

asn_relations_import.py — Technical Documentation

(Auxiliary CAIDA AS-Relationship Importer for `asn_filters_strict.py` and related metrics)

1. Purpose & Role in the Platform

`asn_relations_import.py` is an auxiliary data-ingestion script.

Its sole purpose is to:

- Import AS relationship data from CAIDA (as-rel2 format)
- Populate / update the local `as_rel` table in the SQLite database used by the BGP analytics pipeline

This table is then consumed by higher-level scripts such as:

- `asn_filters_strict.py` – which uses AS relationships to:
 - classify edges as customer, provider, or peer
 - detect valley-free violations
 - reason about where leaks and hijacked routes are most likely to originate or propagate

In short, this script does no security scoring on its own.

It is a simple, robust data collector that provides the structural AS topology context needed for accurate strict-filtering and vulnerability analysis.

2. High-Level Overview

The script performs the following steps:

- Opens (or creates) the project SQLite database in the `database/` folder
 - Ensures the `as_rel` table exists with the correct schema
 - Parses a CAIDA as-rel2 text file line by line
 - Converts CAIDA relation codes to human-readable relationship types:
 - `-1` → provider–customer
 - `0` → peer–peer
 - Inserts directed AS-relationship rows into `as_rel`
 - Optionally clears all previous rows before import
 - Prints basic progress and final row count when `--verbose` is enabled
-

3. Database and File Layout

3.1 Database Location

The script assumes a project layout like:

```
project_root/  
├─ database/  
│   └─ asn_data.db (or a custom name)  
└─ scripts/  
    └─ asn_relations_import.py
```

At runtime:

- It resolves its own directory (`script_dir`)
- Takes the parent directory as `project_root`
- Ensures `project_root/database/` exists

- Opens `<db_dir>/<db_name>` where:
 - `db_name` defaults to `asn_data.db` (or the value of `--db`)

This makes it consistent with the rest of the platform.

3.2 `as_rel` Table Schema

The script creates (or recreates if incompatible) the `as_rel` table:

```
CREATE TABLE IF NOT EXISTS as_rel (  
    asn      INTEGER NOT NULL,  
    neighbor INTEGER NOT NULL,  
    relation TEXT     NOT NULL,  
    source   TEXT     NOT NULL,  
    PRIMARY KEY(asn, neighbor, source)  
);
```

Field meaning:

- `asn` – the local ASN (one endpoint of the relationship)
- `neighbor` – the remote ASN (other endpoint)
- `relation` – relationship type from the perspective of `asn`:
 - `"customer"` – `asn` is a provider of `neighbor`
 - `"provider"` – `asn` is a customer of `neighbor`
 - `"peer"` – `asn` and `neighbor` are peers
- `source` – textual tag (e.g., `"caida-serial2"`) describing the data source or snapshot

The `PRIMARY KEY (asn, neighbor, source)` ensures:

- No duplicate edges are inserted for the same direction, neighbor, and source
- The same relationship can be imported from multiple sources or snapshots by using different `source_tag` values

If there is already a table named `as_rel` but its columns do not contain at least `asn`, `neighbor`, and `relation`, the script:

- Logs a warning when `--verbose` is set
- Drops the old table
- Recreates it with the expected schema

This guarantees schema compatibility for all scripts that depend on `as_rel`.

4. Input Format — CAIDA as-rel2 Files

The script expects the CAIDA as-rel2 format text file as input.

Typical usage:

```
wget
https://data.caida.org/datasets/as-relationships/serial-2/20251201.as-
rel2.txt.bz2
bunzip2 20251201.as-rel2.txt.bz2

python3 asn_relations_import.py \
  --file 20251201.as-rel2.txt \
  --db asn_data.db \
  --verbose
```

Each non-comment line in the CAIDA file usually looks like:

```
<asnA> | <asnB> | <rel_code>
```

where:

- `rel_code = -1` → a provider–customer relationship
- `rel_code = 0` → a peer–peer relationship

Other codes are ignored (with a warning in verbose mode).

The script skips:

- empty lines
 - lines beginning with `#`
 - lines that cannot be parsed as integers
 - lines with unknown `rel_code` values
-

5. Import Logic (Step-by-Step)

The core function `import_asrel()` implements the full import pipeline.

5.1 Table Initialization

`ensure_table(conn, verbose=verbose)`

- Validates that `as_rel` exists with the expected schema
 - Drops and recreates the table if incompatible
-

5.2 Optional Clear of Existing Data

If `clear_old == True` (default):

- Executes `DELETE FROM as_rel`
- Fully clears any previous AS-relationship rows

If `--keep-old` is passed, `clear_old` becomes `False`, and existing data is preserved.

