Creating an RPKI Cache Queried by IRRd

Corey Tessler
ctessler@emich.edu
Fall 2011

Acknowledgements

This project would not have been possible without the generosity of others. First to Andy Rosenzweig at Merit Networks. He was willing to entertain a phone call from a faceless student looking for a scholastic project.

Andy introduced me to Manish Karir, who quickly created an opportunity at Merit. He continually offered his time and expertise throughout this project. Without Manish this project would not have existed, nor would it have succeeded.

Technical assistance was freely volunteered by Kyle Creyts (at all hours) his help was greatly appreciated.

Professor Zeiger's patience and academic guidance in the class room can not be understated.

Lastly, thank you to my fellow class mates this semester. For listening to my reports, providing feedback, and posing challenging questions.

To all of you.

Thank you,
Corey Tessler

Creating an RPKI Cache Queried by IRRd

Introduction

Before starting I must apologize to the reader. The goal of this paper is to explain the function of the project, which is centered around two topics, networking, and security. Both require of the reader more than a cursory understanding. My personal understanding is limited in both domains. However, I will attempt to provide enough background information in this paper to allow you, the reader, to understand the goal, purpose and results of this project. Please forgive my lengthy explanations. In some places there will be some simplifications, in others omissions to avoid even longer explanations. There may be errors due to my lack of experience.

Project Goal

The final goal of this project was to provide RPKI information alongside a WHOIS response served by IRRd. Merit Network offers the Internet Routing Registry daemon (IRRd) as a tool for running an internet routing registry database. One of it's features is the ability to respond to WHOIS queries.

A WHOIS query is a request for information about an internet resource assigned by the Internet Assigned Numbers Authority (IANA); such as domain names, Autonomous System (AS) numbers, and network blocks. It is a question to a registry database "WHO IS the owner of this resource, as allocated by IANA." Notice that there is no stated relationship between IRRd and IANA. Meaning, that a registry database, such as IRRd, could respond with answers that do not align with IANAs allocations..

The Resource Public Key Infrastructure (RPKI) attempts to address this deficiency. Giving the consumer of the registry database an opportunity to verify that a resource has been allocated by IANA, and used appropriately by others. RPKI is a system which is flexible enough to handle all of the resource types allocated by IANA. Not all of those types have been specified, nor are they relevant to this project. This project focuses on the validation of network blocks and AS numbers.

To restate the goal of this project, it is to modify IRRd when responding to a WHOIS query for a network block to provide all of those ASes which may advertise the block, and those which have been excluded from doing so.

Organization

The remainder of this document is divided into three sections. Starting with an overview of the concepts leading up to and comprising the RPKI. Followed by an explanation of the steps involved in meeting the goal of the project. Concluding with considerations for improvements and extensions.

RPKI Organization

RPKI attempts to solve a deficiency found in the current generation of routing protocols, specifically in the Border Gateway Protocol (BGP.) Without RPKI, a BGP peer has no method for evaluating whether a route originating from another peer should be originated by that peer. RPKI provides a method for a BGP peer to verify a received originating route announcement. When a received route is verified against RPKI information, the receiver can validate that the announcement comes from an authorized source, and that authority is ultimately derived from IANA.

As mentioned before the RPKI is a general mechanism for different types of resources allocated by IANA. As the author of this paper and contributor to this project I have ignored the other resource types in meeting the goal of the project. Which is to provide RPKI information regarding network block and AS allocations. Descriptions of information within the RPKI has been limited to these data types. For different data types the RPKI may be organized differently.

Network Blocks and Autonomous Systems

Before trying to explain the organization of the RPKI, it's helpful to understand some of it's contents. Of interest to this project are network bolcks and AS numbers. A network block is a contiguous range of IP addresses, in this context IPv4 or IPv6 addresses. Blocks are commonly represented in the CIDR format. It's a compact way to represent a set of addresses.

An Autonomous System (AS) is a more flexible concept. A simple definition that will serve for our purposes is that an AS is a number that identifies a single entity. In context, an entity that owns network blocks. Your local Internet Service Provider (ISP) would be one such entity. BGP operates at the boundaries of these AS's trading information about network blocks. Creating paths for traffic to flow from one block to another.

IANA typically allocates it's resources, such as network blocks and AS numbers, in large blocks. They are allocated to subordinate authorities that parcel their allocations and assign them to others, who in turn may be permitted to divvy out smaller portions. This creates a hierarchy of resource allocation, with the smallest allocations given to local ISPs.

