INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2026, VOL. 72, NO. 1, PP. 1–7
Manuscript received November 11, 2025; revised February, 2026. doi: 10.24425/ijet.2026.157886

Severity is not exploitability: A simple taxonomy and context score for better CVE triage

Grzegorz Siewruk, and Tomasz Bondaruk

***Abstract*—Modern security teams face a constant backlog of vulnerabilities and limited engineering time for patching. In practice, organizations turn this into a triage problem: deciding which findings to remediate first, which to monitor, and which to postpone, often encoding these choices into SLAs, dashboards, and automated patching workflows driven by scanner output. Today this prioritization is usually based on Common Vulnerability Scoring System (CVSS) base severity, even though severity does not equal exploitability. This paper presents a compact, single-tag taxonomy for exploitability preconditions (exposure, environment, configuration, authentication, cryptography, and related factors) and a transparent context score that estimates how easy a vulnerability is to exploit in a given deployment. We enriched a dataset of 2,426 Common Vulnerabilities and Exposures (CVE) with constraint annotations and compared the context score against CVSS severity and the Exploit Prediction Scoring System (EPSS). The score shows weak association with both signals, indicating it captures complementary information about situational ease rather than impact or ecosystem pressure. Grouped by severity, notable shares of medium- and even low-severity findings emerge as easy to exploit under common configurations. In a telecom self-care platform case study (100 findings), reordering by the context score surfaced 28 straightforward fixes across severities, reducing immediate exposure and consolidating root causes that a severity-only plan would postpone. We conclude that combining EPSS with the proposed context score yields a more effective, auditable triage process for applied informatics settings.**

***Keywords*—devsecops; vulnerability analysis; vulnerability management; cybersecurity**

I. Introduction

SOFTWARE teams face a relentless backlog of CVEs. Triage still leans on severity scores, yet *severity is not exploitability*: some "High" issues never materialize as attacks, while select "Medium" items are exploited quickly [1], [2]. Public catalogs such as National Vulnerability Database (NVD) provide identifiers, Common Weakness Enumeration (CWE) classes, and CVSS vectors, but rarely encode the *concrete conditions* that must hold in real deployments for exploitation to succeed [3], [4].

We addressed this gap with a curated dataset of CVEs *enriched* by explicit **exploitability preconditions**: human- and machine-readable constraints about environment, configuration, privilege boundaries, and code paths (e.g., network reachability, default settings, feature flags, required library/build options) [5], [6]. These factors often determine whether a theoretical weakness becomes a practical incident, yet they are rarely modeled alongside CVSS or used systematically in triage [7], [8].

In most cases the prioritization of the addressed vulnerabilities typically chooses between *coarse* signals (severity) and *late* signals such as: Known Exploited Vulnerabilities (KEV) or public Proof of Concepts (PoCs). [9]. Encoding preconditions on CVEs provides an *ex-ante* view: when required conditions do not hold, risk drops immediately; when they do, the item warrants elevation even before threat-intel arrives [10].

In order to alter this approach and propose a more comprehensive and holistic approach, we gathered CVEs from public sources and documented our enrichment pipeline. Later, based on gathered data we introduced a taxonomy to categorize preconditions to them, and finally proposed an *exploitability score* derived from that taxonomy. The score maps to CVE/CWE/CVSS fields and, where useful, operational frameworks such as Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK) [4], [5]. We analyzed distribution of preconditions and compared prioritization using *Severity-only* vs. *Severity+Preconditions* and our score.

When conducting the analysis and introducing the taxonomy, we focused our approach around the following research questions (RQ1–RQ3). They center the study on practical usability and improvement in triage of the most important vulnerabilities:

- **Prevalence (RQ1).** How often do specific preconditions occur across CVEs/CWEs, and how do they co-occur?
- **Predictive value (RQ2).** Do preconditions (and our score) improve prediction of downstream exploitation (e.g., Severity, EPSS) over severity-only baselines [2], [9]?
- **Practical triage (RQ3).** How do preconditions change prioritization in realistic enterprise settings, and with what impact on time-to-mitigation?

This paper makes the following contributions to vulnerability management and treatment research:

- **Dataset & enrichment.** A reproducible pipeline and curated dataset of CVEs annotated with exploitability preconditions [3].

G. Siewruk is with Warsaw University of Technology; IDEAS Research Institute, Warsaw, Poland (e-mail: grzegorz.siewruk@pw.edu.pl).
T. Bondaruk is with IDEAS Research Institute Warsaw, Poland.

- **Taxonomy.** A concise categorization scheme for vulnerability preconditions, aligned with CVE/CWE/CVSS and optionally ATT&CK.
- **Exploitability score.** A calculation method based on the taxonomy that outperforms severity-only CVSS in triage-oriented evaluations.

In our investigation, we have focused on software vulnerabilities recorded in CVE/NVD and described by CWE. External indicators (KEV, public PoCs) are treated as *outcomes*, used to evaluate preconditions and the proposed score [3], [9].

