Methods for defining the severity of security vulnerabilities*

Smiljana Knežev[1,*], István Bozó[1] and Melinda Tóth[1]

[1]ELTE, Eötvös Lóránd University, Faculty of Informatics, Budapest, Hungary

Abstract

Not all security vulnerabilities are equal. Some pose a greater threat and can cause more harm than others. To effectively navigate security reports, different methods are used for defining the severity of vulnerabilities. This paper provides an overview of common approaches to prioritizing security vulnerabilities from the academic and industrial perspectives. Special attention is given to the Common Vulnerabilities and Exposures (CVE) and Common Vulnerability Scoring System (CVSS) used by the Erlang Ecosystem Foundation (EEF). These systems are relevant because the checkers already implemented in RefactorErl for Erlang and Elixir are based on the EEF guidelines. Furthermore, we describe how security vulnerabilities are currently scored in RefactorErl. Finally, we propose a system to further develop the scoring mechanism across already implemented and future security checkers.

Keywords

Erlang, security vulnerability categorisation, severity level

1. Introduction

We can observe that in today's world, security vulnerabilities are increasingly reported using artificial intelligence. To help make sense of what to act on and how quickly to act, vulnerability prioritization is necessary.

Many methods have been developed that assign a severity level to a potential security exploit. There has been an increasing amount of literature [1] in the past few years that bears this out.

Different methodologies [1] have their own strengths and weaknesses. This paper aims to provide a literature review of existing methods, standards, and frameworks for assessing the severity of security vulnerabilities.

This work focuses on methods for statically analyzing security vulnerabilities in Erlang and Elixir, programming languages that run on the BEAM virtual machine and are used to build scalable, reliable, and fault-tolerant systems. The goal is to evaluate which methods can be applied to RefactorErl, a static analysis tool for Erlang that supports BEAM analysis [2].

The rest of the paper is structured as follows. Subsection Background describes the historical development of industrial standards. Section 3 lists the most prominent security standards and frameworks. Section 4 describes the most widely used systems for assigning scores to security vulnerabilities. Further, Section 2 gives an overview of academic research in this field, and the Section 5 focuses on scoring systems used in industrial security analysis tools. Conclusions are provided in Section 6.

1.1. Background

MITRE is a U.S. not-for-profit organization that develops and maintains cybersecurity bases and classification systems. In 1999, the MITRE organization launched an initiative to collect data from various vulnerability databases and standardize it, so that each publicly disclosed vulnerability would have a unique identifier for reference [3]. This is how the Common Vulnerability Enumeration (CVE)

SQAMIA 2026: 13th Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications, September 16–18, 2026, Serbia, Niš, Serbia

t16wzn@inf.elte.hu (S. Knežev); bozoistvan@elte.hu (I. Bozó); tothmelinda@elte.hu (M. Tóth)

https://www.inf.elte.hu (S. Knežev); https://www.inf.elte.hu (I. Bozó); https://www.inf.elte.hu (M. Tóth)

0009-0006-2947-0390 (S. Knežev); 0000-0001-5145-9688 (I. Bozó); 0000-0001-6300-7945 (M. Tóth)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-11-19

CEUR-WS.org/Vol-4336/paper8.pdf

List was developed. The CVE Records are widely adopted as a standard by many organizations and vulnerability databases [1]. Numerous international organizations include CVE IDs in their alerts as soon as they are released, and CVE Records are used to uniquely identify vulnerabilities in OWASP Top 10 Web Application Security Issues and other similar watch lists [4].

CVE is also used by the National Institute of Standards and Technology (NIST) [5]. NIST's National Vulnerability Database (NVD) [6] is a comprehensive cybersecurity vulnerability database that includes all publicly available U.S. Government vulnerability resources.

Soon after the CVE List was published, MITRE recognized a need to categorize security vulnerabilities. This initiative led to the publication of the Common Weakness Enumeration (CWE) List and an associated classification taxonomy in 2006 [7].

CVE Lists are now built with the help of CVE Numbering Authorities (CNAs). Each CVE record added by a CNA includes a CWE ID indicating which vulnerability group it belongs to. The CVEs are rated using the Common Vulnerability Scoring System, an independent system developed by the Forum of Incident Response and Security Teams (FIRST). FIRST is an international non-profit association that brings together government, industry, academia, and other organizations in the field of cybersecurity.

The Erlang Ecosystem Foundation became a CNA in 2025[1].

2. Related Work

This work aims to cover the academic background, industry standards, and security analysis tools used to prioritize security vulnerabilities.

A survey published in February this year [1] served as the basis for this literature review. We chose it as a recent publication that summarizes relevant work on prioritizing security vulnerabilities from reputable sources such as ACM Digital Library, IEEE Xplore, Scopus, and Google Scholar [1].

