Document version: v1.0 (Proof of Concept)
Date: December 2025
Status: Informational

Public Smart Contract Vulnerability Database (SCVD)

Technical Whitepaper

Abstract

The Public Smart Contract Vulnerability Database (SCVD) is a proof-of-concept system that demonstrates how fragmented smart contract security findings can be transformed into a standardized, machine-readable, and openly accessible dataset. Today, vulnerability intelligence is dispersed across audit PDFs, Markdown reports, GitHub issues, and bespoke JSON formats, limiting the ecosystem's ability to identify recurring weaknesses early in the design phase or build interoperable security tooling.

SCVD addresses this gap by providing an end-to-end, read-only data pipeline that ingests heterogeneous reports, normalizes individual findings into a unified schema, validates structural integrity, and applies semantic deduplication across sources. SCVD uses smart-contract native taxonomies (Smart Contract Weakness Classification (SWC) as the primary tag, with optional Common Weakness Enumeration (CWE) mappings where applicable) and borrows architectural ideas from traditional vulnerability data systems, most notably the CVE/National Vulnerability Database (NVD) pattern of canonical records, enrichment, and machine-readable distribution.

This whitepaper provides a detailed technical overview of the SCVD Proof of Concept. It describes the system architecture, the SCVD v0.1 schema, the semantic deduplication methodology, and the public API that exposes the resulting dataset. The goal is not to establish a new authority, but to demonstrate a neutral, reproducible foundation for shared security knowledge in the Web3 ecosystem.

1) Introduction: Standardizing Smart Contract Vulnerability Intelligence

The smart contract security ecosystem produces a large and growing body of vulnerability intelligence, primarily through audit reports and competitive review processes. However, this information is typically published in heterogeneous formats ranging from unstructured PDFs to semi-structured Markdown and repository issues making it difficult to aggregate, query, and analyze programmatically at scale.

As a result, identical or closely related vulnerabilities are repeatedly rediscovered, severity definitions vary across organizations, and historical data remains largely inaccessible to

Truscova GmbH, Germany
hello@truscova.com | www.truscova.com | Document: SCVD-whitepaper

automated analysis. This fragmentation stands in contrast to traditional software security, where decades of standardization through CVE, CWE, and the NVD have enabled shared tooling, metrics, and research.

To explore whether similar benefits can be achieved for smart contracts without forcing ill-fitting abstractions, Truscova developed the Public Smart Contract Vulnerability Database (SCVD) Proof of Concept. SCVD is an open, read-only system designed to ingest heterogeneous security reports and normalize individual findings into a unified, queryable schema while preserving provenance and source integrity.

2) Design Principles

The SCVD Proof of Concept is guided by a small set of explicit design principles intended to ensure long-term usefulness, neutrality, and ecosystem compatibility.

2.1) Smart-contract Aware

SCVD does not analyze on-chain behavior or smart contract execution. Instead, it models reported vulnerabilities using the contextual identifiers commonly present in smart contract security disclosures, such as blockchain networks, contract addresses, repository references, and code locations when available. Where such context is absent, SCVD preserves the finding as reported without attempting to infer on-chain properties. This approach avoids imposing assumptions about deployment state or exploitability while remaining compatible with the way smart contract vulnerabilities are typically described in practice.

2.2) Classification-first

SCVD treats the source report as the system of record for key labels such as severity and type/category when they are explicitly provided. During extraction and normalization, these labels are captured verbatim and stored with provenance. When a report does not provide a type/category (or uses inconsistent terminology), SCVD may apply conservative inference to populate a normalized classification field, while preserving the original text and clearly marking inferred values. The schema is designed to support structured taxonomies such as SWC (and optional CWE mappings where applicable) as an enrichment layer. In the Proof of Concept, taxonomy fields may be left empty or populated only when the mapping is reliable. Please note, SCVD does not invent taxonomy tags without evidence.

2.3) Provenance-preserving

Every SCVD record maintains a clear link to its original source document, including extraction timestamps and tool versions. The pipeline never modifies or overwrites source material.

2.4) Read-only

SCVD does not validate findings, assign blame, or determine exploitability. It records what has been reported, by whom, and where, allowing downstream consumers to apply their own judgment.

2.5) Open by default

All code is released under an open-source MIT license, and curated data snapshots are published under CC0 to maximize reuse and independent verification.

3) Relationship to CVE and NVD

SCVD is not a replacement for the CVE program or the NVD, nor does it attempt to assign CVE identifiers. Instead, it draws inspiration from architectural patterns that have proven effective in traditional software vulnerability data systems. In particular, SCVD adopts the concepts of a minimal canonical record, a clear separation between raw disclosures and normalized data, and machine-readable formats designed for automation and reuse. Where source reports provide contextual information such as blockchain networks, contract addresses, or repository references, SCVD preserves this information as part of the normalized record without attempting to infer deployment state or authoritative identifiers. In this sense, SCVD should be viewed as complementary infrastructure, applying established vulnerability data engineering principles while remaining faithful to the structure and limitations of the underlying disclosures.