The paper has the following organization. Section II provides background and related work on vulnerability prioritization and exploitability modeling. Section III details data sources and enrichment, and Section IV presents the taxonomy and scoring method. Section V reports the main results, analyses, and comparisons. Section VI gives a case study. Section VII discusses implications and limitations, and Section VIII concludes our research.

II. Background & Related Work

The *Common Vulnerabilities and Exposures (CVE)* system provides standardized identifiers (e.g., `CVE-2024-12345`) for publicly known cybersecurity vulnerabilities, so that different tools and databases can refer to the same issue in a consistent way. It is maintained by the *CVE Program*, a community-driven initiative that coordinates the creation and publication of these identifiers rather than a software "program" in the executable sense. Within this initiative, the CVE Program assigns a stable identifier and a structured JavaScript Object Notation (JSON 5.x) record to each vulnerability. A CVE Record contains a CVE Numbering Authority (CNA) authored container and may include enrichments from *Authorized Data Publishers (ADPs)* in the optional `adp` container array; the CVE Program may also publish its own ADP entry (the "CVE Program Container") [11]. The *National Vulnerability Database (NVD)* consumes CVE records, adds standardized mappings (e.g., *CWE*), applicability via *Common Platform Enumeration (CPE)*, curated references and tags, and exposes versioned JSON 2.0 Application Programming Interfaces (API) for reproducible analysis [12]. In this study we treat CVE/NVD as the canonical spine (identifiers, descriptions, CWE/CPE mappings, references) and then add what they intentionally do not capture: explicit *exploitability preconditions*—the conditions that must hold in a real target for an attack to work.

On top of this identifier and metadata layer, modern practice relies on severity and likelihood signals. CVSS v4.0 is a transparent *severity* standard: Base metrics are environment-agnostic, while the Environmental and Threat metric groups let operators adjust scores using local context and current threat intelligence [13]. This still does not answer the question "how likely is this to be exploited soon?" Signals closer to likelihood include the *Exploit Prediction Scoring System (EPSS)*, which estimates the short-term probability of exploitation [14], and CISA's KEV catalog, which is effectively a ground-truth list of vulnerabilities known to be exploited in the wild [15]. In short, CVSS tells us *how bad* a vulnerability could be, EPSS and KEV indicate *how likely* exploitation is, and our preconditions aim to describe *whether it is actually reachable here* in a given system.

A complementary ecosystem has emerged around software inventories and advisories. An *SBOM* (Software Bill of Materials) is a structured, machine-readable inventory of the components and dependencies that make up a software product; it allows producers and consumers to understand which libraries, packages, and versions are present in a given build. In the SBOM/advisory world, *CISA Vulnerability Exploitability eXchange (VEX)* documents already express "affected" or "not affected" status for a given product and CVE, together with justifications. *Common Security Advisory Framework (CSAF)* v2.0 defines a formal VEX profile, and CycloneDX offers a practical VEX representation used alongside SBOMs [16], [17]. CISA has also published minimum content requirements to keep VEX useful and consistent across producers [18]. Our contribution aligns with this ecosystem: we make the *conditions* explicit at CVE granularity (e.g., feature use, network exposure, authentication mode) so that product teams can prove "not affected" when those conditions do not hold—and verify "affected" when they do.

Empirical studies show that high severity does not automatically translate into frequent exploitation; the relationship is weak and strongly context dependent [19]. At the feasibility layer, prerequisite-based *attack graphs* (e.g., MulVAL and its follow-ups) model "if these preconditions hold, the next step is possible," which mirrors our per-CVE precondition view—but we drive this idea down into code and configuration checks suitable for continuous integration (CI) pipelines [20]. To prioritize effectively, organizations therefore need to *join* standardized severity information, real-world threat signals, and concrete feasibility constraints. That is the gap our precondition-enriched CVE dataset is designed to close.

In this study we anchor our data model on CVE/NVD, which provide the identifiers and baseline facts we rely on, including descriptions, CWE/CPE mappings, and curated references [11], [12]. On top of this canonical layer we use CVSS as the standardized severity framework that supplies transparent and reproducible scoring [13]. We then align with EPSS and KEV to incorporate threat-aware signals that capture how vulnerabilities are exploited in practice, rather than only how severe they could be in theory [14], [15]. Finally, we extend this stack with *validated preconditions* that connect these existing signals to the concrete reality of a given system, so that we can answer the practical question: "Can this be exploited *here*, given our code, configuration, and deployment?"

III. Dataset Sources

Our research is based on a large, curated dataset of public software vulnerabilities that augments standard (published) records with a structured set of exploitability preconditions. While public vulnerability feeds are rich in identifiers and severity vectors, they rarely capture the concrete circumstances that make exploitation feasible in practice. This section explains how we assembled, validated and enriched the dataset. It

explains and describes the data sources we drew from. Later, it describes the precondition schema we applied using our semi-automated annotation workflow. Finally, it covers the resulting statistics and their meaning to the triage.