We selected the security standards and frameworks based on their applicability, Beam security guidelines, potential use in RefactorErl, OWASP, and detailed severity-prioritization descriptions that are useful and adaptable for RefactorErl. Finally, we chose the different scoring systems described because they show different ways to measure and prioritize vulnerabilities. Some are widely used and considered standard.

The survey [1] examined 82 papers on vulnerability prioritization metrics published up to December 2024. The selection criteria were the topic of vulnerability prioritization, methodological depth, use of structured risk metrics, and validation techniques. As a result, a framework for evaluating vulnerability prioritization was developed. A new proposed taxonomy organizes vulnerability metrics into the following groups:

- Impact,
- Exploitability,
- Contextual and environmental factors,
- Predictive metrics,
- Aggregated system-level metrics.

Impact metrics focus on the intrinsic risk of exploiting a vulnerability and evaluate the consequences of doing so. Exploitability metrics look into the ease of exploiting a vulnerability. Aspects such as availability, attack complexity, user interaction, and required privileges are measured. Contextual and environmental metrics use deployment-specific and other organizational information to evaluate a component's level of compromise from multiple perspectives, including business impact, operational constraints, and exposure levels. Predictive metrics assess the likelihood of future exploitation. This is done using statistical and machine learning models. Aggregated system-level metrics integrate multiple approaches to calculate a single score for the entire system.

An empirical study [8] compared four leading scoring systems: the CVSS, the Stakeholder-Specific Vulnerability Categorization (SSVC), the Exploit Prediction Scoring System (EPSS), and the Exploitability

[1] https://cna.erlef.org

Index on a dataset of 600 real-world vulnerabilities from Microsoft's Patch Tuesday disclosures. They found little to no correlation between vulnerability scores for the same vulnerabilities. The authors describe this as a result of each system being designed to measure different aspects of vulnerabilities. The study highlights the need for more empirically grounded frameworks.

A 2024 study [9] conducted preliminary interviews with seven people and later a follow-up survey with 71 participants to examine users' attitudes towards the National Vulnerability Database (NVD) for vulnerability management. The authors claim that the users' experience was generally positive. The survey showed that NVD is most commonly used to collect details about a vulnerability and to use the CVSS score as a reference. Later, within companies, that CVSS score is used to prioritize vulnerabilities.

The issues users listed were missing or incorrect entries, incomplete descriptions, or incomprehensible CVSS ratings. The NVD members are making an ongoing effort to fix these issues and improve the system. One of the possible solutions mentioned was creating a uniform language for describing vulnerabilities.

Impact and exploitability metrics are important in technical assessments; contextual metrics are used to assess the relevance of compliance. Progressive and aggregated metrics are used to predict future risks [1].

Methodologies used for prioritizing vulnerabilities are divided into 5 groups [1]:

- Graph-based methods,
- ML and AI-based approaches,
- Multi-objective optimization methods,
- Rule-based and expert systems,
- Statistical methods.

Graph-based methods are good for analyzing vulnerabilities in interconnected infrastructures. These methods include analysis of attack graphs, dependency graphs, and Bayesian networks. ML and AI-based approaches use historical and real-time data to anticipate security risks. Here, logistic regression, decision trees, and neural networks are used. Multi-objective optimization methods combine several factors to assess vulnerability. This involves optimization algorithms, genetic algorithms, integer programming, and evolutionary models. Rule-based and expert systems rely on human expertise to define predefined rules and heuristics for assessing vulnerabilities. Statistical methods use regression, probabilistic models, and distribution analysis to measure risk factors and rank vulnerabilities.

RefactorErl focuses on developing severity-assigning metrics that taxonomically belong to the Impact group. These metrics assess how a vulnerability affects the entire project or system. To achieve this, a rule-based methodology is used. Currently, not all rules are refined for all available security checkers. For the missing rules, a default severity of the highest level is assigned.

Ongoing work is underway to develop Exploitability metrics in RefactorErl that calculate the reachability of detected vulnerabilities.

Furthermore, development of Predictive metrics using ML and AI-based approaches is being considered for future work.

3. Security Standards and Frameworks

Security standards and frameworks provide users with established practices for preventing vulnerabilities and assigning their severities. Among the most relevant sources for RefactorErl are the BEAM security guidelines by EEF and Erlang and Elixir security recommendations in the language documentation. OWASP secure development resources and the SEI CERT Secure Coding Standards offer valuable approaches to prioritizing vulnerabilities that can be used and adapted for security checkers in RefactorErl.

3.1. OWASP

The Open Worldwide Application Security Project (OWASP) is a nonprofit foundation that provides free resources for software security [10].

