Mind the Data: A Measurement Study on Public Multi-Source Vulnerability Data

YUXIN ZHANG, Tianjin University, China
SEN CHEN*, Nankai University, China
CHENGWEI LIU, Nankai University, China
LINGLING FAN, Nankai University, China
ZHELI LIU, Nankai University, China
YANG LIU, Nanyang Technological University, Singapore
XIAOHONG LI, Tianjin University, China

Public vulnerability databases serve as critical infrastructure for vulnerability analysis; however, quality issues, such as incomplete entries and conflicting or erroneous fields, remain a challenge. Researchers and maintainers face significant difficulties in selecting databases and integrating data from multiple sources. The lack of systematic measurement studies leads to source selection relying on convention rather than comparative evaluation, while cross-database integration is hindered by the inconsistent nature of vulnerability information across sources. This gap introduces risks to downstream applications and limits quality improvement efforts.

To this end, we conduct a measurement study on 16 vulnerability databases, examining variations in completeness, consistency, and accuracy. To enable cross-database comparison, we define a unified field schema and employ LLM-based extraction to align heterogeneous representations. Our analysis reveals three integration challenges. First, vulnerability coverage and field richness are uneven: general databases cover around 70% of vulnerabilities but provide limited analytical fields, while ecosystem databases contribute 29% non-CVE records and richer details that remain largely missing across sources. Second, cross-database consistency is low, with over 70% of database pairs showing weak agreement and substantial divergence in weakness, severity, and affected component annotations, particularly due to heterogeneous classification and version reporting practices, while descriptions vary in contextual richness. Third, accuracy varies substantially by field: while Weaknesses and Severity exceed 80%, Affected remains lowest at 62.06%, with wide variation across databases. And different fields benefit from different integration strategies, indicating that effective integration requires field-aware design rather than uniform aggregation. These findings provide a foundation for researchers and maintainers in selecting and integrating vulnerability data and offer practical guidance for improving database quality.

CCS Concepts: • **Security and privacy**;

Additional Key Words and Phrases: Multi-Source Vulnerability Data, Cross-Database Analysis

*Corresponding author

Authors' Contact Information: Yuxin Zhang, Tianjin University, Tianjin, China, yuxinzhang@tju.edu.cn; Sen Chen, Nankai University, Tianjin, China, senchen@nankai.edu.cn; Chengwei Liu, Nankai University, Tianjin, China, chengwei.liu@ntu.edu.sg; Lingling Fan, Nankai University, Tianjin, China, linglingfan@nankai.edu.cn; Zheli Liu, Nankai University, Tianjin, China, liuzheli@nankai.edu.cn; Yang Liu, Nanyang Technological University, Singapore, Singapore, yangliu@ntu.edu.sg; Xiaohong Li, Tianjin University, Tianjin, China, xiaohongli@tju.edu.cn.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
Manuscript submitted to ACM

ACM Reference Format:
Yuxin Zhang, Sen Chen, Chengwei Liu, Lingling Fan, Zheli Liu, Yang Liu, and Xiaohong Li. 2018. Mind the Data: A Measurement Study on Public Multi-Source Vulnerability Data. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 44 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Vulnerability databases serve as the foundation for security analysis, risk assessment, and patch management [33]. They provide structured records of security vulnerabilities, enabling researchers and practitioners to track, assess, and mitigate security risks. While the Common Vulnerabilities and Exposures (CVE) [10] system offers a standardized identifier for vulnerabilities, multiple databases, such as NVD [46], OSV [51], IBM X-Force [27], and GitHub Advisory [21], extend this information with additional metadata, severity scores, and exploitability details. These databases differ in scope, maintenance strategies, and data enrichment mechanisms, leading to a fragmented environment with varying degrees of completeness, consistency, and accuracy.

Recent research has examined the quality of vulnerability databases [5, 9, 36, 42, 43, 48, 56, 71]. Studies on individual databases have revealed critical data quality issues. Nguyen et al. [49] found that 25% of Google Chrome CVEs in NVD contained incorrect version information, while Malone et al. [40] pointed out significant patch omissions and data errors in NVD. Anwar et al. [1] highlighted inconsistencies in publication dates, vendor names, and product information within NVD. Moving beyond single-database studies, cross-database research has revealed broader inconsistencies. Roland et al. [12] examined discrepancies in severity ratings across several databases, Dong et al. [19] found that only 59.82% of NVD reports matched CVE summaries and external sources on affected versions, and Sun et al. [63] demonstrated inconsistencies in affected products, attack vectors among NVD, IBM X-Force, and ExploitDB. Despite these insights, most prior studies focus on individual databases or isolated vulnerability fields, and even multi-source analyses are typically limited to partial comparisons. As a result, there is still a lack of a holistic understanding of how vulnerability databases differ in composition, field completeness, consistency, and accuracy, which limits the ability to derive principled source selection and maintenance strategies.

This gap is further worsened by three technical challenges. First, the heterogeneity of multi-source databases in schema design, field granularity, and data representation means that without rigorous schema alignment, downstream analyses cannot be meaningfully conducted. Second, field-level inconsistencies, both semantic and syntactic, hinder cross-database comparisons and the systematic analysis of discrepancies. Third, validating the correctness of key fields is time-consuming and labor-intensive due to the lack of authoritative ground truth and messy, duplicate references requiring manual effort. These challenges pose a critical barrier for downstream applications such as modern software supply chain security, where tools like SCA depend on integrating heterogeneous vulnerability sources. Without cross-source awareness, simple CVE-ID–based integration risks semantic conflicts, version drift, and reference ambiguity, undermining the reliability of downstream vulnerability detection and remediation.

To address this, we conduct a systematic measurement study of 16 multi-source vulnerability data, spanning general-purpose, vendor-maintained, and ecosystem-specific sources. Our analysis is guided by three core dimensions, completeness, consistency, and accuracy, aiming to uncover the cross-database variations that introduce risks to downstream applications and hinder effective data selection, integration, and quality improvement. First, we define a unified schema and use LLM-assisted extraction to align database formats and field definitions, establishing a normalized representation and enabling large-scale comparison across sources. Second, we perform cross-database analysis on key vulnerability data fields such as description, affected components, versions, severity scores, and weakness by

normalizing field formats and applying rule-based field matching to uncover typical inconsistency types and their causes. Third, we take substantial manual effort to assess the correctness of reported information using a manually validated dataset of 2,193 vulnerability entries, revealing variation in validation practices and reporting quality across databases. We further evaluate the effectiveness of three integration strategies, majority consensus, reliability ranking, and LLM-based integration, in improving data accuracy.

Our analysis reveals three integration challenges across vulnerability databases. **(1) Coverage and Field Completeness.** There is a clear trade-off between coverage and field richness. General databases (e.g., CVE and NVD) cover around 70% of total vulnerabilities but are largely CVE-centric with limited field dimensions. In contrast, ecosystem- and enterprise-oriented databases contribute most of the 29% non-CVE records and provide richer analytical fields such as impacts, remediation, and exploitation (often >0.50 coverage), yet these fields remain largely missing. **(2) Cross-Database Consistency.** More than 70% of database pairs exhibit consistency below 0.3. Description structures diverge in both entity type (11–22 types) and entity count per vulnerability (5.22–17.94 on average). Only 12.5% of weaknesses achieve full agreement due to differences in CWE hierarchy selection and classification strategies. 58.6% of severity assessments significantly disagree, with impact-related metrics (C/I/A) reaching inconsistency up to 36.4%. The Affected field shows the lowest stability, with component granularity ranging from 1 to 16.58 per vulnerability and over 70% of version records missing or conflicting. **(3) Accuracy and Integration Effectiveness.** The *Affected* field is the least reliable (62.06%) due to component scope and version ambiguity. Constrained fields (*Weaknesses*) benefit most from reliability ranking (94.23%) and also from majority consensus (91.94%). Context-dependent fields (*Severity*) are better addressed by LLM-based reasoning (84.25%), as majority voting reduces accuracy (63.91%). Structurally complex and source-sensitive fields (*Affected*) are better improved by reliability ranking (76.59%) and LLM-based integration (80.91%). No single strategy universally resolves cross-database discrepancies. These results directly inform source selection by revealing each database's strengths and blind spots across completeness, consistency, and accuracy, and guide maintenance priorities by pinpointing where standardization, cross-source supplementation, and validation are most needed to improve data quality and integration efficiency.

In summary, we make the main contributions as follows.

- **Comprehensive Multi-source Analysis.** We systematically evaluate 16 public vulnerability databases across completeness, consistency, and accuracy, revealing critical gaps that affect data reliability and downstream usability.
- **Unified Schema and Data Alignment.** We develop a unified schema[1] and employ LLM to extract and align heterogeneous vulnerability data and enable large-scale field-level comparison across sources.
- **Insights for Database Selection and Maintenance.** We identify divergence types and their root causes across sources, offering practical guidance for informed data selection and targeted maintenance.

2 Problem Statement

Vulnerability data has become increasingly fragmented, as numerous public databases, vendor advisories, and community platforms independently publish and maintain vulnerability information. While this ecosystem collectively enriches vulnerability intelligence, the lack of unified curation leads to substantial redundancy, inconsistency, and conflicts across sources, posing significant challenges for downstream security tasks.

[1]https://sites.google.com/view/vuln-harmonization/home

Table 1. Cross-database representation of CVE-2026-34379 (a) and SCA tool detection results (b). Shaded cells (gray) indicate absent fields; highlighted cells (yellow) mark notable inconsistencies; ✓ = vulnerability detected; ✗ = no vulnerability detected.

(a) Cross-database representation of CVE-2026-34379.

Database	Description	Affected	Severity	Weaknesses
NVD	OpenEXR provides the specification and reference implementation of the EXR file format, an image storage format for the motion picture industry. From 3.2.0 to before 3.2.7, 3.3.9, and 3.4.9, …	openexr/openexr: `[3.2.0, 3.2.7)`; `[3.3.0, 3.3.9)`; `[3.4.0, 3.4.9)`	NVD: N/A;	CWE-704, CWE-787, CWE-843, CWE-475
GitHub	A misaligned memory write vulnerability exists in `LossyDctDecoder_execute()` in src/lib/OpenEXRCore/internal_dwa_decoder.h:749. When decoding a DWA or DWAB-compressed EXR file …	academysoftwarefoundation/openexr: `[3.2.0, 3.2.6]`; `[3.3.0, 3.3.8]`; `[3.4.0, 3.4.8]`	CVSS v3.1: 7.1 `AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:L/A:H` High	CWE-704, CWE-787, CWE-843
OSV	OpenEXR provides the specification and reference implementation of the EXR file format, an image storage format for the motion picture industry. From 3.2.0 to before 3.2.7, 3.3.9, and 3.4.9, …	academysoftwarefoundation/openexr: `[3.2.0, 3.2.6]`; `[3.3.0, 3.3.9-rc]`; `[3.4.0, 3.4.9-rc]`	CVSS v3.1: 7.1 `AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:L/A:H` High	CWE-704, CWE-787, CWE-843
Snyk	Affected versions of this package are vulnerable to Incorrect Type Conversion or Cast in the `LossyDctDecoder_execute` process when decoding DWA or DWAB-compressed files containing FLOAT-type …	conan/openexr: `[3.2.1, 3.2.7)`; `[3.3.0, 3.3.9)`; `[3.4.2, 3.4.9)`	CVSS v4.0: 7.1 `AV:N/AC:L/AT:N/PR:N/UI:P/VC:N/VI:L/VA:H/SC:N/SI:N/SA:N` High	CWE-704
Debian	OpenEXR provides the specification and reference implementation of the EXR file format, an image storage format for the motion picture industry. From 3.2.0 to before 3.2.7, 3.3.9, and 3.4.9, …	`openexr`: bullseye `2.5.4-2+deb11u1`: vulnerable; bookworm `3.1.5-5`: vulnerable; trixie `3.1.13-2`: vulnerable; forky/sid `3.4.6+ds-4`: vulnerable	—	—
Ubuntu	OpenEXR provides the specification and reference implementation of the EXR file format, an image storage format for the motion picture industry. From 3.2.0 to before 3.2.7, 3.3.9, and 3.4.9, …	`openexr` in Ubuntu 16.04–26.04: *Not affected*	CVSS v3.1: 7.1 High `AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:L/A:H` Ubuntu priority: Medium	—

(b) SCA tool detection results for CVE-2026-34379.

SCA Tool	Data Source	`openexr` v3.4.1 (vulnerable)	`openexr` v3.4.9-rc (non-vulnerable)
Snyk	Snyk	✗	✓
Dependabot	GitHub	✓	✗
osv-scanner	OSV	✓	✓
OWASP DC	NVD	✓	✓

Table 1 illustrates this problem using CVE-2026-34379, a memory-alignment vulnerability in OpenEXR disclosed in April 2026, as represented across six vulnerability databases. Although all databases refer to the same vulnerability, notable inconsistencies remain across key fields. For *Description*, NVD, OSV, Debian, and Ubuntu largely reuse the upstream description, whereas GitHub provides more detailed information on the vulnerable function, file location, triggering conditions, and memory-access behavior. Snyk instead characterizes the issue as an "Incorrect Type Conversion or Cast" in `LossyDctDecoder_execute`. More importantly, the databases disagree on the *Affected* field. NVD reports three affected ranges, `[3.2.0, 3.2.7)`, `[3.3.0, 3.3.9)`, and `[3.4.0, 3.4.9)`, while GitHub lists the corresponding stable affected versions through 3.2.6, 3.3.8, and 3.4.8. OSV further includes the release candidates `3.3.9-rc` and `3.4.9-rc` as affected. In contrast, Snyk narrows the first and third ranges to `[3.2.1, 3.2.7)` and `[3.4.2, 3.4.9)`, excluding versions 3.2.0, 3.4.0, and 3.4.1 that are considered affected by the other upstream-oriented databases. Meanwhile, Debian and Ubuntu use distribution-specific package names with distribution release labels instead of upstream version ranges. For *Severity*, GitHub and OSV report CVSS v3.1 7.1 (High) with `UI:R`, while NVD provides no NVD-assigned score. Snyk also reports 7.1 (High) but uses CVSS v4.0 with `UI:P`. Ubuntu reports CVSS v3.1 7.1 (High) but assigns a Medium

priority, reflecting inconsistent severity assessments of the same vulnerability. For *Weaknesses*, GitHub and OSV report CWE-704, CWE-787, and CWE-843, while NVD additionally includes CWE-475. Snyk reports only CWE-704, and Debian and Ubuntu provide no structured weakness information.

These inconsistencies directly affect downstream vulnerability detection. As shown in Table 1(b), we evaluated four SCA tools, each relying on a different vulnerability data source, using two OpenEXR versions whose vulnerability status was manually verified from the source code and fixing commits. Version `v3.4.1` contains the vulnerable misaligned memory-access code and is therefore affected by CVE-2026-34379, whereas `v3.4.9-rc` already contains the corresponding fix and is non-vulnerable. For `v3.4.1`, Dependabot, osv-scanner, and OWASP Dependency-Check correctly detect the vulnerability, while Snyk misses it because its affected range starts from version 3.4.2. Conversely, for the fixed `v3.4.9-rc`, Dependabot correctly reports no vulnerability, whereas Snyk, osv-scanner, and OWASP Dependency-Check still flag it as vulnerable. Beyond detection errors, inconsistent severity scores may lead to different vulnerability prioritization decisions, while conflicting weakness classifications (e.g., CWE-704 versus CWE-475) may misdirect remediation efforts toward different root causes. This highlights that cross-database inconsistency is not merely a data quality issue: it encompasses not only missing or conflicting information, but also potentially incorrect records, directly affecting vulnerability detection, prioritization, and remediation in practice.

Although prior studies have revealed data-quality issues in vulnerability databases, they typically focus on a small number of sources, specific fields, or individual quality dimensions. As a result, we still lack a systematic understanding of how vulnerability information differs across diverse databases and how these differences affect vulnerability-data integration. To address this gap, we conduct a large-scale measurement study of 16 representative vulnerability databases. We first construct a unified vulnerability schema to reconcile heterogeneous data representations and then systematically evaluate vulnerability data quality in terms of completeness, consistency, and accuracy.

3 Data Collection and Preliminary Study

In this section, we detail the process of collecting and aligning data from multiple vulnerability databases, addressing differences in schema design and field representation to enable comprehensive and cross-source comparison, as well as the preliminary analysis conducted to prepare for our study.

3.1 Data Collection

Vulnerability Database Selection. To enable a comprehensive analysis of multi-source vulnerability data, we selected databases in two stages: (1) widely used general-purpose databases and (2) well-known specialized databases. For general-purpose databases, we started from the union of databases identified in two prior studies. Li et al. [36] analyzed the usage of publicly available vulnerability databases in academic research, summarizing 25 databases and indicating their citation frequency in research papers, while Imtiaz et al. [28] examined databases commonly used by software composition analysis (SCA) tools, contributing additional 3 databases including GitHub Advisory [21], Mend.io (formerly WhiteSource) [41], and Snyk [61]. Then, from the 18 candidate databases, we selected the final set based on three criteria: ① **Publicly available**, freely accessible for research. This excluded **8** databases that were commercial, internally maintained, or no longer publicly accessible, including commercial paid platforms (*1337Day*, *Vupen*), internal organizational systems (*CoopVDB*, *DoE-CIRC*), and databases that are no longer publicly accessible (*NVDB*, *Secunia*). ② **Widely adopted**, cited in at least two papers in Li et al. or used by at least two SCA tools in Imtiaz et al., to ensure representativeness. This excluded **8** databases with limited adoption in prior research and practice, including *Android Security Bulletins*, *CERT-FR*, *JVN*, *XSA*, *VMware Security Advisories*, *CNVD/CNNVD*, and *Securiteam*. ③

Actively maintained, with updates in the past two years. This excluded **2** databases: *SecurityFocus*, which has not been actively maintained since 2023, and *SARD*, which is a static synthetic test dataset rather than a continuously maintained vulnerability database. Applying these criteria yielded a final set of **10** general-purpose databases: CVE [10], NVD [46], OSV [51], the Exploit Database (EDB) [20], Cert [11], Snyk [61], GitHub Advisory (GitHub) [21], GitLab Gemnasium (GitLab) [22], IBM X-Force (IBM) [27], and Mend.io [41].

To capture ecosystem-specific vulnerability reporting practices that are underrepresented in general-purpose databases, we further included 6 well-known specialized databases spanning three categories: ① community-driven bug bounty platform, Huntr [26] (targeting open-source AI/ML projects); ② security advisory databases, including RedHat [55], Debian [18], and Ubuntu [67], all maintained by major Linux distributions; and ③ ecosystem-maintained vulnerability trackers, including RustSec [59] (the official security advisory database for the Rust ecosystem) and Curl [14] (a project-maintained tracker). These sources are commonly referenced in previous empirical studies and real-world vulnerability management workflows as complementary channels to general-purpose databases [3, 4, 38, 82], providing ecosystem-specific perspectives on vulnerability disclosure and remediation. In total, we selected 16 vulnerability databases that cover both mainstream vulnerability management platforms and specialized ecosystems, providing a balanced and representative basis for cross-database analysis.

Vulnerability Database Collection. We used public APIs and open datasets to collect vulnerability data where available. For databases with API access [45], automated scripts were developed to retrieve structured vulnerability data efficiently. These APIs provide standardized formats tailored to each database, streamlining data extraction. For databases lacking API support, such as EDB [20] and Huntr [26], we employed a dynamic web crawler to extract vulnerability-related metadata by leveraging webpage structural patterns, thereby converting unstructured content into a structured dataset. To maintain data timeliness, we implemented a scheduled crawling mechanism to periodically update the collected vulnerability records from these databases. All experiments reported in this paper are based on the snapshot collected by November 4th, 2025.

