considers objects with same hash values as identical.

This algorithm uses the content of a manifest object to discover other objects issued by a specified CA. It verifies that the manifest is located in the publication point designated in the CA Certificate. However, if there are other (not listed 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 3.3 tries to minimise removal and subsequent re-fetch of objects that are published in a 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 between a validating tool and a repository could force an implementation to fetch and store those objects in the object store before they are validated and discarded, leading to an out-of-memory or out-of-disk-space conditions, and, subsequently, a denial of service.

9. References

9.1. 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>.

[RFC6485] Huston, G., "The Profile for Algorithms and Key Sizes for Use in the Resource Public Key Infrastructure (RPKI)", RFC 6485, DOI 10.17487/RFC6485, February 2012, <http://www.rfc-editor.org/info/rfc6485>.

[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>.

9.2. 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