The goal of the OWASP Risk Rating Methodology is to propose a system for rating security risks that saves time and reduces disagreements over risk priorities, helping businesses make informed decisions about how to address those risks. The proposed basic framework is designed to be customized, since a particular vulnerability may be more relevant to some organizations than to others.

The standard risk model is calculated by multiplying likelihood and impact. Multiple factors go into assigning a likelihood and impact. The user of the framework is guided to apply the following steps to determine them:

- Step 1: Identifying a Risk,
- Step 2: Factors for Estimating Likelihood,
- Step 3: Factors for Estimating Impact,
- Step 4: Determining Severity of the Risk,
- Step 5: Deciding What to Fix,
- Step 6: Customizing Your Risk Rating Model

Users are advised to err on the side of caution, looking into the worst-case scenario that will lead to the highest overall risk.

Steps 2 and 3 help the evaluator assign numerical values to different likelihood and impact factors, which are then entered into a formula to produce a final score. In step 4, the final severity is estimated based on the final score:

- 0 to 3: Low,
- 3 to 6: Medium,
- 6 to 9: High.

This process can be automated with tools.

Step 5 recommends fixing the highest-scoring risks first, and Step 6 describes how to customize options for a more effective model.

OWASP also produces OWASP Top 10[2] and OWASP API Security Top 10 documents[3]. The first identifies critical categories of web application security risks, while the second focuses on API-specific risks. These documents do not have their own severity levels assigned to individual findings but provide a consensus-based categorization of common and important risks.

3.2. SEI-CERT

The SEI-CERT standards describe rules to help developers write safe, reliable, and secure code in C, C++, Java, and Perl. The standards contain risk assessments. Each rule assigns priority to severity, likelihood, and remediation cost. Numbers from 1 to 3 are assigned to each priority based on the Failure Modes, Effects, and Criticality Analysis (FMECA) [11].

Severity metrics measure the consequences of ignoring a rule. The possible values are presented in Table 1

The likelihood metric measures how likely it is for a vulnerability to lead to an exploitable vulnerability. The corresponding scores are shown in Table 2

Remediation cost consists of detectable and repairable metrics. Detectable metrics measure whether a static analysis tool can be used to detect the violation, and the repairable metrics check whether the violation can be repaired by an automated tool without breaking the existing code. The scores are presented in Table 3

The Remediation Cost, Priority, and Severity metric values are multiplied together for each rule, and the result is used to assign a priority and severity level. Possible values are presented in Table 4.

[2]https://owasp.org/Top10/2025/

[3]https://owasp.org/API-Security/editions/2023/en/0x11-t10/

Table 1
Severity

Value	Meaning	Examples of Vulnerability
1	Low	Denial-of-service attack, abnormal termination
2	Medium	Data integrity violation, unintentional information disclosure
3	High	Run arbitrary code

Table 2
Likelihood

Value	Meaning
1	Unlikely
2	Probable
3	Likely

Table 3
Remediation cost

Automatically...	Not Repairable	Repairable
Not Detectable	1	2
Detectable	2	3

Table 4
Priorities and Levels

Level	Priorities	Possible Interpretation
L1	**12 , 18 , 27**	High severity, likely, inexpensive to repair
L2	**6 , 8 , 9**	Medium severity, probable, medium cost to repair
L3	**1 , 2 , 3 , 4**	Low severity, unlikely, expensive to repair

It is recommended that projects start by remediating security issues by applying rules at the highest level before moving to those at lower levels.

3.3. Beam security guidelines

Erlang Ecosystem Foundation (EEF) [12] provides guidelines for developing secure BEAM applications in Erlang and Elixir. These guidelines include the following documents:

- Secure Coding and Deployment Hardening Guidelines,
- Web Application Security Best Practices for BEAM languages,
- Security Vulnerability Disclosure,
- Meeting Notes.

While these documents do not include information on the severity levels of the security vulnerabilities, the EEF foundation has the CNA and discloses vulnerability records in EEF advisories. Here, the CVSS score is included for each disclosed record.

Erlang's documentation for the OTP 29.0 includes a Secure Coding Guidelines section. In this section, rules for writing secure Erlang code are listed [13]. Each rule has its own ID, a textual explanation of the vulnerability with remediation instructions where applicable, priority, and related CWE and OWASP risk IDs.

When assigning priorities, the likelihood of a code flaw leading to a vulnerability, the severity of the vulnerability, and the ease of remediation are considered. Rules are assigned the following priorities: *Critical, High, Medium and Recommendation.* The rules are also divided into the following categories:

Code style, Deployment, Design, Language and Miscellaneous. The documentation also includes sections on the Top 40 CWEs in 2025 and their relation to Erlang/OTP, as well as the OWASP Top 10 and its relation to Erlang/OTP.