3.2 Schema Identification

Existing vulnerability databases lack unified standards in data organization and representation, with different databases adopting independently defined data schemas that exhibit significant variations in field definitions, data formats, and structural organization. This heterogeneity makes it difficult to directly compare vulnerability information from different data sources. To address this, we construct a unified schema that unifies vulnerability information from multiple sources through a three-stage process: (1) Reference Schema Identification, (2) Cross-schema Field Grouping, and (3) Unified Schema Construction.

3.2.1 Reference Schema Identification. To identify reference schemas for unified schema construction, we manually reviewed the documentation of all 16 databases and identified 5 schemas that explicitly define structured data formats with clear field semantics, data types, and format constraints: ① CVE Record Format (CVE-Schema) [15], ② NVD CVE API Schema (NVD-Schema) [44], ③ OSV Schema (OSV-Schema) [50], ④ Red Hat Security Data API (RedHat-Schema) [54], ⑤ RustSec Advisory Format (RustSec-Schema) [58]. These schemas were selected as reference schemas because they provide sufficiently detailed structural specifications to support systematic semantic alignment and schema construction. Collectively, they represent the major vulnerability representation paradigms used across the selected databases, including official numbering systems (CVE/NVD), operating system distributions (Red Hat), and open source ecosystems (OSV, RustSec). The remaining 11 databases do not provide explicit schema documentation, and their fields

Table 2. Field Coverage and Semantic Mapping Across 5 Vulnerability Schemas

Unified Field	CVE-Schema	NVD-Schema	OSV-Schema	RedHat-Schema	RustSec-Schema	Cov.
id	cveMetadata.cveId	id	id	Bugzilla.id,	advisory.id	**5/5**
source	source	sourceIdentifier	-	-	-	2/5
aliases	-	-	aliases	-	aliases	2/5
related	-	-	related	-	related	2/5
published	datePublic	published	published	PublicDate	date	**5/5**
modified	dateUpdated	lastModified	modified	-	-	3/5
withdrawn	-	-	withdrawn	-	withdrawn	2/5
vuln_status	cveMetadata.state	vulnStatus	-	-	-	2/5
title	title	-	summary	-	-	2/5
description	descriptions	descriptions	details	Details	-	**4/5**
weaknesses	problemTypes	weaknesses	-	CWE	categories	**4/5**
severity	metrics	metrics	severity	ThreatSeverity, CVSS, CVSS3	cvss	**5/5**
impacts	impacts	evaluatorImpact	-	-	-	2/5
exploitability	exploits	-	-	-	-	1/5
affected	affected	configurations	affected	AffectedRelease, PackageState	affected, package, versions	**5/5**
cpe_configurations	cpeApplicability	configurations	-	-	-	2/5
remediation	solutions, workarounds[]	evaluatorSolution	-	Mitigation, UpstreamFix	versions.patched	**4/5**
references	references	references	references	References	references	**5/5**
vendor_comments	-	vendorComments, evaluatorComment	-	Statement	-	2/5
acknowledgements	credits	-	credits	Acknowledgements	-	3/5
tags	tags	cveTags	database_specific	-	informational, keywords	**4/5**
copyrights	-	-	-	-	license	1/5
vendor_url	-	-	-	-	url	1/5
timeline	timeline	-	-	-	-	1/5
upstream	-	-	upstream	-	-	1/5
taxonomy_mappings	taxonomyMappings	-	-	-	-	1/5
configurations	configurations	-	-	-	-	1/5
cisa_info	-	cisaExploitAdd cisaActionDue cisaRequiredAction cisaVulnerabilityName	-	-	-	1/5
Total Fields	**20**	**19**	**14**	**14**	**15**	**-**

Note: "-" indicates the field is not present in the schema. "Cov." shows the number of schemas supporting each field out of 5 total. Underlined fields in the Schema-Specific section were excluded from the final unified schema.

were inferred from crawled records. As detailed in Section 3.3.1, we verified that all 16 databases can be fully mapped to the resulting unified schema, confirming that the 5 reference schemas provide sufficient coverage for all 16 databases without information loss.

3.2.2 Cross-Schema Field Grouping. To identify a consolidated set of distinct field types across the 5 reference schemas, three co-authors grouped all fields from these schemas into candidate categories based on semantic equivalence. Specifically, two co-authors independently extracted all fields defined in the 5 schemas, covering 82 fields in total, and recorded their names, semantic definitions, data structures, and format constraints. The annotators then compared

fields across schemas based on their documented semantics, information scope, and grouped semantically equivalent fields into candidate categories. To ensure consistency, we established three decision criteria: ① fields were grouped only when they described the same vulnerability concept; ② fields with substantially different information scopes were treated separately; and ③ unified field names prioritized widely adopted terminology among those schemas. For example, fields such as `summary`, `title` were grouped into the unified `title` field, as both provide a concise one-sentence description of the vulnerability. Similarly, fields such as `credits`, `reporter`, and `acknowledgements` were unified under `acknowledgements`, as both record individuals or organizations credited for discovering or reporting the vulnerability. The two annotators independently performed semantic grouping before reconciliation. Out of 82 annotation fields covering all fields across the 5 reference schemas (total 82 fields), the two annotators agreed on 73 fields before conflict resolution, yielding an agreement rate of 89.0% (Cohen's κ = 0.88). Disagreements mainly arose for fields with partially overlapping semantics, such as RustSec `categories` versus CWE classifications and OSV `affected` ranges versus NVD `configurations`. All disagreements were resolved through discussion with a third co-author until consensus was reached. Through this process, we identified 28 distinct field types, as shown in Table 2, which presents the classification of fields from each schema into these field types.

However, different schemas exhibit considerable variation. In terms of field count, CVE-Schema and NVD-Schema are the most comprehensive, containing 20 and 19 fields respectively, while OSV-Schema, RedHat-Schema, and RustSec-Schema contain 14-15 fields each. Among these, only a small number of fields (`id`, `description`, `published`, `severity`, `weaknesses`, `references`, `affected`, `remediation`) appear in all or most (≥4) schemas; 11 fields are supported by 2-3 schemas, demonstrating good generality; while 8 fields are supported by only single schema, representing platform-specific or low-frequency fields. Furthermore, fields with equivalent semantic definition exhibit significant representational differences across schemas. By comparing these 28 fields across different schemas, we identified the following major patterns of variation:

- **Naming and organization variations.** Fields with identical semantics use different naming and nesting structures across schemas. For example, modification time is represented as `cveMetadata.dateUpdated` in CVE-Schema, `lastModified` in NVD-Schema, and `modified` or `updated` in OSV-Schema.
- **Structure and composition variations.** Even when field definitions are identical, different schemas may adopt completely different data structures such as arrays, objects, or strings. For instance, severity information is represented as a `metrics` array (supporting multiple scoring scenarios) in CVE-Schema, nested as a multi-layer object structure (`metrics.cvssMetricV2/V3/V31/V40`) in NVD-Schema, as a `severity` array in OSV-Schema, and simplified to a single object or string in RustSec-Schema.
- **Format and constraint variations.** Different schemas impose different data types and format restrictions on the same field. For example, vulnerability IDs in CVE-Schema are strictly limited to the `CVE-YYYY-NNNN` format with a maximum of 19 digits for the numeric portion, NVD-Schema has no upper limit on digit count, OSV-Schema supports over 20 different prefixes (e.g., GHSA-, RUSTSEC-), while Vulners-API imposes no format restrictions.
- **Scope and granularity variations.** The same field name may encode different semantic scopes or degrees of granularity across schemas. For example, remediation information in CVE-Schema is split into `solutions` (official fixes) and `workarounds` (temporary mitigations), while in other schemas it is unified as a single `remediation` or `patched_versions` field. Similarly, the semantic concept of affected products corresponds to significantly different numbers of fields across schemas: CVE-Schema uses a single `affected` array, while RustSec-Schema splits it into three parallel fields (`affected`, `package`, `versions`).

3.2.3 Unified Schema Construction. Based on the identified field types and the observed variations in naming, structure, format, and scope across schemas, the three co-authors jointly finalized the unified field set and their corresponding definitions through consensus discussion. For each field type, the two annotators first independently proposed a unified field definition, field name, and structural representation based on the aligned source fields. The third co-author then reviewed cases with inconsistent decisions and facilitated discussion until agreement was reached.

During this process, the authors determined whether each field type should be retained, merged with another, or excluded, guided by three principles: semantic completeness, structural expressiveness, and cross-database compatibility. Specifically: ① For fields included in most (≥4) schemas with consistent semantics (9 fields), we retained these fields and standardized their data structures and format constraints. Data structure selection follows the principle of maximum information capacity: when some schemas use arrays while others use single objects, we uniformly adopt array structures to support multi-value scenarios. Format constraints follow the principle of maximum compatibility, designing unified representations that cover all source format variants and ensuring cross-source data comparability and computability through explicit normalization rules (e.g., timestamps unified to ISO 8601 UTC, identifier regularization). ② For fields supported by 2-3 schemas but with varying definitions (11 fields), we adopted a strategy combining semantic priority with structural standardization. After identifying the shared vulnerability concept represented by each field, we selected the schema with the richest structured representation as the structural design baseline, and added optional subfields to accommodate special requirements from other schemas. For example, the `severity` field was unified as an object array containing scoring source, scoring scheme version, numeric score, scoring vector, and severity rating, which can express NVD-Schema's multi-version CVSS scores while also accommodating RedHat-Schema's qualitative ratings and OSV-Schema's multi-source scores. ③ For proprietary fields supported by only single schema (8 fields), we evaluated their relevance to vulnerability information and the necessity of field retention. We excluded 6 fields: `vendor_url` and `upstream`, which are redundant with `references`; `timeline`, whose information overlaps with `published`, `modified`, and `withdrawn`; `taxonomy_mappings`, which records references to external classification systems, while the vulnerability classifications are already represented by `weaknesses` field; `configurations`, whose affected-scope semantics are captured by `affected`; and `cisa_info`, which records CISA-specific metadata such as KEV catalog status and due dates rather than vulnerability semantics. The remaining fields, `exploitation` and `copyrights`, were retained as optional extensions because `exploitation` captures evidence of exploit availability and usage for risk prioritization, while `copyrights` preserves advisory licensing and attribution requirements that are legally necessary for data redistribution and reuse.

Through this layered alignment and selection process, we ultimately converged the original 28 field semantics into a unified schema containing 22 fields. Taking CVE-2015-3226 as an example in Figure 1, the schema organizes multi-source vulnerability information into structured fields that capture the essential semantics from heterogeneous data sources. The complete definition of each unified field, including its semantics, data type, format constraints, and sub-field structure, is provided in Appendix B. The detailed specifications of the 5 reference schemas, including field names, definitions, sub-fields, data structures, and instantiation examples, are available on our project website [2].

3.3 Data Alignment of multi-source vulnerability data

Even with a unified data schema constructed, mapping heterogeneous multi-source vulnerability data to standardized representation still faces two main challenges. First, semantic alignment in field mapping: different databases use varying field names, structural organization, and data formats to represent the same concepts, requiring accurate field correspondence. Second, granularity alignment in data representation: some databases aggregate multiple types of

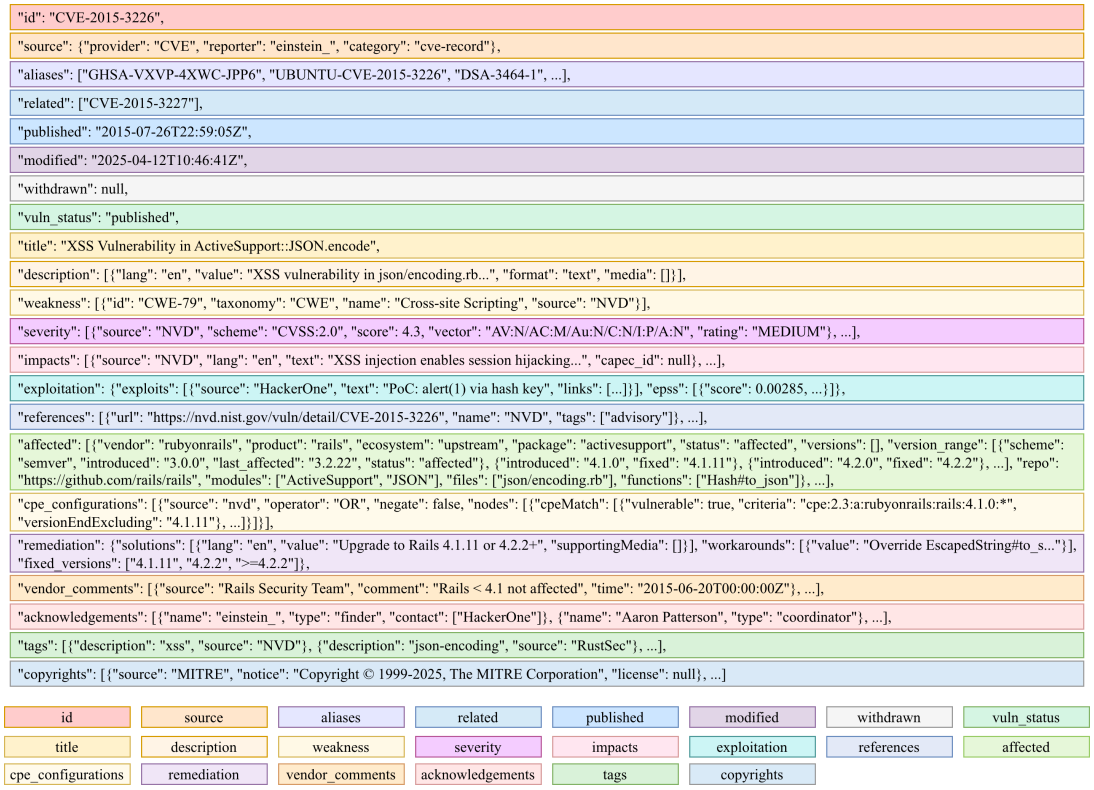


Fig. 1. An Example of Unified Schema Fields: CVE-2015-3226.

information into coarse-grained fields instead of organizing them into distinct fields, necessitating the extraction and separation of mixed content. To address these challenges, we implement a structured field mapping approach based on the unified schema constructed in the previous section, mapping heterogeneous vulnerability data into a standardized representation. This process consists of two main steps.

3.3.1 Field Mapping. Based on the unified schema, we map fields from each vulnerability database to standardized fields. For databases with explicitly defined schemas (including CVE, NVD, OSV, Red Hat, and RustSec), we leverage the predefined correspondence between their official schemas and the unified schema (as shown in Table 2) to directly parse and map original fields to standardized fields, while normalizing data structures and formats. For databases that do not provide explicit schema definitions, we extract field names and field contents from structured and semi-structured records to construct a candidate field set. Based on the semantic meanings of these fields, we follow the same semantic alignment procedure described in Section 3.2 to map them to corresponding standard fields in the unified schema. For example, the "Summary" field in CERT is mapped to the `title` field in the unified schema; fields describing vulnerability types or root causes are mapped to `weaknesses`; and product names and version information are unified under the `affected` field. **After field mapping, all fields in those databases are normalized to the 22 standard fields defined in the unified schema, without introducing additional fields, indicating the completeness of the unified schema.** The complete field mapping for all 16 databases is provided in Appendix A.

3.3.2 Field Completion. Although field mapping aligns field names and data formats, discrepancies remain in field completeness across databases. Specifically, some databases omit certain structured fields entirely, while the corresponding information is instead embedded in unstructured or semi-structured text. For example, product names and affected versions may only appear in free-text descriptions in some databases, whereas other databases explicitly represent this information using dedicated structured fields. As a result, direct field mapping alone is insufficient, necessitating the structured extraction of missing information from unstructured content. To address this issue, we employ a few-shot learning (FSL)–based approach leveraging large language models (LLMs) to extract and populate absent fields from existing vulnerability information within each record. In particular, we use standardized records mapped from databases with explicitly defined schema (CVE, NVD, OSV, Red Hat, and RustSec) as few-shot examples to guide the extraction.

The process consists of two key steps: (1) Identifying missing fields based on the unified schema and (2) Extracting and completing them using LLMs. First, we identify fields that require completion by comparing the available fields in each record against the unified schema. Second, for each identified missing field, we extract relevant information from existing content within the same record, including both structured attributes and unstructured textual descriptions. Importantly, rather than introducing external information, the extraction process is constrained to the original record. We then apply an LLM-driven extraction approach to populate the missing fields, where the LLM is used solely as an information extractor and is restricted to the provided record content, preventing the introduction of information not present in the record. This involves: 1) Schema-Aware Extraction. LLM is guided by a structured prompt that defines expected fields, data formats, and constraints, and is further provided with standardized few-shot examples for each field derived from schema-defined databases to ensure schema adherence while preserving original records. 2) Context-Aware Processing. The model analyzes available fields such as descriptions, references, and metadata to identify and extract missing details from free-text while preserving contextual meaning and formatting them according to the predefined schema. To prevent hallucinations, it strictly derives information from provided data, returning "None" when no relevant content is found. The output is structured in JSON for seamless schema integration.

3.3.3 Evaluation of Field Completion. **Data preparation (Dataset-A).** To evaluate the effectiveness of LLMs in completing missing fields, we constructed a dedicated dataset for field completion based on the unified schema. Specifically, we first identified vulnerability records in which certain fields were missing, as described in Section 3.3.2. We then manually annotated these missing fields to establish the ground truth. During the annotation, we observed that only the fields *Title*, *Weaknesses*, *Severity*, *Affected*, *Impacts*, *Exploitation*, *Remediation*, and *References* could be extracted using information available within the vulnerability record. The remaining fields were excluded from annotation because they either represent identifiers or temporal attributes (e.g., *Source*, *Published*), or unique representations and provenance information of certain databases (e.g., *CPE*, *Acknowledgements*) that cannot be derived from other fields.

To ensure annotation reliability, two co-authors independently annotated each missing field following a shared set of annotation guidelines. Specifically, annotators were instructed to follow the unified schema definitions, use the field descriptions and annotation examples provided for each field, and annotate values solely based on information explicitly available in the original vulnerability record. No external sources, prior knowledge, or implicit assumptions were allowed during annotation. Out of 1,600 annotation instances (200 records $\times$ 8 fields), the two annotators reached agreement on 1,526 instances, yielding an inter-annotator agreement rate of 95.4%. Disagreements were more frequent for fields requiring subjective expression, particularly *Title* (where annotators used different phrasings to describe the same vulnerability) and *Impacts* (where annotators described consequences at different levels of granularity). In cases of disagreement, a third senior co-author reviewed the original record and adjudicated the final annotation. Finally,

Table 3. Results for Effectiveness Evaluation of Field Completion

	Title	Weaknesses	Severity	Affected	Impacts	Remediation	Exploitation	References	Total
Correct (#) / Total (#)	200 / 200	171 / 200	200 / 200	174 / 200	187 / 200	182 / 200	170 / 200	200 / 200	1,496 / 1,600
Accuracy (%)	100.0	85.5	100.0	87.0	93.5	91.0	85.0	100.0	93.5

we annotated 200 records for each of the eight fields, resulting in a total of 1,600 annotated records as **Dataset-A** for evaluating field completion performance.

Setup. To evaluate the performance of Field Completion in Section 3.3.2, we compare the fields after completion with manually annotated ground truth. First, for the vulnerabilities in Dataset-A, we mapped the fields across different databases to unify field names and data formats. Then, we used LLM ("GPT-4o-mini") [52] to identify and complete missing fields, achieving data alignment. For each field, we measure the success rate, defined as the proportion of records for which the completed field exactly matches the annotated ground truth.