A. Data Sources and Curation

To balance realism with coverage, we combined a repository-driven sample with an NVD-driven sample.

a) Repository sample.: We analyzed *200* actively maintained public GitHub repositories across three ecosystems: JavaScript (npm), Java (Maven) and Python (PyPI). Findings were produced with widely used tools and anchored in real codebases. In practice, the analysis processes (SCA, SAST and IaC scanning) were implemented using the following concrete tools:

- **cdxgen + OWASP Dependency-Track (SCA) [21]:** we used *cdxgen*[1] to generate CycloneDX SBOMs for each repository, and imported these SBOMs into *OWASP Dependency-Track* [2] to surface known issues in third-party components.
- **Bearer (SAST) [22]:** we used *Bearer*[3] as the static analysis tool to flag insecure code patterns in first-party application code.
- **KICS (IaC scanning) [23]:** we used *Keeping Infrastructure as Code Secure (KICS)*[4] to detect misconfigurations in infrastructure-as-code and deployment artifacts.

This bottom-up pass yielded roughly 1,000 CVE-backed findings, each tied to a specific software context.

b) NVD sample.: For breadth, we ingested CVEs from the NVD published between January 2022 and October 2025. For each record we retained the CVE ID, mapped CWEs, CVSS v2/v3 vectors and base scores, the official description, and references. Using CPE metadata and keyword filtering, we excluded entries focused on hardware, operating systems or closed-source enterprise appliances to keep the corpus aligned with the ecosystems above. This top-down pass contributed approximately 1,500 CVEs.

After de-duplication, the combined corpus contains **2,426** unique CVEs and forms the basis for our precondition analysis.

B. Precondition Taxonomy and Schema

Our central contribution is a taxonomy of exploitability preconditions derived from a deep dive into advisories, technical write-ups, and fixing commits. We identified recurring patterns and grouped them into *12* categories (Table I) that cover all possibilities, from network, application configuration to code itself.

To make this concrete, these are the kinds of conditions our taxonomy captures. For instance, a precondition could be related to network access, such as requiring a vulnerable service to be reachable from the public internet. It might involve the runtime environment, like needing a specific library version or a particular Just-in-Time (JIT) compiler feature to be active. Other common examples relate to application state, such as a vulnerability that only triggers if a default password has been left unchanged or if a specific feature flag is toggled on. Finally, many preconditions involve privilege, where an exploit is only possible if it can cross a security boundary, like escaping a sandbox.

The dataset was stored in a simple schema: Entity-Relationship (ER) JSON, linking each CVE to one or more normalized precondition instances. Each instance is cross-referenced to CVSS v4 metrics (e.g., *AV*, *PR*, *UI*, *S*) and, where applicable, to MITRE ATT&CK technique IDs to situate the precondition within a broader attack narrative.

C. Annotation Protocol and Quality Control

Annotating thousands of CVEs at high fidelity does not scale with purely manual effort, so we adopted a semi-automated pipeline with expert oversight. The process consists of two phases.

In the Phase 1, for each of the 2,426 CVEs, we prompted Google's Gemini Large Language Model (LLM) as a *research assistant* to propose candidate preconditions from the NVD description and referenced advisories. The goal was acceleration, not authority: the model surfaces hypotheses for human review.

The following Phase 2 consists of two independent security practitioners reviewing each candidate list. Reviewers first **validated** statements against the source material (advisories, vendor notes, PoCs) and discarded speculative items. Later they **assigned** each validated item to exactly one of the 12 categories in Table I. Agreement was monitored using Cohen's κ statistical measure and disagreements were resolved through adjudication by a senior reviewer following pre-registered decision rules.

D. Final Dataset Composition

The final dataset comprises **2,426** CVEs annotated with **26,924** validated precondition instances (on average 11.1 per CVE), underscoring how multi-factor the notion of "exploitability" is compared to a single severity score.

Table II summarizes key statistics. Severity, taken from CVSS base scores, reflects a typical backlog skew toward `MEDIUM` (39.4%) and `HIGH` (34.9%). Precondition categories are markedly imbalanced: `ISO` (Isolation/Privilege) and `DEPS` (Dependency/Runtime) dominate with a combined 72.7%. In other words, exploitability in modern stacks is most often gated by *how* dependencies are assembled and *where* security boundaries hold (or leak).

Figure 1 shows an example of a single vulnerability record (simplified for readability). The record summarizes `CVE-2013-7285`, rated `CRITICAL` with an EPSS of 0.15054, and enumerates eight defined constraints (preconditions) that must be satisfied for the vulnerability to be exploitable.