Similarly, separate guidelines in Core Elixir's documentation cannot be found. Some pages include descriptive security-related sections, but without severity mappings. However, Elixir's web application framework, Phoenix, has a separate document regarding security [14]. Here, the common web application security vulnerabilities relevant to Phoenix are described. Severity is not assigned a single label or numerical value but is instead described in text. The document contains links to relevant EEF guidelines.

4. Scoring Systems

The selected scoring systems represent different approaches to prioritizing vulnerability assessment. While some focus on intrinsic vulnerability severity and exploitability, others predict likelihood and account for stakeholders' interests. Different systems can be used together, complementing each other and providing a more comprehensive approach.

4.1. CVSS

CVE uses the Common Vulnerability Scoring System (CVSS) to score vulnerabilities. CVSS is owned and managed by FIRST, a US-based, independent non-profit organization separate from MITRE.

CVSS is the most widely used standard for vulnerability prioritization, cited in 70 percent of evaluated papers [1]. CVSS assigns a numerical score to a vulnerability relative to other vulnerabilities, ranging from 0.0 to 10.0. [15].

CVSS consists of four metric groups: Base, Threat, Environmental, and Supplemental [15]. The Base metric measures intrinsic characteristics that do not change over time. The Threat metric checks the availability of active exploitation or proof-of-concept code. The Environmental Metrics adjust the score to the computing environment. The Supplemental Metrics measure extrinsic context-dependent attributes. While the Base metrics are assigned by the provider's analyst, the Threat and Environmental metrics are assigned by the CVSS consumer, and the Supplemental metrics are assigned by the provider, with their importance adjusted by the consumer. The term provider refers to a vendor, maintainer, CNA, or third-party assessor, and a CVSS consumer is an organization that uses CVSS to assess the threat score and has insight into the environment and infrastructure in which the code runs.

The numerical CVSS scores can be mapped to severity levels: none, low, medium, high, and critical. Most static analysis tools use such descriptive labels to prioritize the security findings. Behind these labels doesn't always necessarily stand a CVSS score.

One of the critiques of the CVSS base score is that it does not reflect evolving threats, active exploitations, or new attack vectors. The CVSS base score does not necessarily match the actual vulnerability risk. It can happen that a high CVSS score in a well-isolated environment poses a low risk, and vice versa [1].

Since the CVSS scoring system relies on an authority to provide the evaluation, slight inconsistencies are observed among different organizations, which assign different scores to the same vulnerabilities. Regardless of these issues, most participants have a positive attitude of CVSS [16].

4.2. CWSS

The Common Weakness Scoring System (CWSS), developed by MITRE, is a scoring system for weaknesses within the CWE project [17]. Unlike CVSS, which is assigned after a vulnerability has been disclosed, CWSS aims to provide a score for use during code review before a vulnerability has been proven.

The CWSS score can be calculated either manually or automatically. The automatically calculated score may miss certain environmental/contextual information, so it is possible to combine these approaches: first, automatic calculation, then manual updates to include additional details.

CWSS consists of three metric groups: Base Finding, Attack Surface, and Environmental. Base Finding metrics assess the inherent risk of the weakness, Attack Surface measures the effort an attacker must invest to exploit the weakness, and Environmental metrics examine the relationship between environmental influences and the weakness.

Table 5 shows the metric groups, the metrics included in each group, and their short summaries.

Table 5
CWSS metrics

Group	Name	Summary
Base Finding	Technical Impact (TI)	The possible consequences of exploiting the weakness.
	Acquired Privilege (AP)	The type of privileges that are obtained by an attacker.
	Acquired Privilege Layer (AL)	The operational layer to which the attacker gains privileges.
	Internal Control Effectiveness (IC)	The effectiveness of internal control to prevent exploitation.
	Finding Confidence (FC)	The confidence that the issue is an exploitable weakness.
Attack Surface	Required Privilege (RP)	The type of privileges needed to exploit the weakness.
	Required Privilege Layer (RL)	The operational layer needed to exploit the weakness.
	Access Vector (AV)	The channel an attacker uses to reach the weakness
	Authentication Strength (AS)	The strength of the authentication routine protecting the weakness.
	Level of Interaction (IN)	The actions the victim must take for a successful attack.
	Deployment Scope (SC)	Whether the weakness is present in all deployable instances
Environmental	Business Impact (BI)	The potential impact to the business or mission.
	Likelihood of Discovery (DI)	The likelihood that an attacker can discover the weakness.
	Likelihood of Exploit (EX)	The likelihood that, if the weakness is discovered, an attacker with the required privileges/authentication/access would be able to successfully exploit it.
	External Control Effectiveness (EC)	The capability of controls or mitigations outside the software that may make the weakness more difficult for an attacker to exploit and/or trigger.
	Prevalence (P)	How frequently this type of weakness appears in software.

