Introduction

Network Engineer at Cloudflare in San Francisco

Open-source projects including flows and RPKI

Network data collection (BGP, flows, peering-portal)

https://blog.cloudflare.com/rpki-details/
https://blog.cloudflare.com/rpki/
BGP leaks and cryptocurrencies

24 Apr 2018 by Louis Poirognon.

Over the few last hours, a dozen news stories have broken about how an attacker attempted (and perhaps managed) to steal cryptocurrencies using a BGP leak.
Story 2/4

Authority DNS route hijack in April 2018.

This affected our DNS Resolver.

The route was sent to us on a Chicago peering session.

What should we do?
Story 3/4

At the time...

150+ PoPs, 26000 BGP sessions, IP space in 5 RIRs

Just the RIPE Validator[1]

How to distribute a prefix list efficiently?

[1] Cloudflare is very grateful for the RIPE Validator s/w
July: started deploying internally GoRTR.

August: open-source release.  
https://github.com/cloudflare/gortr

September → December:

- Turn up RTR sessions
- Signing prefixes
Behind the scene (until January 2019)

**RIPE Validator** providing list of prefixes.

Running in a Mesos cluster.

With a cronjob:
- Fetching the data
- Filtering it (remove > /24 and > /48 and duplicates)
- Signing it
- Making it available to our edge.

https://rpki.cloudflare.com/rpki.json was born.
Effects

The question everyone asked us.

How much traffic was affected?

Many invalids. Little traffic in practice (default or valid less specific).

Except in one place. Few gigabits per seconds displaced due to geographical more specific.

https://www.flickr.com/photos/thure/6287816628/
Signing the routes
Signing the routes

IP space in 5 RIRs (*no twnic/jpnic/cnnic*). Not a unified experience.

<table>
<thead>
<tr>
<th>RIR</th>
<th>Features</th>
<th>Ease of use</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFRINIC</td>
<td>⭐️</td>
<td>⭐️</td>
<td>⭐️</td>
</tr>
<tr>
<td>APNIC</td>
<td>⭐⭐️</td>
<td>⭐⭐️</td>
<td>⭐⭐️</td>
</tr>
<tr>
<td>ARIN</td>
<td>⭐⭐️</td>
<td>⭐⭐⭐</td>
<td>⭐⭐⭐</td>
</tr>
<tr>
<td>LACNIC</td>
<td>⭐️</td>
<td>⭐⭐⭐</td>
<td>⭐️</td>
</tr>
<tr>
<td>RIPE</td>
<td>⭐⭐⭐⭐⭐</td>
<td>⭐⭐⭐</td>
<td>⭐⭐⭐</td>
</tr>
</tbody>
</table>
Rankings

Features: RRDP, 2 factors, extra info, CA.

Ease of use: steps to sign a ROA, multi user.

API: functional, complete and accessible.
Comparison - AFRINIC

Hard to set up: client TLS certificate to create (BPKI) in order to do RPKI.

Buggy.

No RRDP.

No API.

No auto-renew.

Hosted CA possible.

Extensive certificate informations.
Comparison - APNIC

Two factors or client certificate.
RRDP.
Auto-renew.
Allow BGP batch signing.
(slight bugs with big amount of prefixes).
Hosted CA possible.

Draft for API:
https://www.apnic.net/manage-ip/apnic-services/services-roadmap/public-api-draft-for-members/
Comparison - ARIN

Two factors. Separate signing key.
No RRDP.
No auto-renew.
Semi-functional API (add).
Dashboard not easy to find.
Hosted CA possible.
Slow rsync update (4 times a day).

Some certificate information.
Comparison - LACNIC

No two factors. Single user.
No RRDP.
No API.
Auto-renew opt-in.
Allow BGP batch signing.
Based off RIPE.
No Hosted CA.
Some extra info (revoked, path).
Incorrect certificate encoding (BER). High turnover of certificate (few days).
Comparison - RIPE

Two factors.
RRDP.
Auto-renew.
Nice API.
Allow BGP batch signing.
No Hosted CA (theoretically).
No extra information. But history.
Incorrect certificate encoding (BER).
Automation

We automated prefixes adding on ARIN and RIPE with a Salt state.

Two secrets to store (API key and signing key).

Cannot delete or list via API for ARIN: very prone to mistakes if user wants to reduce the amount of ROA files.

def_format_payload(roas, signature):
    template = """------BEGIN ROA REQUEST------
    {roas}
    ------END ROA REQUEST------
    ------BEGIN SIGNATURE------
    {signature}
    ------END SIGNATURE------
    """
    payload = template.format(
        roas=roas, signature="\n".join(textwrap.wrap(signature, width=64))
    )
    return payload

def__make_roa(name, asn, t, start_val, end_val, prefix, length, maxlength):
    template = (
        '\1\(time\)\(name\)\(asn\)\(start_val\)\(end_val\)\(prefix\)\(length\)\(maxlength\)'
    )
    time_str = calendar.timegm(t.timetuple())
    start_val_str = start_val.strftime(_TIME_FORMAT)
    end_val_str = end_val.strftime(_TIME_FORMAT)
    roa = template.format(
        time=time_str,
        name=name,
        asn=asn,
        start_val=start_val_str,
        end_val=end_val_str,
        prefix=prefix,
        length=length,
        maxlength=maxlength,
    )
    return roa

def__sign(pkey, roas):
    signature = pkey.sign(roas.encode('utf-8'), padding.PKCS1v15(), hashes.SHA256())
    return base64.b64encode(signature).decode('utf-8')
Validator
Why making a validator?

