

prefix_data.py — Full Technical Documentation

AS-Level Prefix Enumeration & Database Population

1. Purpose & Role in the Platform

`prefix_data.py` is the **root collector** that enumerates *all prefixes announced by every ASN* in the `asn_data` table.

This script establishes the foundational dataset for all prefix-level metrics computed later (ROA, IRR, vantage visibility, prefix length, etc.).

It answers the essential question:

“What exact prefixes does each ASN announce globally?”

This makes `prefix_data.py` the **first step** in the prefix-pipeline.

No ML, security scoring, or vulnerability modeling is possible without this base dataset.

2. High-Level Behavior

At a high level, the script:

1. Loads all ASNs from `asn_data.asn`.
2. Determines which ASNs still need prefix collection (resume mode).
3. Queries two independent data sources:
 - **RIPEstat announced-prefixes**
 - **BGPView ASN prefixes** (fallback)

4. Normalizes all prefixes and assigns families (IPv4/IPv6).
5. Inserts them into `prefix_data` with `(asn, prefix)` as primary key.
6. Runs continuously in rounds (default: every 3600 seconds).

3. Metrics Produced

For each `(asn, prefix)`, the script populates:

Column	Meaning
<code>asn</code>	Origin ASN owning the prefix entry
<code>prefix</code>	Canonical prefix string (e.g., <code>192.0.2.0/24</code>)
<code>family</code>	4 (IPv4) or 6 (IPv6)
<code>source</code>	<code>"ripestat"</code> , <code>"bgpview"</code> , or <code>"mixed"</code>
<code>created_at</code>	First time prefix was inserted
<code>updated_at</code>	Last time the row was refreshed

No security metrics are calculated here — this script populates the **raw structural dataset** required by all others.

4. Database Contract

4.1 Input Requirements

Requires a valid table:

```
asn_data(asn INTEGER NOT NULL)
```

4.2 Output Schema (created automatically)

The script creates the following table if it doesn't exist:

```
prefix_data (  
  asn INTEGER NOT NULL,  
  prefix TEXT NOT NULL,  
  family INTEGER NOT NULL,  
  source TEXT NOT NULL,  
  created_at TEXT NOT NULL,  
  updated_at TEXT NOT NULL,  
  PRIMARY KEY (asn, prefix)  
);
```

Indexes automatically created:

- `idx_prefix_data_asn`
- `idx_prefix_data_family`

4.3 Insert/Replace Contract

The writer performs:

```
INSERT OR REPLACE INTO prefix_data(...)
```

This guarantees:

- no duplicate prefixes
- atomic replacement on re-collection
- consistent timestamps

5. Data Flow Overview

5.1 Load ASNs

- `read_all_asns()` retrieves all ASNs from `asn_data`.
- Resume mode skips ASNs already present in `prefix_data`.

5.2 Fetch Prefixes

Two sources are used:

Primary Source: RIPEstat announced-prefixes

`https://stat.ripe.net/data/announced-prefixes/data.json?resource=AS<asn>`

Fallback: BGPView

`https://api.bgpview.io/asn/<asn>/prefixes`

If RIPEstat fails, BGPView is used.

If both fail → AS is logged as having 0 visible prefixes.

5.3 Normalize

`normalize_prefix_list()` ensures:

- prefixes are strings
- unwanted entries are removed
- families (IPv4/IPv6) assigned

5.4 Write to DB

An async writer queue batches commits (`db_writer()`).

6. Performance & Concurrency Architecture

6.1 Adaptive Rate Limiters

Two independent adaptive limiters:

- `ripestat_rps` (default 12 RPS)
- `bgpview_rps` (default 3 RPS)

These automatically increase or decrease based on API responses.

6.2 Dual aiohttp Sessions

Two independent HTTP clients:

- one for RIPEstat
- one for BGPView

6.3 Massive Concurrency

- Up to **400 concurrent RIPEstat sessions**
- Up to **40 concurrent BGPView sessions**
- Total semaphore capped around **≤ 1200 tasks**

6.4 Efficient Writer

Asynchronous DB batch writer with:

`INSERT OR REPLACE`

to minimize lock contention.

7. Control Loop Behavior

`main_loop()`:

1. Runs one full collection round.
2. Sleeps for `--interval` seconds (default 3600).
3. Repeats indefinitely.
4. Exits cleanly on SIGINT/SIGTERM.

This ensures **fully autonomous 24/7 operation**.

8. CLI Arguments

Flag	Description	Default
<code>--db</code>	SQLite path	asn_data.db
<code>--timeout</code>	HTTP timeout seconds	30
<code>--db-batch</code>	Rows per DB commit	5000
<code>--ripestat-rps</code>	RPS target	12
<code>--ripestat-conc</code>	Concurrent connections	400
<code>--bgpview-rps</code>	RPS limit	3
<code>--bgpview-conc</code>	Concurrent connections	40
<code>--sources</code>	Priority list	ripestat bgpview
<code>--resume</code>	Skip ASNs already populated	ON

<code>--force</code>	Re-scan all ASNs	OFF
<code>--interval</code>	Sleep between rounds	3600

9. Reliability Rationale — Why These Data Sources Were Selected

RIPEstat announced-prefixes

RIPEstat is the most widely used global routing dataset for BGP research and operations:

- Uses RIPE RIS + global collectors
- Extremely high uptime and capacity
- Standard source for prefix visibility in academic papers
- Industry-trusted across IXPs, vendors, research labs
- Delivers structured, normalized prefix data ideal for automation

BGPView Fallback

BGPView is used because:

- Data model complements RIPEstat
- Independent routing feeds increase accuracy
- Provides redundancy when RIPEstat rate-limits or API glitches occur
- Excellent for IPv6 coverage

Why Dual-Sourcing Matters

This platform is a *security* product.

Redundant data sources eliminate blind spots:

- increases completeness
- avoids single-point-of-failure
- ensures higher prefix discovery rate
- guarantees robustness in production environments

This dual-source design is a major enterprise feature, not a hobbyist choice.

10. Usage Examples

Collect prefixes for all ASNs:

```
python3 prefix_data.py
```

Force refresh every ASN:

```
python3 prefix_data.py --force
```

Use BGPView as primary:

```
python3 prefix_data.py --sources bgpview ripestat
```

11. Integration With the Platform

The outputs feed directly into:

- **prefix_has_roa.py**
- **prefix_vrp_count.py**
- **prefix_roa_maxlen.py**

- `prefix_irr_exact.py`
- `prefix_vantage.py`
- `prefix_length.py`

Meaning:

This script generates the authoritative prefix list used by all subsequent vulnerability and ML stages.

`prefix_data.py` is the foundational ingestion step of the prefix-processing pipeline. All subsequent scripts require its dataset, and none of them can run correctly until `prefix_data.py` has populated the `prefix_data` table.

12. Limitations

- Dependent on RIPEstat/BGPView availability
- Does not validate routing correctness (just enumeration)
- Prefers RIPEstat even if BGPView may show more/alternate prefixes