Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 6, 2011

S. Weiler
A. Sonalker
SPARTA, Inc.

July 5, 2010

A Publication Protocol for the Resource Public Key Infrastructure (RPKI)

draft-weiler-sidr-publication-00

Abstract

This document defines a protocol for publishing Resource Public Key Infrastructure (RPKI) objects. Even though the RPKI will have many participants issuing certificates and creating other objects, it is operationally useful to consolidate the publication of those objects. This document provides the protocol for that.

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 6, 2011.

Copyright Notice

Table of Contents

1. Introduction

This document assumes a working knowledge of the Resource Public Key Infrastructure (RPKI), which is intended to support improved routing security on the Internet. [I-D.ietf-sidr-arch]

In order to make participation in the RPKI easier, it is helpful to have a few consolidated repositories for RPKI objects, thus saving every participant from the cost of maintaining a new service. Similarly, relying parties using the RPKI objects will find it faster and more reliable to retrieve the necessary set from a smaller number of repositories.

These consolidated RPKI object repositories will in many cases be outside the administrative scope of the organization issuing a given RPKI object. Hence the need for a protocol to publish RPKI objects.

This document defines the RPKI publication protocol, including a sub-protocol for configuring the publication engine.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

"Publication engine" and "publication server" are used interchangeably to refer to the server providing the service described in this document.

"Business Public Key Infrastructure" ("Business PKI" or "BPKI") refers to a PKI, separate from the RPKI, used to authenticate clients to the publication engine.

2. Context

This protocol was designed specifically for the case where an internet registry, already issuing RPKI certificates to its children, also wishes to run a publication service for its children.

We use the term "Business PKI" here to suggest that an internet registry might already have a PKI, separate from the RPKI, for authenticating its clients and might wish to reuse that PKI for this protocol. Such reuse is not a requirement.

3. Protocol Specification

In summary, the publication protocol uses XML messages wrapped in CMS, carried over HTTP transport.

The publication procotol consists of two separate subprotocols. The first is a control protocol used to configure a publication engine. The second subprotocol, which we refer to by the overloaded term "publication protocol", is used to request publication of specific objects. The publication engine operates a single HTTP server on a single port. It distinguishes between the two protocols by using different URLs for them.

3.1. Common Details

This section discusses details that the two subprotocols have in common, including the transport and CMS wrappers. This portion of the protocol is largely inherited from the provisioning protocol ([I-D.ietf-sidr-rescerts-provisioning]).

Both protocols use a simple request/response interaction. The client passes a request to the server, and the server generates a corresponding response. A message exchange commences with the client initiating an HTTP POST with content type of "application/x-rpki", with the message object as the body. The server's response will similarly be the body of the response with a content type of "application/x-rpki".

The content of the POST, and the server's response, will be a well-formed Cryptographic Message Syntax (CMS) [RFC5652] object with OID = 1.2.840.113549.1.7.2 as described in Section 3.1 of [I-D.ietf-sidr-rescerts-provisioning].

3.1.1. Common XML Message Format

The publication protocol uses the same message passing design as the provisioning protocol. The XML schema for this protocol (including both subprotocols) is below in Section 3.5. Both subprotocols use the same basic XML message format, which looks like:

```
<?xml version='1.0' encoding='us-ascii'?>
<msg xmlns="http://www.hactrn.net/uris/rpki/publication-spec/"
         version="1"
         type="message type">
        [one or more PDUs]
</msg>
```

version:
The value of this attribute is the version of this protocol. This document describes version 1.

type:
The possible values of this attribute are "reply" and "query".

3.2. Control Protocol

The control protocol is used to configure a publication server. It can set global variables (at the moment, limited to a BPKI CRL) and manage clients who are allowed to publish data on the server.

The control protocol has two objects: the <config/> object, and the <client/> object.

3.2.1. Config Object

The <config/> object allows configuration of data that apply to the entire publication server rather than a particular client. There is exactly one <config/> object in the publication server, and it only supports the "set" and "get" actions -- it cannot be created or destroyed.

The <config/> object only has one data element that can be set: the bpki_crl. This is used by the publication server when authenticating clients.

3.2.2. Client Object

Unlike the <config/> object the <client/> object represents one client authorized to use the publication server.

The <client/> object supports five actions: "create", "set", "get", "list", and "destroy". Each client has a "client_handle" attribute, which is used in responses and must be specified in "create", "set", "get", or "destroy" actions.

Payload data which can be configured in a <client/> object include:

- base_uri (attribute): This attribute represents the base URI below which the client will be allowed to publish data. Additional constraints may be imposed by the Publication Server in certain cases, for e.g., a child publishing directly under its parent.
- bpki_cert (element): This represents the X509 BPKI CA certificate for this client. This should be used as part of the certificate chain when validating incoming TLS and CMS messages. Two valid approaches exist. If the optional bpki_glue certificate is being used, then the bpki_cert certificate should be issued by the

bpki_glue certificate; otherwise, the bpki_cert certificate should be issued by the publication engine's bpki_ta certificate.