RPKI and X.509

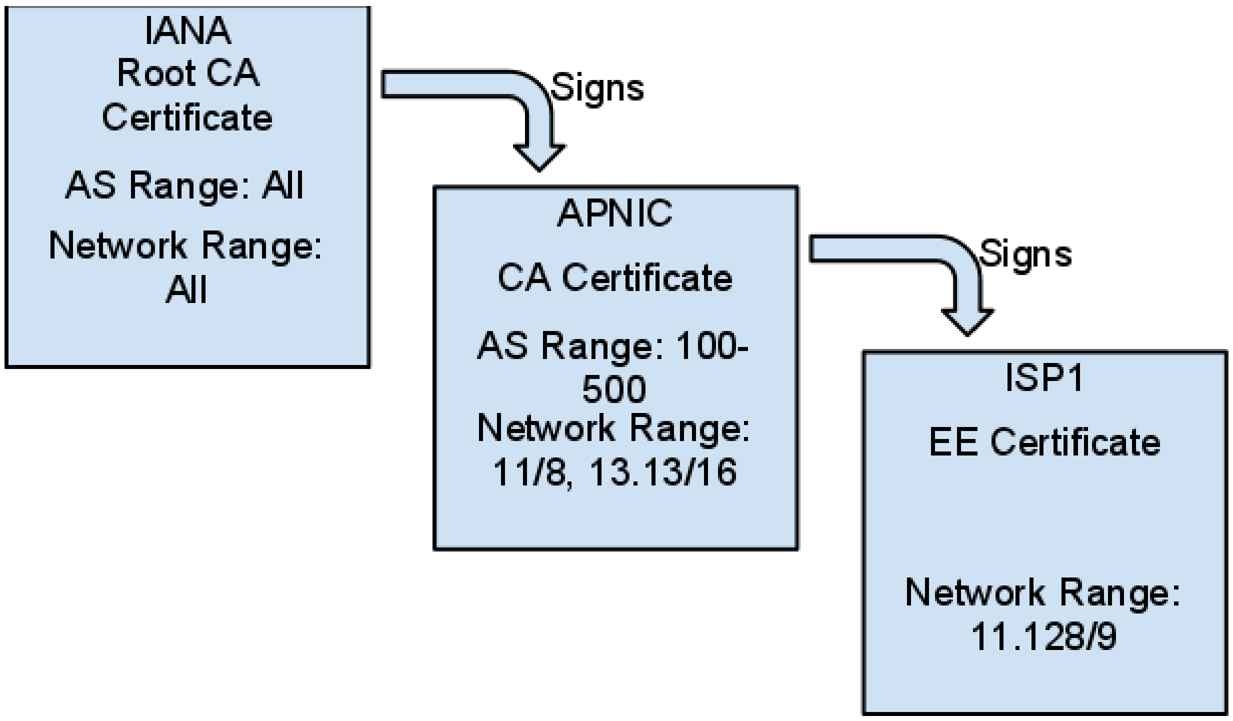
Information in the RPKI is organized in an identical hierarchy, with the added property that ownership of a resource can be validated. RPKI's hierarchy is created from signed X.509 certificates. An X.509 certificate is a container with the ability to be signed by an encryption key. X.509 certificates have many extensions to carry different data types. An important data type the certificates can carry is an encryption key, which will be refered to as a public key.

RPKI (and other applications) utilize this feature to publish certificates which can be validated. Each certificate is signed by the private key of an issuer. This signature is generated by combining a hash value of the contents and the issuers private key. Also contained within a certificate is an identifier and location of the issuers certificate. To verify that a certificate has been authorized by the issuer, the inspector performs five operations. First, calculating the hash of the data area of the signed certificate. Second, obtaining the issuers certificate. Third, extracting the public key from the issuers certificate. Fourth, decrypting the signature from the signed certificate with the obtained public key. Fifth, comparing the computed and decrypted values. When the values match the certificate is valid.

An X.509 certificate used to sign another certificate may have been signed by another certificate, which may have been signed by yet another certificate, and on and on. This creates a certificate chain that can only be validated by checking the signatures of each certificate in turn. At the end of each chain is a self signed root certificate, a certificate with no issuer. This certificate cannot be validated, it must be obtained through a trusted mechanism.

There are data type extensions to X.509 certificates that allow network block and AS numbers to be included. Certificates containing network block and AS number information are used within the RPKI as an authorization of allocation. RPKI places IANA at the root of all certificate chains, each subordinate certificate contains a subset of the resources allocated by IANA. Each resource must be allocated in a strict subset, overlapping subsets are not permitted.

Within the RPKI there are two types of certificates, a Certificate Authority (CA) and End Entity (EE). A CA certificate grants the certificate owner permission to create subordinate certificates, that include a subset of the resources within. An EE certificate grants the owner permission to originate routes, that include a subset of the network blocks within. These are the major components of the RPKI hierarchy.



From the above example, if ISP1 were to originate a route to 11.128/9, it's authority to do so could be validated. By checking the signature and contents of ISP1's EE certificate against the public key in APNIC's CA certificate. Then by checking the signature and contents of APNIC's CA certificate against the public key found in the IANA CA certificate.

This brief, simplified, slightly inaccurate description of RPKI with it's use of X.509 provides just enough background information to frame the work of the project. For a complete understanding of the RPKI and it's components start by reading *An Infrastructure to Support Secure Internet Routing*.

RPKI Cache & IRRd

Recall the goal of this project is to modify IRRd to return RPKI information within a WHOIS response for a network block. There are two conceptual tasks involved in augmenting the IRRd's WHOIS response. The first is to create an RPKI cache which can respond with the RPKI information given a network block. The second is to modify IRRd to query the RPKI Cache and return the information as part of servicing a WHOIS response. Most of the effort of this project went into understanding RPKI and constructing the cache. Having given a brief explanation of RPKI, the details of the cache implementation will be presented, followed by the IRRd modifications.

RPKI Cache Architecture

The cache, the first part of the project, is divided into four stages; fetching, validating, importing, and serving. Each stage consumes the output of the previous stage. Fetching pulls information from the RPKI Trust Anchors. Validation determines the validity of the certificates from the trust anchors. Import inserts the certificate information into the database. The server accepts network block queries and responds with all of the matches from the database.

Fetching

Earlier it was stated that all certificates are signed by an IANA root certificate, which isn't true. RPKI is designed so that there will be a root certificate provided by IANA and used to sign the larger registries certificates. RPKI has yet to be wholly adopted, which the designers have accommodated for. Their design also avoids a central repository for all certificate information.

A Trust Anchor (TA) is an RPKI concept. It is a publication point for certificates. Currently each TA organize their certificates differently. They all make their repositories available via the rsync protocol. Each TA contains a self signed root certificate, CA certificates, EE certificates, ROAs, manifests and CRLs. The manifests and CRLs were ignored by this project.

Fetching the contents of the TAs is done by the `rpki-fetch.pl` utility. A script written for this project that is driven by a configuration file and invoked by a cron daemon. The configuration file describes where to place the contents of the TA. When repeatedly invoked via cron the fetching utility utilizes rsync to update the fetched repositories.

What follows is an example configuration file for `rpki-fetch.pl`, `rfetch.cfg`. The syntax is simple, it instructs `rpki-fetch.pl` to pull the `apnic`, `lacnic`, and `ripe` repositories. The contents rooted at the URI's are deposited in `rpki-sync/apnic`, `rpki-sync/lacnic`, and `rpki-sync/ripe`.

