

AboutCode

Proposal for Google Summer of Code 2026

Project Name:

VulnerableCode Insights

Live Proposal Link : Google Doc for Proposal

I accept the standard 12-week coding period as default.

Personal Information

Name: Sampurna Pyne

Email: sampurnapyne1710@gmail.com

Gitter username : @samk1710:gitter.im

Github: https://github.com/Samk1710

LinkedIn: https://www.linkedin.com/in/samk1710/

Timezone: Indian Standard Time (GMT +5:30)

Relevant Skills: Python, Django, PostgreSQL, Security, APIs, Scraping, Web

Project Information

Project Title: VulnerableCode Insights (Category B: Usage)

Project Length: Medium (175 hours)

Idea Link: VulnerableCode Insights Project Link

Related Issues: https://github.com/aboutcode-org/vulnerablecode/issues/1276

Project Abstract

VulnerableCode is a free and open-source database of *vulnerable packages*. It provides its users with both a Web UI and a JSON API, letting them know what vulnerabilities a certain 'Package Foo' is affected by, and whether a later version fixes them.

Over the years, it has consumed numerous vulnerability data sources, enriching its coverage and support, yet it does not answer a very fundamental question its users have:

How do I believe VulnerableCode has enough sane data for me to use ?

Vulnerablecode surely can answer this, it has just been too humble to. This project aims to provide visual insights that convince users of VulnerableCode's effectiveness. Through a multi-panel telemetry dashboard built directly into the Web UI, it gives users confidence to trust VulnerableCode and the maintainers' insights to improve it.

Key Deliverables

- A six-panel insights dashboard at `/telemetry/` built with Django templates and Chart.js
- A `TelemetrySnapshotPipeline` that pre-aggregates all dashboard metrics via `DailySnapshot` model every 24 hours
- A `PipelineRunStat` model that makes importer run statistics queryable
- Three REST API endpoints under `/snapshot`, `/epss`, `/pipeline-stats`
- Documentation of the project and its corresponding tutorials

Technical Approach

1.1 Telemetry App

The `telemetry` app is a self-contained Django application that sits alongside `vulnerabilities`. It reads from the existing V2 models.

```
telemetry/
├── __init__.py
├── apps.py
├── models.py
├── migrations/
├── pipelines.py
├── views.py
├── urls.py
└── templates/
    └── telemetry/
```

```
└── dashboard.html
```

1.2 DailySnapshot Model for storing daily pre-computed metrics

Most of the dashboard queries require full-table joins across million-row tables. Running these on every API request would make the dashboard unusable. The solution is to run them **once per day** and store the results with the following model skeleton:

```
class DailySnapshot(models.Model):
    snapshot_date = models.DateField(unique=True,
db_index=True)


    overview = models.JSONField(
        default=dict,
    )
    vulnerability_analytics = models.JSONField(
        default=dict,
    )
    ecosystem_analytics = models.JSONField(
        default=dict,
    )
    importer_analytics = models.JSONField(
        default=dict,
    )
    data_quality = models.JSONField(
```

```
        default=dict,
    )
    class Meta:
        ordering = ["-snapshot_date"]
        get_latest_by = "snapshot_date"
```

`DailySnapshot` is a model with one row per day containing every pre-computed metric the dashboard needs. Each API endpoint reads from `DailySnapshot.objects.latest()` Computing them once and storing them as JSON means zero aggregation work at request time.

1.3 TelemetrySnapshotPipeline to pre-compute ORM query results

`TelemetrySnapshotPipeline` populates `DailySnapshot` every 24 hours at a fixed time. It inherits `VulnerableCodePipeline` and gets a `PipelineSchedule` row at startup, runs every 24 hours via the existing RQ scheduler, and appears in the Pipeline Schedule UI. The dashboard **can monitor its own snapshot pipeline** on the Pipeline Health panel. It is responsible to run the ORM queries and store the result in `DailySnapshot.`

```
class TelemetrySnapshotPipeline(VulnerableCodePipeline):

    pipeline_id =
"telemetry.pipelines.TelemetrySnapshotPipeline"


    @classmethod
    def steps(cls):
        return (
            cls.compute_overview_stats,
            cls.compute_cve_analytics,
            cls.compute_ecosystem_analytics,
            cls.compute_importer_analytics,
            cls.compute_quality_analytics,
            cls.save_snapshot,
        )
```

Each step computes one category of metrics and stores results in instance variables. `save_snapshot` writes the complete row in a single `update_or_create` call wrapped in `@transaction.atomic`. If any step fails, the pipeline exits with a non-zero code and the previous day's snapshot remains intact.

