

Prefix_ml_vuln.py - Machine Learning for Prefix-Level Vulnerability & Origin-Hijack Exposure

1. Purpose & Role in the Platform

`prefix_ml_vuln.py` computes a machine-learning-based vulnerability score **per IP prefix**, estimating how exposed that prefix itself is to BGP origin hijacking, mis-origination, and subprefix abuse.

This component answers a **strictly prefix-centric security question**:

“If this specific IP prefix is targeted, how weak are the validation and policy controls that govern this prefix, independent of who the attacker is?”

Key Design Principle

This ML focuses exclusively on the **control surface attached to the prefix itself**:

- cryptographic authorization (RPKI / ROA),
- prefix-specific policy permissiveness,
- registry anchoring (IRR),
- operational visibility of the prefix in the global routing system.

1.1 Why a Dedicated Prefix ML Exists

In BGP security, **prefix vulnerability** and **attacker capability** are orthogonal dimensions:

- A prefix can be structurally weak even if most ASNs enforce strict filtering.

- A strong attacker ASN cannot succeed if the prefix is cryptographically locked.
- Conversely, a weak or permissive prefix amplifies even mediocre attacker capability.

`prefix_ml_vuln.py` isolates **prefix-level origin-policy weakness** into a single, explainable signal that can later be safely combined with attacker-side ML.

1.2 Output Semantics

The model produces a continuous **risk score** in the range **[0,1]** for each observed prefix, **augmented by an explicit evidence-quality layer**.

Risk Score

The numerical risk score represents the model-estimated probability that a prefix belongs to the **vulnerable / hijack-prone** class.

Score	Interpretation
0.0	Strong prefix protection (strict ROA constraints, valid VRPs, IRR anchoring, high routing visibility)
1.0	High prefix vulnerability (no ROA or permissive ROA, missing IRR anchoring, low or stealth visibility)

Evidence Quality Indicators

In addition to the risk score, the model exposes the **quality of the underlying evidence** used to compute that score:

- **data_completeness** – a normalized value in **[0,1]** representing the fraction of expected security and visibility signals available for the prefix.

- **score_confidence** – a qualitative confidence indicator derived from data completeness:
 - **HIGH** – most expected signals are present,
 - **MEDIUM** – partial signal coverage,
 - **LOW** – limited observable evidence.

These indicators make uncertainty **explicit**, allowing consumers to distinguish between **high-risk prefixes supported by strong evidence** and **scores derived from sparse observational data**.

Downstream Usage

The output is:

- **persisted in the database** together with completeness and confidence metadata,
- **consumed by the ASN–Prefix attack-surface combinator**,
- **usable directly in dashboards and customer-facing reports**, where risk magnitude and evidence quality can be evaluated independently.

2. Data Sources & Feature Space

2.1 Prefix Feature Store (**prefix_data**)

The ML relies on the **prefix_data** table, populated upstream by deterministic collectors. This table represents **policy, authorization, and observability properties of each prefix**.

Column	Meaning
prefix	IP prefix under analysis

<code>asn</code>	ASN currently announcing the prefix
<code>has_roa</code>	Binary flag: does any ROA exist for this prefix
<code>vrp_count</code>	Number of valid VRPs covering the prefix
<code>roa_coverage_scope</code>	How well ROAs cover the prefix address space
<code>roa_maxLength</code>	ROA <code>maxLength</code> constraint
<code>irr_objects_count</code>	Number of exact IRR route objects
<code>prefix_length</code>	Prefix length (/24, /48, etc.)
<code>vantage_points_seen</code>	Number of BGP vantage points observing the prefix

2.2 Final Feature Vector Used by the ML

The exact and complete feature list used for training and inference is:

```
feature_cols = [
    "has_roa",
    "vrp_count",
```

```
"roa_coverage_scope",  
"roa_maxLength",  
"irr_objects_count",  
"prefix_length",  
"vantage_points_seen",  
]
```

3. Preprocessing, Normalization & Missing-Data Handling

Internet routing data is inherently incomplete and unevenly observable across prefixes.

Preprocessing is therefore designed to be **conservative, explicit, and auditable**, avoiding both overestimation of security **and** implicit assumptions about risk.

3.1 Numeric Coercion

All input features are coerced into numeric form using:

```
pd.to_numeric(..., errors="coerce")
```

Any invalid, malformed, or non-parsable values are converted to **NaN**, ensuring a clean and consistent numerical input space for downstream processing.

3.2 Infinite-Value Protection

To guard against corrupted collector output or upstream anomalies, infinite values are explicitly removed:

```
df = df.replace([np.inf, -np.inf], np.nan)
```

This prevents numerical instability, undefined ML behavior, or unintended dominance of invalid observations.

3.3 Explicit Missing-Data Modeling (Conservative but Non-Punitive)

Missing data is handled using a **two-layer strategy** that separates **risk estimation** from **evidence quality**.