`rfetch.cfg`

```
[rpki]
root_dir=rpki-sync
output_cfg=rvalid.cfg

[repo:apnic]
1:uri=rsync://rpki.apnic.net/member_repository
2:uri=rsync://rpki.apnic.net/repository
```

```
[repo:lacnic]
1:uri=rsync://repository.lacnic.net/rpki

[repo:ripe]
1:uri=rsync://rpki.ripe.net/ta
2:uri=rsync://rpki.ripe.net/repository

                     output directories
     rpki-sync/apnic/member_repository
     rpki-sync/apnic/repository
     rpki-sync/lacnic/rpki
     rpki-sync/ripe/ta
     rpki-sync/ripe/repository
```

Also included in the configuration file is the name of an output configuration file, `rvalid.cfg`. This is an output file which is used as input for the next stage. It tells the validation stage where to look for the fetched repositories, and the URI from which they were fetched.

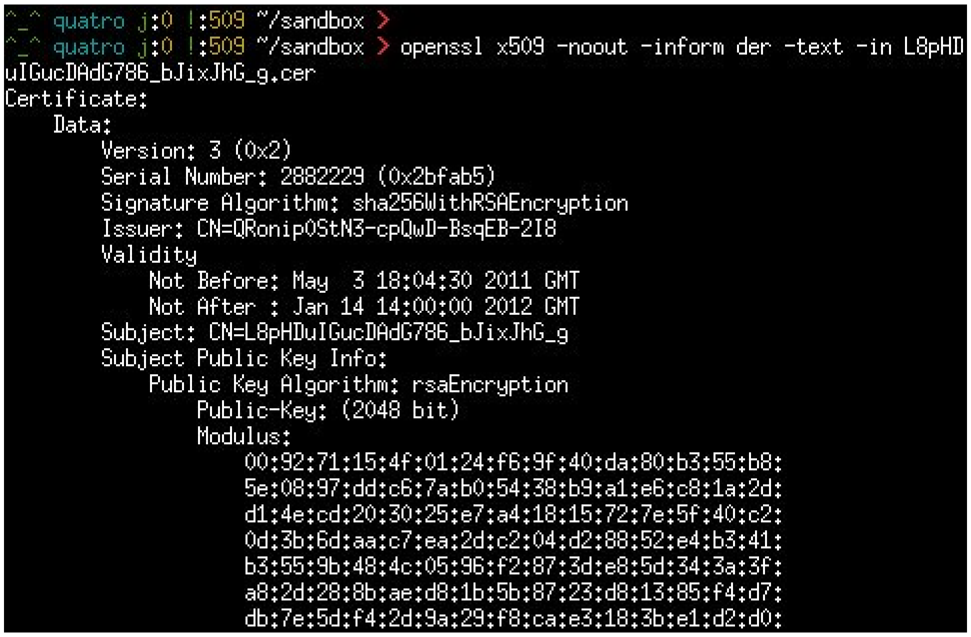
Validating

Validation required the most effort to accomplish in this project. No existing tools were sufficient for performing the required operations. The goal of the validation stage is to sort the certificates and ROAs based on their validity. Valid objects go into the `valid/` directory and invalid to the `invalid/` directory.

The validation process is handled by another script written for this project called `rpki-vdate.pl`. It's configuration file, `rvalid.cfg`, was written by the previous script `rpki-vdate.pl`. From the configuration file the validation script determines the location of the repositories, and the destination directories for the valid and invalid objects. To ensure validation occurs after fetching the cron entry invokes `rpki-fetch.pl` then `rpki-vdate.pl`.

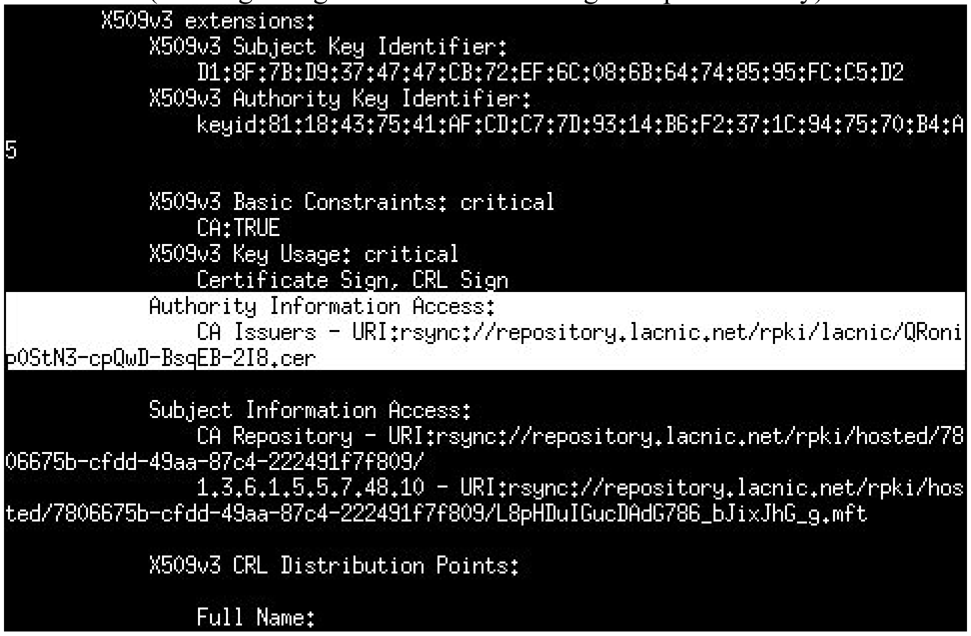
Each repository is validated in turn. First by examining the certificates and then the ROAs. To validate a certificate the entire chain of certificates must be determined. OpenSSL provides a tool for interpreting and extracting certificate information. Each certificate includes a URI reference to it's issuer (the signer.) Carried by the Authority Information Access field of the certificate, shown in the following example from the lacnic TA.

