

prefix_roa_scope.py — Full Technical Documentation

1. Purpose & Role in the Platform

`prefix_roa_scope.py` computes, for each `(asn, prefix)` entry in the `prefix_data` table, a precise metric called:

`roa_coverage_scope` $\in [0.0, 1.0]$

= the fraction of the prefix's IP address space that is covered by ROAs belonging to that ASN.

This script is an **auxiliary data-processing component** used by higher-level analytics such as:

- `asn_filters_strict.py`
- prefix-level strictness checks
- Vulnerability evaluation
- origin-validation consistency metrics

Its job is **not** to determine validity or security directly, but to provide **foundational ROA coverage data** required for strict filtering and hijack detection.

2. Why ROA Coverage Scope Matters

ROAs define which ASNs are authorized to originate which prefixes — but not necessarily the entire prefix space. A prefix may be:

- **fully covered** by ROAs for its origin AS → secure, predictable

- **partially covered** → mixed security, unclear authorization
- **not covered at all** → high vulnerability risk, weak validation guarantees

Coverage on a *per-IP basis* is more precise than simply checking for existence of a ROA. For example:

- A prefix /16 may be covered only by a set of /24 ROAs.
- Another prefix may have ROAs only for half its address space.
- ROAs may overlap partially or fully, requiring merge logic.

`prefix_roa_scope.py` computes the **exact fraction** of the prefix covered by the union of all ROA-authorized IP ranges for the ASN.

3. High-Level Overview of What the Script Does

For every `(asn, prefix)` in `prefix_data`:

1. Fetch all ROAs from RIPEstat using `/rpki-roas?include=more-specifics`.
2. Filter ROAs to keep only those belonging to the **same ASN**.
3. Convert the prefix and relevant ROA prefixes into numeric IP intervals.
4. Compute the **intersection** between the target prefix and each ROA interval.
5. Merge overlapping intervals to prevent double-counting.

Calculate the exact coverage fraction:

```
coverage = covered_IPs / total_IPs_in_prefix
```

- 6.

7. Store the result (`roa_coverage_scope`) back into the database.

If an error occurs (API failure, parsing error, unexpected runtime error), the script records:

- `roa_coverage_scope = NULL`
- `rpki_error_last = <error message>`

and continues.

4. Database Schema

The script ensures the following columns exist in the `prefix_data` table:

Column	Type	Description
<code>roa_coverage_scope</code>	REAL	Fraction of prefix covered by ROAs for that ASN
<code>rpki_checked_at</code>	TEXT	Timestamp of last successful coverage computation
<code>rpki_error_last</code>	TEXT	Last error encountered during processing
<code>updated_at</code>	TEXT	Timestamp of last DB update

If missing, these columns are created automatically.

5. Target Selection Logic

The script determines which prefixes to process via:

```
SELECT asn, prefix FROM prefix_data
WHERE roa_coverage_scope IS NULL      (default)
```

Default mode (no flags):

→ Only rows with **NULL** coverage are processed (incremental mode).

With **--force**:

→ All rows are processed and recalculated, regardless of previous values.

This behavior is implemented explicitly in `load_targets()`.

6. Fetching ROAs from RIPEstat

For each prefix, the script calls:

`/data/rpki-roas/data.json?resource=<prefix>&include=more-specifics`

The response may contain:

- ROAs for the exact prefix
- ROAs for broader prefixes that cover it
- ROAs for more-specific prefixes inside it

The script applies:

- Retry logic for 429 / 5xx / timeouts
 - Adaptive rate limiting (`AdaptiveRL`)
 - Parsing of multiple possible ROA field names (`prefix`, `route`, `roa_prefix`, etc.)
-

7. Detailed Algorithm for Coverage Calculation

This is the core logic and the part most under-documented in the original PDF. Here is the step-by-step version that matches the script precisely.

7.1 Convert the target prefix into a numeric interval

Example:

`192.0.2.0/24 → start=3221225984, end=3221226239`

This allows precise interval math independent of prefix-length boundaries.

7.2 Filter ROAs to those belonging to the target ASN

Only ROAs where:

`asn_from_roa == origin_asn`

are considered.

Reason:

→ We want to compute **how well the prefix is covered by ROAs that authorize *this specific ASN***.

Coverage by ROAs of other ASNs is irrelevant.

7.3 Convert each ROA prefix into an interval

Each ROA prefix becomes `(start, end)` using the same numeric conversion.

7.4 Compute intersection with the target prefix

For each ROA interval:

```
intersection_start = max(roa_start, prefix_start)
intersection_end   = min(roa_end, prefix_end)
```

If intervals do not overlap → skip.

This trims ROA ranges to only the part that is inside the target prefix.

7.5 Merge overlapping intervals (Interval Union)

If we have multiple ROA intervals such as:

[0–99] and [50–150]

the union is:

[0–150]

This step ensures **no double-counting** of overlapping ROAs.

The script uses `interval_union_len(intervals)` to obtain:

→ the exact number of covered IPs.

7.6 Compute total and covered address counts

```
total_IPs    = prefix_end - prefix_start + 1
covered_IPs = union_length(intervals)
```

7.7 Compute the ROA coverage fraction

```
roa_coverage_scope = covered_IPs / total_IPs
```

Always in [0.0, 1.0].

Examples:

- 1.0 → fully covered
 - 0.0 → no ROA coverage
 - 0.25 → only 25% of addresses are authorized for this ASN via ROAs
-

8. Error Handling

If any unexpected failure occurs:

- network/API error
- invalid prefix
- JSON decode failure
- runtime errors

The script stores:

```
roa_coverage_scope = NULL  
rpki_error_last = "<error>"
```

And continues processing.

Successful processing sets:

```
rpki_error_last = NULL  
rpki_checked_at = now()
```

9. Concurrency & Continuous Operation

The script supports:

- asynchronous concurrency (`asyncio`)
- configurable worker count (`--concurrency`)
- rate limiting (`--rps`)
- request timeout (`--timeout`)
- periodic sleep (default 3600s) via `run_forever()`

It can operate as a **daemon**, continuously updating ROA coverage as ROAs evolve.

10. Usage Examples

10.1 Process only missing entries (default incremental mode)

```
python3 prefix_roa_scope.py --db database/asn_data.db
```

10.2 Recompute all prefixes (full refresh)

```
python3 prefix_roa_scope.py --db database/asn_data.db --force
```

10.3 Continuous mode (looping daemon)

```
python3 prefix_roa_scope.py
```

11. Interpretation of the Metric

roa_coverage_scope = 1.0

- The ASN has full ROA coverage for this prefix.
- Strong indicator of clean origin configuration.
- High resistance to hijacking.

0.0 < roa_coverage_scope < 1.0

- Partial coverage.
- Indicates mixed ROA configuration or delegated subranges.
- Important signal for strict filtering.

0.0

- No ROA exists that authorizes this ASN for this prefix.
 - High hijacking risk.
-

12. Summary

`prefix_roa_scope.py` is a precision tool that converts raw ROA data into **accurate per-prefix coverage scores**. It uses:

- interval math
- overlap resolution
- ASN-specific filtering
- precise per-IP accounting

to deliver a metric essential for the vulnerability framework.