5.3 Line-by-Line Parsing

For each line in the input file:

- Strip whitespace
- Skip comments (`#...`) and empty lines
- Split on `" | "` and validate at least 3 fields
- Parse:
 - `a = int(parts[0])`
 - `b = int(parts[1])`
 - `rel_code = int(parts[2])`

Depending on `rel_code`:

`rel_code == -1` (provider-customer)

- `provider = a`
- `customer = b`

Insert two directed edges:

- `(provider, customer, "customer", source_tag)`
- `(customer, provider, "provider", source_tag)`

`rel_code == 0` (peer-peer)

Insert two directed edges:

- (a, b, "peer", source_tag)
- (b, a, "peer", source_tag)

Unknown rel_code

- Skip line (with warning in verbose mode)

This modeling as two directed rows per relationship is crucial for downstream algorithms:

- It allows `asn_filters_strict.py` to query relationships from the perspective of a single ASN
- Directionality is explicit while preserving the semantics of provider–customer and peer–peer relationships

5.4 Batch Inserts

To avoid performance issues, the script collects rows into a batch and uses:

```
INSERT OR IGNORE INTO as_rel(asn, neighbor, relation, source)
```

- `BATCH_SIZE = 10000`
- Batches are committed periodically
- `INSERT OR IGNORE` ensures duplicates are silently skipped

After the file is fully processed, any remaining rows in the batch are also committed.

5.5 Final Reporting (Verbose)

If `--verbose` is enabled:

- Logs intermediate batch commits with running totals
 - Executes `SELECT COUNT(*) FROM as_rel`
 - Prints final total row count
-

6. Command-Line Interface

The `main()` function wires everything together and provides a simple CLI.

Arguments

`--db`

- SQLite DB filename (default: `asn_data.db`)
- Stored in `project_root/database/<db_name>`

`--file` (*required*)

- Path to the CAIDA `as-rel2.txt` input file

`--source-tag`

- Tag stored in the `source` column
- **Default value:** `caida-serial2`
- If not provided, this default value is used
- The script **does not derive** the tag automatically from the CAIDA file name
- To import multiple CAIDA snapshots side-by-side, the user must explicitly provide distinct `--source-tag` values

`--verbose`

- Enables detailed logging and warnings

--keep-old

- If set, does not clear existing rows in `as_rel`
- New rows are added on top using `INSERT OR IGNORE`

Example Usage

Fresh import (clear previous data):

```
python3 asn_relations_import.py \  
  --db asn_data.db \  
  --file 20251201.as-rel2.txt \  
  --source-tag caida-2025-12-01 \  
  --verbose
```

Appending a new snapshot without clearing old rows:

```
python3 asn_relations_import.py \  
  --db asn_data.db \  
  --file 20251101.as-rel2.txt \  
  --source-tag caida-2025-11-01 \  
  --keep-old \  
  --verbose
```

7. How `asn_filters_strict.py` Uses `as_rel`

Once imported, the `as_rel` table becomes a core input for strict-filtering and vulnerability metrics:

- Valley-free checking

- Distinguishing uphill (provider/peer) vs downhill (customer) segments
- Detecting valley violations correlated with leaks and misconfigurations
- Reasoning about origin and near-origin responsibility
- Adjusting suspicion weights based on AS relationships
- Contextualizing suspicious AS paths while avoiding blame on passive transit nodes

Without high-quality `as_rel` data, `asn_filters_strict.py` would lose a critical structural signal.

8. Why the AS-Relationship Data Source Is Reliable

`asn_relations_import.py` relies on CAIDA's AS-relationship (as-rel2) dataset, one of the most widely used and cited sources for Internet AS-level topology. These relationships are inferred using long-term BGP observations and well-documented heuristics, and are actively maintained by CAIDA.

While no inferred dataset is perfect, CAIDA's AS-relations are the de-facto standard in both academia and industry for modeling the global AS graph, making them a solid foundation for valley-free checks, leak detection, and routing-topology reasoning.

9. Summary

`asn_relations_import.py` is a simple but essential auxiliary script that:

- Reads CAIDA as-rel2 dumps
- Translates them into a directional `as_rel` schema
- Populates a normalized AS-relationship graph in SQLite
- Provides the structural foundation for advanced BGP security metrics like `asn_filters_strict.py`

It is designed to be:

- **robust** (schema checks, input validation)
- **efficient** (batch inserts)
- **repeatable** (multiple snapshots via explicit `source_tag`)
- **safe** (optional `--keep-old` and `INSERT OR IGNORE` semantics)