1.4 Making Pipeline Run Statistics Queryable

Currently each V2 Importer run logs out how many advisories were handled, but this information currently exists only as plain text inside `PipelineRun.log`. While useful for debugging, this is **neither structured nor queryable** and hence difficult to aggregate into graphs.
Related file links:

- vulnerabilities/pipelines/__init__.py
- https:/vulnerabilities/pipes/advisory.py

In/pipelines/__init__.py in collect_and_store_advisories()

```
if _obj := insert_advisory_v2(
    advisory=advisory,
    pipeline_id=self.pipeline_id,
    logger=self.log,
    precedence=self.precedence,
)
```

This treats all advisories the same, regardless of whether they were newly inserted into the database or already present. From the definition of `insert_advisory_v2()` , each advisory is resolved using:

```
advisory_obj, created = AdvisoryV2.objects.get_or_create(
            advisory_id=advisory.advisory_id,
            datasource_id=pipeline_id,
            unique_content_id=content_id,
            defaults=default_data,
        )
```

Here when `created` = True means advisory is newly inserted and `created` = False means advisory already exists. However, this information is not exposed to the pipeline layer as `created` is never returned.

Suggested Update:

We return created as well: `return advisory_obj, created` . This allows `advisory.py` to count the newly appended advisories separately:

```
created_count = 0
skipped_count = 0

result = insert_advisory_v2(..., return_created=True)
if result:
    advisory_obj, created = result
    if created:
        created_count += 1
    else:
        skipped_count += 1
```

Previously, all advisories were counted equally. Now, ingestion distinguishes between **data added** and **data already known**.

1.5 PipelineRunStat to persist the improvement

```
class PipelineRunStat(models.Model):
    pipeline_run_id = models.UUIDField(unique=True,
db_index=True)
    pipeline_id = models.CharField(max_length=600,
db_index=True)

    advisories_created = models.IntegerField(default=0)
    advisories_skipped = models.IntegerField(default=0)
    run_start = models.DateTimeField(db_index=True)
    run_end = models.DateTimeField(null=True)
    duration_seconds = models.FloatField(null=True)
    run_exitcode = models.IntegerField(null=True)
```

```
    class Meta:
        ordering = ["-run_start"]
        indexes = [
            models.Index(fields=["pipeline_id", "-run_start"]),
        ]
```

This model stores the advisories created per run and additional fields needed in Pipeline health Panel.

2.0 The REST API endpoints

We require the following endpoints:

- `GET /telemetry/snapshot/` returns the complete `DailySnapshot` in a single response. The dashboard loads this once on page open and each panel reads its corresponding key.
- `GET /telemetry/epss/` for EPSS feeds and `GET /telemetry/pipeline-stats/` for additional pipeline statistics.

Response skeleton of `/snapshot:`

```
{
  "metadata": { ... },

  "overview": { ... },

  "vulnerability_analytics": { ... },

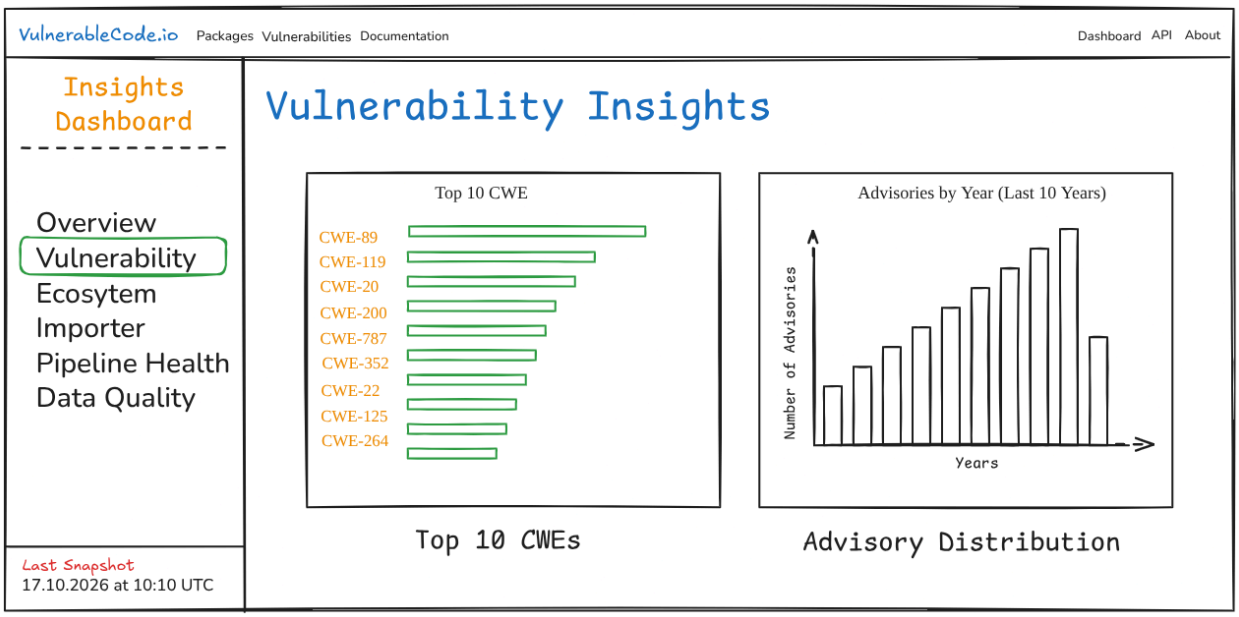
  "ecosystem_analytics": { ... },
```

```
  "importer_analytics": { ... },

  "data_quality": { ... }
}
```