4) System Architecture Overview

A robust, auditable data pipeline is the cornerstone of the SCVD project. To ensure data quality, consistency, and traceability, SCVD is implemented as a multi-stage, read-only processing architecture. Each stage has a distinct responsibility, and intermediate artifacts are preserved to allow independent verification of every transformation step.

4.1) High-Level Data Flow

The end-to-end pipeline as shown in Figure 1 is designed so that original source documents are never modified. Each stage consumes the output of the previous one and produces artifacts that can be inspected, validated, and reproduced.

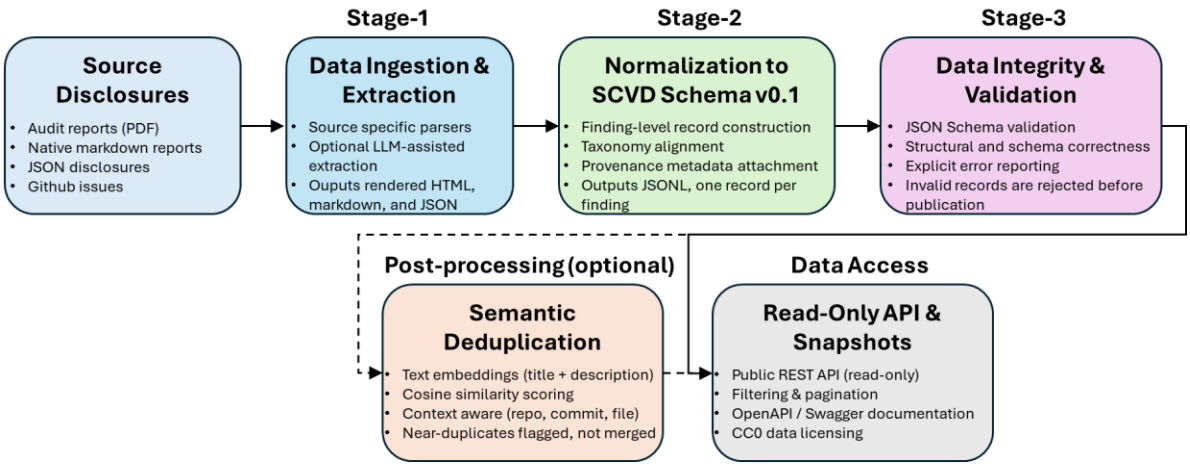

Figure 1: SCVD Architecture Overview

The SCVD pipeline consists of three core processing stages (data ingestion and extraction, normalization, and validation), an optional post-processing stage for semantic deduplication, and a read-only data access layer. The data ingestion stage ingests raw source documents in various formats and converts them into a standardized, intermediate JSON representation for each report. The normalization stage processes the intermediate JSON and transforms each identified vulnerability into a canonical record conforming to the SCVD v0.1 schema. The validation stage acts as a linter, verifying that the normalized records strictly adhere to the formal JSON Schema definition, ensuring data integrity for all downstream purposes. The optional semantic deduplication stage computes semantic embeddings to identify near-duplicate findings across the entire dataset, enriching records with canonical linkage. In the following sections, each of the aforementioned stages are discussed in detail.

4.2) Stage 1: Data Ingestion and Extraction

The first challenge in building SCVD is converting heterogeneous source formats into a consistent, structured intermediate representation. The pipeline currently supports audit PDFs, native Markdown reports, and findings sourced from Code4rena repositories via GitHub Issues. The extraction stage processes each source document and produces several artifacts, including rendered HTML, normalized Markdown, and a structured *report.json* file. This JSON object serves as the standardized input for the normalization stage, decoupling source-specific parsing logic from schema mapping.

PDF processing uses the Marker Python package to handle complex layouts such as multi-column text, tables, code extraction, and headings. Where heuristic methods are insufficient, the pipeline can optionally use a locally hosted Large Language Model (LLM) via

Ollama to infer report-specific metadata schemas and assist with finding segmentation. All such assistance is non-destructive and recorded in provenance metadata.

4.3) Stage 2: Normalization to the SCVD Schema v0.1

Normalization is the core of the SCVD system. During this stage, structured information extracted from source reports is mapped into a canonical SCVD record representing a single vulnerability finding. The normalization process outputs data in JSON Lines (JSONL) format, with one record per finding. Each record conforms to the SCVD v0.1 schema, which defines a predictable structure for downstream consumers while remaining extensible for future enrichment.