NVD content is used under its published license; we redistribute only derived annotations and identifiers, not full

[1]https://github.com/CycloneDX/cdxgen
[2]https://dependencytrack.org/
[3]https://github.com/Bearer/bearer
[4]https://github.com/Checkmarx/kics

name	base_severity	epss	constraint_value	nstraint
CVE-2013-7285	CRITICAL	0.1505400000	Xstream API version is up to 1.4.6 or 1.4.10.	DEPS
CVE-2013-7285	CRITICAL	0.1505400000	Xstream security framework has not been initialized.	ISO
CVE-2013-7285	CRITICAL	0.1505400000	Application unmarshals XML or any supported format (e.g., JSON).	INPUT
CVE-2013-7285	CRITICAL	0.1505400000	Input stream is manipulated by a remote attacker.	ISO
CVE-2013-7285	CRITICAL	0.1505400000	Xstream API version is up to 1.4.6 or 1.4.10.	DEPS
CVE-2013-7285	CRITICAL	0.1505400000	Xstream security framework has not been initialized.	ISO
CVE-2013-7285	CRITICAL	0.1505400000	Application unmarshals XML or any supported format (e.g., JSON).	INPUT
CVE-2013-7285	CRITICAL	0.1505400000	Input stream is manipulated by a remote attacker.	ISO

Fig. 1. Single record from database describing CVE-2013-7285

upstream texts. Where public PoCs are referenced, we redact details that would materially lower the bar for exploitation.

IV. Method

Our goal is to move from "this CVE is HIGH" to "this CVE is *actually easy or hard to exploit in our setup*". To do that, we started from the recorded *constraints* for each vulnerability (for example: endpoint must be public, debug mode must be enabled, default password must be present) and mapped each constraint to exactly one of twelve single-tag categories: EXP, ENV, CFG, AUTH, DATA, DEPS, EXT, INPUT, TIME, ISO, LOG, CRYPTO - see Table I. Intuitively, these tags tell us *what needs to be true* for exploitation to work: exposure, environment, configuration, authentication, cryptography, and so on. We then turned these tags into a single *context score* per vulnerability, and, when available, combine it with an external prior such as EPSS.

a) Constraint tagging (deterministic).: Each constraint row receives exactly one tag. We used a fixed precedence to break ties when a constraint could fit several categories:

$$\texttt{ISO} > \texttt{AUTH} > \texttt{EXP} > \texttt{CFG} > \texttt{ENV} > \texttt{INPUT},$$

$$\texttt{CRYPTO} > \texttt{DEPS} > \texttt{EXT} > \texttt{DATA} > \texttt{TIME} > \texttt{LOG}.$$

For example, if a constraint mentions both authentication and exposure, it is tagged as `AUTH`, because identity requirements are usually more decisive than simple reachability. We also prefer developer–controlled levers (`CFG`, `DEPS`, `INPUT`) over ops–controlled ones (`ENV`, `EXT`) in a situation when both appear. Finally, we favor property-like keys (e.g., `spring.foo=bar`) over generic phrases. The result of the tagging process is a single reproducible label (category) for each constraint. This tagging simplifies analysis of the vulnerability and enables quick understanding of the condition types to exploit it. For example: "Vulnerability ABC is easily exploitable, because it has several preconditions related to network exposure (EXP) and weak cryptography (CRYPTO)".

b) Tag polarity and weights.: Once every constraint was tagged, we asked a simple question: does this tag *make life easier for an attacker*, or does it *stand in the way*? We called the first type *enablers* (they increase ease of exploitation) and the second type *gates* (they make exploitation rarer or more difficult). We assigned small weights $w(t)$ to each tag:

- Enablers: `EXP`, `CFG`, `ENV`, `INPUT` (common formats), `LOG`, `CRYPTO`, `EXT`.
- Gates: `AUTH`, `TIME`, `ISO`, `DATA`, `DEPS`, `INPUT` (exotic encodings).

We then applied a few intuitive overrides. For example, an `AUTH` constraint that says "anonymous/guest/no auth" clearly makes the attacker's life easier, so we treated it as an enabler. A `DATA` constraint that requires default credentials or a hard-coded API key is also an enabler. For `INPUT`, common formats such as JSON behave as enablers (it is easy for an attacker to send JSON), whereas niche formats (UTF-7, SOAP/XML-only) behave more like gates. We kept these weights small, interpretable, and easy to tune.

c) Vulnerability-level aggregation.: A single vulnerability usually has more than one constraint. Some of them *must all hold together* (AND-style), while others describe *alternative paths* (ANY-of). Let C_v be the set of constraints for vulnerability v, each tagged $t(c)$. We aggregated tag weights into a context logic:

$$z_v \;=\; b_0 \;+\; \sum_{c \in C_v^{\mathrm{ALL}}} w\big(t(c)\big) \;+\; \sum_{G \in \mathcal{A}_v^{\mathrm{ANY}}} \max_{c \in G} w\big(t(c)\big),$$

where C_v^{ALL} are *AND*-style preconditions and $\mathcal{A}_v^{\mathrm{ANY}}$ are groups of alternative paths. If we did not know the grouping, we conservatively treated all constraints as AND-style. We then converted z_v into a probability using the standard logistic function:

$$p_{\mathrm{ctx}}(v) = \sigma(z_v) = \frac{1}{1 + e^{-z_v}}.$$

Intuitively, more and stronger enablers push p_{ctx} towards 1 (easy to exploit), while more and stronger gates push it towards 0 (hard to exploit).

d) Fusion with a prior (optional).: In many environments we already have a "global" signal about exploitation, such as EPSS, which tells us how likely a CVE is to be exploited somewhere on the Internet. We did not want to override that signal, but to complement it with our local view. When a prior probability $p_0(v)$ (e.g., EPSS) is available, we combine it with the context probability using a simple noisy-OR:

$$p_{\mathrm{final}}(v) \;=\; 1 - \big(1 - p_0(v)\big)\big(1 - p_{\mathrm{ctx}}(v)\big).$$

In plain terms, p_{final} becomes high if *either* the global model says "this CVE is often exploited" *or* our context score says "this CVE is easy in our setup" (or both). If no prior is available, we simply use $p_{\mathrm{ctx}}(v)$.