lacnic Certificate Example



```
^_^ quatro j:0 !:509 ~/sandbox >
^_^ quatro j:0 !:509 ~/sandbox > openssl x509 -noout -inform der -text -in L8pHD
uIGucDAdG786_bJixJhG_g.cer
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 2882229 (0x2bfab5)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN=QRonip0StN3-cpQwD-BsqEB-2I8
        Validity
            Not Before: May  3 18:04:30 2011 GMT
            Not After : Jan 14 14:00:00 2012 GMT
        Subject: CN=L8pHDuIGucDAdG786_bJixJhG_g
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:92:71:15:4f:01:24:f6:9f:40:da:80:b3:55:b8:
                    5e:08:97:dd:c6:7a:b0:54:38:b9:a1:e6:c8:1a:2d:
                    d1:4e:cd:20:30:25:e7:a4:18:15:72:7e:5f:40:c2:
                    0d:3b:6d:aa:c7:ea:2d:c2:04:d2:88:52:e4:b3:41:
                    b3:55:9b:48:4c:05:96:f2:87:3d:e8:5d:34:3a:3f:
                    a8:2d:28:8b:ae:d8:1b:5b:87:23:d8:13:85:f4:d7:
                    db:7e:5d:f4:2d:9a:29:f8:ca:e3:18:3b:e1:d2:d0:
```

(The beginning of the certificate using the openssl utility)



```
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                D1:8F:7B:D9:37:47:47:CB:72:EF:6C:08:6B:64:74:85:95:FC:C5:D2
            X509v3 Authority Key Identifier:
                keyid:81:18:43:75:41:AF:CD:C7:7D:93:14:B6:F2:37:1C:94:75:70:B4:A
5

            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Key Usage: critical
                Certificate Sign, CRL Sign
            Authority Information Access:
                CA Issuers - URI:rsync://repository.lacnic.net/rpki/lacnic/QRoni
p0StN3-cpQwD-BsqEB-2I8.cer

            Subject Information Access:
                CA Repository - URI:rsync://repository.lacnic.net/rpki/hosted/78
06675b-cfdd-49aa-87c4-222491f7f809/
                1.3.6.1.5.5.7.48.10 - URI:rsync://repository.lacnic.net/rpki/hos
ted/7806675b-cfdd-49aa-87c4-222491f7f809/L8pHDuIGucDAdG786_bJixJhG_g.mft

            X509v3 CRL Distribution Points:

                Full Name:
```

(Highlighting the Authority Information Access section of the certificate)

By using openssl in a very similar way to the above example, `rpki-vdate.pl` extracts from a certificate the URI of the issuer. A URI isn't useful to the validation tool, each certificate is stored on local disk. To obtain the path to the issuer's certificate, the original repository is removed from the URI and replaced with the local directory. From the image above `rsync://repository.lacnic.net/rpki` would be replaced with `rpki-cache/lacnic/rpki`, yielding a path to `rpki-cache/lacnic/rpki/QRonip0StN3-cpQwD-BsqEB-2I8.cer.` This certificate is examined for an issuer, starting a chain to be validated. Parenting certificates are added to the chain until the root self signed certificate is reached. When the entire chain has been discovered, only then, can the starting certificate be validated.

Unfortunately the openssl command line utility would not validate chains of arbitrary length. Before this project was undertaken the maximum length was two. To solve this problem a local change was made to openssl-1.0.0e. Modifying openssl's verify command to accept an arbitrary number of certificates.

Within the openssl library there is a concept of an X.509 certificate stack, which the verification code already used. By adding each certificate provided on the command line to the stack the changes were kept to a minimum. The source to openssl-1.0.0e and the patch containing the required changes can be found in the git repository found in the references section.

After the validity of each certificate chain is determined by openssl, `rpki-vdate.pl` places the certificate in the correct output directory; `valid` or `invalid`. Each of these certificates are CA certificates that allow entities to create subsequent certificates. They are not authorizations to announce routes. That's what an ROA is for.

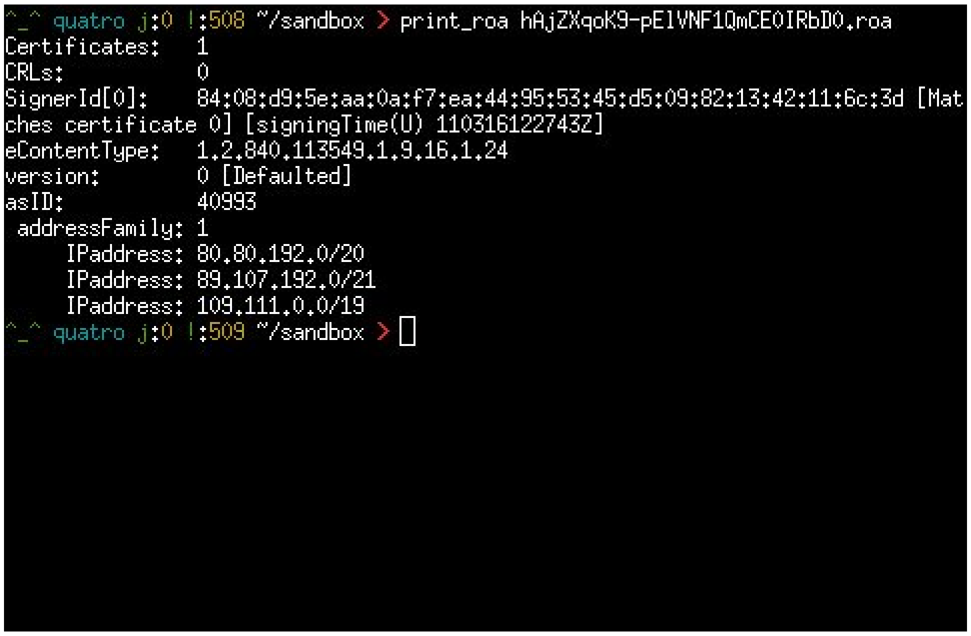
A Route Origin Authorization (ROA) is a signed object. It is an ASN.1 encoded CMS message which contains the final X.509 certificate (EE) in the chain. EE certificates like CA certificates include a URI that points to the issuers certificate. An AS number, and a set of network blocks is included in a ROA. Each ROA is signed with the private key of the EE contained within it. A ROA's signature is checked against the public key of the contained EE, which can in turn be validated up the certificate chain. Valid ROA's are an authorization to originate routes to the included network blocks, by the included AS number.