3.3.1 Availability Tracking (Pre-Imputation)

Before any imputation occurs, the pipeline records **per-feature availability indicators** (`has_*`) for all security and visibility signals.

These indicators are used to compute:

- **data_completeness** – the fraction of expected signals present for a given prefix.
- **score_confidence** – a qualitative assessment (**HIGH / MEDIUM / LOW**) derived from data completeness.

This ensures that incomplete observability is **explicitly exposed**, rather than implicitly encoded into the risk score.

3.3.2 Conservative Imputation Strategy (Exact Code-Aligned)

After availability tracking, missing numeric values are filled to allow model training and inference to proceed, using a conservative and explicitly defined strategy.

For most security and visibility signals, missing values are filled with `0.0`:

- `has_roa` → 0.0
- `vrp_count` → 0.0
- `roa_coverage_scope` → 0.0
- `irr_objects_count` → 0.0
- `vantage_points_seen` → 0.0

For structural fields that describe prefix geometry, median-based imputation is used:

- `roa_maxLength` → median of observed values
- `prefix_length` → median of observed values

If the median of a structural field cannot be computed (e.g., all values are missing or invalid), the implementation explicitly falls back to `0.0`.

This guarantees fully numeric feature vectors under all data conditions while preserving conservative behavior.

Importantly, missing values are **not interpreted as weak security by default**.

Their impact on the final risk score is mediated through explicit availability indicators and the evidence-quality layer (`data_completeness`, `score_confidence`), ensuring that incomplete observability increases uncertainty rather than forcing a high-risk classification.

3.4 Alignment with ASN-Level Modeling

This approach mirrors the strategy used in `asn_ml_risk.py`:

- Risk scores are always produced,
- Missingness is modeled explicitly through availability indicators,
- Evidence quality is surfaced separately from the predicted risk.

This ensures **conceptual symmetry** between prefix-level and ASN-level ML pipelines and enables consistent interpretation across layers of the platform.

4. Automatic Label Generation (Weak Supervision)

There is no authoritative, global dataset of hijacked prefixes.

The model therefore uses **weak supervision**, labeling only **extreme, unambiguous cases**.

4.1 Adaptive Visibility Thresholds

Instead of fixed values, thresholds are derived dynamically:

- low visibility → 30th percentile,
- high visibility → 70th percentile.

This allows the model to:

- adapt as BGP observability changes,
- remain stable over time,
- avoid hard-coded assumptions.

4.2 GOOD Prefixes (Label = 0)

A prefix is labeled **GOOD** if all of the following hold:

- `has_roa >= 1`
- `vrp_count >= 1`
- `roa_maxLength <= prefix_length` (strict ROA)
- `irr_objects_count >= 1`
- `vantage_points_seen >= high_visibility_threshold`

These prefixes exhibit:

- strict cryptographic authorization,
- no subprefix hijack allowance,
- registry anchoring,
- strong global visibility.

They form **high-confidence low-risk ground truth**.

4.3 BAD Prefixes (Label = 1)

Two clearly vulnerable categories are labeled **BAD**.

BAD — Type A: No Controls

- `has_roa == 0`
- `irr_objects_count == 0`
- `vantage_points_seen <= low_visibility_threshold`

Represents:

- no cryptographic protection,
 - no registry anchoring,
 - stealth hijack surface.
-

BAD — Type B: Permissive ROA

- `has_roa >= 1`
- `roa_maxLength >= prefix_length + 2`
- `vantage_points_seen <= low_visibility_threshold`

Represents:

- explicit subprefix hijack allowance,
- weak origin enforcement,

- stealth hijack amplification.
-

4.4 Training Set Construction

Only high-confidence GOOD and BAD prefixes are retained:

```
df_labeled = df[good | bad]
```

All intermediate cases are excluded.

Consequences:

- minimal label noise,
 - no speculative assumptions,
 - strong statistical defensibility.
-

4.5 Deep Dive: Labeling Algorithm & Learning Logic

4.5.1 Why Labeling Is the Core of This ML

There is no clean historical ground truth for prefix hijacks.

The model is designed to identify **structural vulnerability**, not replay incidents.

The ML answers:

“Given only prefix-level policy and visibility, does this prefix structurally allow or resist origin hijack?”

4.5.2 What the Model Actually Learns

LightGBM learns how combinations of prefix-level controls correlate with exposure, including non-linear interactions such as:

- permissive ROA × low visibility,
- IRR absence × missing ROA,
- prefix length × ROA coverage scope.

The learned mapping is:

(prefix policy + visibility) → structural vulnerability probability

4.5.3 GOOD vs BAD Labels vs Final Output

Important distinction:

- GOOD (0) and BAD (1) labels exist **only during training**.
 - They are **never stored** in the database.
 - The final output is a **continuous risk score in [0,1]**.
-

4.6 Why Feature Importance Is Learned, Not Hardcoded

Hardcoding rules like

“ROA > IRR > visibility” would:

- freeze assumptions,
- fail under non-linear interactions,
- break as routing practices evolve.