Results. The results of the effectiveness evaluation are presented in Table 3. The column "Correct (#) / Total (#)" reports the number of correctly completed records over the number of evaluated records. Overall, the LLM demonstrated strong performance in completing missing fields across the unified schema. Across all 1,600 records, we identified 38 false positives (incorrect completions) and 66 false negatives (missing completions), achieving 93.5% (1,496/1,600) overall completion accuracy. The quality of completions varied across different fields. Severity (100%) maintained the highest accuracy, benefiting from standardized CVSS scores readily available in most databases, as did Title and References. Impacts (93.5%, 187/200) demonstrated strong performance, with 16 records omissions when impact information was embedded within lengthy narrative descriptions. Remediation reached 91.0% (182/200) accuracy, with most errors stemming from missing commit link recognition when patches were referenced indirectly. Affected achieved an accuracy of 87.0% (174/200) in component and version extraction, with 26 records having minor errors: 12 version parsing errors when handling descriptions like "up to and including", and 14 missing component extractions. However, Weaknesses (85.5%, 171/200) had notable extraction challenges, with 29 errors: 10 incorrect CWE mappings and 19 missing extractions. While structured CWE IDs were accurately extracted (100% when present), many databases provide weakness information only as textual descriptions without explicit CWE references, contributing to the majority of missing records. Exploitation (85.0%, 170/200) had the lowest performance, mainly due to speculative content in reports and the LLM's conservative "no over-inference" strategy, leading to more missing records. Overall, the LLM performed well in extracting structured fields (such as Weaknesses, Severity, and Affected), but struggled with inferential fields like Exploits and PoCs. Future improvements should integrate external knowledge bases and rule-based enhancements.

4 Multi-source Vulnerability Data Evaluation

4.1 Research Questions

We aim to assess the data quality of multi-source vulnerability data by answering the following three research questions (RQs), focusing on data completeness, consistency, and accuracy in different databases.

- **RQ1: (Completeness Analysis) What completeness gaps exist in multi-source vulnerability data?** To support comprehensive vulnerability discovery and risk analysis, we evaluate the completeness of vulnerability data across databases by examining vulnerability coverage and field availability. This analysis identifies missing vulnerabilities and field-level data gaps that may limit downstream usage.

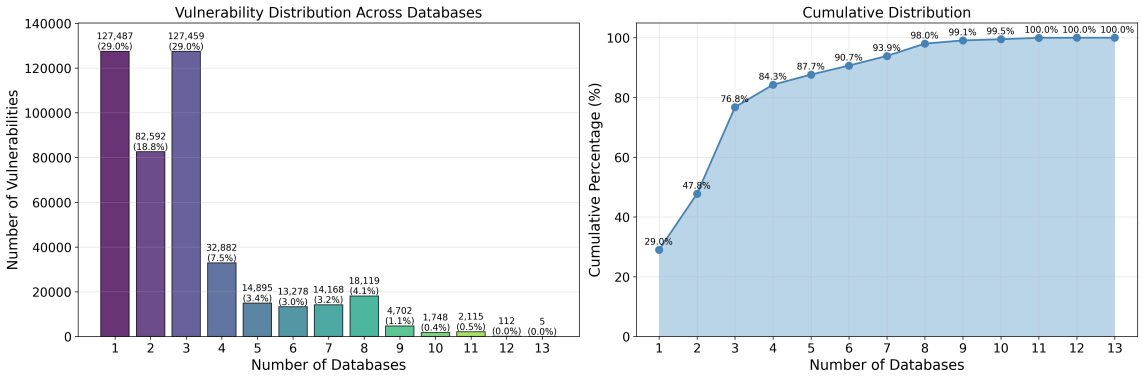


Fig. 2. Coverage Distribution of Vulnerabilities Across 16 Databases.

- **RQ2: (Inconsistency Analysis) What inconsistency patterns and causes hinder cross-source integration?** To enable reliable cross-database integration and comparison, we assess how the same vulnerability is represented across sources. By analyzing inconsistencies and their patterns, we aim to uncover divergence causes that may introduce ambiguity or conflict in multi-source usage.
- **RQ3: (Accuracy Analysis) How accurate is aggregated multi-source vulnerability data by data selection?** To support trustworthy data selection and decision making, we evaluate the field-level accuracy of multi-source vulnerability data and examine whether different aggregation strategies can enhance data accuracy. This analysis reveals the reliability of existing data and informs effective integration practices.

Data preparation (Dataset-B). To support the field completeness analysis (RQ1, Section 4.2.3) and the inconsistency analysis (RQ2, Section 4.3), we construct Dataset-B, which requires vulnerabilities to appear across multiple databases to support cross-database comparisons. To characterize how vulnerabilities are distributed across the 16 databases, we first measure the cross-database coverage of each vulnerability, defined as the number of databases in which it appears.

Figure 2 illustrates the resulting distribution. The left panel shows the number of vulnerabilities reported by different numbers of databases, while the right panel shows the corresponding cumulative distribution. Overall, 76.8% of vulnerabilities appear in fewer than four databases, making them unsuitable for meaningful cross-database comparison. We therefore define vulnerabilities reported by at least eight databases (i.e., at least half of the 16 databases) as the high-coverage core set and include all of them in Dataset-B. Although they account for only about 6% of all vulnerabilities, they represent the most widely shared vulnerabilities and provide sufficient overlap for reliable cross-database analysis, yielding 18,119 CVE vulnerabilities.

To avoid sampling bias toward highly shared vulnerabilities, we also include a portion of vulnerabilities with limited cross-database coverage through stratified sampling in Dataset-B. For vulnerabilities appearing in fewer than eight databases, we group them by cross-database coverage into strata (2–7 databases) and draw an equal number of samples from each stratum per database, targeting 100 vulnerabilities per database in total. When a lower-frequency stratum contains insufficient samples, we supplement from the immediately higher-frequency stratum to maintain the target count. After deduplication, this contributes 1,443 additional vulnerabilities.

After deduplication, we obtain a final dataset containing 19,562 vulnerabilities corresponding to a total of 192,741 vulnerability records across the 16 databases. We note that non-CVE vulnerabilities are not included, as they lack shared

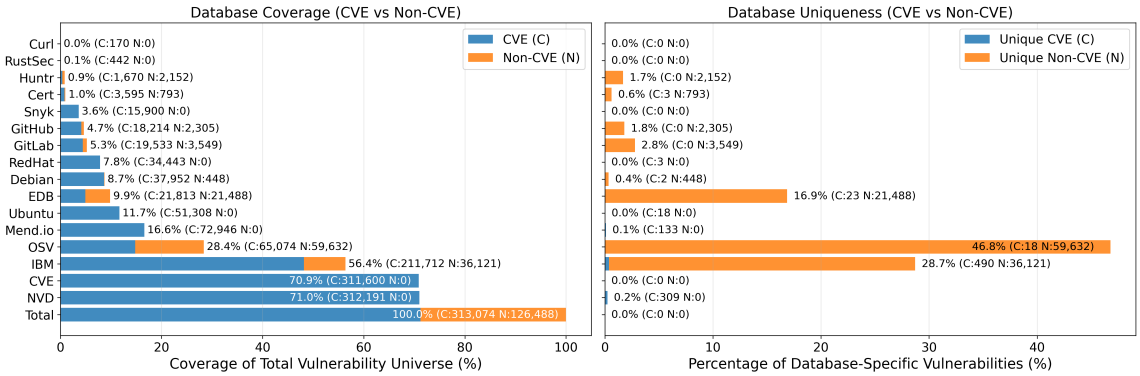


Fig. 3. Vulnerability Coverage and Uniqueness across Databases.

cross-database identifiers and appear in only a single database. We tried to link all 126,488 non-CVE vulnerabilities against all collected vulnerabilities using shared references. The results show that, although 26.68% (33,744) could be linked to at least one vulnerability in another database, only a very small portion of them has adequate evidence to be manually confirmed that they indeed describe the same vulnerability (47 out of 180 by our manual inspection). Therefore, large-scale cross-database linking for non-CVE records is currently infeasible without substantial manual effort and is outside the scope of this dataset. For the selected vulnerabilities, we then mapped the fields across different databases to unify field names and data formats, and used LLM ("GPT-4o-mini") [52] to identify and complete missing fields, achieving data alignment across databases. The resulting dataset is referred to as **Dataset-B**.

4.2 RQ1: Completeness Analysis

To reveal the completeness of vulnerability information recorded in multi-source vulnerability data, this research question analyzes data completeness from two perspectives: vulnerability coverage and field-level information completeness.

4.2.1 Vulnerability Coverage of Multi-source Vulnerability Data. To characterize vulnerability coverage across different sources, we analyze the distribution of vulnerabilities with CVE identifiers and those without CVE identifiers (non-CVE) in each database. Specifically, we quantify the proportion of CVE and non-CVE vulnerabilities covered by each database with respect to the overall vulnerability universe, and further quantify the unique vulnerabilities in each database.

Result. Figure 3 illustrates the vulnerability coverage and uniqueness across multiple databases. The left panel reports the *coverage* of each database with respect to the total vulnerability universe, decomposed into CVE and non-CVE records, while the right panel shows the *uniqueness*, i.e., the proportion of database-specific vulnerabilities not reported by any other source. The entry *Total* represents the merged vulnerability set aggregated from all data sources.

The aggregated vulnerability set (*Total*) contains 439,572 unique vulnerabilities, of which 313,084 (71.0%) are identified by CVE IDs and 126,488 (29.0%) are non-CVE records. Databases such as NVD (312,190, 71.02%), CVE (311,600, 70.89%), and IBM (212,506, 48.34%) achieve high coverage of the total vulnerability set; however, their coverage is almost entirely limited to CVE vulnerabilities and still does not include all CVE vulnerabilities. In contrast, databases including OSV (59,632 non-CVEs), EDB (21,488), GitHub (2,305), Huntr (2,152), CERT (793), and Snyk contribute substantial numbers of non-CVE vulnerabilities, reflecting the diversity of threat intelligence sources. This trend indicates that recent

vulnerability disclosures increasingly originate from ecosystem-driven and platform-specific security intelligence sources that operate outside the traditional CVE assignment process.

The right panel in Figure 3 summarizes unique vulnerabilities in different databases. In total, we identify **127,457** database-specific vulnerabilities across all sources, which refer to vulnerabilities that appear exclusively in a single database. Notably, databases such as OSV, EDB, and IBM X-Force contribute a large share of these database-specific records, but their uniqueness is dominated by non-CVE vulnerabilities (e.g., OSV: 59,632 unique non-CVEs; EDB: 21,488; IBM X-Force: 36,121).

Despite this non-CVE dominance, we still observe that 999 unique CVE vulnerabilities appear exclusively in individual databases, indicating that synchronization of CVE records across sources is not yet complete. Among them, IBM contributes the largest number of unique CVEs (490), largely associated with enterprise and vendor-specific software (e.g., SAP, Oracle, and IBM products), followed by Mend.io with 133 unique CVEs, which are primarily related to Android component vulnerabilities. In addition, OSV, EDB, and Ubuntu each contain around 20 unique CVEs (18, 23, and 18, respectively), highlighting that database size alone is not a sufficient indicator of coverage completeness, and that smaller or specialized databases contain complementary vulnerability information that would otherwise be missed.

4.2.2 Scope-Aware Coverage Analysis. To further understand the observed coverage differences, we analyze the scope characteristics of each database by examining the distribution of the `affected` field, which records the vendors, products, packages, and ecosystems associated with each vulnerability. Because these attributes directly reflect the software components referenced by vulnerability records, their distributions provide an empirical characterization of the types of software covered by each database. Specifically, following the field mapping process described in Section 3.3.1, we normalize the heterogeneous `affected` fields from all 16 databases into a unified representation, and profile the vendor, product, ecosystem, and package dimensions of the `affected` field to reveal differences in database focus and specialization.

Database Scope Identification. The results show that five databases exhibit clear scope preferences. RedHat exclusively covers vulnerabilities in Red Hat Enterprise Linux and related distributions (vendor=Red Hat, ecosystem=RHEL); Debian and Ubuntu each focus on their own distribution packages (vendor=Debian and Canonical respectively, ecosystem=Debian and Ubuntu respectively); RustSec focuses entirely on the Rust ecosystem (ecosystem=`crates.io`); and Curl exclusively tracks vulnerabilities in the curl project (package=`curl`, ecosystem=`C/C++`). Their coverage is naturally bounded by their intended software domain. These empirically observed scope boundaries are consistent with each database's stated scope: RedHat publishes security advisories for its own product portfolio [54], Debian and Ubuntu maintain security trackers for their respective distribution packages [8, 17], RustSec is maintained by the Rust community to track vulnerabilities in the `crates.io` ecosystem [58], and Curl maintains its own vulnerability database exclusively for the curl project [13]. And the remaining 11 databases show broadly dispersed distributions with no well-defined scope boundaries.

Expected Vulnerability Coverage. For the five domain-specific databases, we further compute the expected vulnerability universe by classifying each vulnerability in the full dataset as in-scope based on its `affected` field: vendor=*Red Hat* or ecosystem=*RHEL* for RedHat; vendor=*Debian* or ecosystem=*Debian* for Debian; vendor=*Canonical* or ecosystem=*Ubuntu* for Ubuntu; ecosystem=`crates.io` for RustSec; and package=`curl` for Curl. This yields the expected vulnerability universe for each database, against which we measure actual coverage.

Table 4 presents the scope-aware coverage results. For each specialized database, the *In-Scope Total* columns report the total number of unique vulnerabilities within its defined scope across all 16 databases, while the *Per-Source Contribution*

Table 4. Scope-aware coverage analysis for the five domain-specific databases. *Total* and *CVE/Non-CVE* report the number of unique vulnerability IDs within each database's defined scope across all 16 databases. The remaining columns report the per-source contribution (number of in-scope vulnerabilities found in each database); bold values indicate the target database itself.

Scope	In-Scope Total			Per-Source Contribution														
	Total	CVE	Non-CVE	CVE	NVD	CNVD	CNNVD	EDB	IBM	GitHub	GitLab	OSV	Snyk	Debian	Ubuntu	RustSec	RedHat	Curl
RedHat	35,411	35,131	280	1,239	3,757	575	0	1	1,357	0	0	0	0	0	0	0	**34,443**	0
Debian	49,858	39,949	9,909	27	9,953	45	78	0	0	0	0	9,902	0	**37,952**	0	0	0	
Ubuntu	52,248	52,231	17	171	4,251	63	161	0	0	0	0	0	0	0	**51,308**	0	0	0
RustSec	2,467	848	1,619	0	346	0	0	0	0	670	1,160	1,388	479	0	0	**442**	0	0
Curl	283	181	102	63	142	5	126	2	0	2	3	216	2	155	124	0	141	**170**

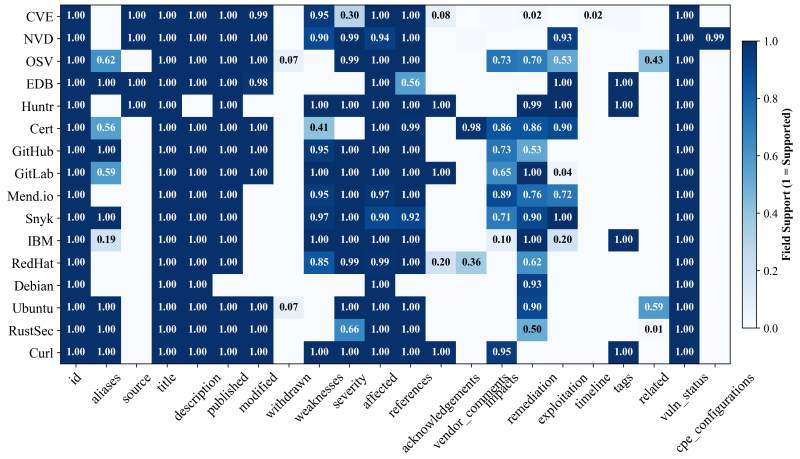
columns show how many of these vulnerabilities are covered by each database. Ubuntu and RedHat achieve high in-scope coverage (98.2% and 97.3% respectively), while Debian reaches 76.1%. However, none achieves complete coverage: the missing in-scope vulnerabilities are distributed across other databases. For RustSec, only 442 of 2,467 in-scope vulnerabilities (17.9%) are collected, with OSV (1,388) and GitLab (1,160) contributing substantially more. For Curl, 170 of 283 in-scope vulnerabilities (60.1%) appear in the Curl database, with OSV (216) and NVD (142) providing complementary coverage. These results show that even databases with clearly defined scopes exhibit non-trivial coverage gaps, underscoring the necessity of cross-database integration even within a single domain.

4.2.3 Field Completeness in Multi-Source Vulnerability Data. To reveal the composition of vulnerability information recorded by different databases, we assess field-level completeness in multi-source vulnerability data by comparing the distribution of information fields across databases in Dataset-B. This analysis examines the availability of fields in vulnerability records and reveals differences in information richness among sources.

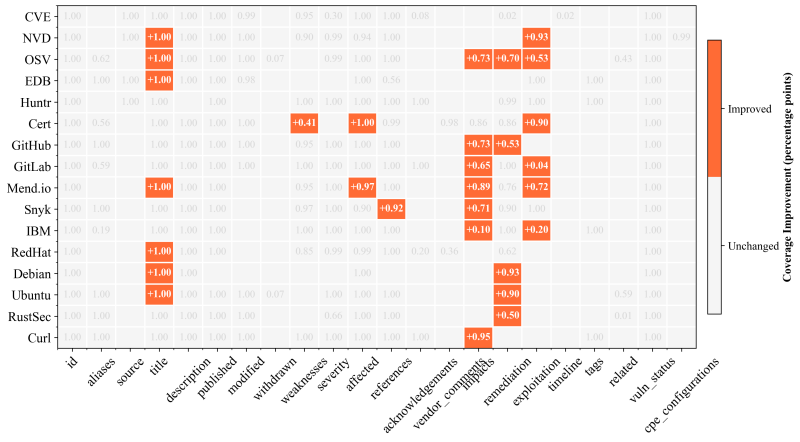
Result. Figure 4 illustrates the field-level completeness of multi-source vulnerability data after field completion (Figure 4a), where darker colors indicate higher field availability, together with the coverage improvements introduced by completion (Figure 4b). Overall, substantial variation exists in field coverage across databases, both in terms of the number of supported fields and their completeness.

Core identification fields, such as *id*, *title*, *description*, and *affected* are consistently well supported across all databases, with support ratios close to 1.00. Standardized classification fields, including *severity* and *weaknesses*, also exhibit high coverage in most databases, typically above 0.90. In contrast, analytical fields show pronounced disparities across sources. Coverage of *impacts*, *remediation*, and *exploitation* varies widely between databases. Databases such as OSV, Mend.io, GitHub, GitLab, Cert, and Snyk provide comparatively rich support for these fields. For example, OSV and Mend.io achieve support ratios above 0.70 for *impacts* and *remediation*, and above 0.50 for *exploitation*. GitHub and GitLab also demonstrate substantial coverage, with support ratios for these fields generally ranging from 0.60 to 0.80.

By contrast, CVE and NVD focus on a narrower set of fields. Although both maintain near-complete coverage for fields such as *id*, *severity*, *affected*, and *references*, their support for analytical fields remains limited, often below 0.10. This reflects a design emphasis on identifier management and vulnerability classification rather than downstream analysis or mitigation guidance. Notably, while EDB and Huntr are widely recognized as specialized sources for exploitation, their coverage is largely limited to this information, with limited support for complementary contextual fields such as *impacts* and *remediation*. In contrast, databases including Cert, Mend.io, and Snyk exhibit more balanced field coverage, capturing exploitation information alongside richer contextual and mitigation-related data. These observations suggest that exploitation-related information is not exclusive to exploitation-focused databases, but is widely distributed across diverse vulnerability sources. Overall, the results show that centralized databases prioritize consistency and standardized



(a) Field Coverage in Dataset-B across Databases



(b) Field Coverage Improvements from Completion

Fig. 4. Field Coverage across Vulnerability Databases.

metadata, whereas ecosystem-oriented and security-vendor databases tend to provide richer contextual and actionable vulnerability information.