To validate a ROA the EE needs to be extracted, which can then be used to authenticate the signature within the ROA. The RPKI project created a tool set, which includes a utility called `print_roa`. When run with a ROA as input it displays the abbreviated contents of the ROA. Including information about the signature matching the embedded EE certificate. The contents of the EE certificate are omitted from the command output.

The utility (`print_roa`) has access to the EE certificate to perform the validation of the ROA. Which `rpki-vdate.pl` needs to extract to create a certificate chain that will determine if the ROA is valid or not. To complete the validation stage `print_roa` was modified to take a new command line argument, -e, that places the EE certificate in the same directory as the ROA. The RPKI project source, and patch used to make these changes can be found in the git repository found at the end of the references.

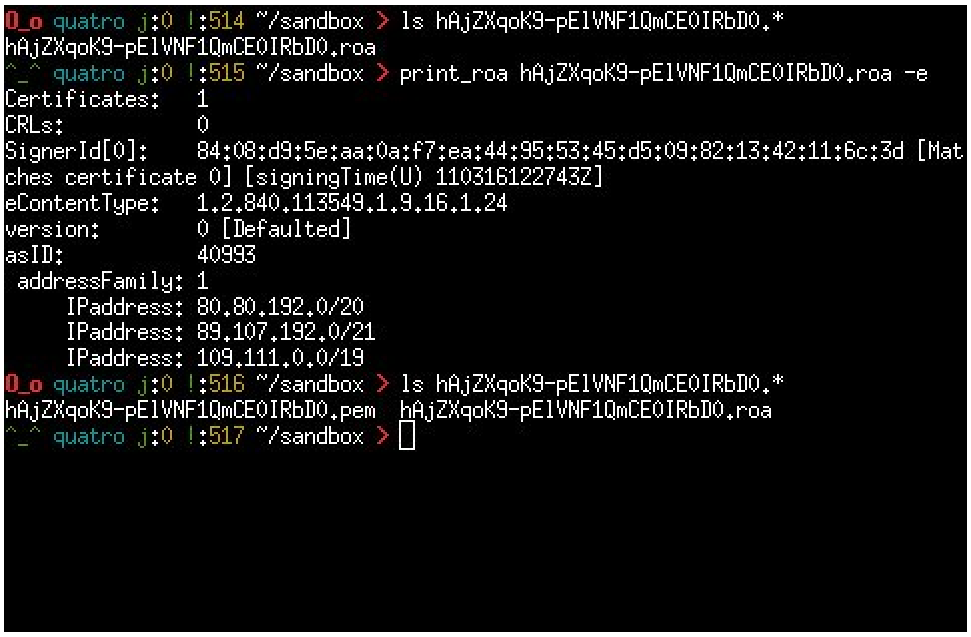
There are several subtleties to ROA validation, including the lack of AS numbers in EE certificates. The process of validation was left the OpenSSL library. Please refer to the working group documentation for a better understanding of how the contained resources are validated.

print_roa example output



```
^_^ quatro j:0 !:508 ~/sandbox > print_roa hAjZXqoK9-pElVNF1QmCEOIRbD0.roa
Certificates:   1
CRLs:           0
SignerId[0]:    84:08:d9:5e:aa:0a:f7:ea:44:95:53:45:d5:09:82:13:42:11:6c:3d [Mat
ches certificate 0] [signingTime(U) 110316122743Z]
eContentType:   1.2.840.113549.1.9.16.1.24
version:        0 [Defaulted]
asID:           40993
 addressFamily: 1
     IPaddress: 80.80.192.0/20
     IPaddress: 89.107.192.0/21
     IPaddress: 109.111.0.0/19
^_^ quatro j:0 !:509 ~/sandbox > 
```

(Example print_roa output)



```
0_o quatro j:0 !:514 ~/sandbox > ls hAjZXqoK9-pElVNF1QmCEOIRbD0.*
hAjZXqoK9-pElVNF1QmCEOIRbD0.roa
^_^ quatro j:0 !:515 ~/sandbox > print_roa hAjZXqoK9-pElVNF1QmCEOIRbD0.roa -e
Certificates:   1
CRLs:           0
SignerId[0]:    84:08:d9:5e:aa:0a:f7:ea:44:95:53:45:d5:09:82:13:42:11:6c:3d [Mat
ches certificate 0] [signingTime(U) 110316122743Z]
eContentType:   1.2.840.113549.1.9.16.1.24
version:        0 [Defaulted]
asID:           40993
 addressFamily: 1
     IPaddress: 80.80.192.0/20
     IPaddress: 89.107.192.0/21
     IPaddress: 109.111.0.0/19
0_o quatro j:0 !:516 ~/sandbox > ls hAjZXqoK9-pElVNF1QmCEOIRbD0.*
hAjZXqoK9-pElVNF1QmCEOIRbD0.pem  hAjZXqoK9-pElVNF1QmCEOIRbD0.roa
^_^ quatro j:0 !:517 ~/sandbox > 
```

(Extracting the EE certificate with a modified print_roa)

When checking the validity of an ROA, if the signature did not match the EE it is placed in the `invalid` directory. If the certificate chain started by the extracted EE is found to be invalid the ROA is placed in the `invalid` directory. Otherwise the ROA is placed in the `valid` directory.

Importing

Importing brings valid and invalid ROAs into the database. Which indexes ROA and certificate information by network block. The information imported into the database is the certificate URI, validity of the chain ending at the ROA, and certificate duration of the EE contained within the ROA. CA certificates are not imported into the database, their purpose (to validate ROA information) has been fulfilled.

A new PostgreSQL database was created to store the ROA and certificate information. The layout is small and simple. There are four tables: cert, roa, roa_state, and update.