3.0 The Six-Panel Dashboard

The project proposes a **six** panel Dashboard.



Charts will be implemented using Chart.js for lightweight visualization and implementation the following graphs:

3.1 Overview Panel

- **3 KPI cards: Total Advisories, Total Packages, Total data sources**

  Shows the count and delta of key metrics.
- **Sparkline**: **30-day daily advisory ingest trend**

  Shows growth of the database over time.
- **Cross validation donut: Percentage of cross-validated advisories**

Shows overlap of advisories across data sources pointing to data accuracy.

3.2 Vulnerability Analytics Panel

- **Horizontal bar: Top 10 CWEs by advisory count**

  Shows the weightage of weakness categories like SQL injection, XSS, CSRF etc.
- **Bar chart: Advisory counts by year of publication for the last 10 years**

  Shows vulnerability data coverage from over the years.
- **Heatmap table and Multi-line Chart: Advisory count by CWE type and year**

  Draws inspiration from: https://www.cvedetails.com/vulnerabilities-by-types.php
- **Scatter Plot: CVSS Score Distribution**

  Draws inspiration from: https://www.cvedetails.com/cvss-score-charts.php
- **Line Graph and Table: EPSS scores and their deltas over time**

  Similar: https://www.cvedetails.com/epss/CVE-2021-25118/epss-score-history.html and will aim to resolve issue#2231.

  We can also use : https://www.first.org/epss/api

3.3 Ecosystem Analytics Panel

- **Type Donut: Package count by ecosystem type from `PackageV2.type`**

  Shows *total* ecosystem coverage.
- **Namespace donut with ecosystem dropdown: For a selected ecosystem type (like pypi, maven), shows the top 10 namespaces by package count from `PackageV2.namespace`**

  Show *per* ecosystem coverage.
- **Grouped bar with ecosystem dropdown: Affected packages vs packages with a known fix per ecosystem**

  Shows for affected vs fixed packages per ecosystem.
- **Severity Donut with ecosystem dropdown: For a selected ecosystem type shows the severity distribution**

  A scatter plot for the same can also be used.

3.4 Importer Analytics Panel

- **Horizontal bar: Advisory count per datasource importer**

  Shows which sources are primary contributors.

- **Line chart with datasource dropdown: 90-day daily ingest volume for all importers or a selected datasource**

  Shows importer consistency. The dropdown lets us isolate a specific importer's behavior.

- **Stacked bar with importer dropdown: For all importers or a selected one, shows exclusive vs cross-validated advisory counts**

  Visual aggregate for cross validated packages like Vulntotal.

3.5 Pipeline Health Panel

VulnerableCode already ships Pipeline Schedule in the Web UI at https://public.vulnerablecode.io/pipelines/schedule/

Right now it lists Pipeline ID with its corresponding status, interval, last run and next run time stamps.

We can enhance this with addition of the following columns or tooltips:

- Average run duration
- New advisories appended
- Total advisories
- Streak tracker indicating success/failure of last 5 runs

3.6 Data Quality Analytics Panel

- **Horizontal bar: Open issue count by type from `AdvisoryToDoV2.issue_type`**

  Shows the weighted scale of each quality problem category.

- **Donut chart with issue_type type dropdown: For a selected `issue type`, shows the contribution of each datasource**

  Shows how much a particular importer contributes to a given data quality issue.