Figure 4b quantifies field coverage improvements introduced by field completion, revealing which types of vulnerability information are commonly omitted from structured fields but can be recovered from free-text descriptions. Across multiple databases, the improvements are concentrated in analytical fields, including *impacts*, *remediation*, and *exploitation*. The *impacts* field shows large gains in OSV (+0.73), Mend.io (+0.89), GitHub (+0.73), GitLab (+0.65), Snyk (+0.71), and Curl (+0.95), while *remediation* coverage increases markedly in Cert (+0.90), Ubuntu (+0.90), OSV (+0.70), and Mend.io (+0.76). Similarly, notable improvements in *exploitation* are observed in OSV (+0.53), Mend.io (+0.72), GitHub (+0.53), and Cert (+0.90). Together, these patterns indicate that impact descriptions, mitigation guidance, and exploitability details are frequently present in narrative text, but are omitted from structured fields. Field completion is therefore effective at recovering information that is available but structurally missing. In contrast, other fields

exhibit little or no improvement across databases, indicating information that is universally missing rather than merely unstructured, such as *acknowledgements*, *related*.

Finding 1: Over 29% of vulnerabilities appear only in a single data source outside NVD, and fields such as Impacts, Exploitation, and Remediation are largely missing across most databases; moreover, beyond EDB and Huntr, Cert, Mend.io, Snyk, and GitLab also contain exploitation-related information, which is mostly recorded in unstructured free-text descriptions. This underscores the importance of unified modeling to capture comprehensive vulnerability and field information.

4.3 RQ2: Inconsistency Analysis

Although multiple vulnerability databases may report the same vulnerability, the recorded information is often not identical across sources. To reveal the consistency of multi-source vulnerability data, this research question analyzes how the same vulnerability is represented across different databases. By comparing field-level values for shared vulnerabilities, we aim to identify common inconsistency patterns, assess their prevalence across fields, and understand the causes of divergence in vulnerability reporting.

To ensure comparability across databases, we first select fields that are supported by more than ten databases, including `id`, `description`, `published`, `affected`, `weaknesses`, `severity`, and `references`, which are also core fields appearing in all or most (≥4) database schemas. We then exclude fields that primarily serve identification or time-management purposes and focus on four fields: *Description*, *Affected*, *Weaknesses*, and *Severity*. Using Dataset-B, we evaluate the consistency of these fields across 19,562 shared vulnerabilities reported by multiple databases.

4.3.1 Consistency Across Databases. To evaluate the consistency of four fields across multi-source vulnerability data, we compare the four fields of shared vulnerabilities between each database pair. (1) For **Description**, we encode vulnerability descriptions using the Sentence-BERT model (all-MiniLM-L6-v2) and compute cosine similarity; a similarity score above 0.95 is considered consistent. (2) For **Affected**, we compare component information defined by the tuple *(product, vendor, package, ecosystem)* via exact matching, and assess version consistency by normalizing affected version ranges and explicit version lists into unified version sets using package management platforms. (3) For **Weaknesses**, we compare the assigned CWE label lists for exact match. (4) For **Severity**, we compare CVSS information reported under the same CVSS version (v2/v3/v4), including numerical scores, vector strings, and severity ranks (Low/Medium/High/Critical).

Based on the above field-specific consistency criteria, we compute the pairwise similarity between two databases as follows:

$$\mathrm{sim}(D_i, D_j) = \frac{|\{v \in V_{ij} \mid \forall f \in F, \mathrm{consistent}(D_i(v, f), D_j(v, f))\}|}{|V_{ij}|} \tag{1}$$

where $V_{ij} = V_i \cap V_j$ denotes the set of vulnerabilities reported by both D_i and D_j (i.e., the overlapping vulnerabilities), $F = \{Description, Affected, Weaknesses, Severity\}$, and a vulnerability v is counted as consistent only when all four fields are consistent between D_i and D_j. The function $\mathrm{consistent}(\cdot, \cdot)$ determines whether the values reported by two databases are consistency according to the field-specific criterion above. Therefore, $\mathrm{sim}(D_i, D_j)$ represents the proportion of shared vulnerabilities whose *Description*, *Affected*, *Weaknesses*, and *Severity* are all consistent between the two databases.

Additionally, beyond pairwise consistency analysis, we further quantify the overall degree of inconsistency for each vulnerability by measuring the number of distinct values reported across databases for the four fields. Specifically, for a given vulnerability and field, we compare the values reported by different databases, merge identical values, and count the number of distinct values remaining. For each field, value equivalence is determined using the same criteria defined above: semantic similarity (cosine > 0.95) for *Description*, exact matching on normalized component tuples and version

sets for *Affected*, exact matching on CWE label lists for *Weaknesses*, and exact matching on CVSS scores, vector strings, and severity ranks under the same CVSS version for *Severity*.

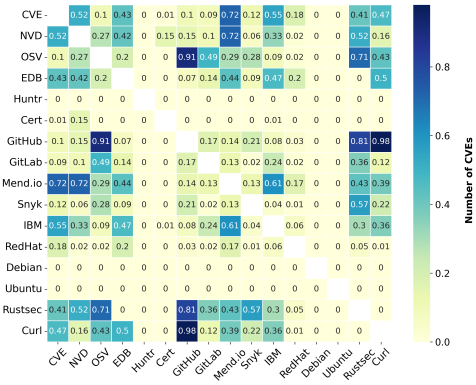


Fig. 5. Pairwise consistency across databases.

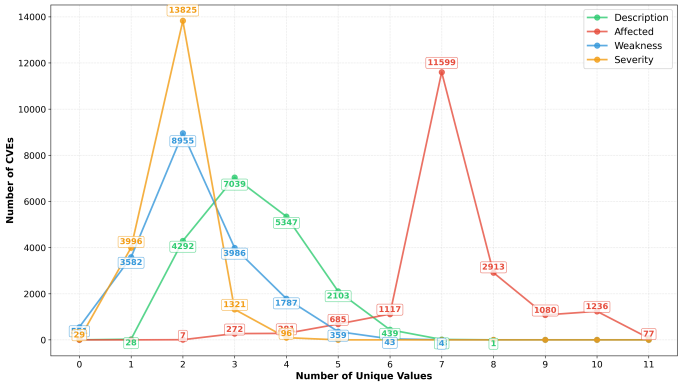


Fig. 6. Distribution of field-level value diversity across vulnerabilities.

Result. Figure 5 shows the pairwise similarity of 16 databases across four fields, measured as the ratio of vulnerabilities with consistent information to their shared vulnerabilities. The results indicate overall poor consistency across databases: over 70% of database pairs exhibit similarity below 0.3. Higher consistency is observed only among a few database pairs, mainly between ecosystem-driven databases such as GitHub–Curl (0.98), OSV–GitHub (0.91), and GitHub–RustSec (0.81), where substantial data reuse exists. In contrast, databases such as Debian, Ubuntu, and Huntr exhibit consistently low similarity with almost all other sources, with most pairwise values equal to or near 0.00. These databases either provide limited field coverage or focus on specific types of information; for example, Huntr emphasizes exploit records but lacks detailed information on affected versions. These results suggest that multi-source vulnerability database integration should explicitly account for field-level inconsistency patterns and source-specific biases, rather than assuming uniform reliability or consistency across databases.

While the pairwise analysis reveals substantial inconsistency across databases, Figure 6 further shows the distribution of the number of semantically distinct values for *Description*, *Affected*,*Weaknesses*, and *Severity* reported across all databases. The x-axis represents the number of distinct values observed for a vulnerability in a given field after applying the field-specific equivalence rules, while the y-axis represents the proportion of vulnerabilities with that count. The results reflect the degree of fragmentation of vulnerability information across sources. The *Affected* field exhibits the highest inconsistency: more than 60.2% of vulnerabilities have 7 distinct values, with the maximum reaching 11, indicating substantial discrepancies in how databases define affected components and version ranges. In contrast, Description values are mainly concentrated between 2~4, indicating the presence of differences while exhibiting a relatively stable variation pattern overall. By comparison, Weaknesses and Severity exhibit higher consistency. Weaknesses values mostly fall within 1~3 entries, while Severity shows the strongest consensus, with 71.8% of vulnerabilities having only two distinct values and very few exceeding 4, reflecting the standardized nature of CVSS scoring and the widespread adoption of NVD-provided severity ratings. Those results indicate that multi-source vulnerability data largely agree on whether a vulnerability exists, but diverge on who is affected, to what extent, and how severe the risk is.

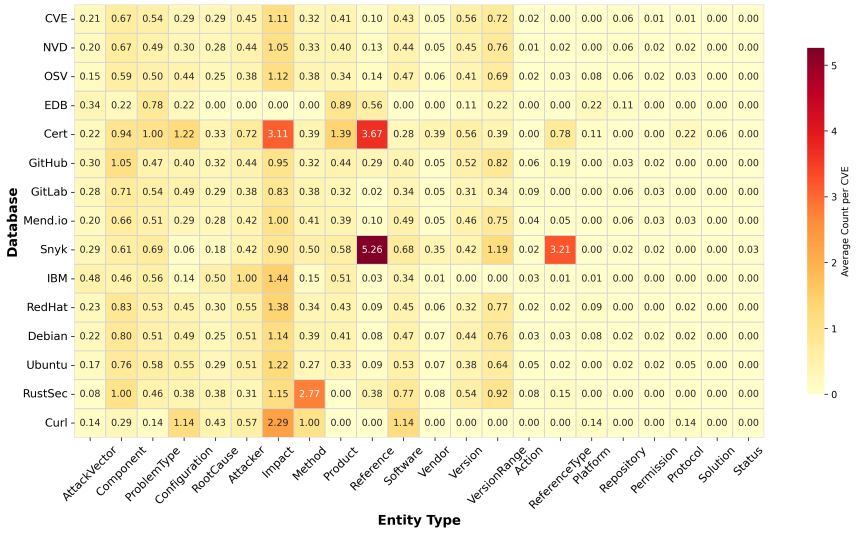


Fig. 7. The Distribution of Entity Counts across Databases by Entity Type.

Finding 2: More than half of the database pairs exhibit consistency below 0.3, indicating uncertainty in data selection and integration. In contrast, GitHub, Curl, OSV, GitLab, and RustSec frequently share identical field values. Where *Affected* exhibits the highest divergence, with 60.2% of vulnerabilities having 7 or more distinct values; *Description* values are stably concentrated in the range of 2–4, and *Weaknesses* and *Severity* are the most consistent fields, with over 40% of vulnerabilities exhibiting only 2 values.

4.3.2 Character of Inconsistencies in different fields. To investigate the root causes of field-level inconsistencies, we further analyze the inconsistency characteristics of each field by examining how its values diverge across databases.

(1) Composition of Description Field. To examine how vulnerability descriptions are structured, we utilize Neo4j Labs' LLM Graph Builder [47] to automatically extract entities and their semantic relationships from unstructured description texts and organize them into knowledge graphs. This allows us to compare vulnerability descriptions across databases based on their extracted entities rather than raw text. In total, 22 entity types are extracted, and Figure 7 shows the distribution of entity counts across databases by entity type.

From a commonality perspective, most databases consistently record *AttackVector*, *Component*, *ProblemType*, *RootCause*, and *Impact* as the core composition of vulnerability descriptions, with *Component* and *ProblemType* appearing at stable frequencies of approximately 0.5~0.8 per vulnerability and *Impact* occurring around or above 1.0 across major databases. Beyond this shared core, Snyk and Cert contain more entities per description (both exceeding 16; see AvgEntities(#) in Table 5, driven by frequent *Impact* (5.26 and 3.11), *Action* (3.21), and *Reference* (3.67) entities. In contrast, EDB adopts a concise description, supporting only 11 entity types with an average of 5.22 entities per vulnerability, focusing mainly on *Component*, *Version*, and *Product*. In addition, some databases introduce extra contextual information, such as *Repository* and *Permission* in GitHub, GitLab, and Snyk, and *Solution* in Cert, providing richer context for vulnerability scope and remediation paths.

Table 5. Subfield composition of multi-source vulnerability database

	CVE	NVD	OSV	EDB	Huntr	Cert	GitHub	GitLab	Mend.io	Snyk	IBM	RedHat	Debian	Ubuntu	Rustsec	Curl
Vulns(#)	19,320	19,287	16,257	1,163	1,161	330	11,207	11,600	18,822	11,064	19,168	10,555	10,618	10,623	411	155
Description(%)	100.0	100.0	100.0	100.0	-	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
EntityType(#)	20	21	22	11	-	18	20	19	21	20	16	20	22	21	16	12
AvgEntities(#)	6.77	6.54	6.50	5.22	-	17.94	7.66	6.15	6.71	16.00	6.25	7.36	7.10	7.11	9.69	8.86
Weaknesses(%)	32.0	88.7	-	-	5.9	-	54.7	59.2	91.4	55.0	-	45.7	-	-	-	80.0
AvgCWEs(#)	1.06	1.13	-	-	1.00	-	1.17	2.98	1.33	1.03	-	1.11	-	-	-	1.00
Severity(%)	29.6	99.4	99.2	-	99.7	-	100.0	99.7	100.0	99.5	100.0	99.5	-	100.0	66.4	100.0
CVSS v4.0(%)	-	-	-	-	-	-	0.6	-	-	-	-	-	-	-	-	-
CVSS v3.1(%)	21.1	68.1	73.4	-	-	-	68.2	72.2	99.5	80.9	-	48.3	-	-	-	-
CVSS v3.0(%)	9.2	31.9	25.8	-	-	-	22.4	21.1	-	21.8	93.4	38.4	-	-	-	-
CVSS v2.0(%)	0.4	81.1	-	-	-	-	-	76.2	81.0	-	6.6	-	-	-	-	-
Other Scheme(%)	-	-	-	-	-	-	100.0	-	-	-	100.0	99.1	-	100.0	66.4	100.0
One-vs-Many(%)	0.9	0.2	-	-	-	-	-	12.1	-	0.7	0.2	26.2	11.7	-	-	-
Many-vs-Many(%)	3.1	1.7	-	-	-	-	-	4.2	-	1.6	1.7	1.5	2.7	-	-	-
AvgComps(#)	1.35	3.41	1.59	1.00	1.00	1.11	1.22	1.06	1.12	1.00	2.66	16.58	1.36	10.22	1.00	1.84
Product(%)	100.0	100.0	-	100.0	-	-	-	-	100.0	-	100.0	100.0	-	-	-	-
Vendor(%)	91.0	100.0	-	100.0	100.0	100.0	-	-	100.0	-	100.0	100.0	-	-	-	100.0
Package(%)	5.0	-	100.0	-	100.0	-	100.0	100.0	-	100.0	-	100.0	100.0	100.0	100.0	100.0
Ecosystem(%)	5.0	-	100.0	-	-	-	100.0	100.0	-	100.0	-	78.0	100.0	100.0	100.0	42.0

Note: Vulns(#): total vulnerabilities per database. Description(%): coverage rate; EntityType(#): number of entity types; AvgEntities(#): average entities per vulnerability. Weaknesses(%): CWE coverage; AvgCWEs(#): mean CWEs per vulnerability. Severity(%): severity coverage; CVSS version (%): distribution of scoring standards; One-vs-Many(%): database deviates from consensus alone; Many-vs-Many(%): no clear majority exists. AvgComps(#): average affected components per vulnerability; Product/Vendor/Package/Ecosystem(%): component subfield coverage.

(2) Annotation of the Weaknesses Field. To examine the consistency of weakness annotation across databases, we compare how different databases assign CWE labels to the same vulnerabilities. Specifically, for each database, we first quantify the presence of the Weaknesses field and compute the average number of CWEs assigned per vulnerability. We then evaluate cross-database agreement by comparing the CWE label sets reported across all databases.

According to the calculation, only 12.5% of vulnerabilities achieve full agreement across databases, 42.4% exhibit high overlap with a single outlier, and 40.9% show substantial conflicts involving multiple databases. As shown in Table 5, most databases (e.g., CVE, NVD, GitHub, and Snyk) assign a single or very small number of CWEs per vulnerability (with averages close to 1), emphasizing a dominant weakness interpretation. In contrast, GitLab adopts a more multidimensional annotation strategy, assigning a larger number of CWEs on average (2.98), which often combine specific vulnerability types (e.g., *CWE-79: Cross-Site Scripting*) with higher-level OWASP category tags (e.g., *CWE-937: OWASP Top 10 2013* and *CWE-1035: OWASP Top 10 2017*).

These findings show that although vulnerability descriptions across databases exhibit high semantic similarity, consistency drops markedly at the level of weakness annotation. This suggests that the process of mapping rich, multi-dimensional description texts, containing multiple information such as attack vectors, affected components, root causes, and impacts, into a limited set of CWE categories constitutes a major source of divergence across databases.

(3) Partial Discrepancies of Severity Field. To analyze inconsistencies in the *Severity* field, we first examine the adoption of different CVSS versions, and then assess cross-database agreement on reported CVSS information, including numerical scores, vector strings, and severity ranks. We further conduct a fine-grained comparison of individual CVSS vector components to identify which aspects of severity assessment are most prone to divergence.

Table 6. Inconsistency Rates of CVSS Metrics by Version

CVSS Ver.	AV	AC	PR/AU	UI	S	C	I	A
v2.0	3.4%	26.9%	0.0%	–	–	25.6%	19.4%	23.8%
v3.0	17.3%	10.6%	12.7%	14.8%	5.2%	30.7%	29.7%	36.4%
v3.1	9.2%	16.5%	9.1%	12.1%	7.1%	16.2%	15.1%	19.3%

For CVSS version adoption for the *Severity* field, as shown in Table 5, OSV and Mend.io combine near-complete severity coverage (99.2% and 100.0%) and almost exclusively use CVSS v3.1 (73.4% and 99.5%, respectively), while GitHub also predominantly adopts v3.1 (68.2%) but supplements it with non-CVSS severity schemes (e.g., MODERATE, LOW) for a subset of records. In contrast, NVD distributes severity information across multiple CVSS versions, with v3.1 covering only 68.1% despite high severity coverage of 99.4%, and GitLab primarily relies on CVSS v2.0 (76.2%). In addition, Ubuntu, Rustsec and Curl only has custom severity labels (e.g., *MODERATE*, *LOW*) and lack rating criteria. Such missing severity information and asymmetric CVSS version adoption hinder direct comparison and aggregation of severity information across databases.

For vulnerabilities with Severity ratings from multiple databases, 41.4% achieve perfect agreement (all databases assign identical ratings), 33.6% show high agreement (only one database differs), while 27.1% exhibit significant conflicts (two or more databases disagree). Among them, IBM (26.2%), GitHub (12.1%), and RedHat (11.7%) show the highest inconsistency rates relative to other databases.

A finer-grained analysis of CVSS vector components reveals substantial variation across metrics. As shown in Table 6, attack-path-related metrics such as Attack Vector (AV) and Scope (S) exhibit relatively low inconsistency across CVSS v2.0, v3.0, and v3.1 (e.g., v3.1: AV 9.2%, S 7.1%), whereas impact-related metrics, such as Confidentiality, Integrity, and Availability, show much higher inconsistency (e.g., v3.0: Availability 36.4%, Confidentiality 30.7%).

This pattern is consistent with observations from the Description and Affected fields. For example, Cert and Snyk record more detailed Impact information in descriptions (averaging 3.50 and 1.47 Impact entities per vulnerability, respectively), while the Affected field exhibits the lowest overall consistency. Together, these results indicate systematic differences in how databases interpret and model the scope and severity of vulnerability impacts.

(4) Divergent Interpretation of the Affected Field. Differences in how *Impact*, *Method*, and *Repository* information are recorded in the Description field, together with inconsistencies in Severity assessment across the C/I/A impact metrics, collectively indicate systematic divergence in how different databases interpret a vulnerability's impact. This divergence culminates in the Affected field, which exhibits the lowest consistency among all fields, as shown in Figure 6. To further identify the sources of this inconsistency, we analyze the *Affected* field by examining how each vulnerability database records affected components, and by categorizing the factors that lead to version information discrepancies across databases.