TABLE I
THE EXPLOITABILITY PRECONDITION TAXONOMY. THE 12 CATEGORIES USED TO ANNOTATE CVES, WITH DESCRIPTIONS AND REPRESENTATIVE EXAMPLES

Code	Category Name	Description and Examples
EXP	Exposure/Reachability	Conditions related to network reachability and service exposure. *Examples: a vulnerable port is internet-exposed; CORS policy permits cross-origin access.*
ENV	Environment/Deployment	Conditions tied to the deployment environment or OS. *Examples: a specific environment variable is set; running in a container without a hardening profile.*
CFG	Application Configuration	Application-level settings, flags, or modes. *Examples: a feature toggle is enabled; application runs in debug mode.*
AUTH	Authentication/Access	Identity, access control, and session state. *Examples: vulnerable endpoint accessible to anonymous users; attacker holds a specific role.*
DATA	Data State	Data-dependent preconditions. *Examples: default credentials present; object size exceeds a threshold.*
DEPS	Dependency/Runtime	Dependency presence/version or runtime traits. *Examples: vulnerable library in use; specific JVM/JIT behavior required.*
EXT	External Service	Preconditions involving external/cloud services. *Examples: overly permissive S3/IAM policy; misconfigured webhook endpoint.*
INPUT	Input Semantics	Crafted input formats or protocol sequences. *Examples: specific* `Content-Type`*; XML parser quirk leveraged.*
TIME	Timing/Concurrency	Timing or lifecycle constraints. *Examples: race window in TOCTOU; overlapping batch jobs.*
ISO	Isolation/Privilege	Security boundary or sandbox assumptions. *Examples: runs outside a sandbox; container escape requires kernel capability.*
LOG	Logging/Observability	Observability as an enabler. *Examples: verbose error logs leak secrets.*
CRYPTO	Cryptography/Trust	Weak or misconfigured cryptography. *Examples: weak TLS ciphers accepted; certificate validation disabled.*

TABLE II
DESCRIPTIVE STATISTICS OF THE CURATED DATASET (N = 2,426 CVES)

Metric	Count	Percentage
Vulnerability Severity Distribution		
CRITICAL	392	16.2%
HIGH	847	34.9%
MEDIUM	956	39.4%
LOW	88	3.6%
INFO/NONE	142	5.9%
Total CVEs	**2,426**	**100%**
Precondition Category Distribution		
ISO (Isolation)	11,993	44.5%
DEPS (Dependency)	7,578	28.1%
INPUT (Input Semantics)	1,773	6.6%
CFG (Configuration)	1,425	5.3%
EXP (Exposure)	1,232	4.6%
AUTH (Authentication)	1,097	4.1%
ENV (Environment)	975	3.6%
DATA (Data State)	419	1.6%
CRYPTO (Cryptography)	228	0.8%
EXT (External Service)	77	0.3%
LOG (Logging)	74	0.3%
TIME (Timing)	53	0.2%
Total Preconditions	**26,924**	**100%**

e) Fallback index.: Some teams prefer a simpler, non-probabilistic signal that they can reason about on a whiteboard. For them, we define an *Exploitability Ease Index*:

$$\mathrm{EEI}(v) \;=\; \#\{\text{enabler tags in } v\} \;-\; \#\{\text{gate tags in } v\}.$$

If EEI is strongly positive, the vulnerability has many enablers and few gates (it looks easy); if it is strongly negative, there are many gates and few enablers (it looks hard). We map EEI to {Low, Medium, High} using fixed thresholds, for example: *High* if EEI < -1, *Medium* if $-1 \le$ EEI ≤ 1, and *Low* if EEI > 1. EEI is less precise than p_{ctx}, but it is extremely easy to audit and explain.

f) Outputs.: For each vulnerability we therefore report:

1) the per-constraint tag (what has to be true),
2) the context probability p_{ctx} (and p_{final} if a prior exists),
3) the EEI value and its bucket (Low/Medium/High).

Together, these outputs provide a compact, reproducible ranking of vulnerabilities by context-driven exploitability, while still keeping the reasoning human-readable ("this CVE is easy because it is public, has no auth, and runs with weak TLS").

V. RESULTS

We labelled each constraint with a single tag and aggregated per-CVE into a context score (Section IV). A vulnerability is considered *easy to exploit* if its context probability satisfies $p_{\text{ctx}} \ge 0.5$; we also reported a conservative check using the Exploitability Ease Index (EEI > 0). In total, we scored 2,426 CVEs. The distribution of p_{ctx} is clearly right-skewed: the median is 0.047, with the first and third quartiles at approximately 0.002 and 0.574, respectively. In other words, many vulnerabilities are gated by one or more strong preconditions (very low p_{ctx}), while a non-trivial tail emerges as clearly easy under typical configurations.