SCVD Schema v0.1 (Selected Fields)

Field Name	*Description*
schema_version	The version of the SCVD schema this record conforms to (e.g., "0.1").
scvd_id	A unique, stable identifier for the SCVD record.
doc_id	Identifier of the source document
finding_index	Index of the finding within the source
title	Original finding title
description_md	Normalized description in Markdown
severity	Severity as assigned by the source report
target	Structured information about affected code, e.g., file path
repo	Repository context and commit hash, if available
taxonomy	Structured classification (e.g., SWC, CWE)
references	A list of URLs or other references cited in the original finding.
provenance	Extraction and normalization metadata

Certain fields, such as taxonomy mappings and fix status, are intentionally defined as placeholders in the Proof of Concept and are populated only when reliable information is available, i.e., if the fix status is mentioned in the report.

4.4) Stage 3: Data Integrity and Validation

To ensure structural integrity, all normalized records are validated against a formal JSON Schema definition. This validation step acts as a quality gate, ensuring that every record adheres to the published data contract before it is exposed through the API or included in data snapshots. Validation is performed by a dedicated linting utility designed for automation. Records that fail validation are rejected with explicit, actionable error messages, preventing malformed data from propagating downstream.

4.5) Post-Processing: Semantic Deduplication

A key challenge in aggregating vulnerability data from multiple sources is the presence of duplicate or near-duplicate findings describing the same underlying issue. SCVD addresses this through a semantic deduplication process that combines natural language similarity with contextual signals. The process computes text embeddings for finding titles and descriptions using a configurable transformer model. More concretely, this is performed using a specified Hugging Face model, which defaults to *Snowflake/snowflake-arctic-embed-l-v2.0* for its robust performance. Pairwise similarity is calculated using cosine similarity, and the resulting score is augmented with contextual boosts when repository URLs, commit hashes, or file paths match.

$$final\_score = text\_similarity + hard\_boost \times (repo\_match + commit\_match + path\_match)$$

Findings whose final score exceeds a configurable threshold are flagged as near-duplicates. The output of this process enriches the SCVD records with three additional fields to make the deduplication results transparent and useful:

Field name	*Description*
duplicate_of	If a record is a duplicate, this field contains the scvd_id of the canonical record. It is null for canonical or unique findings.
dedup	A metadata object containing details about the deduplication decision, including the model used, thresholds, and when the analysis was run.
duplicate	A list containing the top-K nearest neighbors for the record, including their similarity scores and the specific contextual signals that contributed to the match.

Please note, records are not automatically merged, instead, deduplication metadata is added to preserve differing interpretations while enabling accurate aggregate analysis.

5) Data Access: Read-Only API and Snapshots

The primary goal of the SCVD is to make normalized smart contract vulnerability data programmatically accessible to developers, security researchers, and automated tooling. To this end, we provide a lightweight, read-only REST API built with FastAPI. By providing programmatic access via a standardized API, we lower the barrier for community members to build and share security tooling.

The API is publicly deployed on Google Cloud Run and is available at the base URL: *https://api.scvd.dev*.

In addition to the live API, SCVD publishes periodic data snapshots in JSONL and CSV formats to support reproducibility, offline analysis, and archival use. These snapshots are released under a CC0 license to encourage unrestricted reuse and independent verification.

6) Non-Goals and Scope Boundaries

The SCVD Proof of Concept is intentionally limited in scope. It does not attempt to validate vulnerability claims, determine exploitability, assign responsibility, or replace responsible disclosure processes. SCVD records what has been reported, not what is necessarily true. By maintaining a clear separation between data aggregation and security judgment, SCVD avoids becoming a centralized authority and remains compatible with diverse perspectives and use cases.

Document version: v1.0 (Proof of Concept)
Date: December 2025
Status: Informational

7) Future Work

The Proof of Concept establishes a foundation rather than a finished system. As outlined in the project proposal, future work will focus on refining the SCVD schema, stabilizing the normalized and canonical data representation, and operating a robust read-only API based on the refined schema. Building on this foundation, subsequent phases envision the introduction of a secure partner intake API, improved search and discovery capabilities, and a moderated community submission process. Together, these phases aim to evolve SCVD from a small proof of concept into a community-driven vulnerability data resource, while preserving the neutrality and provenance guarantees established in the initial implementation.

8) Conclusion

The Smart Contract Vulnerability Database Proof of Concept demonstrates that a unified, open, and reproducible vulnerability knowledge base for smart contract security disclosures is both feasible and valuable. By combining careful schema design, rigorous data engineering practices, and lessons learned from traditional vulnerability data systems, SCVD shows how fragmented audit reports can be transformed into a standardized, machine-readable dataset without compromising provenance or neutrality. This Proof of Concept is not an endpoint, but a starting point. It establishes a technical and organizational foundation upon which more refined schemas, improved access mechanisms, and community-driven processes can be built. By providing an open and reproducible source of normalized vulnerability data, SCVD lowers the barrier for collaboration, research, and tooling development, contributing to greater transparency and resilience across the smart contract security ecosystem.

Truscova GmbH, Germany
hello@truscova.com | www.truscova.com | Document: SCVD-whitepaper