In terms of component information, databases differ substantially in both the number of affected objects recorded and the granularity of their representation. As shown by the row 'AvgComps" (average components per vulnerability) in Table 5, CVE and NVD typically record only 3~4 upstream components centered on product/vendor identifiers, whereas OS vendor databases such as RedHat (16.58) and Ubuntu (10.22) expand affected components to finer-grained OS package–level entities. In contrast, Snyk and Mend.io, as SCA-driven platforms, organize affected objects around package/ecosystem identifiers but often retain only a single dependency component. As a result of these differences in component granularity and field emphasis, component records referring to the same vulnerability are frequently misaligned across databases and difficult to reconcile directly.

Version information further amplifies this inconsistency. Among 179,427 component records associated with 19,562 vulnerabilities, we identified 28,203 unique components, yet only 2,302 exhibit consistent version records across databases. While component-level inconsistencies largely stem from differences in granularity choice, discrepancies in version information reflect more substantive divergences in how databases define the scope of a vulnerability. A finer-grained analysis categorizes version inconsistencies into five types: *missing version information* (39.30%), *completely divergent version scopes* (33.67%), *partial omissions* (15.77%), *conflicts between range and list* (6.72%), and *unclear version boundaries* (4.06%). The dominance of missing version records highlights fundamental gaps in vulnerability lifecycle management, where tracking of affected versions is either omitted or deferred. And other consistent records pose the greatest barrier to multi-source integration, as they reflect incompatible interpretations of the vulnerability's scope.

Finding 3: Vulnerability descriptions share a stable structural core (e.g., *Component*, *ProblemType*), but differ in contextual richness, particularly in *Impact*-related content (AvgEntities ranges from 5.22 in EDB to 17.94 in Cert). Only 12.5% of vulnerabilities achieve full agreement in *Weaknesses*, where different annotation levels lead to 1–2.98 CWEs assigned on average across databases. For *Severity*, 27.1% exhibit substantial conflicts, with impact-related CVSS metrics showing high inconsistency (e.g., v3.0 Availability 36.4%). The greatest divergence occurs in *Affected*: databases differ in component granularity (AvgComps from 1.00 to 16.58), and only 2,302 out of 28,203 components maintain consistent version records, with 39.30% of discrepancies caused by missing version information. These divergent annotations necessitate semantic reconciliation beyond simple merging.

4.4 RQ3: Accuracy Analysis

Consistency analysis shows that differences in focus, annotation strategies, and data sources across vulnerability databases lead to varying degrees of inconsistency among four fields. However, inconsistency does not necessarily imply inaccuracy. To further address whether multi-source vulnerability database data is accurate, where inaccuracies arise, and how accuracy can be improved, we conduct a systematic accuracy analysis from four aspects: (1) field-level accuracy assessment for each database, (2) majority-consensus-based integration as a commonly used baseline, (3) reliability-ranking-based integration that prioritizes empirically more accurate sources, and (4) an LLM-based integration approach that jointly reasons over multi-source evidence to resolve conflicts.

Ground Truth Construction (Dataset-C). To evaluate the accuracy of multi-source vulnerability data, we construct a manually validated ground-truth dataset. First, We randomly sample 400 vulnerabilities, each corresponding to a distinct CVE ID, from Dataset-B, resulting in a total of 2,193 vulnerability records across sources. Following prior studies on validating the accuracy of weaknesses and severity information [60, 73], we then manually verify and annotate the accuracy of four fields in every vulnerability records: *Description*, *Weaknesses*, *Severity*, and *Affected*.

Specifically, (1) *Weaknesses* field was validated by examining the consistency between the assigned CWE identifiers and the root cause as described in the vulnerability description. (2) *Severity* field was assessed by examining the correctness of CVSS v3.1 vectors, including whether the assigned metrics and impact dimensions align with the vulnerability's behavior and impact descriptions. (3) For the *Affected* field, we validated affected components and version information using multiple reference sources, including official security advisories, developer discussions, issue records in source code repositories, component version histories, and, in particular, patch and exploit details. Vendor-provided documentation for the relevant software or hardware products was also used as an important validation reference. (4) Based on the verified information from these fields, we further evaluated the correctness of the *Description* entries on

Table 7. Accuracy of Different Fields Across Vulnerability Databases (Dataset-C)

Field	CVE	NVD	OSV	EDB	Huntr	GitHub	GitLab	Mend.io	Snyk	IBM	RustSec	Curl	Overall
Total (#)	400	400	183	43	31	234	183	157	271	186	58	47	2,193
Description (%)	82.00	82.00	81.97	62.79	100	85.90	85.25	81.53	91.88	98.39	100	100	86.00
Weaknesses (%)	92.75	92.25	-	-	93.55	88.03	88.52	79.62	74.17	-	-	97.87	84.25
Severity (%)	72.75	90.00	77.05	-	-	85.47	86.34	82.17	80.07	-	-	-	81.84
Affected (%)	60.06	77.00	65.03	11.63	0	81.20	53.01	59.24	65.31	14.52	100	100	62.06

three databases: NVD, GitHub, and Snyk, selected due to their distinct composition patterns identified in the previous analysis.

Three co-authors, each with four years of experience in vulnerability analysis and validation, cross-verified the reported components and versions against reference sources. When discrepancies arise, a fourth researcher with over ten years of experience will be involved to review and validate the evidence and decide the ground truth. Overall, the manual annotation process reaches a Cohen's Kappa coefficient of 0.86, which indicates reasonable annotation consistency and reliability of the dataset. And the final manual annotations constitute the Dataset-C.

4.4.1 Accuracy of different fields in each vulnerability database. Table 7 presents the results of manual validation. The row "Total (#)" reports the number of vulnerability records evaluated for each database in Dataset-C, and each subsequent row reports the accuracy (%) of each field across databases. Cert, Debian, and Ubuntu were excluded as Cert does not contain detailed component version information, while Debian and Ubuntu only involve operating system version fixes, lacking more component version records. Overall, the accuracy differences across the four fields are highly consistent with the field complexity and sources of ambiguity revealed in the previous consistency analysis.

The *Description* field demonstrates relatively high accuracy, with an overall accuracy of 86.00%, particularly in Huntr, RustSec, and Curl. These databases typically include fewer entity types in their descriptions, mainly focusing on problem types, affected products, and impact information, and these entities contain relatively fewer errors. In contrast, errors in the *Description* field are mainly concentrated in descriptive statements related to affected components and version ranges, which closely mirrors the error distribution observed in the *Affected* field. Notably, the *Affected* field exhibits substantially lower overall accuracy (62.06%) and the largest variation across databases, consistent with its lowest consistency identified in the previous analysis. As a result, databases that include more detailed component and version information in their descriptions tend to exhibit lower accuracy in this field, such as EDB (62.79%) and NVD (82.00%). These observations suggest that databases with higher accuracy may be report a narrower subset of entity types, whereas databases with lower accuracy may contain richer or more unique contextual information. Therefore, simply merging descriptions or selecting a single source, such as prioritizing majority consensus or high-accuracy databases, is insufficient; effective integration must strike a balance between correctness and comprehensive information coverage.

For the *Affected* field, Curl and RustSec achieve 100% accuracy in Dataset-C, largely due to their focus on single components and their reliance on fine-grained evidence such as code commits and patch information. In contrast, EDB, Huntr, and IBM exhibit very low accuracy, as their data emphasizes exploit descriptions, advisories, or proof-of-concept analysis rather than systematic tracking of upstream components and version ranges, resulting in imprecise annotations. Databases centered on open-source ecosystems generally perform better: GitHub achieves an accuracy of 81.20%, while OSV (65.03%), GitLab (53.01%), and Snyk (65.31%) show slightly lower accuracy due to duplicated or conflicting records, yet still remain above 50%.

Table 8. Accuracy Comparison of Integration Strategies on Dataset-C (%)

Field	Dataset-C	Majority Consensus	Reliability Ranking	LLM-Based
Weaknesses	84.25	91.94	**94.23**	92.08
Severity	81.84	63.91	78.63	**84.25**
Affected	62.06	60.30	76.59	**80.91**

Additionally, the *Weaknesses* and *Severity* fields exhibit relatively uniform performance across databases, with most databases achieving accuracies of around 80%. The overall accuracy is 84.25% for *Weaknesses* and 81.84% for *Severity*. This aligns with earlier findings: although annotation discrepancies exist across databases, the value spaces of these fields are relatively constrained (i.e., CWE enumerations and CVSS vectors), and errors primarily arise from annotation bias resulting from insufficient contextual information. For the *Weaknesses* field, inaccuracies are largely due to over-generalized CWE assignments such as CWE-119, CWE-20, CWE-284, and CWE-Other, which fail to capture precise vulnerability characteristics.

In the *Severity* field, errors are primarily associated with impact-related metrics (e.g., Confidentiality, Integrity, and Availability), reflecting uncertainty in impact assessment when contextual details are limited. More importantly, beyond annotation errors, a key source of inconsistency lies in contextual variation: the same vulnerability may be evaluated under different reporting perspectives across databases. For example, some databases (e.g., IBM) assess severity at the product, while others (e.g., Snyk) focus on specific components or modules. As a result, a single vulnerability may legitimately have multiple CVSS vectors and scores corresponding to different affected contexts. These observations further confirm that inaccuracies in description text are not caused by linguistic deficiencies, but instead stem from inconsistent interpretations of a vulnerability's affected scope and impact across databases.

Finding 4: While *Weaknesses* (84.25%) and *Severity* (81.84%) show higher accuracy across databases, and *Affected* remains lower (62.06%) with the largest variation (0–100%), the *Description* field exhibits accuracy differences driven by variation in entity composition across databases, with errors concentrated in component and version related content. Database strengths are field-specific: some sources perform well on certain fields (e.g., Curl and RustSec reach 100% on *Affected*), but not across others, indicating effective vulnerability data integration requires field-aware source selection strategies.

4.4.2 Accuracy Evaluation Based on Majority Consensus. **Setup.** In practical applications of multi-source vulnerability data, majority consensus is a commonly adopted strategy to resolve field-level conflicts. Accordingly, we use majority consensus as a baseline to evaluate its effectiveness in correcting erroneous field values across databases. Specifically, this strategy is applied to the three structured fields, *Weaknesses*, *Severity*, and *Affected*, while excluding the informative field *Description*, as it is a free text field with rich internal semantics and heterogeneous information composition for which simple value voting is not meaningful and may suppress complementary contextual information. For each vulnerability and each field, we construct a consensus-integrated result as follows. If the field is reported by only one database, or all reported values are identical across databases, the value is directly retained as the integrated result. When multiple databases report conflicting values, we select the value supported by the largest number of databases as the consensus result. If no strict majority exists, the value reported by NVD is selected as the result. The resulting value is treated as the *consensus-integrated result* for that vulnerability and field. Based on the ground truth established in Section 4.4, we evaluate the accuracy of the consensus-integrated dataset.

Result. Table 8 reports the accuracy results based on majority consensus, evaluated against the manually established ground truth in Dataset-C, as shown in the "Dataset-C" column.. Overall, the majority consensus is effective for the *Weaknesses* field, achieving an overall accuracy of 91.94%, which is substantially higher than the data in multi-source databases. This suggests that for fields with constrained value spaces and relatively standardized annotation schemes, such as vulnerability weakness categories, databases tend to converge on consistent and correct labels, allowing majority voting to correct deviations in individual sources.

However, the majority consensus performs noticeably worse for the *Severity* field. As shown in Table 8, the overall accuracy of *Severity* drops to 63.91% under majority consensus, compared with 81.84% in the original Dataset-C. This degradation indicates that severity assessment is highly context-dependent and cannot be reliably resolved through simple voting. Different databases may assign severity scores based on distinct threat models, exploit assumptions, or deployment contexts, and the majority consensus fails to account for these contextual differences, leading to the suppression of valid but minority assessments.

Similarly, for the *Affected* field, majority consensus also degrades accuracy. The overall accuracy decreases from 62.06% to 60.30% after consensus replacement. This demonstrates that for fields with high internal complexity, such as affected components, ecosystems, and version ranges, the majority does not imply correctness, and that extensive unverified data reuse exists across multi-source vulnerability data. Differences in component granularity, inconsistent version scopes, and missing information prevent effective convergence through simple voting. Moreover, complex fields with rich internal composition, such as *Description*, are also unsuitable for majority-consensus-based correction.

4.4.3 Accuracy Evaluation Based on Reliability Ranking. **Setup.** Based on the manual validation results, we find databases exhibit significant differences in accuracy, forming a distinguishable reliability ranking. For the *Weaknesses* field, the ranking is: Curl, Huntr, CVE, NVD, GitLab, GitHub, Mend.io, and Snyk. For the *Severity* field: NVD, GitLab, GitHub, Mend.io, Snyk, OSV, and CVE. For the *Affected* field: Curl, RustSec, GitHub, NVD, OSV, Snyk, Mend.io, GitLab, CVE, RedHat, IBM, EDB, and Huntr. Using these rankings, we construct a reliability-based accuracy evaluation strategy for the three structured fields above, excluding the informative field *Description*. For each vulnerability and each field, if the field is reported by only one database, or all reported values are identical, the value is directly retained as the integrated result. When conflicting values are reported by multiple databases, we select the value provided by the highest-ranked database among those that report this field. The selected value is treated as the *reliability-integrated result* and evaluated against the manually annotated ground truth.

Result. Table 8 reports the accuracy results of the reliability-based evaluation. Overall, reliability ranking improves *Weaknesses* and *Affected* compared to both the original data and the majority-consensus baseline: *Weaknesses* increases to 94.23% (vs. 84.25% original and 91.94% majority), and *Affected* rises to 76.59% (vs. 62.06% original and 60.30% majority). The gain on *Affected* is substantial (+14.53 pp), indicating that prioritizing empirically more reliable sources can partially suppress errors propagated by cross-database reuse and noisy component/version records. However, this improvement is constrained by the limited coverage of highly reliable sources; for example, Curl and RustSec, despite their high accuracy, account for less than 1% of the total vulnerabilities, limiting their global impact.

In contrast, reliability ranking does not consistently improve *Severity*. While it outperforms the majority-consensus result (78.63% vs. 63.91%), it remains below the original accuracy (78.63% vs. 81.84%, −3.21 pp). This suggests that for *Severity*, directly selecting a single source can discard correct alternative CVSS assessments that arise from different reporting contexts, and thus may reduce accuracy even when the chosen source is generally reliable.

Overall, reliability ranking is effective for fields where errors are strongly source-dependent and can be reduced by source prioritization (notably *Affected* and *Weaknesses*), but it is less suitable for fields where multiple values may be context-dependent or legitimately vary across databases (e.g., *Severity*).

4.4.4 Accuracy Evaluation Based on LLM. **Setup.** To explore the capability of large language models in improving the accuracy of multi-source vulnerability data, we introduce an LLM-based accuracy evaluation method. Specifically, for each vulnerability, we aggregate all available field values reported by different databases and provide them to a large language model (GPT-5) as a unified contextual input. The model jointly reasons over vulnerability descriptions, conflicting field values, and cross-field semantic cues to evaluate the semantic consistency and correctness of each candidate value, and integrates the correct value(s) into a unified result for each field. To improve reproducibility, we implement a two-stage prompting pipeline following the validation principles used in manually constructing Dataset-C. First, the LLM extracts supporting evidence from multi-source references, including vulnerability behavior, patch information, official clarifications, reproduction evidence, and component relationships. It then verifies conflicting field values against the extracted evidence, labels them as correct, incorrect, or uncertain with supporting sources and explanations, and integrates the validated values into a unified result for each field. The complete prompt templates are provided in Appendix C. The correctness of the integrated results is then evaluated by comparing them against the manually annotated ground truth described in Section 4.4.

Result. Table 8 presents the accuracy results of the LLM-based method. The LLM achieves the best performance among the three approaches on the *Affected* field, reaching an accuracy of 80.91%. Beyond the contextual cues provided by vulnerability descriptions, this improvement benefits from the LLM's reasoning capability, which enables it to integrate component, version, and semantic cues from multi-source descriptions to holistically resolve conflicts in affected information. However, when evidence is insufficient or descriptions are incomplete, the LLM tends to favor values with higher frequency or those it implicitly treats as "authoritative," which can introduce bias in judgments.

For the *Weaknesses* field, the LLM does not outperform the reliability-based strategy. Although it improves over the original data (84.25%) and the majority-consensus baseline (91.94%), its accuracy (92.08%) remains lower than that achieved by explicit reliability ranking (94.23%). This is because the *Weaknesses* field is highly standardized with a constrained value space (CWE labels), where empirically validated high-reliability sources are more effective than semantic inference.

In contrast, for the *Severity* field, the LLM outperforms both the original data and the reliability-based approach. Specifically, the LLM achieves an accuracy of 84.25%, exceeding the original accuracy (81.84%) as well as the reliability-based result (78.63%). This improvement suggests that, despite the standardized nature of CVSS metrics, contextual cues in vulnerability descriptions (e.g., impact narratives) can help the LLM correct misestimated severity assignments. However, the gain remains limited compared to that observed for the *Affected* field, indicating that semantic reasoning provides diminishing returns as field ambiguity decreases.

Cost and Reproducibility. Processing each vulnerability required an average of 290.1 seconds at a cost of $0.83 USD, reflecting the two-stage evidence extraction and field verification pipeline. To ensure reproducibility, we used a fixed temperature of 0 and provide the complete prompt templates in Appendix C.

Finding 5: Fields require different integration strategies rather than selecting a single source value. Informative fields, such as *Description*, which contain heterogeneous and complementary entities, require semantic consolidation rather than simple source prioritization or voting. Fields with constrained value spaces, such as *Weaknesses*, are best improved by reliability ranking (94.23%). Structurally complex and source-sensitive fields, such as *Affected*, benefit more from reliability ranking (76.59%) and LLM-based integration (80.91%) than from majority consensus. Context-dependent fields such as *Severity* are better suited to LLM-based reasoning (84.25%). Overall, no single strategy works for all fields, so the choice of the most appropriate strategy depends on the field characteristics.

5 Discussion

Our study reveals substantial gaps and discrepancies in multi-source vulnerability data by measuring completeness, consistency, and accuracy across 16 databases. These findings expose critical challenges in how such data is used and maintained. In this section, we discuss how the results support more informed decisions in database selection and cross-source integration, further explore the data quality of non-CVE vulnerabilities as an important and underexplored aspect of the vulnerability landscape, and provide a practical framework for multi-source vulnerability data management.

5.1 Toward Informed Source Selection in Vulnerability Data Usage

Completeness analysis (as discussed in RQ1) reveals a clear trade-off between coverage and field richness in multi-source vulnerability data, making it difficult for any single source to achieve both. Even CVE and NVD each miss over 30% of known vulnerabilities, while many non-CVE cases are concentrated in ecosystem- and enterprise-oriented databases such as IBM, GitHub, and Mend.io. Field completeness varies widely: general-purpose databases (e.g., CVE and NVD) focus on unified identifiers and standardized classifications with limited depth, whereas ecosystem-driven databases (e.g., OSV, GitHub, Snyk, and Mend.io) provide richer information on impacts, remediation, and exploitation. Although EDB and Huntr are known for exploit-focused data, Cert, Mend.io, and Snyk also contain substantial but often overlooked exploitation-related information. Consequently, database selection should be task-driven: use CVE/NVD as a coverage baseline, complement non-CVE and remediation data with ecosystem and enterprise sources, and incorporate exploit- or patch-oriented databases as needed to form a complementary data set.

Consistency analysis (as discussed in RQ2) reveals significant field-level discrepancies across databases. 47.2% of weakness labels, 66.5% of severity scores, and large portions of version and reference data are inconsistent, driven by unsynchronized update policies, inconsistent taxonomy use, and semantic variation. These conflicts show that blindly integrating sources can introduce noise or error. Therefore, source selection should prioritize consistency in the fields most critical to downstream applications. For example, the wide variation in recorded components, ranging from 1 to 16.58, highlights distinct database focuses. NVD links both OS-level and general-purpose components, while GitHub, OSV, and Mend.io capture cross-ecosystem open-source dependencies. Severity-based risk scoring should rely on databases like NVD, OSV, and Snyk, which exhibit more synchronized CVSS updates and broader field coverage. For weakness classification, NVD, Mend.io, and GitHub offer relatively aligned CWE taxonomies, whereas GitLab introduces higher variance and should be used with caution. These findings support task-specific, field-aware selection strategies grounded in empirical consistency data.