- bpki_glue (element): This is an additional (optional) type of X509 certificate for this client. It may be used in certain pathological cross-certification cases which require a two-certificate chain due to issuer name conflicts. When being used, issuing order is that the bpki_glue certificate should be the issuer of the bpki_cert certificate. Otherwise, it should be issued by the publication engine's bpki_ta certificate. Since this is an optional use certificate, it may be left unset if not needed.

3.3. Publication Protocol

The publication protocol is structured differently from the control protocol in that objects in the publication protocol represent objects to be published or objects to be withdrawn from publication.

Each kind of object supports two actions: "publish" and "withdraw". In each case the XML element representing the object to be published or withdrawn has a "uri" attribute which contains the publication URI. For "publish" actions, the XML element body contains the DER object to be published, encoded in Base64; for "withdraw" actions, the XML element body is empty.

The publication protocol uses four types of objects:

- Certificate Object: The <certificate/> object represents an RPKI certificate to be published or withdrawn.
- CRL Object: The <crl/> object represents an RPKI CRL to be published or withdrawn.
- Manifest Object: The <manifest/> object represents an RPKI publication manifest to be published or withdrawn. See [I-D.ietf-sidr-rpki-manifests] for more information on manifests.
- ROA Object: The <roa/> object represents a ROA to be published or withdrawn. See [I-D.ietf-sidr-roa-format] for more information on ROAs.

Note that every publication or withdrawal action requires a new manifest, thus every publication or withdrawal action will involve at least two objects.

3.4. Error handling

Errors are handled similarly in both subprotocols, and they're handled at two levels.

Since all messages in this protocol are conveyed over HTTP connections, basic errors are indicated via the HTTP response code.

4xx and 5xx responses indicate that something bad happened. Errors that make it impossible to decode a query or encode a response are handled in this way.

Where possible, errors will result in an XML <report_error/> message which takes the place of the expected protocol response message. <report_error/> messages are CMS-signed XML messages like the rest of this protocol, and thus can be archived to provide an audit trail.

<report_error/> messages only appear in replies, never in queries. The <report_error/> message can appear in both the control and publication subprotocols.

The <report_error/> message includes an optional "tag" attribute to assist in matching the error with a particular query when using batching.

The error itself is conveyed in the error_code (attribute). The value of this attribute is a token indicating the specific error that occurred. [TODO: define these tokens]

The body of the <report_error/> element itself is an optional text string; if present, this is debugging information. At present this capabilty is not used, debugging information goes to syslog.