First release of Routinator in November 2018.
We were still using RIPE Validator.

We wanted something more custom: with monitoring and RRDP.

By building it in Go:

- Many APIs and easy for concurrency
- Community doing cryptography
- Cloudflare uses Go a lot (cfssl, sidh, etc.)
Challenges

Juniper bugs: Routing Validation disabled.

Difficulties: rsync, BER encoded instead of DER, conditions in cryptography

3) a subjectPublicKeyInfo [RFC5280] in DER format [X.509], encoded in Base64 (see Section 4 of [RFC4648]). To avoid long lines, <CRLF> or <LF> line breaks MAY be inserted into the Base64-encoded string.

where the URI section is comprised of one of more of the ordered sequence of:
Cloudflare’s RPKI Toolkit

Sets of libraries and tools written in Go.

Including *OctoRPKI* 🦀

https://blog.cloudflare.com/cloudflares-rpki-toolkit/
Cloudflare’s RPKI Toolkit

Libraries

- CER/ROA/MFT decoder
- PKI manager (exploring, validating)
- RRDP/rsync fetcher
- Validation of prefixes
Cloudflare’s RPKI Toolkit

Software

- Local validator (without RRDP/Rsync)
- API tools for a distributed version without filesystem
- OctoRPKI
- Certificate Transparency tool
OctoRPKI - Features (1/2)

- Decodes TAL/CER/ROA/MFT
- Explore via Manifest or directory.
- RRDP support (and failover to Rsync)
- Monitoring (Prometheus and JSON API which includes logs)
- Dockerizeable
- Handle stability (generate file when done)
OctoRPKI - Features (2/2)

- Full compatibility with GoRTR (including signing the JSON file)
- Server + caching options for generated file (CDN friendly)
- Configuration options
  - Disable/Enable components
  - Modes (server, one-off)
- ~5-15 minutes for a full cold-start sync
OctoRPKI - Compute footprint

CPU

[Graph showing CPU usage with peaks and valleys indicating load variations over time.]

RAM

[Graph showing memory usage with a steady line indicating average usage over time.]

avg

- rpki-benchmark-octorpki: 428 MB
- rpki-benchmark-ripe: 906 MB
- rpki-benchmark-routinator: 511 MB

OctoRPKI v1.1.3
RIPE Validator v2.25
Routinator v3.3.0
Monitoring

Resource synchronization

Last stable validation

Number of ROAs
OctoRPKI - Run it yourself

```bash
$ docker run -ti \
  -p 8080:8080 \
  -v $PWD/cache:/cache \
  -v $PWD/tals/arin.tal:/tals/arin.tal \
  cloudflare/octorpki
```

- **Container image**: `cloudflare/octorpki`
- **Open port 8080 on host**
- **Use cache folder on host**
- **Adding ARIN TAL**
- **Adding ARIN TAL**
- **Adding ARIN TAL**
- **Adding ARIN TAL**
GoRTR

OctoRPKI does not embed a RTR server. Modular and independence!
Fully compatible with GoRTR https://github.com/cloudflare/gortr
Signs the prefix list to ensure a safe distribution of the file.
Can run natively on Juniper!

```
$ docker run -ti \
- p 8082:8082 \ 
- v $PWD/example.pub:/example.pub \ 
cloudflare/gortr \ 
- verify.key /example.pub \ 
- cache https://YOUR_ROA_URL
```
GoRTR

Only software to support plaintext, SSH and TLS.

Future plans:
an open RTR service available in our 160+ PoPs

Compatibility matrix

A simple comparison between software and devices. Implementations on versions may vary.

<table>
<thead>
<tr>
<th>Device/software</th>
<th>Plaintext</th>
<th>TLS</th>
<th>SSH</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTRdump</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Juniper</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Cisco</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Only SSH password</td>
</tr>
<tr>
<td>Alcatel</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Arista</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>FRRouting</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Only SSH password</td>
</tr>
<tr>
<td>Bird</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Only SSH key</td>
</tr>
<tr>
<td>Quagga</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>
Cloudflare’s Internal Version

Is providing:
https://rpki.cloudflare.com/rpki.json

But also a **GraphQL API**

...which powers a **dashboard**
Dashboard
Certificate Transparency

Historical records of certificates.

Contains a chain.

Sent by our validator.
Other data - so how fresh are those ROAs?

ARIN uses ten year expire

LACNIC random expires

RIPE regenerates certificates!
Future projects or ideas

RPKI validation tester using our CDN:

- Using a /23 (/47 IPv4) valid and a /24 (/48 IPv6) invalid certificate encoder

ASPA

More toolings and visualizations around RPKI (BGP collection)
Thank you / Gracias / Obrigado

Questions?

louis@cloudflare.com
@lpoinsig (twitter)