Finally, the observed accuracy variations among different sources (as discussed in RQ3) may offer deeper insights into the trade-offs between coverage and accuracy. Databases that integrate third-party data without structured verification, such as Snyk or IBM, may exhibit lower correctness rates, while sources anchored in code repositories, like GitHub and RustSec, achieve higher precision by leveraging direct code-level evidence, albeit with narrower coverage. Thus,

database selection is not only about inclusion but also about understanding the source's provenance and its implications for data trustworthiness.

Based on these findings, we recommend the following steps for task-driven source selection: ① Start with CVE and NVD as the coverage baseline, as they provide the broadest CVE coverage and standardized *Severity* and *Weaknesses* classifications. ② Complement with ecosystem-oriented sources (OSV, GitHub, GitLab, Mend.io, Snyk) when non-CVE vulnerabilities, remediation guidance, or exploitation context are required. ③ Incorporate exploit-focused databases (EDB, Huntr) when exploit intelligence is the primary need, while noting that Cert, Mend.io, and Snyk also provide valuable exploitation-related information. ④ Prioritize domain-specific databases (RedHat, Debian, Ubuntu, RustSec, Curl) when the target vulnerabilities fall within their defined scope, as they achieve high in-scope coverage with greater field accuracy.

5.2 Toward Better Integration in Vulnerability Data Maintenance

First, completeness analysis (RQ1) reveals a fundamental trade-off between coverage breadth and field richness across multi-source vulnerability data. While NVD (71.02%) and CVE (70.89%) maintain the highest coverage, they primarily rely on CVE records with limited field sets, emphasizing standardized classifications (Severity, Weaknesses) but lacking exploitation details or remediation guidance. In contrast, Non-CVE vulnerabilities (28.77%) are predominantly captured by enterprise security and open-source ecosystem sources (GitHub, Mend.io, OSV), which, despite narrower coverage with CVE vulnerabilities, provide richer fields—particularly fine-grained version ranges, remediation solutions, and ecosystem-specific metadata. However, all databases exhibit systematic gaps in critical information. Exploitation-related fields (Exploits, PoCs) remain sparse due to insufficient disclosure focus rather than technical limitations. High-value fields like vendor comments and acknowledgements show extremely low coverage, limiting vulnerability response traceability. Therefore, integrating multi-source information into comprehensive vulnerability intelligence systems and developing task-driven strategies for data source selection are essential for effective vulnerability management.

Consistency analysis (RQ2) indicates that inconsistencies in core fields reflect the lack of data source independence and insufficient standardization, posing clear requirements for integration strategies. Description field requires establishing standardized paradigms to clarify the boundaries between core elements (trigger conditions, impact scope, attack vectors) and supplementary information (exploitation details, remediation guidance). Affected field requires constructing schema-aware component mapping mechanisms to establish explicit bridges between product-vendor and package-ecosystem namespaces, and to normalize version range expressions to address the 70% missing/conflicting issues. Severity field requires establishing CVSS version conversion rules and semantic mappings for custom ratings to handle the 58.6% deviation cases. Weaknesses field requires constructing CWE parent-child mapping ontologies and normalizing annotations at different abstraction levels to improve the 12.5% consistency rate. Effective integration must shift from field-level conflict resolution to schema-aware semantic normalization: unifying namespaces, normalizing abstraction levels, establishing scoring conversion rules, etc.

Accuracy analysis (RQ3) reveals that cross-database consistency does not necessarily imply correctness, as agreement often reflects shared upstream sources rather than independent validation. Majority consensus improves constrained fields such as *Weaknesses* (91.94%) but degrades context-sensitive fields like *Severity* (63.91%) and fails to correct structural inconsistencies in *Affected* (60.30%). In contrast, reliability-based selection improves source-dependent fields (e.g., *Affected* to 76.59%), and LLM-based reasoning further enhances semantically complex fields (e.g., *Affected* 80.91%,

Table 9. Field consistency of the 47 confirmed cross-database non-CVE pairs.

Field	# Both Available	# One-sided	# Both Missing	# Consistent	# Inconsistent	Consistency (%)
Description	45	2	0	3	42	6.7
Weaknesses	2	19	26	2	0	100.0
Severity	8	39	0	6	2	75.0
Affected	34	12	1	5	29	14.7

Consistency is calculated only for pairs where both records provide the corresponding field.

Severity 84.25%). These results suggest that effective integration should prioritize empirically reliable and ecosystem-specific sources, avoid blind reliance on majority voting, and combine source reliability with semantic reasoning to mitigate propagation bias in multi-source vulnerability data.

5.3 Toward Data Quality Analysis of Non-CVE Vulnerability

Non-CVE records account for a substantial portion of several databases: OSV (59,632 non-CVE vs. 65,074 CVE), IBM (36,121 vs. 211,712), EDB (21,488 vs. 21,813), GitLab (3,549 vs. 19,533), GitHub (2,305 vs. 18,214), Huntr (2,152 vs. 1,670), Cert (793 vs. 3,595), and Debian (448 vs. 37,952). However, as described in Section 4.1, these records are not included in Dataset-B because they lack shared cross-database identifiers.

Consistency Analysis of Non-CVE Vulnerability. To evaluate the cross-database consistency of non-CVE vulnerabilities, we further analyzed the 47 cross-database pairs that were manually confirmed to describe the same vulnerability in Section 4.1. Following the same methodology as RQ2, we compared their *Description*, *Affected*, *Weaknesses*, and *Severity* fields using the same field-specific consistency criteria. Table 9 presents the field coverage and consistency of the 47 confirmed cross-database non-CVE pairs. These records exhibit incomplete field coverage. While *Description* and *Affected* are available in both databases for 45 and 34 pairs, respectively, *Weaknesses* and *Severity* are jointly available for only 2 and 8 pairs. This suggests that non-CVE vulnerabilities likely have incomplete and less standardized field information across databases. For pairs where both databases provide the corresponding field, *Description* and *Affected* show low consistency, with only 6.7% (3/45) and 14.7% (5/34) being consistent, respectively. In contrast, *Weaknesses* and *Severity* achieve consistency rates of 100.0% (2/2) and 75.0% (6/8), respectively, but these results are based on very few comparable pairs. Overall, the results show that confirmed non-CVE records also exhibit both limited field coverage and substantial cross-database inconsistencies.

Accuracy Analysis of Non-CVE Vulnerability. To evaluate the accuracy of non-CVE vulnerability, we randomly sampled 50 non-CVE vulnerabilities from databases that provide verifiable source evidence, including source code references, patch commits, or reproduction steps. Databases without accessible source references (IBM, Cert, Debian) were excluded as systematic field-level verification is infeasible without such evidence. Following the same annotation procedure used for Dataset-C, two co-authors independently verified the *Weaknesses*, *Severity*, and *Affected* fields for each sampled record against available source evidence, with disagreements resolved by a third co-author. The results, summarized in Table 10, reveal that non-CVE vulnerability records exhibit lower and more uneven field quality than CVE records. Despite the availability of source code and patch evidence in the sampled records, *Affected* achieves only 66% accuracy overall. In contrast, *Weaknesses* (40% correct, 18% missing) and *Severity* (34% correct, 42% missing) show substantially lower quality, with a large proportion of records either missing these fields entirely or assigning overly generalized labels (e.g., CWE-937, CWE-1035 in GitLab records). Compared to CVE records in Dataset-C, where *Weaknesses* and *Severity* achieve overall accuracies of 84.25% and 81.84% respectively, non-CVE records show a pronounced quality gap, reflecting the lack of multi-source corroboration and cross-database enrichment that

Table 10. Manual annotation results for 50 sampled non-CVE vulnerabilities. C = Correct; P = Partially correct; I = Incorrect; M = Missing.

Vulnerability ID	DB	W	S	A	Vulnerability ID	DB	W	S	A
GHSA-9gxv-x7rp-r2hc	GitHub	M	C	I	GMS-2020-538	GitLab	P	M	C
GHSA-mrrw-grhq-86gf	GitHub	M	I	I	GMS-2024-287	GitLab	I	M	C
GHSA-5gmm-6m36-r7jh	GitHub	C	C	C	GMS-2023-5978	GitLab	I	M	C
GHSA-446q-xxg5-3vhh	GitHub	M	C	C	GMS-2019-23	GitLab	I	M	C
GHSA-cv3v-7846-6pxm	GitHub	I	C	I	GMS-2023-6499	GitLab	I	M	C
GHSA-rjvj-673q-4hfw	GitHub	C	C	I	GMS-2021-62	GitLab	I	M	I
GHSA-cc65-xxvf-f7r9	GitHub	C	C	C	GMS-2020-378	GitLab	P	M	C
GHSA-w4x6-hh3x-wjrx	GitHub	M	I	C	GMS-2020-9	GitLab	I	M	C
GHSA-c35v-qwqg-87jc	GitHub	C	I	C	GMS-2021-175	GitLab	I	M	C
GHSA-hfmc-7525-mj55	GitHub	C	C	C	GMS-2023-2358	GitLab	P	M	C
GHSA-9wx7-jrvc-28mm	GitHub	C	C	I	GMS-2022-4708	GitLab	I	M	C
GHSA-qcff-ffx3-m25c	GitHub	C	I	C	GMS-2020-239	GitLab	P	M	C
GHSA-74hv-qjjq-h7g5	GitHub	C	I	C	GMS-2019-145	GitLab	I	M	I
GHSA-5j5w-g665-5m35	GitHub	C	I	C	GMS-2022-7821	GitLab	I	M	C
GHSA-22rr-f3p8-5gf8	GitHub	M	C	C	GMS-2020-774	GitLab	P	M	C
GHSA-36jr-mh4h-2g58	GitHub	I	C	C	GMS-2015-7	GitLab	I	M	C
GHSA-r32x-jhw5-g48p	GitHub	C	I	C	EDB-48876	EDB	M	M	C
GHSA-5v72-xg48-5rpm	GitHub	C	C	I	EDB-49957	EDB	M	M	I
GHSA-78m5-jpmf-ch7v	GitHub	C	I	C	OSV-2023-1117	OSV	M	M	C
GHSA-cpgr-wmr9-qxv4	GitHub	C	C	C	UVI-2021-1002015	OSV	M	M	C
GMS-2022-6760	GitLab	I	M	C	1-npm-safe-obj	Huntr	C	I	I
GMS-2020-405	GitLab	I	C	I	dedf1720-3e63-46d7-8163-56b86a21a982	Huntr	C	C	M
7a62b1bf-b959-4344-b78a-3b683a7c6d38	Huntr	C	C	I	49240e58-e804-4325-a752-39ec1b4d49d2	Huntr	C	I	I
84cd7753-3302-413a-aae6-15d7e70fdbf8	Huntr	I	C	C	7840cd32-af15-40cb-a148-7ef3dff4a0c2	Huntr	C	C	I
e3f4fe2c-dcdc-4273-9e54-aba500aebfda	Huntr	C	I	I	d8968782-2c85-450f-aa9d-49c61ddde7a1	Huntr	C	I	I

W = Weaknesses; S = Severity; A = Affected.

CVE records benefit from. These findings highlight the need for quality evaluation and benchmarking of non-CVE vulnerabilities in future work.

5.4 Toward a Practical Framework for Multi-source Vulnerability Integration

Our findings suggest that effective multi-source vulnerability management requires more than simply aggregating records from different databases. Based on the observations from RQ1–RQ3, we summarize a practical framework consisting of four stages: schema alignment, field completion, conflict resolution, and unified record generation.

First, map heterogeneous databases into a unified schema to establish consistent semantics across sources. Second, recover missing information hidden in free-text descriptions through field completion. Third, resolve conflicts among different sources in a field-specific manner rather than by applying a uniform strategy. Our results show that no single integration strategy is universally optimal: reliability-based selection achieves the highest accuracy for *Weaknesses* (94.23%), whereas LLM-based reasoning performs best for semantically complex fields such as *Severity* (84.25%) and *Affected* (80.91%). Finally, consolidate the validated values into unified vulnerability records.

This framework provides actionable guidance for building vulnerability intelligence systems and can be readily extended to incorporate additional databases. Due to source-specific distribution requirements, we do not redistribute the integrated data; instead, we publicly release the unified schema along with the mapping and completion code, enabling others to reproduce and extend the unified dataset from the original sources on our website [2]. However, because non-CVE vulnerabilities lack shared identifiers across databases, systematic quality evaluation and benchmarking of non-CVE records currently rely heavily on manual verification and remain an important direction for future work.

6 Threats to Validity

Several threats may affect the validity of our results.

Table 11. *Weaknesses* accuracy (%) per database across vulnerability types in Dataset-C.

CWE Type	# Vulnerabilities	CVE	NVD	Huntr	GitHub	GitLab	Mend.io	Snyk	Curl
CWE-79	35	93.6	93.6	–	93.1	90.7	90.0	85.7	–
CWE-125	32	92.9	92.9	–	100.0	100.0	66.7	–	80.0
CWE-22	27	95.2	100.0	–	94.8	92.5	90.0	38.5	–
CWE-400	25	94.5	94.5	–	88.2	76.5	70.6	56.2	–
CWE-674	23	82.2	88.2	100.0	81.8	75.0	76.5	72.7	100.0
CWE-276	21	93.8	93.8	–	77.8	88.9	84.6	88.9	–
CWE-502	19	86.7	93.3	–	92.9	100.0	83.3	60.0	–
CWE-78	18	85.7	92.9	–	90.0	100.0	80.0	80.0	–
CWE-287	18	92.9	92.9	–	88.9	83.3	87.5	88.9	–
CWE-94	17	91.7	75.0	–	83.3	75.0	66.7	50.0	–
Others	165	93.8	92.9	93.3	88.2	91.7	75.0	78.9	97.7

 # Vulnerabilities = number of sampled vulnerabilities in each CWE category.

(1) Database representativeness. The selected databases may not cover all vulnerability sources. To mitigate this, we followed two prior studies that summarized existing vulnerability databases from two complementary perspectives: academic research usage [36] and industrial SCA tool adoption [28]. We filtered for databases that are publicly available, widely adopted, and actively maintained, and further supplemented with 6 well-known specialized databases to ensure broad coverage across different types of vulnerability sources.

(2) Accuracy of field completion. LLM-based field completion may introduce extraction errors. To mitigate this, we restricted LLM to extract specific fields that are usually documented in unstructured text from existing vulnerability records, and evaluated its reliability on Dataset-A (1,600 manually annotated records across eight fields), achieving an overall accuracy of 93.5%. Moreover, the extracted fields are primarily *Impacts*, *Exploitation*, and *Remediation*, which are not involved in our main analyses. The remaining completed fields (*Affected* and *Weaknesses*) account for only a small proportion of the completed data, and most extraction failures result in missing values rather than incorrect ones. Consequently, the influence of the failures on our analysis is very limited.

(3) Dataset representativeness. Dataset-B is constructed for field completeness (RQ1) and inconsistency analysis (RQ2), requiring vulnerabilities to appear in multiple databases. We therefore included all vulnerabilities reported by at least eight databases (half of the 16 databases) and supplemented with stratified samples of lower-coverage vulnerabilities. This design improves the feasibility of cross-database comparison while preserving diversity across coverage levels. Dataset-C is randomly sampled from Dataset-B with 400 vulnerabilities spanning 1999–2025. While the sample size is limited by the cost of manual verification, the sampled vulnerabilities provide a diverse benchmark for accuracy analysis.

(4) Ground-truth quality. Manual annotation may introduce subjectivity and inconsistency. To mitigate this, all annotation tasks were performed by two co-authors independently, with disagreements resolved by a third co-author through consensus discussion. The inter-annotator agreement across all three annotation tasks indicates strong reliability: schema construction reached 89.0% (Cohen's κ = 0.88) across 82 items; Dataset-A annotation achieved 95.4% (Cohen's κ = 0.91) across 1,600 instances; and Dataset-C annotation achieved a Cohen's κ of 0.86.

(5) Stability of reliability rankings. The reliability rankings derived from Dataset-C are based on 400 sampled vulnerabilities. To assess their stability across vulnerability types, we grouped the sampled CVEs by their ground-truth CWE labels into 11 categories: the 10 most frequent CWE types in Dataset-C (CWE-79, CWE-125, CWE-22, CWE-400, CWE-674, CWE-276, CWE-502, CWE-78, CWE-287, and CWE-94), and a residual *Others* category covering the

remaining vulnerabilities. For each category, we computed the *Weaknesses* accuracy of each database and assessed ranking stability by measuring the proportion of pairwise database orderings consistent with the overall reliability ranking. As shown in Table 11, which presents the *Weaknesses* accuracy of each database across vulnerability types, the reliability ranking remains largely stable. The ordering CVE/NVD ≥ GitHub ≥ GitLab ≥ Mend.io ≥ Snyk is fully preserved for CWE-79, CWE-22, and CWE-400, and more than 70% of pairwise database orderings remain consistent with the overall ranking in 10 of the 11 categories. Although inversions occur in CWE categories with relatively limited samples (17–35 vulnerabilities), such as GitLab outperforming GitHub by 11.1% in CWE-276 (# Vulnerabilities = 21) and by 10.0% in CWE-78 (# Vulnerabilities = 18), the substantially larger *Others* category (# Vulnerabilities = 165) preserves 85.7% of the pairwise ordering, with only minor inversions within 4%. These results suggest that the observed inversions may partly reflect the limited per-category sample sizes, while the overall reliability ranking remains largely stable across vulnerability types.

7 Related Work

7.1 Data Consistency in Multi-Source Vulnerability Data

Previous studies have examined data consistency in vulnerability databases. Anwar et al. [1] highlighted significant inconsistencies and missing information in key fields such as vendor names, and product names within NVD. Sun et al. [62] reported severe issues in software name recognition and normalization in vulnerability reports, showing that the matching rate between NVD and other mainstream databases (e.g., CVE) is only 20.3%. Dong et al. [19] compared the structured information of vulnerability in NVD with CVE summaries and other external reports, revealing widespread inconsistencies in software versions. They found that only 59.82% of vulnerability reports fully matched the CVE summaries on version information. Sun et al. [63] further examined inconsistencies and missing information in seven key aspects of descriptions across NVD, IBM X-Force, and ExploitDB databases, including affected products, vulnerability types, and attack vectors. These studies primarily focused on the consistency of vulnerability descriptions.

Additionally, Roland et al. [12] reported inconsistencies in severity ratings across multiple sources. Zhang et al. [70, 73, 78] identified CVSS score inconsistencies in 10.44% of NVD entries through clustering analysis, where vulnerabilities with identical descriptions were assigned severity levels ranging from Medium to Critical. And Wu et al. [71] analyzed discrepancies in affected libraries and their ecosystems across four databases, emphasizing the severity of inconsistency issue. To assess vulnerability database quality, Kuehn et al. [32] proposed the OVANA framework, which evaluates NVD across multiple dimensions such as completeness and uniqueness. Jiang et al. [29] conducted a cross-database comparison of CVE, NVD, and vendor repositories, finding substantial inconsistencies among open-source vulnerability databases, which negatively impact cybersecurity prioritization and decision-making. Zhao et al. [80] investigated the effectiveness of SCA tools in detecting vulnerable third-party libraries, highlighting challenges in vulnerability identification. Zhang et al. [76] further explored vulnerable API detection to assess whether third-party library vulnerabilities affect downstream projects.

Although prior studies have revealed inconsistencies in vulnerability data, they remain limited in scope, often focusing on a single database, fewer fields, or surface-level comparisons, without a systematic multi-source measurement or cause analysis. As a result, they cannot offer a clear basis for source selection or actionable directions for database maintenance, particularly in improving data quality. In this work, we address this gap by analyzing 16 major vulnerability databases across completeness, consistency, and accuracy, characterizing the underlying causes of cross-database discrepancies, and providing a basis that can guide more informed source selection and effective maintenance practices.