Table III summarizes how many vulnerabilities, grouped by CVSS base severity, meet the *easy* criterion. The probability-based view shows a clear gradient: *CRITICAL* has the largest fraction flagged as easy (33.4%), followed by *HIGH* (28.3%), *MEDIUM* (26.2%), and *LOW* (23.9%). The EEI view yields lower rates—as intended for a sign-only index—yet preserves the same general ordering.

Two observations stand out. First, the gap between *MEDIUM* and *HIGH* is small a roughly one in four *MEDIUM* issues already looks easy in context. This suggests that a strict "fix HIGH first, MEDIUM later" policy will postpone a non-trivial set of vulnerabilities that are straightforward to exploit in common deployments. Second, even *LOW* severity is not uniformly harmless: almost a quarter of *LOW* entries cross the $p_{\text{ctx}} \ge 0.5$ threshold. In practice, many of these correspond to

TABLE III
EASY-TO-EXPLOIT COUNTS BY SEVERITY (CVSS BASED). PROBABILISTIC VIEW USES $p_{\text{CTX}} \geq 0.5$; INDEX VIEW USES EEI > 0

Severity	N	Easy (prob)	Rate	Easy (EEI)	Rate
LOW	88	21	23.9%	6	6.8%
MEDIUM	956	250	26.2%	84	8.8%
HIGH	847	240	28.3%	104	12.3%
CRITICAL	392	131	33.4%	28	7.1%

TABLE IV
TELECOM CASE: ITEMS FLAGGED AS *easy* ($p_{\text{CTX}} \geq 0.5$) BY SEVERITY

Sev.	N	Easy	Rate	$\tilde{p}_{\text{ctx}}$
LOW	10	2	20%	0.018
MEDIUM	44	12	27%	0.019
HIGH	34	10	29%	0.039
CRITICAL	12	4	33%	0.119
Total	**100**	**28**	—	—

configuration or exposure issues that do not change impact but strongly affect reachability.

The EEI-based view provides a sanity check. Because EEI only tracks the difference between counts of enabler and gate tags, it is less sensitive than the probabilistic score. However, it still highlights the same pattern: a growing share of "easy" items as we move from *LOW* through *MEDIUM* and *HIGH* to *CRITICAL*. This consistency across two simple metrics increases confidence that the signal is not an artefact of a particular threshold.

The context score provides complementary signal to EPSS: the correlation between p_{ctx} and EPSS is weak (Pearson correlation $= 0.081$, Spearman close to zero). In practice, EPSS reflects global threat pressure (how likely a CVE is to be exploited somewhere), while p_{ctx} captures deployment and configuration enablers in a specific environment. Prioritization benefits from considering both: act first where *high* severity aligns with *high* EPSS and *high* p_{ctx}; re-check *MEDIUM/LOW* items that surface as easy under local preconditions; and monitor high-EPSS but low-p_{ctx} items while hardening the relevant gates. In this way, the context score does not replace existing metrics, but fills a missing piece between theoretical impact and observed attacks.

VI. CASE STUDY (APPLIED INFORMATICS / TELECOM)

We applied the method to a telecom self-care platform (Java/Spring microservices behind an API gateway, Kafka for events, object storage for artifacts). A routine SAST/SCA scan reported **100** unique vulnerabilities. The company's standing policy is *severity-first* (fix CRITICAL, then HIGH), which is typical in large enterprises [24].

Base severities were: **12 CRITICAL**, **34 HIGH**, **44 MEDIUM**, and **10 LOW**. Under the baseline policy, developers would tackle the *46* CRITICAL+HIGH issues first.

We tagged constraints with a single category and computed the per-CVE context probability p_{ctx}. Using a clear criterion ($p_{\text{ctx}} \geq 0.5$) to flag items as *easy to exploit*, we obtained the counts in Table IV. Context surfaces "easy" items outside the CRITICAL/HIGH bucket (e.g., exposed endpoints with weak crypto or permissive configs) that a severity-only queue would postpone.

A severity-only plan would fill the next development sprint (Sprint 1, i.e., the first available change window for the team) with 46 CRITICAL+HIGH items, yet only about *14* of those are context-easy (4 CRITICAL + 10 HIGH). At the same time, there are *14* MEDIUM/LOW findings that are *also* context-easy (12 MEDIUM + 2 LOW). Reordering to *fix all 28 easy items first* yields two wins: (i) **faster exposure reduction** (we remove reachable/misconfigured pathways early), and (ii) **fewer hotfix loops** (shared root causes like reverse-proxy trust or insecure defaults neutralize multiple CVEs at once).

Gateway rule hardening (drop trust in `X-Forwarded-*`) eliminated 5 `EXP/INPUT` cases; *TLS hostname verification on* removed 3 `CRYPTO` vulns; *disable debug/actuator exposure* closed 4 `CFG/EXP` findings. All landed within one sprint with minimal code churn.