For each metric group, each factor is assigned a value, which is converted to an associated weight, and a subscore is calculated. The subscores for each metric group range from 0 to 100. These three subscores are multiplied together to produce a CWSS score between 0 and 100.

Although individual CWSS factors use descriptive values such as Critical, High, Medium, Low, and None, CWSS does not provide a mapping between the final score and descriptive severity level.

The Common Weakness Risk Analysis Framework (CWRAF), also developed by MITRE, provides a means to score weaknesses using CWSS in a customizable way. With the help of CWRAF, users can generate a targeted ”Top-N” list of weaknesses tailored to their needs. The framework is also used by CWE/SANS Top 25 and OWASP Top Ten [18].

CWRAF uses vignettes for context-dependent scoring of a system. A vignette is a shareable, semi-formal description of an operational or business environment. For an environment, the vignette describes

aspects such as which assets matter, the technologies involved, and the organization's priorities for software security [19].

4.3. SSVC

Stakeholder-Specific Vulnerability Categorization (SSVC) is a system developed by Carnegie Mellon University's Software Engineering Institute (SEI) in collaboration with the Cybersecurity and Infrastructure Security Agency (CISA). The aim of SSVC is to help organizations prioritize actions during vulnerability management [20].

SSVC uses a decision tree model to prioritize vulnerabilities into decisions. Possible decisions are: *Track, Track*, Attend and Act.*

Track means monitoring vulnerabilities and reassessing when more information becomes available. Track* means monitor more closely for changes and fix within the standard timeline. Attend requires remediation in sooner-than-standard time. Finally, Act means remediate as soon as possible.

4.4. EPSS

The Exploit Prediction Scoring System (EPSS) is a machine learning model used to estimate the probability that, within 30 days of disclosure, the published CVE will be exploited. EPSS publishes a probability for each CVE every day [21].

4.5. Exploitability Index

The Microsoft Exploitability Index calculates the exploitability of every vulnerability in a Microsoft security update and publishes the information in the monthly Microsoft security update details [22].

Table 6 shows index values and their short definitions.

Table 6
Exploitability Index assessment

Exploitavility index assessment	Short definition
0	Exploitation detected
1	Exploitation more likely
2	Exploitation less likely
3	Exploitation unlikely

5. Security Analysis Tools

This section describes different security analysis tools, mostly SAST tools, and how they report security severity levels. Some of these tools were chosen because of their popularity: Semgrep and SonarQube. Sobelow was chosen because it analyzes Elixir, and the other tools were chosen because they calculate and present severity levels differently.

Tables 7 to 11 compare these tools and how they present security vulnerability severities.

5.1. CodeChecker

CodeChecker is a static analysis tool built on top of the LLVM/Clang Static analyser toolchain [23]. When presenting results, CodeChecker uses the following labels: *profile, guideline, sei-cert-c, sei-cert-cpp and severity.*

The label representing the severity of the bug has possible values: style, low, medium, critical, and unspecified. The style label indicates that the code violates a specific guideline, and the unspecified label indicates that the severity for that checker is not set. The critical label is used for compilation

errors. The low label usually means the code is hard to read or comprehend. The medium label indicates a possible runtime error, while the high label means the code will cause a runtime error.

5.2. Sobelow

Sobelow is a security-focused static analysis tool for Elixir and Phoenix [24]. There are three possible severity levels for each reported vulnerability: high, medium, and low. The tool color-codes its findings according to confidence in the insecurity. Red means high, yellow medium, and green low. If it cannot be reliably determined whether a function accepts user-supplied input, it is marked as having low insecurity confidence.

5.3. Semgrep

Semgrep has Semgrep Code, a SAST tool that combines static analysis with AI; Semgrep Supply Chain, an SCA tool that uses reachability analysis; and the Semgrep Secret Scanning tool, which uses semantic and entropy analysis to prevent secret leakage [25]. Semgrep uses rules that describe vulnerabilities it searches for.

Semgrep uses a severity level for each rule [26]. Possible values of these levels are: critical, high, medium, and low.

For Semgrep Code and Semgrep Secret Scanning Tool, the severity level reflects how critical the issue the rule detects is. Here, the rule author assigns the severity. When it comes to Semgrep Supply Chain, severity is tied to the CVSS score or the score set by the GitHub Advisory Database. Also, EPSS scoring is provided.

In addition to the severity level, the confidence level is provided. The confidence level is set by the rule author. The purpose of the confidence level is to assure the user that the pattern accurately captures the vulnerability and does not produce many false positives.