7.2 Data Refinement in Vulnerability Databases

Beyond examining the quality and reliability of vulnerability databases, researchers have also focused on improving data completeness and structure [6, 53, 65, 69, 72, 74]. Early methods relied on rule-based techniques to extract vulnerability information from unstructured descriptions [7, 57], but they struggled with complex text structures. Recent advancements in NLP introduced BERT-based extraction models to improve named entity recognition (NER) accuracy [25]. Guo et al. [23] proposed an approach to detect and augment missing key aspects in vulnerability descriptions. Other approaches leveraged vertical search techniques [37] and integrated multiple data sources such as open-source vulnerability databases, issue tracking systems, and version control systems [53]. Dang et al. [16] explored cross-source information integration to enhance the completeness of vulnerability PoC reports. For patch propagation, Machiry et al. [39] introduced Spider, which leverages static analysis to rapidly disseminate security patches across related repositories, addressing incomplete vulnerability fix information in ecosystems. However, these works primarily focused on collecting and supplementing data from single sources, assuming the reliability of existing records, without systematically evaluating their accuracy.

7.3 Data Integration in Vulnerability Databases

Variations in data sources and collection methodologies result in heterogeneity and redundancy across vulnerability databases [31]. To address this, researchers have explored data fusion and cross-database integration [68, 81]. Some studies applied deep learning-based NER techniques to extract key information from multiple databases to supplement CVE data [24], while Li et al. [35] introduced the IFVD framework to map heterogeneous data by computing software name and version similarity. Yuan et al. [75] leveraged graph attention networks to predict entity relationships across different security databases. Others employed NLP models, such as BERT with a question-answering (QA) framework, to enhance cross-database correlation [64].

Beyond textual and metadata alignment, Chen et al. [30] further studied scalable vulnerability ID tracking and alignment in heterogeneous SAST environments, focusing on maintaining consistent identifiers across tools and versions. At the patch level, several studies sought to integrate vulnerability records with version control data by linking vulnerabilities to fixing commits [66, 79]. PatchFinder [34] systematically traces disclosed vulnerabilities to security patches at scale, while Nguyen et al. [83] identify silent vulnerability fixes from large commit histories. Zhang et al. [77] investigated the association of multiple security patches with a single vulnerability, highlighting the complexity of vulnerability-to-patch mapping. Despite these efforts, existing methods primarily focus on enriching CVE data by mapping information across different databases rather than identifying cross-database inconsistencies and inaccuracies, underscoring the need for a systematic assessment of multi-source vulnerability data.

8 Conclusion

We conduct a systematic measurement study of 16 vulnerability databases, focusing on completeness, consistency, and accuracy. By aligning heterogeneous data through a unified schema and LLM-assisted extraction, we enable large-scale comparison across sources. Our study reveals substantial coverage gaps, field-level inconsistencies, and correctness issues that vary by database type and provenance. These findings provide actionable insights for selecting trustworthy sources and designing more robust integration pipelines, offering a data-driven foundation for improving vulnerability data usage and maintenance.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant Nos. 62472309 and 62332005).

References

[1] Afsah Anwar, Ahmed Abusnaina, Songqing Chen, Frank Li, and David Mohaisen. 2021. Cleaning the NVD: Comprehensive quality assessment, improvements, and analyses. *IEEE Transactions on Dependable and Secure Computing* 19, 6 (2021), 4255–4269.

[2] Anonymous Author. 2025. Mind the Data: A Measurement Study on Public Multi-Source Vulnerability Databases. https://sites.google.com/view/vuln-harmonization/home. https://sites.google.com/view/vuln-harmonization/home

[3] Jessy Ayala, Yu-Jye Tung, and Joshua Garcia. 2025. Investigating Vulnerability Disclosures in Open-Source Software Using Bug Bounty Reports and Security Advisories. *arXiv preprint arXiv:2501.17748* (2025).

[4] Yechan Bae, Youngsuk Kim, Ammar Askar, Jungwon Lim, and Taesoo Kim. 2021. Rudra: finding memory safety bugs in rust at the ecosystem scale. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. 84–99.

[5] Lingfeng Bao, Xin Xia, Ahmed E Hassan, and Xiaohu Yang. 2022. V-SZZ: automatic identification of version ranges affected by CVE vulnerabilities. In *Proceedings of the 44th international conference on software engineering*. 2352–2364.

[6] Guru Bhandari, Amara Naseer, and Leon Moonen. 2021. CVEfixes: automated collection of vulnerabilities and their fixes from open-source software. In *Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering*. 30–39.

[7] Robert A Bridges, Corinne L Jones, Michael D Iannacone, Kelly M Testa, and John R Goodall. 2013. Automatic labeling for entity extraction in cyber security. *arXiv preprint arXiv:1308.4941* (2013).

[8] Canonical. 2024. Ubuntu Security Notices. https://ubuntu.com/security/notices. Accessed: 2025.

[9] Oscar Chaparro, Jing Lu, Fiorella Zampetti, Laura Moreno, Massimiliano Di Penta, Andrian Marcus, Gabriele Bavota, and Vincent Ng. 2017. Detecting missing information in bug descriptions. In *Proceedings of the 2017 11th joint meeting on foundations of software engineering*. 396–407.

[10] Common Vulnerabilities and Exposures (CVE). 2024. CVE Database. https://www.cve.org/

[11] Computer Emergency Response Team (CERT). 2024. CERT Vulnerability Database. https://www.kb.cert.org/

[12] Roland Croft, M Ali Babar, and Li Li. 2022. An investigation into inconsistency of software vulnerability severity across data sources. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 338–348.

[13] curl project. 2024. curl Vulnerability Disclosure Policy. https://curl.se/dev/vuln-disclosure.html. Accessed: 2025.

[14] Curl Security Advisories. 2024. Curl Vulnerability Database. https://curl.se/docs/security.html

[15] CVE Project. [n. d.]. CVE Record Format. https://github.com/CVEProject/cve-schema/blob/main/schema/CVE_Record_Format.json. Accessed: 2025-12-12.

[16] Wenjing Dang, Lyuye Zhang, Lingxiao Wang, Kaixuan Li, Mengyao Zhao, Xiaohong Li, and Sen Chen. 2026. Synchronizing Key Aspects: Enhancing Vulnerability PoC Reports from Multiple Sources. *IEEE Transactions on Information Forensics and Security* (2026).

[17] Debian Security Team. 2024. Debian Security Bug Tracker. https://security-team.debian.org/security_tracker.html. Accessed: 2025.

[18] Debian Security Tracker. 2024. Debian Vulnerability Database. https://security-tracker.debian.org/

[19] Ying Dong, Wenbo Guo, Yueqi Chen, Xinyu Xing, Yuqing Zhang, and Gang Wang. 2019. Towards the detection of inconsistencies in public security vulnerability reports. In *28th USENIX security symposium (USENIX Security 19)*. 869–885.

[20] Exploit Database (EDB). 2024. Exploit Database. https://www.exploit-db.com/

[21] GitHub Security Advisories. 2024. GitHub Advisory Database. https://github.com/advisories

[22] GitLab Advisory Database. 2024. GitLab Vulnerability Database. https://gitlab.com/security/advisories

[23] Hao Guo, Sen Chen, Zhenchang Xing, Xiaohong Li, Yude Bai, and Jiamou Sun. 2022. Detecting and augmenting missing key aspects in vulnerability descriptions. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 3 (2022), 1–27.

[24] Hao Guo, Zhenchang Xing, Sen Chen, Xiaohong Li, Yude Bai, and Hu Zhang. 2021. Key aspects augmentation of vulnerability description based on multiple security databases. In *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 1020–1025.

[25] Fateme Hashemi Chaleshtori and Indrakshi Ray. 2022. Automation of vulnerability information extraction using transformer-based language models. In *European Symposium on Research in Computer Security*. Springer, 645–665.

[26] Huntr by Protect AI. 2024. Huntr Database. https://huntr.dev/

[27] IBM X-Force Exchange. 2024. IBM X-Force Vulnerability Database. https://exchange.xforce.ibmcloud.com/

[28] Nasif Imtiaz, Seaver Thorn, and Laurie Williams. 2021. A comparative study of vulnerability reporting by software composition analysis tools. In *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 1–11.

[29] Yuning Jiang, Manfred Jeusfeld, and Jianguo Ding. 2021. Evaluating the data inconsistency of open-source vulnerability repositories. In *Proceedings of the 16th International Conference on Availability, Reliability and Security*. 1–10.

[30] James Johnson, Julian Thome, Lucas Charles, Hua Yan, and Jason Leasure. 2025. A Scalable, Effective and Simple Vulnerability Tracking Approach for Heterogeneous SAST Setups Based on Scope+Offset. In *2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 540–550. doi:10.1109/ICSE-SEIP66354.2025.00053

[31] Wooseok Kang, Byoungho Son, and Kihong Heo. 2022. TRACER: signature-based static analysis for detecting recurring vulnerabilities. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 1695–1708.

[32] Philipp Kuehn, Markus Bayer, Marc Wendelborn, and Christian Reuter. 2021. OVANA: An approach to analyze and improve the information quality of vulnerability databases. In *Proceedings of the 16th International Conference on Availability, Reliability and Security*. 1–11.

[33] Triet HM Le, Huaming Chen, and M Ali Babar. 2022. A survey on data-driven software vulnerability assessment and prioritization. *Comput. Surveys* 55, 5 (2022), 1–39.

[34] Kaixuan Li, Jian Zhang, Sen Chen, Han Liu, Yang Liu, and Yixiang Chen. 2024. PatchFinder: A Two-Phase Approach to Security Patch Tracing for Disclosed Vulnerabilities in Open-Source Software. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis* (Vienna, Austria) *(ISSTA 2024)*. Association for Computing Machinery, New York, NY, USA, 590–602. doi:10.1145/3650212.3680305

[35] Ruike Li, Shichong Tan, Chensi Wu, Xudong Cao, Haitao He, and Wenjie Wang. 2020. Ifvd: Design of intelligent fusion framework for vulnerability data based on text measures. In *2020 29th International conference on computer communications and networks (ICCCN)*. IEEE, 1–6.

[36] Xiaozhou Li, Sergio Moreschini, Zheying Zhang, Fabio Palomba, and Davide Taibi. 2023. The anatomy of a vulnerability database: A systematic mapping study. *Journal of Systems and Software* 201 (2023), 111679.

[37] Xiao Hong Li, Chang Hu, Zhi Yong Feng, Hong Wei Du, and Gang Gang Ding. 2012. An Approach to Obtain Software Security Vulnerabilities Based on Vertical Search. *Advanced Materials Research* 403 (2012), 3203–3206.

[38] Jiahuei Lin, Haoxiang Zhang, Bram Adams, and Ahmed E Hassan. 2023. Vulnerability management in linux distributions: An empirical study on debian and fedora. *Empirical Software Engineering* 28, 2 (2023), 47.

[39] Aravind Machiry, Nilo Redini, Eric Camellini, Christopher Kruegel, and Giovanni Vigna. 2020. Spider: Enabling fast patch propagation in related software repositories. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1562–1579.

[40] Mac Malone, Yicheng Wang, Kevin Snow, and Fabian Monrose. 2021. Applicable Micropatches and Where to Find Them: Finding and Applying New Security Hot Fixes to Old Software. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 394–405.

[41] Mend.io Security Database. 2024. Mend Vulnerability Database. https://www.mend.io/

[42] Dongliang Mu, Alejandro Cuevas, Limin Yang, Hang Hu, Xinyu Xing, Bing Mao, and Gang Wang. 2018. Understanding the reproducibility of crowd-reported security vulnerabilities. In *27th USENIX Security Symposium (USENIX Security 18)*. 919–936.

[43] Antonio Nappa, Richard Johnson, Leyla Bilge, Juan Caballero, and Tudor Dumitras. 2015. The attack of the clones: A study of the impact of shared code on vulnerability patching. In *2015 IEEE symposium on security and privacy*. IEEE, 692–708.

[44] National Institute of Standards and Technology. [n. d.]. NVD CVE API 2.0 Schema. https://csrc.nist.gov/schema/nvd/api/2.0/cve_api_json_2.0.schema. Accessed: 2025-12-12.

[45] National Vulnerability Database (NVD). 2024. NVD API - Accessing Vulnerability Data. https://nvd.nist.gov/developers Accessed: 2024-03-03.

[46] National Vulnerability Database (NVD). 2024. NVD Database. https://nvd.nist.gov/

[47] Neo4j Labs. 2023. LLM Graph Builder. https://github.com/neo4j-labs/llm-graph-builder. Accessed: 2025-01-18.

[48] Viet Hung Nguyen, Stanislav Dashevskyi, and Fabio Massacci. 2016. An automatic method for assessing the versions affected by a vulnerability. *Empirical Software Engineering* 21 (2016), 2268–2297.

[49] Viet Hung Nguyen and Fabio Massacci. 2013. The (un) reliability of nvd vulnerable versions data: An empirical experiment on google chrome vulnerabilities. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*. 493–498.

[50] Open Source Security Foundation. [n. d.]. Open Source Vulnerability Schema. https://github.com/ossf/osv-schema/blob/main/validation/schema.json. Accessed: 2025-12-12.

[51] Open Source Vulnerabilities (OSV). 2024. OSV Database. https://osv.dev/

[52] OpenAI. 2024. GPT-4o Mini: OpenAI's Lightweight Multimodal Model. https://openai.com/research/gpt-4o

[53] José D'Abruzzo Pereira, João Henggeler Antunes, and Marco Vieira. 2022. A Software Vulnerability Dataset of Large Open Source C/C++ Projects. In *2022 IEEE 27th Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE, 152–163.

[54] Red Hat Product Security. [n. d.]. Red Hat Security Data API. https://access.redhat.com/hydra/rest/securitydata. Accessed: 2025-12-12.

[55] Red Hat Security Advisories. 2024. Red Hat Vulnerability Database. https://access.redhat.com/security/security-updates/

[56] Luis Gustavo Araujo Rodriguez, Julia Selvatici Trazzi, Victor Fossaluza, Rodrigo Campiolo, and Daniel Macêdo Batista. 2018. Analysis of vulnerability disclosure delays from the national vulnerability database. In *Workshop de Segurança Cibernética em Dispositivos Conectados (WSCDC)*. SBC.

[57] Ernesto Rosario Russo, Andrea Di Sorbo, Corrado A Visaggio, and Gerardo Canfora. 2019. Summarizing vulnerabilities' descriptions to support experts during vulnerability assessment activities. *Journal of Systems and Software* 156 (2019), 84–99.

[58] RustSec. [n. d.]. RustSec Advisory Database Format. https://github.com/RustSec/advisory-db. Accessed: 2025-12-12.

[59] RustSec Advisory Database. 2024. RustSec Database. https://rustsec.org/

[60] Şevval Şimşek, Zhenpeng Shi, Howell Xia, David Sastre Medina, and David Starobinski. 2024. Poster: Analyzing and Correcting Inaccurate CVE-CWE Mappings in the National Vulnerability Database. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. 5042–5044.

[61] Snyk Vulnerability Database. 2024. Snyk Database. https://security.snyk.io/

[62] Hongyu Sun, Guoliang Ou, Ziqiu Zheng, Lei Liao, He Wang, and Yuqing Zhang. 2023. Inconsistent measurement and incorrect detection of software names in security vulnerability reports. *Computers & Security* 135 (2023), 103477.

[63] Jiamou Sun, Zhenchang Xing, Xin Xia, Qinghua Lu, Xiwei Xu, and Liming Zhu. 2023. Aspect-level information discrepancies across heterogeneous vulnerability reports: Severity, types and detection methods. *ACM Transactions on Software Engineering and Methodology* 33, 2 (2023), 1–38.

[64] Jiamou Sun, Zhenchang Xing, Xiwei Xu, Liming Zhu, and Qinghua Lu. 2022. Heterogeneous vulnerability report traceability recovery by vulnerability aspect matching. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 175–186.

[65] Xiaobing Sun, Zhenlei Ye, Lili Bo, Xiaoxue Wu, Ying Wei, Tao Zhang, and Bin Li. 2023. Automatic software vulnerability assessment by extracting vulnerability elements. *Journal of Systems and Software* 204 (2023), 111790.

[66] Zixuan Tan, Jiayuan Zhou, Xing Hu, Shengyi Pan, Kui Liu, and Xin Xia. 2024. Similar but Patched Code Considered Harmful – The Impact of Similar but Patched Code on Recurring Vulnerability Detection and How to Remove Them. arXiv:2412.20740 [cs.SE] https://arxiv.org/abs/2412.20740

[67] Ubuntu Security Notices. 2024. Ubuntu Vulnerability Database. https://ubuntu.com/security/notices

[68] Xinda Wang, Shu Wang, Pengbin Feng, Kun Sun, and Sushil Jajodia. 2021. Patchdb: A large-scale security patch dataset. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 149–160.

[69] Ying Wei, Lili Bo, Xiaobing Sun, Bin Li, Tao Zhang, and Chuanqi Tao. 2023. Automated event extraction of CVE descriptions. *Information and Software Technology* 158 (2023), 107178.

[70] Qiushi Wu, Yue Xiao, Xiaojing Liao, and Kangjie Lu. 2022. OS-Aware Vulnerability Prioritization via Differential Severity Analysis. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 395–412. https://www.usenix.org/conference/usenixsecurity22/presentation/wu-qiushi

[71] Susheng Wu, Wenyan Song, Kaifeng Huang, Bihuan Chen, and Xin Peng. 2024. Identifying affected libraries and their ecosystems for open source software vulnerabilities. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–12.

[72] Xiaoxue Wu, Wei Zheng, Xiang Chen, Fang Wang, and Dejun Mu. 2020. CVE-assisted large-scale security bug report dataset construction method. *Journal of Systems and Software* 160 (2020), 110456.

[73] Julia Wunder, Andreas Kurtz, Christian Eichenmüller, Freya Gassmann, and Zinaida Benenson. 2024. Shedding light on CVSS scoring inconsistencies: A user-centric study on evaluating widespread security vulnerabilities. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1102–1121.

[74] Congying Xu, Bihuan Chen, Chenhao Lu, Kaifeng Huang, Xin Peng, and Yang Liu. 2022. Tracking patches for open source software vulnerabilities. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 860–871.

[75] Liu Yuan, Yude Bai, Zhenchang Xing, Sen Chen, Xiaohong Li, and Zhidong Deng. 2021. Predicting entity relations across different security databases by using graph attention network. In *2021 IEEE 45th annual computers, software, and applications conference (COMPSAC)*. IEEE, 834–843.

[76] Fangyuan Zhang, Lingling Fan, Sen Chen, Miaoying Cai, Sihan Xu, and Lida Zhao. 2024. Does the vulnerability threaten our projects? Automated vulnerable API detection for third-party libraries. *IEEE Transactions on Software Engineering* 50, 11 (2024), 2906–2920.

[77] Fangyuan Zhang, Lyuye Zhang, Lingling Fan, Chengwei Liu, Yinan Li, Liang Huang, Yang Liu, Zheli Liu, and Sen Chen. 2026. One Is Not Enough: The Untold Story of Multiple Security Patches for One Vulnerability. In *Proceedings of the 41st IEEE/ACM International Conference on Automated Software Engineering*. 2011–2023.

[78] Siqi Zhang, Minjie Cai, Mengyuan Zhang, Lianying Zhao, and Xavier de Carné de Carnavalet. 2023. The flaw within: Identifying CVSS score discrepancies in the NVD. In *2023 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 185–192.

[79] Zheng Zhang, Hang Zhang, Zhiyun Qian, and Billy Lau. 2021. An Investigation of the Android Kernel Patch Ecosystem. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 3649–3666. https://www.usenix.org/conference/usenixsecurity21/presentation/zhang-zheng

[80] Lida Zhao, Sen Chen, Zhengzi Xu, Chengwei Liu, Lyuye Zhang, Jiahui Wu, Jun Sun, and Yang Liu. 2023. Software composition analysis for vulnerability detection: An empirical study on Java projects. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 960–972.