LightGBM learns importance dynamically, allowing:

- ROA to dominate when it truly blocks hijacks,
- IRR to matter when ROA is absent,

- visibility to amplify or dampen both.

This is **more accurate**, **more stable**, and **more defensible**.

5. Model Training (LightGBM)

5.1 Why LightGBM Was Chosen

LightGBM is:

- state-of-the-art for tabular security data,
- robust to heterogeneous feature scales,
- capable of modeling non-linear interactions,
- explainable and auditable.

This makes it ideal for BGP metadata.

5.2 Train / Validation Split

```
train_test_split(  
    test_size=0.25,  
    stratify=y  
)
```

- 75% training,
- 25% validation,
- class balance preserved.

5.3 Model Configuration

```
params = {  
    "objective": "binary",  
    "metric": ["auc", "binary_logloss"],  
    "boosting_type": "gbdt",  
    "learning_rate": 0.05,  
    "num_leaves": 31,  
    "feature_fraction": 0.9,  
    "bagging_fraction": 0.9,  
    "bagging_freq": 5,  
}
```

Training:

- `num_boost_round = 500`
- `early_stopping_rounds = 50`

This prevents overfitting and ensures stability.

6. Risk Scoring for All Prefixes

After training, the model is applied to **all observed prefixes**:

```
proba_bad = model.predict(X_all)
```

Output Columns

Column	Meaning
risk_score	ML-estimated probability that the prefix belongs to the BAD (vulnerable) class
data_completeness	Fraction of expected security and visibility signals available for the prefix
score_confidence	Qualitative confidence indicator (HIGH / MEDIUM / LOW) derived from data completeness

Interpretation

- **risk_score** represents the estimated prefix-level vulnerability to origin hijack and mis-origination.
- **data_completeness** quantifies how much observational evidence supports the computed score.
- **score_confidence** makes the reliability of the score explicit, allowing consumers to distinguish between:
 - high-risk prefixes supported by strong evidence, and
 - scores produced under partial or sparse observability.

Risk magnitude and evidence quality are intentionally **decoupled**, enabling conservative yet transparent risk assessment.

7. Database Persistence

7.1 Output Table

The model persists its output in the following table:

```
prefix_ml_vuln (  
    prefix          TEXT NOT NULL,  
    origin_asn      INTEGER NOT NULL,  
    risk_score      REAL NOT NULL,  
    data_completeness REAL NOT NULL,  
    score_confidence TEXT NOT NULL,  
    label_note      TEXT,  
    updated_at      TEXT NOT NULL,  
    PRIMARY KEY (prefix, origin_asn)  
)
```

Each row represents the latest ML-derived assessment for a given `(prefix, origin_asn)` pair.

Important Note: Handling of Missing or Incomplete Data

Prefixes with missing security or visibility data are **not treated as vulnerable by default**, nor are they assumed to be safe.

Instead, the system follows a **defensive but explicit strategy**:

- Missing data **does not suppress scoring**.
- Missing data **does not directly inflate the risk score**.
- Missing data is **explicitly exposed** via `data_completeness` and `score_confidence`.

From an Internet routing security perspective, incomplete observability limits confidence, not necessarily security posture.

This approach avoids both:

- **false negatives** (treating unknown prefixes as safe), and
- **false positives** (treating missing data as definitive evidence of vulnerability).

As additional security controls or observability signals become available (e.g., ROAs, IRR objects, increased visibility), the model automatically incorporates them, updates `data_completeness`, and adjusts the risk score accordingly.

7.2 Semantics

- `risk_score` → ML-estimated prefix vulnerability score
- `data_completeness` → proportion of expected signals present
- `score_confidence` → qualitative reliability of the score
- `label_note` → explanation of the scoring regime (currently set to `"weak_supervision_same_pot"`)
- **UPSERT semantics** → safe for repeated runs and incremental updates

8. Model Stability, Retraining & Operational Safety

This ML is designed for long-term operational use:

- percentile-based thresholds adapt automatically,
- conservative imputations prevent false safety,
- early stopping avoids drift,

- retraining does not change semantics.

Scores remain stable, interpretable, and comparable over time.

9. Operational Usage & CLI Interface

```
python3 prefix_ml_vuln.py --db asn_data.db [--min-labeled N]
[--print-summary]
```

Argument	Description
<code>--db</code>	Path to SQLite database (required)
<code>--min-labeled</code>	Minimum labeled prefixes required (default: 200)
<code>--print-summary</code>	Prints training & scoring summary

If insufficient labeled data exists, the script exits safely without training.

10. Summary

`prefix_ml_vuln.py`:

- compresses prefix-level policy and visibility into a single continuous score,
- uses conservative, defensible ML logic,
- avoids speculative assumptions,

- remains stable across time,
- provides a core building block for prefix-level attack-surface modeling.