We still use EPSS as global threat pressure. Items with both *high* EPSS and *high* p_{ctx} go first. When EPSS is low but p_{ctx} is high, we still act because local preconditions make exploitation easy in *our* deployment; conversely, high-EPSS but low p_{ctx} items are monitored while we harden the gates.

For a telecom codebase, context-aware triage preserved focus on CRITICAL/HIGH but also surfaced MEDIUM/LOW vulnerabilities that were *easier* to exploit in situ. Fixing those 28 first reduced immediate risk and delivered diagnostic insight—outcomes aligned with Applied Informatics' fault detection and diagnostics.

VII. THREATS TO VALIDITY AND LIMITATIONS

Our dataset is centered on CVE/NVD. This skews coverage toward what is disclosed and standardized, omitting private advisories, narrow product lines, and configuration-only faults. Prior empirical work shows that public vulnerability corpora carry selection effects and do not fully capture exploitation dynamics [19], [25], [26].

We compressed each constraint into exactly one tag for auditability. Multi-faceted preconditions (e.g., auth *and* protocol) can be under-specified, and our enabler/gate polarity and weights reflect curated heuristics. Light-weight adjudication (spot checks, agreement metrics) would reduce coder bias; nonetheless, the single-tag rule trades nuance for consistency.

Many practical preconditions live in deployment notes, IaC, proxies, or Identity and Access Management (IAM) policies—not in CVE text. Inferring them from code/config patterns can miss or overstate gates/enablers. Exploitation in the wild further depends on attacker incentives and ecosystem conditions that public records under-represent [2], [19].

Our score estimates *situational ease of exploitation* given local preconditions; it is not an impact metric. Weak association with CVSS and modest association with exploit-in-the-wild signals are expected, because these constructs differ (technical impact vs. realized or likely exploitation) [25]. We therefore interpret it as complementary, not substitutive.

The "easy" cutoff ($p_{\text{ctx}} \geq 0.5$) and the sign-only EEI rule are pragmatic defaults. Different risk appetites or noise levels

may justify tighter/looser cutoffs; our reporting of medians and simple correlations is intended to make such tuning transparent.

To sum up these limits argue for using the context score alongside CVSS (impact) and exploit-likelihood signals (e.g., EPSS analyses), yielding a triage view that is both auditable and sensitive to local configuration.

VIII. Conclusions

This work introduced a compact, single-tag taxonomy for vulnerability *preconditions* and a transparent context score that estimates how easy a finding is to exploit *in situ*. Across our dataset and the telecom case study, we observed that base severity alone is a weak guide for action: a sizable fraction of *MEDIUM* (and even *LOW*) issues become practically easy to exploit once exposure, configuration, and crypto preconditions are considered. This aligns with our quantitative result that the context probability shows only weak association with CVSS, because it measures a different construct (situational ease rather than impact).

In practice, prioritization improves when teams combine three complementary signals: *CVSS* for technical impact, *EPSS* for ecosystem pressure, and *our context score* for local enablers and gates. Items that are simultaneously high-impact (CVSS), likely (EPSS), and easy in context (high p_{ctx}) should lead. Conversely, context can justifiably elevate certain *MEDIUM*/*LOW* findings ahead of harder-to-exploit *HIGH* ones, cutting exposure faster and reducing hotfix churn.

Our **Research summary** is that p_{ctx} adds signal beyond EPSS/CVSS (RQ1), flags sizable MEDIUM/LOW subsets as easy in context (RQ2), and—when used for reordering—cuts exposure faster than severity-only triage (RQ3).

Our recommendation is to not base vulnerability management solely on base severity. Fuse EPSS with the context score to obtain a triage queue that is both auditable and sensitive to local deployment realities.

For future work, the largest expected lift comes from *code reachability analysis*. Integrating static call-graphs and request-path coverage (e.g., API routing to vulnerable sinks) would down-weight unreachable code and up-weight paths demonstrably exercised in the product, sharpening p_{ctx} and further reducing noise in day-to-day triage. Complementary directions include light calibration of tag weights on internal incident data and tighter linkage to gateway/proxy policies to reflect true exposure.

References

[1] "Ghost report: Exorcising the sast demons," *Future Generation Computer Systems*, 2025.

[2] J. Jacobs, S. Romanosky, B. Edwards, I. Adjerid, and M. Roytman, "Exploit prediction scoring system (epss)," *Digital Threats: Research and Practice*, vol. 2, no. 3, pp. 1–17, 2021.

[3] National Vulnerability Database (NVD). (2025) Nvd dashboard. National Institute of Standards and Technology. [Accessed: 2025-09-02]. [Online]. Available: https://nvd.nist.gov/general/visualizations/vulnerability-metrics/cvss-severity-distribution-over-time

[4] M. C. Francesco Marchiori, Denis Donadel, "Can llms classify cves? investigating llms capabilities in computing cvss vectors," *arXiv preprint arXiv:2504.10713*, 2025.