5.4. Checkmarx

Checkmarx is an AI-powered, cloud-native application security platform. It uses hybrid scanning to identify vulnerabilities, prioritize risks based on exploitability, and fix critical issues [27]. Checkmarx One has multiple scanners: SAST, SCA, API Security, IaC Security, and Container Security [27]. Each discovered vulnerability is assigned a severity level.

The initial severity level is assigned by the Checkmarx One AppSec research team, drawing on their expertise in threat assessment. Later, users can fine-tune the severity level based on their own judgment. Furthermore, overall project and application risk are assigned [28]. Possible severity levels are: critical, high, medium, low, and info.

Critical severity is assigned to vulnerabilities that demand immediate attention. These vulnerabilities lead to a complete system compromise, unauthorized access, or data breaches, resulting in breaches of the system's confidentiality, availability, and integrity. High severity means the vulnerability can lead to unauthorized access to resources, theft of sensitive data, or theft of session information. For the vulnerability to be exploited, additional conditions must be met, such as a privileged user successfully interacting with the attack. Medium Severity is assigned to vulnerabilities that can be a potential security weakness, usually misconfigurations that may result in limited unauthorized access or data exposure. Low severity is used for vulnerabilities that have minimal impact on immediate security but, if not addressed, can escalate into more significant security issues. The info label provides information about a lack of compliance with security best practices.

5.5. Coverity

Coverity is a static analysis SAST tool for a large collection of programming languages, libraries, and frameworks [29].

Coverity's security report includes a section titled "Security Details". Here, the number of issues associated with each Technical Impact category and corresponding severity level are listed. When the report generator is configured, it is possible to specify custom severity mapping. Later in the "Detailed Issues Ranked by Severity" section, all issues are listed along with their severity levels. Possible severity levels are: very high, high, medium, low, and very low [30].

5.6. Klockwork

Klockwork is a SAST tool that reports security and reliability issues for C, C++, C#, Rust, Java, JavaScript, Python, and Kotlin. For each reported issue, a severity level is assigned, from 1 to 10, with a label [31]. Severity levels are:

- 1 - Critical,
- 2 - Error,
- 3 - Warning,
- 4 - Review,
- 5 to 10 - Configurable severity levels.

Here, numbers from 1 through 10 are part of a severity label and not a calculated severity score.

Each checker is assigned a default severity level. Levels 1 through 4 are assigned by default. Users can edit severity levels and labels for individual checkers.

5.7. SonarQube

SonarQube is a platform for automated code quality and security analysis. It supports more than 40 programming languages and frameworks.

SonarQube offers the Multi-Quality Rule (MQR) and Standard Experience Severity Types [32]. MQR severity type assigns separate severities for different qualities per one rule. The qualities are Security, Reliability, and Maintainability.

Standard Experience Severity Types assign a single label per severity to a rule.

Rule authors assign a default severity to their rules. When assigning a value, the authors consider whether a rule breach could cause significant damage to the system and its users, and how likely exploitation is.

SonarQube also allows users with the appropriate permissions to set a custom severity for a rule.

5.8. Aikido

Aikido is an application security platform for developers. It includes multiple tools, including AI, to scan code, dependencies, infrastructure, containers, and cloud configurations [33].

For dependency issues, the tool looks into public vulnerability databases and sources, such as NVD and its own threat intelligence source, Aikido Intel. Further, Aikido adjusts severity based on context, considering active exploitation and code reachability.

For SAST results, Aikido has an AutoTriage layer that is executed after scanning and calculates severity. It first tries to rule out exploitability and then prioritizes the remaining findings [34].

Aikido uses an internal scoring system from 0 to 100 to prioritize risk. From 90 to 100, the vulnerability is critical. Between 70 and 89, the vulnerability is high. From 40 to 69, it is medium, and below 39 is low.

5.9. RefactorErl

RefactorErl [2] is a static analysis tool for Erlang that supports analysis of compiled Beam code, so Elixir can also be analyzed. Currently, it assigns severity levels using a rule-based system for several vulnerability checkers; for the rest, the severity is high by default.

Rules are defined for race-condition and denial-of-service vulnerability checkers. The severity is returned as a tuple, where the first element has possible values: high, medium, and low, and the second element is a description of why that priority level is assigned.

To check for a race condition, the first argument of the *process_flag/2* function is analyzed; if the possible value is priority, this indicates that the process priority is manually set, and the vulnerability is high; otherwise, it is moderate.

Dynamically creating atoms can lead to atom exhaustion, resulting in a denial-of-service vulnerability in Beam languages. Depending on whether a function that creates atoms accepts user input and whether that function is in a loop, different severity levels are assigned. If such a function is in a loop, high severity is assigned; otherwise, low severity is assigned. If such a function is used by a lambda expression that is passed to a function whose behavior the analyzer does not know, the severity is medium.

