

prefix_vantage.py — Prefix Visibility Collector (Full Technical Documentation)

1. Purpose & Role

`prefix_vantage.py` computes **vantage_points_seen**, representing how many **RIPE RIS peers** have observed a given prefix.

This metric captures global **visibility**, **monitorability**, and **propagation footprint** of a prefix.

It is used in:

- prefix-level ML vulnerability classification
- risk-surface modeling (prefix visibility vs. ASN weaknesses)
- hijack-stealth estimation
- anomaly scoring (low-visibility = high risk)
- propagation-profile clustering

The script provides a **direct, real-time measurement** of how many external collectors see the prefix.

2. High-Level Behavior

At a high level, the script:

1. Loads prefixes (ASN + prefix) from the SQLite table `prefix_data`.
2. Ensures all required vantage-related columns exist.

3. For each prefix:
 - Connects to `riswhois.ripe.net:43`.
 - Sends the prefix as a WHOIS-style query.
 - Receives a text dump with routing metadata.
 - Extracts the number of **RIPE RIS peers** that have seen the prefix.
4. Writes the result to the DB with timestamps and error info.
5. Runs continuously (hourly loop) unless `--once` is used.

The script is a **lightweight, I/O-optimized visibility probe**, designed for millions of prefixes.

3. Metrics Produced

3.1 `vantage_points_seen` (INTEGER)

The number of **distinct RIS peers** that have observed the prefix, extracted via regex patterns:

- `num-rispeers:`
- `number-of-rispeers:`
- `peers seen:`

3.2 `vantage_sources_used` (TEXT)

Always set to `"RISWhois"`.

3.3 `vantage_checked_at` (TEXT)

Timestamp (ISO 8601) of the latest check.

3.4 `vantage_error_last` (TEXT)

Stores the last error message if the WHOIS request failed.

3.5 updated_at (TEXT)

Timestamp of any update.

4. Database Contract

4.1 Requirements

Input table must include:

- `asn` INTEGER
- `prefix` TEXT

Script automatically adds the following columns if missing:

- `vantage_points_seen`
- `vantage_sources_used`
- `vantage_checked_at`
- `vantage_error_last`
- `updated_at`

4.2 Update Logic

Rows are *updated*, not inserted:

```
UPDATE prefix_data
SET vantage_points_seen=?,
    vantage_sources_used='RISWhois',
    vantage_checked_at=?,
    vantage_error_last=NULL,
```

```
    updated_at=?  
WHERE asn=? AND prefix=?;
```

On error:

```
UPDATE prefix_data  
SET vantage_error_last=?,  
    vantage_checked_at=?,  
    updated_at=?  
WHERE asn=? AND prefix=?;
```

5. Data Flow Overview

5.1 Target Loading

`load_targets()` selects prefixes with missing `vantage_points_seen` unless `--force` is used.

5.2 Query Execution

For each prefix:

`prefix` → `RISWhois TCP query` → `text dump` → `regex extract` → `number of RIS peers`

This is done using:

- direct TCP connection (`asyncio.open_connection`)
- WHOIS-style text protocol
- robust multi-regex parsing
- retry logic (up to `DEF_RETRIES`)

5.3 Result Writing

Results are applied immediately via individual DB updates.

No batch merges — this matches the script's IO model exactly.

6. Performance & Concurrency Architecture

Concurrency

- Uses `asyncio.Semaphore` with configurable `--concurrency` (default: 200).

Rate Limiting

Custom rate limiter:

```
RateLimiter(rps)
```

Enforces global RPS limit to RISWhois.

IO Architecture

- Fully asynchronous TCP connections
- Sequential WHOIS text reading
- Error retry per prefix
- Immediate commit after each prefix

Scalability

The script handles:

- hundreds of concurrent requests
- tens of thousands of prefixes per hour
- continuous operation in hourly loops

7. Control Loop Behavior

Single Round

`run_once(args):`

- processes all targets
- writes DB state
- prints per-prefix logs
- returns cleanly

Continuous Mode

`run_forever(args):`

- executes `run_once()`
- sleeps 3600 seconds
- repeats indefinitely
- supports graceful SIGINT

8. CLI Arguments

Argument	Meaning	Default
<code>--db</code>	Path to SQLite DB	asn_data.db
<code>--force</code>	Recompute all rows	False

<code>--limit</code>	Process N rows only	0
<code>--concurrency</code>	Max concurrent workers	200
<code>--rps</code>	Global requests per second	20
<code>--timeout</code>	TCP timeout	15
<code>--db-batch</code>	Legacy (unused)	2000
<code>--once</code>	Run only one round	False

9. Reliability Justification — Why RISWhois Is the Correct Source

Why RISWhois is an appropriate and reliable source for vantage-count data

1. **Direct access to RIPE RIS infrastructure**

RISWhois exposes metadata from RIPE’s global routing information system — the same data RIPE uses in its research and public tools.

2. **Standard WHOIS-like interface**

RISWhois uses a stable, text-based protocol supported by nearly all route-analysis toolchains.

3. **Specifically optimized for “num-rispeers” lookup**

RISWhois responses include fields like:

- `num-rispeers:`
- `number-of-rispeers:`
- `peers seen:`

These fields *only* exist in RISWhois and are not part of other BGP collectors.

4. **Independent from live BGP feeds**

It avoids issues such as temporary RIS outage, session resets, BMP delays, or

streaming throttling.

5. **Lightweight and scalable**

TCP WHOIS queries are efficient and impose near-zero load on RIS, enabling large-scale visibility polling.

6. **Used widely in operational research**

Many routing-security papers and tools rely on RISWhois when needing vantage counts instead of full BGP dumps.

Conclusion

RISWhois is the correct data source for **vantage_points_seen** because it provides:

- authoritative data
 - consistent formatting
 - low-latency responses
 - global RIS coverage
 - predictable performance at scale
-

10. Usage Examples

Process only missing rows

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

Force recomputation

```
python3 prefix_vantage.py --force
```

Limit processing to first 500 prefixes

```
python3 prefix_vantage.py --limit 500
```


Run only one round

```
python3 prefix_vantage.py --once
```

Notes on Database Path Resolution

When `--db` is supplied with a *relative* path, the script **ignores that value** and falls back to the internal default database location:

```
PROJECT_ROOT/database/asn_data.db
```

Only **absolute paths** are used as-is and override the default.

11. Integration with the Platform

This script feeds:

- ML prefix vulnerability scoring
- combined ASN+prefix risk modeling
- stealth-propagation scoring
- anomaly detection (visibility regressions)
- route-leak detection heuristics

Low visibility = high risk

High visibility = widely monitored prefix (lower stealth risk)

12. Limitations

- Depends on RISWhois TCP service availability.
- WHOIS text outputs are unstructured → regex parsing required.
- Vantage count reflects RIS peers, not all BGP collectors in the world.