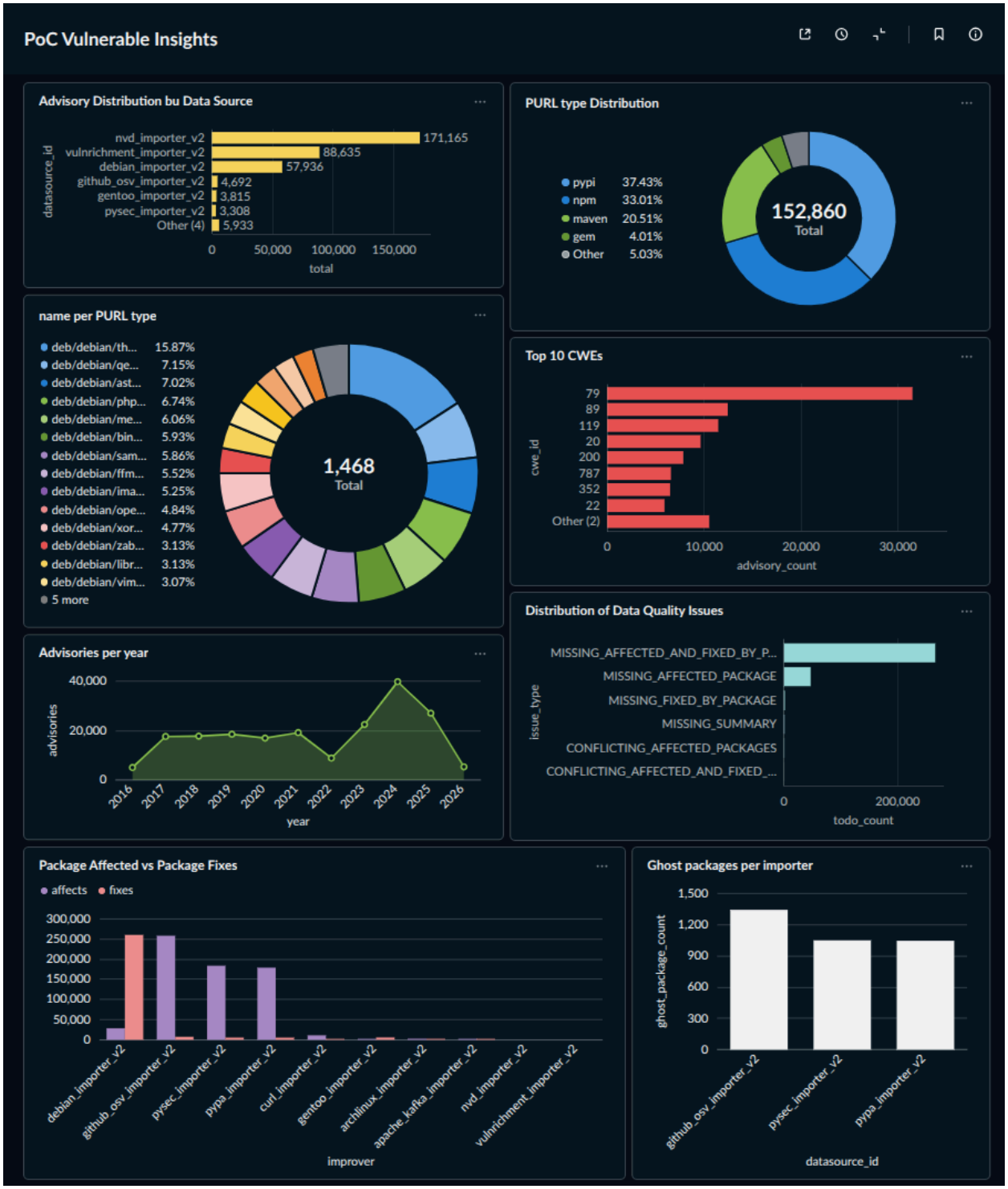
- **Stacked bar with importer dropdown: Ghost package count per importer**

  Shows which importer ingests packages that do not exist upstream, directly pointing to data source quality.

- **Line chart with importer dropdown: Open vs resolved issues over time, for all importers or a selected datasource**

  Shows issue resolution rate.

4.0 Proof of Concept with Metabase



Made with Metabase and querying data from 10 importers.

Project Timeline

Community Bonding Period: May 1 - May 24

Discuss and take suggestions from mentors to enhance architecture and implementation. Explore and research more data metrics.

Week 1 & 2: May 25 - June 7

- Setup the `telemetry` app and project structure.
- Design and implement `DailySnapshot` and `PipelineRunStat` models with migrations.
- Devise a model to store EPSS history
- Add initial unit tests for the models.