The cert table contains information from the EE certificate embedded within the ROA, and some identifying information from the ROA itself.

cert table

Column	Purpose
cert_id	an index and primary key
cert_name	the short portion of the ROA filename
cert_src_uri	the download location of the certificate
cert_issue_date	when the certificate is valid after
cert_expiry_date	when the certificate is valid to

The roa table is incorrectly named, it contains prefix information extracted from the ROAs. Each prefix is associated with a roa_state, and the certificate that authorized the ROAs creation.

roa table

Column	Purpose
roa_pfx	primary key, the shortest prefix for the network block
roa_pfx_len_max	the longest prefix for the network block
roa_as	AS authorized to announce
roa_state_id	foreign key to roa_state table
cert_id	foreign key to the cert table

To normalize the occurence of “valid” and “invalid” in the database the roa_state table was created.

roa_state table

Column	Purpose
roa_state_id	an index and primary key
roa_state_name	a short name for the state
roa_state_descr	a long description of the state

Lastly the update table indicates the last time the database was updated.

update table

Column	Purpose
last_update	the time of the last import

To populate these tables a new script was written, `rpki-import.pl`. It is run by the cron entry, after `rpki-vdate.pl` has completed. Before adding information to the database, the first operation it performs is that of cleaning the entire database.

The import script then walks through the `valid`, and `invalid` directories. When it finds a ROA, it extracts the information from it using the modified print_roa and `openssl` utilities. Finally placing the extracted information into the database.

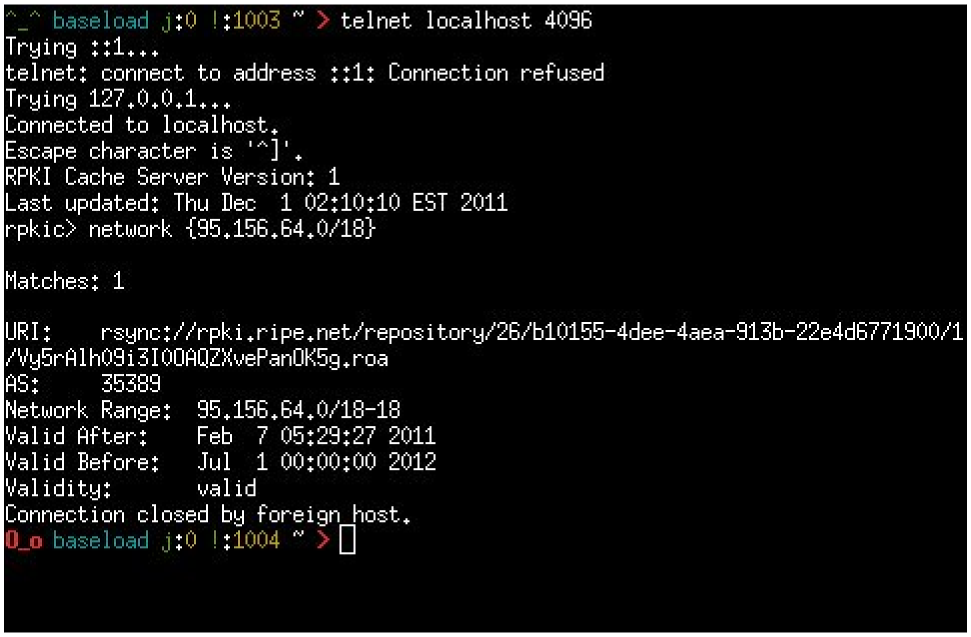
Serving

The last component of the cache is the server. Which expects a network block as input and outputs the RPKI cache information. Implement as a perl script, `rpki-checkd.pl` listens for TCP connections on port 4096. Using a small select loop, the server forks a process per accepted connection. Each accepting process services only one query, then terminates the TCP connection, and exits the process.

There is no protocol for sending a query and receiving a response. All data is passed as ASCII characters through the TCP stream. Instead of a protocol a simple syntax is used, similar to a CLI. There is a single network command that takes one argument, the network block, surrounded by curly braces.

After receiving a network block as input, the database is consulted to find the most specific match. It’s possible that there may be multiple matches of the same specificity. Meaning, that a network block with a prefix length of 16 may be matched by multiple entries. Each entry represents an unique ROA that is authorized to advertise the network block. A response is returned as ASCII starting with total number of matches, for each match the ROA content, EE information, and validity are returned.

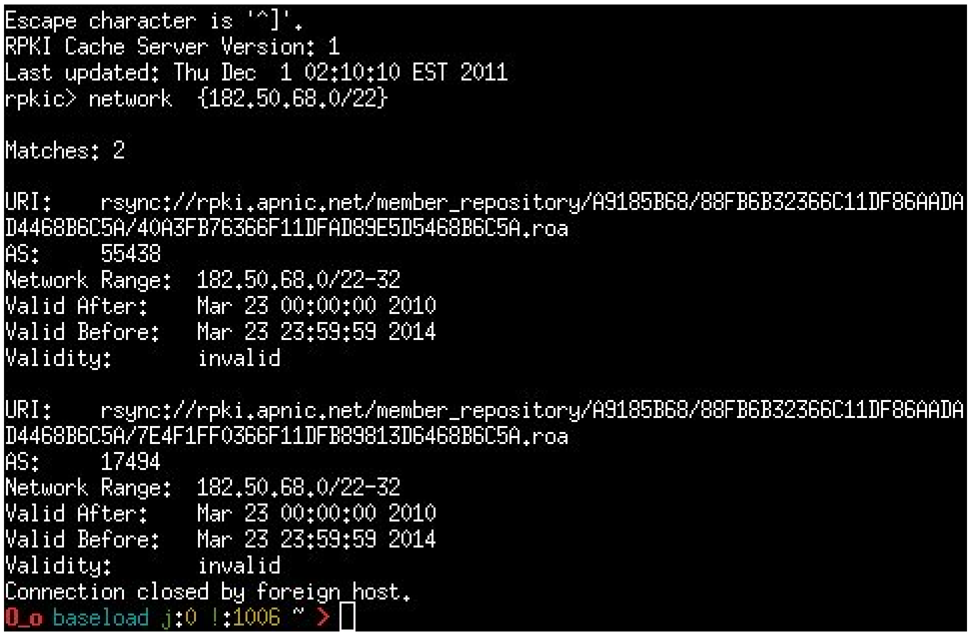
rpki-checkd.pl example



```
^_^ baseload j:0 !:1003 ~ > telnet localhost 4096
Trying ::1...
telnet: connect to address ::1: Connection refused
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
RPKI Cache Server Version: 1
Last updated: Thu Dec  1 02:10:10 EST 2011
rpkic> network {95.156.64.0/18}

Matches: 1

URI:    rsync://rpki.ripe.net/repository/26/b10155-4dee-4aea-913b-22e4d6771900/1
/Vy5rAlh09i3IOOAQZXvePanOK5g.roa
AS:     35389
Network Range:  95.156.64.0/18-18
Valid After:    Feb  7 05:29:27 2011
Valid Before:   Jul  1 00:00:00 2012
Validity:       valid
Connection closed by foreign host.
O_o baseload j:0 !:1004 ~ >
```