Further rules need to be developed to assign severity levels to the remaining vulnerability checkers. Each security vulnerability can be categorized based on its Technical Impact in one of the following groups according to the CWE [35]:

- Read data,
- Modify data,
- Denial-of-Service: unreliable execution,
- Denial-of-Service: resource consumption,
- Execute unauthorized code or commands,
- Gain privileges / assume identity,
- Bypass protection mechanism,
- Hide activities

For different organizations and projects, preventing vulnerabilities from different groups will be of greater importance. Providing additional information to each vulnerability in terms of which Technical Impact group it belongs to, and links to corresponding CWE pages that describe the vulnerability in detail, will help users understand, sort, and prioritize the findings.

Furthermore, in the vulnerability report, we plan to include how each severity level was assigned to each vulnerability, along with the tainted execution path that led to it.

Following a rule-based approach for each security vulnerability, we will assign a severity level with the above-described explanations, so users can easily sort the findings by the aspect they find most important. Future work may include a way for users to configure these values before running the analysis.

We have decided to follow the rule-based approach for its reliability and repeatability. Future work may involve using statistical or machine learning models that leverage known EEF CVE repositories to predict the impact of a security vulnerability.

The advantage RefactorErl has over the other tools mentioned is that it is tailored specifically to languages compiled to the BEAM, making the security checkers more precise and focused on relevant vulnerabilities in this environment. Currently, vulnerability checkers are implemented for Erlang and Elixir programming languages, but the implementation can be extended to other languages compiled to the BEAM.

5.10. Compariosn tables

In tables 7 to 11, we list all the mentioned tools, the severity-level labels they use, the severity scores behind those labels (if they exist), and a short description of these levels.

6. Conclusion

This paper presents a literature overview of security standards, scoring systems for assigning vulnerability severity levels, and how security analysis tools use those scoring systems.

Table 7
Comparison table

Tool	Severity levels	Score	Description
CodeChecker	Style	/	The source code is against a specific coding guideline or could improve readability.
	Low	/	The source code is hard to read or understand, or could be easily optimized.
	Medium	/	The source code that may not cause a run-time error (yet), but goes against intuition and hence is prone to error.
	High	/	The source code will cause a run-time error.
	Critical	/	Currently used for indicating compilation errors.
	Unspecified	/	Checker severity is not specified for a checker.
Sobelow	High	/	Rule-based for every vulnerability
	Medium	/	Rule-based for every vulnerability
	Low	/	Looks like a function could be used insecurely, but it cannot reliably be determined if the function accepts user-supplied input.
Semgrep	Critical	/	Highest on the ordered scale, assigned by the rule author.
	High	/	High the ordered scale, assigned by the rule author.
	Medium	/	Medium on the ordered scale, assigned by the rule author.
	Low	/	Low on the ordered scale, assigned by the rule author.

Table 8
Comparison table (cont.)

Tool	Severity levels	Score	Description
SonarQube MQR	Blocker	/	An issue with a high probability of severe unintended consequences for the application should be fixed immediately. This includes bugs leading to production crashes and security flaws allowing attackers to extract sensitive data or execute malicious code.
	High	/	An issue with a high impact on the application that should be fixed as soon as possible.
	Medium	/	An issue with a medium impact.
	Low	/	An issue with low impact.
	Info	/	There is no expected impact on the application. For informational purposes only.

Table 9
Comparison table (cont.)

Tool	Severity levels	Score	Description
Checkmarx	Critical	9.0 -10.0	Vulnerabilities that demand immediate attention. These vulnerabilities can lead to a complete system compromise, unauthorized access, or data breaches, compromising the system's confidentiality, availability, and integrity.
	High	7.0 - 8.9	The vulnerability can lead to unauthorized access to resources, theft of sensitive data, or theft of session information. For the vulnerability to be exploited, additional conditions must be met, such as a privileged user successfully interacting with the attack.
	Medium	4.0 - 6.9	Vulnerabilities that can be a potential security weakness, usually misconfigurations that may result in limited unauthorized access or data exposure.
	Low	0.1 - 3.9	Low severity is used for vulnerabilities that have minimal impact on immediate security but, if not addressed, can escalate into more significant security issues.
	Info	0.0	Information about a lack of compliance with security best practices.

Table 10
Comparison table (cont.)

SonarQube Standard Experience Severity Types	Blocker	/	An issue with a high probability of severe unintended consequences for the application should be fixed immediately. This includes bugs leading to production crashes and security flaws allowing attackers to extract sensitive data or execute malicious code.
	Critical	/	An issue with a critical impact on the application that should be fixed as soon as possible.
	Major	/	An issue with a significant impact on the application.
	Minor	/	An issue with a minor impact on the application.
	Info	/	There is no expected impact on the application. For informational purposes only.