Week 3 & 4: June 8 - June 21

- Implement enhancement in `insert_advisory_v2()`.
- Update `collect_and_store_advisories()` to track created and skipped advisories.
- Update existing unit tests.

Week 5 & 6: June 22 - July 5

- Implement the `SnapshotPipeline` along with all its steps.
- Implement each step with ORM queries.
- Add units for the pipeline.

Week 7 & 8: July 6 - July 19

- Design and implement the API views.
- Test out the APIs in Postman/CURL.
- Add unit tests for the API.

Week 9 & 10: July 20 - August 2

Implement panels:

- Panel 1: Overview
- Panel 2: Vulnerability Analytics
- Panel 3: Ecosystem Analytics

Week 11 & 12: August 3 - August 16

Implement remaining panels:

- Panel 4: Importer Analytics
- Panel 5: Pipeline Health (real-time updates)
- Panel 6: Data Quality Analytics

Performance optimization tweaks where necessary and UI test the entire dashboard.

Project Submission: August 17 - August 24

Buffer time to complete pending/unprecedented work. Write documentation. Wrap up the project, address review feedback and submit the code for Final Evaluation.

Important Deadlines

- **Midterm Evaluation: July 10**
- **Final Evaluation: August 24**

Post GSOC

Keep contributing to VulnerableCode and other AboutCode projects and help new contributors onboard to our community.

Expected Outcomes

Once implemented, the dashboard *at a glance* will be able to answer **if VulnerableCode is healthy, growing and worth trusting with visual data as evidence.** It will also **visually summarise** various metrics of the database providing a good user experience.

Previous Contributions

I have contributed mainly to four repositories namely vulnerablecode, aboutcode-mirror-euvd, univers and vulnerablecode-ai-experiments, taking review feedback and suggestions from community calls to improve code implementations.

VulnerableCode

- vulnerablecode#2104: Add Importer Pipeline for Tuxcare Advisories, consolidate them into Impact Packages mapped across different flavors of Linux Distributions
- vulnerablecode#2154: Add Importer Pipeline for Openstack Advisories
- vulnerablecode#2161: Fix intermittent error in Github Actions due to linkcheck fails in nixos.wiki
- vulnerablecode#2198: Add Importer Pipeline for Vmware Photon Advisories and consolidate them into Impact Packages
- vulnerablecode#2046: Add Importer Pipeline for EUVD

Aboutcode mirror for EUVD

- aboutcode-mirror-euvd#1: Add Mirror Pipeline for collect and store advisories from EUVD via daily sync
- aboutcode-mirror-euvd#3: Document the readme.md for EUVD Mirror

Univers

- univers#185: Add alpine in RANGE_CLASS_BY_SCHEME
- univers#184: Add support Openstack Advisories

Vulnerablecode-ai-experiments

- experiments#15: Fix missing cwe2 dependency
- experiments#18: Identify and fix env load failures due to improper typecasts

Additional Contributions

- issues#1199: Researched and documented the VulDB API for Vulntotal
- issues#1338: Currently researching and writing an importer to collect Bitnami Advisories

Related Experience

- SDE Intern@Rudrax Remote|London, United Kingdom | Apr - Dec 25

  Worked as a forward-deployed SDE across backend and frontend using Python and TypeScript stacks.

- Security Intern@India Internet Foundation Hybrid| Kolkata, India | Jan - Mar 25

  Worked on container orchestration and automation scripts for DNS in a Box.

- Achieved podium finishes in 12 hackathons, including the national IEEE AIORI Hackathon, which led to a government aided fully sponsored opportunity to participate in the IETF 122 Bangkok (March 2025) where I collaborated with the AIORI IMN team to implement RFC 8250 and evaluate DNS server performance using the IMN distributed measurement platform and an IPv6 DNS testbed.

  Related Link: Official IETF 122 Presentation

- Security Lead@Google Developers Club Oct 24 - Oct 25

  Conducted hands-on cybersecurity workshops, seminars and competitions for students in my college campus.

Academic Studies

University: Heritage Institute of Technology, Kolkata (2023 - 2027)

Degree: B.Tech in Computer Science Engineering

Any other commitments during GSoC?

I can contribute full-time to this project, I don't have any commitments during the coding period. My end-semester examination is supposed to end before the start of the coding period (in May itself).

I will be able to attend the weekly status calls on both Mondays and Tuesdays to discuss progress and feedback.

GSoC participation

This is my first time applying for Google Summer of Code. I did not submit a proposal to any other organization.