[5] B. Martin, M. Brown, A. Paller, D. Kirby, and S. Christey, "2011 cwe/sans top 25 most dangerous software errors," *Common Weakness Enumeration*, vol. 7515, p. 2011, 2011.

[6] H. Hanif, M. H. N. M. Nasir, M. F. Ab Razak, A. Firdaus, and N. B. Anuar, "The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches," *Journal of Network and Computer Applications*, vol. 179, p. 103009, 2021.

[7] S. Elder, M. R. Rahman, G. Fringer, K. Kapoor, and L. Williams, "A survey on software vulnerability exploitability assessment," *ACM Computing Surveys*, vol. 56, no. 8, pp. 1–41, 2024.

[8] Y. Jiang, N. Oo, Q. Meng, H. W. Lim, and B. Sikdar, "A survey on vulnerability prioritization: Taxonomy, metrics, and research challenges," *arXiv preprint arXiv:2502.11070*, 2025.

[9] Cybersecurity and Infrastructure Security Agency (CISA). (2025) Known exploited vulnerabilities (kev) catalog. [Accessed: 2025-09-02]. [Online]. Available: https://www.cisa.gov/known-exploited-vulnerabilities-catalog

[10] R. Badhwar, "Risk-based vulnerability management," in *The CISO's Next Frontier: AI, Post-Quantum Cryptography and Advanced Security Paradigms*. Springer, 2021, pp. 371–378.

[11] CVE Program, "Cve record user guide," 2024, describes CVE JSON 5.x structure including ADP containers. [Online]. Available: https://test.cve.org/CVERecord/UserGuide

[12] NIST NVD, "Nvd vulnerabilities api 2.0 documentation," 2023, versioned JSON 2.0 endpoints and parameters for CVE data. [Online]. Available: https://nvd.nist.gov/developers/vulnerabilities

[13] FIRST CVSS-SIG, "Cvss v4.0 specification," FIRST, Tech. Rep., 2023, defines Base, Environmental, Threat, and Supplemental metric groups. [Online]. Available: https://www.first.org/cvss/v4-0/specification-document

[14] FIRST EPSS SIG, "Exploit prediction scoring system (epss)," 2025, ePSS v4 released 2025-03-17; probability of exploitation in next 30 days. [Online]. Available: https://www.first.org/epss/

[15] CISA, "Known exploited vulnerabilities (kev) catalog," 2025, authoritative list of vulnerabilities known to be exploited in the wild. [Online]. Available: https://www.cisa.gov/known-exploited-vulnerabilities-catalog

[16] OASIS CSAF Technical Committee, "Oasis csaf v2.0 vex profile," 2022, profile defining Vulnerability Exploitability eXchange (VEX) in CSAF. [Online]. Available: https://docs.oasis-open.org/csaf/csaf/v2.0/os/csaf-v2.0-os.html

[17] OWASP CycloneDX, "Cyclonedx vex," 2024, vEX representation aligned with CycloneDX SBOMs. [Online]. Available: https://cyclonedx.org/capabilities/vex/

[18] CISA, "Minimum requirements for vulnerability exploitability exchange (vex)," 2023, defines minimum content needed for useful, consistent VEX documents. [Online]. Available: https://www.cisa.gov/resources-tools/resources/minimum-requirements-vulnerability-exploitability-exchange-vex

[19] L. Allodi and F. Massacci, "Comparing vulnerability severity and exploits using case-control studies," *ACM Transactions on Information and System Security*, vol. 17, no. 1, pp. 1–20, Aug. 2014, shows weak link between CVSS base scores and exploitation; exploit-kit presence is a strong predictor. [Online]. Available: https://doi.org/10.1145/2630069

[20] X. Ou, S. Govindavajhala, A. W. Appel *et al.*, "Mulval: A logic-based network security analyzer." in *USENIX security symposium*, vol. 8. Baltimore, MD, 2005, pp. 113–128.

[21] N. Imtiaz, S. Thorn, and L. Williams, "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)*, 2021, pp. 1–11.

[22] R. Croft, D. Newlands, Z. Chen, and M. A. Babar, "An empirical study of rule-based and learning-based approaches for static application security testing," in *Proceedings of the 15th ACM/IEEE international symposium on empirical software engineering and measurement (ESEM)*, 2021, pp. 1–12.

[23] A. Verdet, *Exploring security practices in infrastructure as code: An empirical study*. Ecole Polytechnique, Montreal (Canada), 2023.

[24] J. M. Spring, E. Hatleback, A. Householder, A. Manion, and D. Shick, "Prioritizing vulnerability response: A stakeholder specific vulnerability categorization," Tech. Rep., 2019.

[25] M. Bozorgi, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond heuristics: learning to classify vulnerabilities and predict exploits," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2010, pp. 105–114.

[26] I. Forain, R. de Oliveira Albuquerque, and R. T. de Sousa Júnior, "Towards system security: What a comparison of national vulnerability databases reveals," in *2022 17th Iberian Conference on Information Systems and Technologies (CISTI)*. IEEE, 2022, pp. 1–6.