[81] Wei Zheng, Manqing Zhang, Hui Tang, Yuanfang Cai, Xiang Chen, Xiaoxue Wu, and Abubakar Omari Abdallah Semasaba. 2021. Automatically identifying bug reports with tactical vulnerabilities by deep feature learning. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 333–344.

[82] Xiaoye Zheng, Zhiyuan Wan, Yun Zhang, Rui Chang, and David Lo. 2023. A closer look at the security risks in the rust ecosystem. *ACM Transactions on Software Engineering and Methodology* 33, 2 (2023), 1–30.

[83] Jiayuan Zhou, Michael Pacheco, Zhiyuan Wan, Xin Xia, David Lo, Yuan Wang, and Ahmed E. Hassan. 2021. Finding A Needle in a Haystack: Automated Mining of Silent Vulnerability Fixes. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 705–716. doi:10.1109/ASE51524.2021.9678720

A Complete Field Mapping Across 16 Vulnerability Databases

Table 12 presents the complete field mapping from each of the 16 selected vulnerability databases to the 22 unified schema fields, showing the original field name(s) in each database that correspond to each unified field.

Table 12. Complete field mapping from each of the 16 vulnerability databases to the 22 unified schema fields. Each cell shows the source field name(s) that map to the unified field; "—" indicates the field is absent. Shaded cells indicate the field is present.

Unified Field	CVE	NVD	OSV	RedHat	RustSec	EDB	Snyk	GitLab	IBM	Mend	Debian	CERT	Curl	Huntr	Ubuntu	GitHub
id	cveMetadata.cveId	id	id	Bugzilla.id	advisory.id	codes	cve_id	identifier	stdcode	vul_id	cve_id	cveids	id	cve	id	id
source	source	sourceIdentifier	—	—	—	author	—	—	—	—	—	—	—	authorFound	—	—
aliases	—	—	aliases	—	aliases	—	snyk id	identifiers	stdcode	—	—	cveids	aliases	—	ubuntu id	aliases
related	—	—	related	—	related	—	—	—	—	—	—	—	—	—	related/upstream	—
published	datePublic	published	published	PublicDate	date	date_published	publicationTime	pubdate	reported	date	—	original_release_date	published	time	published	published
modified	dateUpdated	lastModified	modified	—	—	date_updated	modificationTime	date	—	—	—	last_revised	modified	—	modified	modified
withdrawn	—	—	withdrawn	—	withdrawn	—	—	—	—	—	—	—	—	—	withdrawn	withdrawn
vuln_status	cveMetadata.state	vulnStatus	—	—	—	—	—	—	—	—	—	—	—	status	—	—
title	title	—	summary	—	—	description	title	title	title	—	—	name	summary	title	—	summary
description	descriptions	descriptions	details	details	—	description	description	description	description	desc	description	clean_desc	details	pocmd	details	details
weaknesses	problemTypes	weaknesses	—	CWE	categories	—	CWE	cwe_ids	consequences	cwe_id	—	—	CWE	cwe	—	cwe_ids
severity	metrics	metrics	severity	ThreatSeverity, CVSS, CVSS3	cvss	—	cvssDetails	cvss_v2 / cvss_v3	cvss, risk_level	cvss	—	—	severity	severity	severity	severity
impacts	impacts	evaluatorImpact	—	—	—	—	—	—	—	—	—	impact	—	—	—	—
exploitation	exploits	—	—	—	—	id, file (exploit)	epssDetails	—	—	—	—	—	—	pocmd, source	—	—
affected	affected	configurations	affected	AffectedRelease, PackageState	affected, package, versions	platform, description	packageName, vulnerableVersions	package_slug, affected_range	platforms_affected	—	package, releases	—	affected	repo, versions	affected	affected
cpe_configuration	cpeApplicability	configurations	—	—	—	—	—	—	—	—	—	—	—	—	—	—
remediation	solutions, workarounds	evaluatorSolution	—	Mitigation, UpstreamFix	versions.patched	—	remediation	solution, fixed_versions	remedy	fix	—	resolution, workarounds	—	patch	—	—
references	references	references	References	references	references	source_url, application_url	references	urls	references	related_resources	—	references	issue, www	source	references	references
vendor_comments	—	vendorComments, evaluatorComment	—	Statement	—	—	—	—	—	—	—	vendor_info	—	—	—	—
acknowledgements	credits	—	credits	Acknowledgement	—	—	—	credit	—	—	—	—	credits	authorFound, authorFixed	—	—
tags	tags	cveTags	db_specific	—	informational, keywords	tags, type, platform	—	—	report_confidence	—	—	—	award	visibility, status	—	—
copyrights	—	—	—	—	license	—	—	—	—	—	—	—	—	—	—	—

B Unified Vulnerability Schema Field Definitions

Table 13 presents the definitions of all 22 unified fields in the proposed vulnerability schema. For each field, the table specifies its name, sub-fields, data type, and definition.

Table 13. Definitions of the 22 unified vulnerability schema fields. *Field*: top-level field name; *Sub-field*: sub-fields of the preceding top-level field (shaded rows); *Type*: data type; *Definition*: semantic scope and aggregation rationale.

Field	Sub-field	Type	Definition
`id`		string	A unique, stable identifier for the vulnerability record, using the most widely-recognized public identifier available. Priority: CVE > GHSA > RUSTSEC > OSV > source-scoped ID. Pattern: `^[A-Za-z0-9._:-]+$`.
`source`		object	Metadata describing the origin of this record: which platform provided it, who reported it, and what document type it represents.
	`.provider`	string	Primary database name (e.g., `CVE`, `NVD`, `OSV`, `RedHat`, `RustSec`).
	`.reporter`	string	Individual or organization credited as disclosing the vulnerability.
	`.category`	string	Document type: `advisory` | `cve-record` | `database-entry` | `bug-bounty` | `exploit` | `scanner` | `news` | `info`.
`aliases`		string[]	Alternative identifiers for the same vulnerability across different databases, used for cross-database deduplication and linking.
`related`		string[]	Identifiers of related but non-equivalent vulnerabilities—e.g., a vulnerability in a C library wrapped by a Rust crate.
`published`		string	Date/time the vulnerability was first publicly disclosed through an official channel. Format: ISO 8601 UTC (`YYYY-MM-DDThh:mm:ssZ`). Date-only sources normalized by appending `T00:00:00Z`.
`modified`		string	Date/time this vulnerability record was last updated in its original data source. Null if source provides no modification timestamp.
`withdrawn`		string	Date/time this record was officially withdrawn or retracted by its source. Null if the record remains active.
`vuln_status`		string	Current lifecycle state of the record. Values: `published` (default) | `rejected` | `analyzing` | `modified` | `unknown`.
`title`		string	Short one-sentence summary of the vulnerability (≤256 chars). Generated from description when not provided by the source.
`description`		object[]	Multi-language natural-language explanations of the vulnerability including its cause, affected components, impact, and exploitation conditions.
	`.lang`	string	BCP 47 language code (e.g., `en`, `zh-CN`).
	`.value`	string	Description text, ≤4096 characters.
	`.format`	string	Content format: `text` | `markdown` | `html`.
	`.media`	array	Supporting media objects `{type, base64, value}`.
`weaknesses`		object[]	Standardized weakness classifications associated with the vulnerability (CWE IDs, OWASP categories, RustSec-specific categories).

(continued on next page)

(continued from previous page)

Field	Sub-field	Type	Definition
	.id	string	Weakness identifier, e.g., CWE-79, crypto-failure, A01:2021.
	.taxonomy	string	Classification system: CWE | RustSecCategory | OWASP | Text | Other.
	.name	string	Human-readable weakness title.
	.source	string	Provider that asserted this classification.
severity		object[]	Severity scores and ratings from different providers, unifying CVSS metrics (v2/v3/v4), vendor-specific qualitative ratings (e.g., RedHat ThreatSeverity), and distribution-specific ratings (e.g., Ubuntu).
	.source	string	Provider: CVE | NVD | OSV | RedHat | RustSec | …
	.scheme	string	Scoring scheme: CVSS:2.0 | CVSS:3.1 | CVSS:4.0 | RedHat:ThreatSeverity | Ubuntu | Other.
	.score	number	Numeric base score 0.0–10.0; null for non-numeric schemes.
	.vector	string	Full CVSS vector string, e.g., CVSS:3.1/AV:N/AC:L/. . .
	.rating	string	Severity band: NONE | LOW | MEDIUM | HIGH | CRITICAL or vendor-specific label.
impacts		object[]	Natural-language descriptions of what an attacker can achieve if the vulnerability is exploited, optionally linked to CAPEC attack patterns.
	.source	string	Provider of this impact description.
	.text	string	Impact description, ≤4096 characters.
	.capec_id	string	CAPEC identifier, e.g., CAPEC-100.
exploitation		object	Information about real-world exploitation: available exploit code, proof-of-concept (PoC) links, and EPSS exploitability probability scores.
	.exploits[]	array	Exploit descriptions: {source, lang, text, links[]}.
	.epss[]	array	EPSS scores: {source, cve_id, score, percentile, last_updated}. Score range: 0.0–1.0.
affected		object[]	Unified list of affected software products, packages, and platforms. Consolidates CVE's vendor/product model, NVD's CPE-based configurations, OSV's ecosystem/package model, RedHat's RPM-centric AffectedRelease/PackageState, and RustSec's crate-level model.
	.vendor	string	Organization, project, or maintainer name.
	.product	string	Human-readable product or software name.
	.ecosystem	string	Package ecosystem: npm | pypi | maven | crates.io | rpm | upstream | …
	.package	string	Package identifier within the ecosystem.
	.status	string	Overall status: affected | unaffected | unknown.
	.versions[]	string[]	Explicit list of affected version numbers.
	.version_range[]	object[]	Version ranges: {scheme, introduced, fixed, last_affected, limit, status}. Scheme: semver | rpm | deb | git | ecosystem | custom.
	.os / .arch	string[]	OS names and CPU architectures constraining applicability.
	.cpe / .purl	string	CPE 2.3 string or Package URL for machine-readable identification.
	.functions[]	string[]	Affected function/method paths (fine-grained, RustSec-style).

(continued on next page)

(continued from previous page)

Field	Sub-field	Type	Definition
cpe_configurations		object[]	CPE-based logical expression trees using AND/OR/NOT operators for precise product and version applicability matching, preserving full NVD/CNA configuration semantics.
	.operator	string	Top-level logic: AND | OR.
	.negate	boolean	Whether to invert the entire tree's logic.
	.nodes[].cpeMatch[]	object[]	CPE match entries: {vulnerable, criteria (CPE 2.3), versionStart/EndIncluding/Excluding}.
remediation		object	All information about mitigating or resolving the vulnerability: formal fixes, temporary workarounds, and specific versions containing the fix.
	.solutions[]	object[]	Formal fix descriptions: {lang, value, supportingMedia}.
	.workarounds[]	object[]	Temporary mitigation descriptions: {lang, value, supportingMedia}.
	.fixed_versions[]	string[]	Version specifiers containing the fix, e.g., 1.2.4, >=1.2.4, RUSTSEC:>=1.2.0.
references		object[]	Structured list of external resources: advisories, vendor security bulletins, patches, PoC write-ups, and discussion threads.
	.url	string	Absolute URL to the resource (required).
	.name	string	Human-readable title for the link.
	.tags[]	string[]	Resource type: exploit | patch | vendor-advisory | advisory | misc.
vendor_comments		object[]	Statements, assessments, or comments provided by vendors, security teams, or NVD evaluators about the vulnerability.
	.source	string	Organization or individual providing the comment.
	.comment	string	The statement or assessment text (required).
	.time	string	ISO 8601 timestamp when the comment was made.
acknowledgements		object[]	Credits to individuals, organizations, or tools that contributed to discovering, reporting, analyzing, or coordinating the vulnerability response.
	.name	string	Person, organization, or tool name (required).
	.type	string	Role: finder | reporter | analyst | coordinator | remediation_developer | sponsor | tool | other.
	.contact[]	string[]	Email addresses, URLs, or social handles.
tags		object[]	Keywords and classification labels for filtering and categorization, e.g., disputed, informational, or domain keywords.
	.description	string	Tag text, 1–256 chars, recommended lowercase or kebab-case.
	.source	string	Platform that applied this tag.
copyrights		object[]	Copyright statements, license identifiers, and attribution requirements associated with the vulnerability data, e.g., MITRE attribution requirements or RustSec CC-BY-4.0 license.
	.source	string	Provider of this copyright notice.
	.notice	string	Formal copyright notice text.
	.license	string	SPDX identifier, e.g., CC0-1.0, CC-BY-4.0.

(continued on next page)

(continued from previous page)

Field	Sub-field	Type	Definition
	.license_url	string	URL to full license text.

C LLM Prompt Templates for Accuracy Evaluation

This appendix presents the two-stage prompt templates used in the LLM-based accuracy evaluation described in Section 4.4.4. Both prompts were designed to mirror the annotation guidelines used in Dataset-C, ensuring that the LLM reasoning process is grounded in the same validation criteria applied during manual annotation.

Stage 1: Reference Information Extraction

The first-stage prompt instructs the LLM to extract all validation-supporting facts from multi-source reference data for a given CVE, organizing them into seven structured dimensions before any field-level verification is performed.

Stage 1 Prompt: Reference Extraction

```
You are tasked to perform: Reference Information Extraction and Preparation for {cve_id}.
Goal: Extract all validation-supporting facts from the multi-source reference data to support later reasoning. Each
↪  piece of extracted data should be accurate, well-categorized, and conflict-resolved.

Extraction Dimensions:
1. Vulnerability Behavior
   - Technical description: trigger conditions, affected code paths, attack methods (e.g., injection points).
   - Consequences (e.g., crash, XSS, RCE, DoS).
   - Origin: commit/version introducing the vulnerability; exploit descriptions or reproduction steps (if any).
2. Patch and Fix Data
   - All patches, commits, pull requests:
     * Distinguish patch vs. introducing commit.
     * Link each patch to: version fixed, affected component, ecosystem (e.g., PyPI, Debian, Ubuntu), and distribution
     ↪  branch if applicable (e.g., Ubuntu 20.04).
   - Track changelogs, release notes, or commit diffs.
   - Note packaging system differences (e.g., RubyGems vs. Debian package).
3. Clarification & Official Statements
   - Official vendor or maintainer communication: mailing lists, GitHub issues, security trackers.
   - Explicit affected or not-affected version ranges.
   - Developer confirmations, test outcomes, or conclusions.
4. Reproduction and Testing Data
   - PoC references or reproduction attempts: which version was tested, outcome (vulnerable/not vulnerable), links to
   ↪  PoC or reproduction environments.
5. Component & Ecosystem Relationships
   - Component aliases, naming variations.
   - Ecosystem-specific identifiers (e.g., Python module vs. Debian package).
   - Upstream vs. downstream relationships; library within framework; packaging system distinctions.
6. Other Evaluable Information
   - Any other information that might assist in validation but does not fall into the above categories.
7. Conflict Resolution
   - If data is contradictory, apply this priority:
     (1) Newer timestamp wins (entries with no timestamp are not prioritized).
     (2) Reproduction/Test > Patch > Official Clarification > Community Discussion > Database Entry.
     (3) Source credibility: Commit repo > Official advisory > Core mailing lists/forums > Third-party databases.
```

```
Output Format (Structured Sections):
  - Vulnerability Behavior Description
  - Patch & Fix Collection
  - Clarification & Communication Collection
  - Reproduction & Testing Collection
  - Component/Ecosystem Relationships
  - Other Evaluable Information
  - Conflict Points and Resolution Summary

Now process the following reference data:
{reference_data}
```

Stage 2: Field Verification

The second-stage prompt instructs the LLM to verify each conflicting field value against the extracted evidence from Stage 1, following a structured Chain-of-Thought (CoT) reasoning process. Each entry is assigned one of three labels: `1` (correct), `0` (incorrect), or `-1` (uncertain).

Stage 2 Prompt: Field Verification

```
You are a vulnerability verification assistant. Validate whether the field-level information for CVE ID {cve_id} is
↪  correct (tag: 1), incorrect (tag: 0), or uncertain (tag: -1). Follow the field-specific rules below and reason
↪  step-by-step before assigning a tag.

--- Multi-source Reference Data ---
{support_data}
--- Conflict Data to Verify ---
{conflict_data}

=== Field-Specific Verification Rules ===
[Weakness]
Verify whether the assigned CWE identifier(s) correctly reflect the root cause in the vulnerability description.
Steps:
1. Extract the root cause (e.g., injection, memory corruption, improper access control).
2. Map it to the expected CWE category.
Tag rules:
- tag: 1  CWE(s) precisely match the root cause (e.g., CWE-79 for XSS, CWE-787 for OOB write).
- tag: 0  CWE is over-generalized (e.g., CWE-119, CWE-20, CWE-284) when a specific CWE is clearly applicable, or CWE
↪  contradicts the described root cause.
- tag:-1  Description lacks sufficient detail, or multiple CWEs are plausibly applicable without clear basis.
Conflict resolution: authoritative CWE mapping (NVD, CVE) > multi-source consensus > single-source assignment.

---
[Severity]
Verify whether the CVSS vector and score are consistent with the described vulnerability behavior. Only compare
↪  entries under the same CVSS version (v2/v3/v4).
Steps: Check each metric against the description:
- AV: Network if remote; Local/Physical if local access.
- AC: Low if straightforward; High if special conditions.
- PR: None/Low/High per authentication requirement.
- UI: None or Required per victim-action requirement.
- S:  Changed if impact extends beyond vulnerable component.
```

```
- C/I/A: None/Low/High per described consequence.
Tag rules:
- tag: 1  Vector consistent with described exploitation conditions and impact.
- tag: 0  One or more metrics clearly contradict the description (e.g., AV:L for a remote issue; C:H when only partial
↪  disclosure is described).
- tag:-1  Description is ambiguous about exploitation conditions or impact; or different reporting contexts (e.g.,
↪  product-level vs. component-level) legitimately justify different vectors.
Note: context-driven CVSS differences across databases are not errors unless the vector misrepresents the description.
Conflict resolution: NVD or vendor-authoritative CVSS > multi-source consensus > single-source assignment.

---
[Affected]
Verify whether the reported components and version ranges are correct and complete.
Component check:
- tag: 0  Component unrelated to the described vulnerability.
- tag:-1  Component identity is ambiguous or naming varies.
Version range check:
- tag: 1  Range directly supported by patch evidence (authoritative_patch, patch_diff) or official advisory.
- tag: 0  if ANY of:
  * Range overlaps a known fixed version.
  * Range includes versions confirmed as not affected.
  * Range omits confirmed affected versions.
  * Fixed version cannot be matched to ecosystem patch data.
- tag:-1  if ANY of:
  * Versions potentially affected but unconfirmed.
  * Fixed version has no ecosystem-specific patch evidence.
  * Branch information (e.g., Debian 8 vs. 9) is unclear.
Cross-ecosystem rule: never assume patches from one ecosystem (e.g., Ubuntu) apply to another (e.g., Debian, RubyGems)
↪  without explicit evidence. If uncertain: tag:-1.
Conflict resolution: authoritative_patch + patch_diff > official advisory > package_changelog > ecosystem_database >
↪  third-party summaries.

=== Judging Principle ===
- tag: 1  information is complete, directly supported by evidence, and passes all applicable rules above.
- tag: 0  information clearly contradicts evidence or violates a rule above.
- tag:-1  evidence is insufficient, ambiguous, or correctness cannot be determined.
=== Output Format (JSON array) ===
[
  {
    "source": ["..."],
    "url": ["..."],
    "content": "...",
    "field": "Weakness | Severity | Affected",
    "tag": 1 / 0 / -1,
    "reason": "..."
  },
  ...
]
```

Received 20 December 2025