3.5. XML Schema

The following is a RelaxNG compact form schema describing the Publication Protocol.

```
default namespace = "http://www.hactrn.net/uris/rpki/publication-spec/"

# Top level PDU

start = element msg { attribute version { xsd:positiveInteger {
   maxInclusive="1" } }, ((attribute type { "query" }, query_elt*) |
   (attribute type { "reply" }, reply_elt*)) }

# PDUs allowed in a query
query_elt = ( config_query | client_query | certificate_query |
   crl_query | manifest_query | roa_query )

# PDUs allowed in a reply
reply_elt = ( config_reply | client_reply | certificate_reply |
   crl_reply | manifest_reply | roa_reply | report_error_reply )

# Tag attributes for bulk operations
```

```
    tag = attribute tag { xsd:token {maxLength="1024" } }

    # Base64 encoded DER stuff
    base64 = xsd:base64Binary { maxLength="512000" }

    # Publication URLs
    uri_t = xsd:anyURI { maxLength="4096" }
    uri = attribute uri { uri_t }

    # Handles on remote objects (replaces passing raw SQL IDs).  NB:
    # Unlike the up-down protocol, handles in this protocol allow "/" as a
    # hierarchy delimiter.
    object_handle = xsd:string { maxLength="255" pattern="[\-_A-Za-z0-9/]*" }

    # <config/> element (use restricted to repository operator)
    # config_handle attribute, create, list, and destroy commands omitted
    # deliberately, see code for details

    config_payload = (element bpki_crl { base64 }?)

    config_query |= element config { attribute action { "set" }, tag?,
       config_payload }
    config_reply |= element config { attribute action { "set" },     tag? }
    config_query |= element config { attribute action { "get" },     tag? }
    config_reply |= element config { attribute action { "get" }, tag?,
       config_payload }

    # <client/> element (use restricted to repository operator)

    client_handle = attribute client_handle { object_handle }

    client_payload = (attribute base_uri { uri_t }?, element bpki_cert {
       base64 }?, element bpki_glue { base64 }?)

    client_query |= element client { attribute action { "create" }, tag?,
       client_handle, client_payload }
    client_reply |= element client { attribute action { "create" }, tag?,
       client_handle }
    client_query |= element client { attribute action { "set" }, tag?,
       client_handle, client_payload }
    client_reply |= element client { attribute action { "set" }, tag?,
       client_handle }
    client_query |= element client { attribute action { "get" }, tag?,
       client_handle }
    client_reply |= element client { attribute action { "get" }, tag?,
       client_handle, client_payload }
    client_query |= element client { attribute action { "list" }, tag? }
    client_reply |= element client { attribute action { "list" }, tag?,
```

```
      client_handle, client_payload }
   client_query |= element client { attribute action { "destroy" }, tag?,
      client_handle }
   client_reply |= element client { attribute action { "destroy" }, tag?,
      client_handle }

   # <certificate/> element

   certificate_query |= element certificate { attribute action {
      "publish" }, tag?, uri, base64 }

   certificate_reply |= element certificate { attribute action {
      "publish" }, tag?, uri }

   certificate_query |= element certificate { attribute action {
      "withdraw" }, tag?, uri }

   certificate_reply |= element certificate { attribute action {
      "withdraw" }, tag?, uri }

   # <crl/> element

   crl_query |= element crl { attribute action { "publish" }, tag?, uri,
      base64 }
   crl_reply |= element crl { attribute action { "publish" }, tag?, uri }
   crl_query |= element crl { attribute action { "withdraw" }, tag?, uri }
   crl_reply |= element crl { attribute action { "withdraw" }, tag?, uri }

   # <manifest/> element

   manifest_query |= element manifest { attribute action { "publish" },
      tag?, uri, base64 }
   manifest_reply |= element manifest { attribute action { "publish" },
      tag?, uri }
   manifest_query |= element manifest { attribute action { "withdraw" },
      tag?, uri }
   manifest_reply |= element manifest { attribute action { "withdraw" },
      tag?, uri }

   # <roa/> element

   roa_query |= element roa { attribute action { "publish" }, tag?, uri,
      base64 }
   roa_reply |= element roa { attribute action { "publish" }, tag?, uri }
   roa_query |= element roa { attribute action { "withdraw" }, tag?, uri }
   roa_reply |= element roa { attribute action { "withdraw" }, tag?, uri }

   # <report_error/> element
```

```
error = xsd:token { maxLength="1024" }

report_error_reply = element report_error {
   tag?,
   attribute error_code { error },
   xsd:string { maxLength="512000" }?
   }
```

4. Operational Considerations

Placeholder section to talk about nesting children under parents in the sameso repository, to allow for a single rsync to fetch both (observing that the rsync setup times tends to dominate over the sync time). And, more distressingly, talk about the access control impacts of that nesting.

5. IANA Considerations

This document specifies no IANA Actions.

6. Security Considerations

7. References

7.1. Normative References

[I-D.ietf-sidr-res-certs]
Huston, G., Michaelson, G., and R. Loomans, "A Profile for X.509 PKIX Resource Certificates", draft-ietf-sidr-res-certs-18 (work in progress), May 2010.

[I-D.ietf-sidr-rescerts-provisioning]
Huston, G., Loomans, R., Ellacott, B., and R. Austein, "A Protocol for Provisioning Resource Certificates", draft-ietf-sidr-rescerts-provisioning-06 (work in progress), May 2010.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[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, May 2008.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", RFC 5652, September 2009.

[X.690] Postel, J., "ITU-T Recommendation X.690: ISO/IEC 8825-1:2002, Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", 2002.

7.2. Informative References

[I-D.ietf-sidr-arch]
Lepinski, M. and S. Kent, "An Infrastructure to Support Secure Internet Routing", draft-ietf-sidr-arch-09 (work in progress), October 2009.

[I-D.ietf-sidr-roa-format]
Lepinski, M., Kent, S., and D. Kong, "A Profile for Route Origin Authorizations (ROAs)", draft-ietf-sidr-roa-format-06 (work in progress), October 2009.

[I-D.ietf-sidr-rpki-manifests]
Austein, R., Huston, G., Kent, S., and M. Lepinski, "Manifests for the Resource Public Key Infrastructure", draft-ietf-sidr-rpki-manifests-07 (work in progress), May 2010.

Appendix A. Acknowledgments

We acknowledge the editors of [I-D.ietf-sidr-rescerts-provisioning] (Geoff Huston, Robert Loomans, Byron Ellacott, and Rob Austein), from whom we took some of the text for this document.

We especially thank Rob Austein, who implemented the publication protocol and helped us to understand it.

Authors' Addresses

Samuel Weiler
SPARTA, Inc.
7110 Samuel Morse Drive
Columbia, Maryland 21046
US

Email: weiler@tislabs.com

Anuja Sonalker
SPARTA, Inc.
7110 Samuel Morse Drive
Columbia, Maryland 21046
US

Email: Anuja.Sonalker@sparta.com