(Accessing the rpki-checkd.pl server over a TCP socket)

multiple matches example



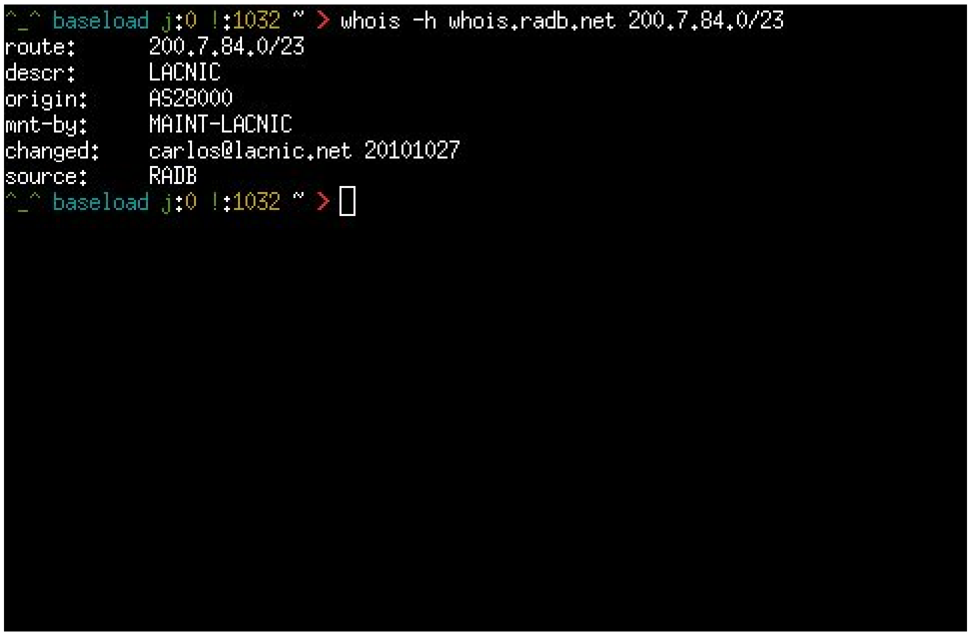
```
Escape character is '^]'.
RPKI Cache Server Version: 1
Last updated: Thu Dec  1 02:10:10 EST 2011
rpkic> network  {182.50.68.0/22}

Matches: 2

URI:    rsync://rpki.apnic.net/member_repository/A9185B68/88FB6B32366C11DF86AADA
D4468B6C5A/40A3FB76366F11DFAD89E5D5468B6C5A.roa
AS:     55438
Network Range:  182.50.68.0/22-32
Valid After:    Mar 23 00:00:00 2010
Valid Before:   Mar 23 23:59:59 2014
Validity:       invalid

URI:    rsync://rpki.apnic.net/member_repository/A9185B68/88FB6B32366C11DF86AADA
D4468B6C5A/7E4F1FF0366F11DFB89813D6468B6C5A.roa
AS:     17494
Network Range:  182.50.68.0/22-32
Valid After:    Mar 23 00:00:00 2010
Valid Before:   Mar 23 23:59:59 2014
Validity:       invalid
Connection closed by foreign host.
O_o baseload j:0 !:1006 ~ >
```

(Multiple RPKI matches for a single network block)

IRRd Modifications

Returning to the goal of the project, including RPKI information within a WHOIS response served by IRRd. This final step is accomplished by modifying IRRd to connect and query `rpki-checkd.pl`, delivering the RPKI response after the WHOIS information. Before any modifications were made to IRRd, an example WHOIS query and response looks like this.

Unmodified WHOIS query and response



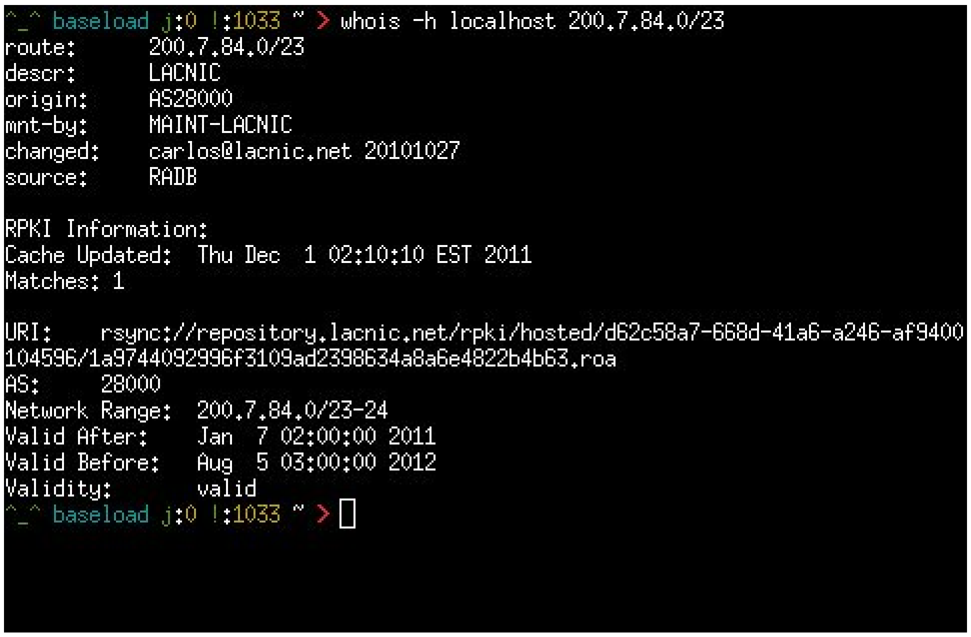
```
^_^ baseload j:0 !:1032 ~ > whois -h whois.radb.net 200.7.84.0/23
route:      200.7.84.0/23
descr:      LACNIC
origin:     AS28000
mnt-by:     MAINT-LACNIC
changed:    carlos@lacnic.net 20101027
source:     RADB
^_^ baseload j:0 !:1032 ~ >
```

WHOIS queries are sent as ASCII text, and returned in the same format. Making response modifications to IRRd straightforward. After the existing WHOIS response has been collected and formatted, a new response is inserted before closing the TCP socket to the client.

To create the response, a single function call (`irr_rpki`) is added to the end of the existing WHOIS code path. Two new source files are added to IRRd `rpkic.c` and `rpkic.h`. These two files handle all of the interactions with the `rpki-checkd.pl` server, and the WHOIS client. Initiating a new TCP connection to the RPKI cache server, executing the query, and transmitting the response (verbatim) to the WHOIS client.

With all of these pieces in place the goal of the project is achieved. Shown in the next example image, a single ROA is associated with the 200.7.84.0/23 network block. It came from the lacnic repository, identified by the URI portion of the response. Since the ROA is valid, it authorizes AS 28000 to originate routes to the network block from January 7, 2011 until August 5, 2012.

WHOIS response with RPKI information



```
^_^ baseload j:0 !:1033 ~ > whois -h localhost 200.7.84.0/23
route:       200.7.84.0/23
descr:       LACNIC
origin:      AS28000
mnt-by:      MAINT-LACNIC
changed:     carlos@lacnic.net 20101027
source:      RADB

RPKI Information:
Cache Updated:  Thu Dec  1 02:10:10 EST 2011
Matches: 1

URI:    rsync://repository.lacnic.net/rpki/hosted/d62c58a7-668d-41a6-a246-af9400
104596/1a9744092996f3109ad2398634a8a6e4822b4b63.roa
AS:     28000
Network Range:  200.7.84.0/23-24
Valid After:    Jan  7 02:00:00 2011
Valid Before:   Aug  5 03:00:00 2012
Validity:       valid
^_^ baseload j:0 !:1033 ~ > 
```

Future Work

While this project has successfully reached it's goal, there is always room for improvement on the results themselves, as well as further extensions. At it's conclusion this project is too fragile for heavy use by network operators. The following list enumerates some, but not all, of the improvements and features that could be added to this system. They are ordered by importance, starting with the most important.

1. Configuration option for the RPKI cache - The modifications to IRRd did not include a configuration option for the rpki-checkd.pl server address or port. It is currently hard coded to be on the loopback at 4096.
2. Cross repository validation support - At the time this work was completed there was no root certificate in use by IANA. The validation tool does not support cross repository validation of certificates. Which will be required if IANA creates and begins using such a certificate
3. Utilize the RPKI Cache protocol - There is a draft in the progress of being standardized (http://tools.ietf.org/html/draft-ietf-sidr-rpki-rtr-13) which specifies a protocol to be used between routers and an RPKI cache. `rpki-checkd.pl` could be modified to speak this protocol, which would then require IRRd to communicate with the cache server using the same protocol.
4. More efficient database management - During import merge the new certificates, do not clear the entire database on the start of a new import.
5. More efficient certificate validation - During the validation process when an intermediate CA certificate is found to be invalid it is not recorded. Instead subordinate certificates are validated with the invalid certificate in the chain, which fails validation. While this yields the correct result it is inefficient. The CA could be marked as invalid and all subordinate certificates would not require a validation.

Project Sources

A git repository has been published on github, including this paper and all the related materials. Including the versions of the packages that were modified, and the modifications as patch sets. These sources can be cloned out from:

https://github.com/ctessler/irrd-to-rpkicache

References

Merit

1. Welcome to Merit Neroa_state_name. 2011/11/27. <http://www.merit.edu>
2. IRRd - Internet Routing Registry Daemon. 2011/11/27. <http://www.irrd.net>
3. Welcome to Merit RADb. 2011/11/27 <http://www.radb.net>

RFC

4. WHOIS Protocol Specification. 2011/11/27. <http://tools.ietf.org/rfc/rfc3912.txt>
5. A Border Gateway Protocol 4 (BGP-4). 2011/11/27 <http://tools.ietf.org/rfc/rfc4271.txt>
6. Classless Inter-domain Routing (CIDR). 2011/11/27 <http://tools.ietf.org/rfc/rfc4632.txt>
7. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. 2011/11/27. <http://tools.ietf.org/rfc/rfc5280.txt>

RPKI

8. Resource Public Key Infrastructure (RPKI). 2011/11/27. <https://www.arin.net/resources/rpki.html>
9. Internet Society - IETF Journal - Resource Certification. 2011/11/27. <http://isoc.org/wp/ietfjournal/?p=597#more-597>
10. An Infrastructure to Support Secure Internet Routing draft-ietf-sidr-arch-13.txt. 2011/11/27. <http://tools.ietf.org/id/draft-ietf-sidr-arch-13.txt>
11. A Profile for X.509 PKIX Resource Certificates draft-ietf-sidr-res-certs-22. 2011/11/27. <http://tools.ietf.org/id/draft-ietf-sidr-res-certs-22.txt>
12. A Profile for Route Origin Authorizations (ROAs) draft-ietf-sidr-roa-format-12.txt. 2011/11/27. <http://tools.ietf.org/id/draft-ietf-sidr-roa-format-12.txt>
13. Signed Object Template for the Resource Public Key Infrastructure draft-ietf-sidr-signed-object-04.txt. 2011/11/27. <http://tools.ietf.org/id/draft-ietf-sidr-signed-object-04.txt>

MISC

10. IANA - Internet Assigned Numbers Authority. 2011/11/27. <http://www.iana.org>
11. OpenSSL: The Open Source toolkit for SSL/TLS. 2011/11/27. <http://openssl.org>
12. Resource PKI Software. 2011/11/27 <http://rpki.net>
13. PostgreSQL: Welcome. 2011/11/27 <http://postgresql.org>