In our previous work, we aimed to implement all recommendations in the EEF guidelines and integrate them into RefactorErl with a focus on reducing false-positive security analysis results. Even as new

Table 11
Comparison table (cont.)

Tool	Severity levels	Score	Description
Coverity	Very High	/	Highest in the ordered scale associated with Technical impact.
	High	/	High in the ordered scale associated with Technical impact.
	Medium	/	Medium in the ordered scale associated with Technical impact.
	Low	/	Low in the ordered scale is associated with Technical impact.
	Very Low	/	Lowest in the ordered scale associated with Technical impact.
	Informational	/	Informational classification.
Klockwork	1 - Critical	/	Level 1 highest on ordered scale.
	2 - Error	/	Level 2 on ordered scale.
	3 - Warning	/	Level 3 on ordered scale.
	4 -Review	/	Level 4 on ordered scale.
	5 - 10 - Configurable	/	Configurable levels on ordered scale.
Aikido	Critical	90 - 100	Highest on ordered scale.
	High	70 - 89	High on ordered scale.
	Medium	40 - 69	Medium on ordered scale.
	Low	1-39	Low on ordered scale.
RefactorErl	High	/	Highest on ordered scale.
	Medium	/	Medium on ordered scale.
	Low	/	Low on ordered scale.

recommendations emerge, we recognize the importance of assigning an appropriate severity level to each detected vulnerability.

Looking into different security standards, scoring systems, and tools, we note that the combination of a value set by an authority, a rule-based system that automatically checks security risk, and end-user or organization configuration determines the most appropriate severity level for static code analysis. Some rules are already defined in RefactorErl; in future work, we aim to implement rules for the remaining security checkers, assign initial severities to them, and enable users to configure the results by incorporating organizational and environmental information.

Acknowledgments

Project no. KDP-25-II-ELTE-246 has been implemented with the support provided by the Ministry of Culture and Innovation of Hungary from the National Research, Development and Innovation Fund, financed under the 2025-2.1.1 Univeristy Research Scholarship Program - Cooperative Doctoral Program funding scheme.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] Y. Jiang, N. Oo, Q. Meng, H. Lim, B. Sikdar, A survey on vulnerability prioritization: Taxonomy, metrics, and research challenges (2025). doi:10.48550/arXiv.2502.11070.

[2] M. Tóth, I. Bozó, Static analysis of complex software systems implemented in erlang, Central European Functional Programming Summer School – Fourth Summer School, CEFP 2011, Revisited

Selected Lectures, Lecture Notes in Computer Science (LNCS), Vol. 7241, pp. 451-514, Springer-Verlag, ISSN: 0302-9743, 2012.

[3] D. E. Mann, S. M. Christey, Towards a common enumeration of vulnerabilities, 1999. URL: https://www.cve.org/Resources/General/Towards-a-Common-Enumeration-of-Vulnerabilities.pdf, presented at the 2nd Workshop on Research with Security Vulnerability Databases, Purdue University, West Lafayette, IN, USA, Jan. 21–22, 1999.

[4] C. Program, Cve history, 1999. URL: https://www.cve.org/about/history.

[5] N. I. of Standards, Technology, Cybersecurity framework, 2024. URL: https://www.nist.gov/cyberframework.

[6] N. I. of Standards, Technology, National vulnerability database, 2026. URL: https://nvd.nist.gov/.

[7] MITRE, Cwe history, 2026. URL: https://cwe.mitre.org/about/history.html.

[8] V. Koscinski, M. Nelson, A. Okutan, R. Falso, M. Mirakhorli, Conflicting scores, confusing signals: An empirical study of vulnerability scoring systems, in: Proceedings of the 32nd ACM Conference on Computer and Communications Security, CCS '25, Association for Computing Machinery, 2025, pp. 1904–1918. doi:10.1145/3719027.3765210.

[9] J. Wunder, A. Corona, A. Hammer, Z. Benenson, On nvd users' attitudes, experiences, hopes, and hurdles, Digital Threats 5 (2024). URL: https://doi.org/10.1145/3688806. doi:10.1145/3688806.

[10] O. Foundation, Owasp foundation, 2026. URL: https://owasp.org/.

[11] S. E. Institute, How this coding standard is organized, 2016. URL: https://github.com/semgrep/semgrep.

[12] E. E. Foundation, Security working group, 2026. URL: https://erlef.org/wg/security.

[13] Erlang/OTP, Secure coding guidelines, 2025. URL: https://erlang.org/github-pr/10447/doc/system/secure_coding.html.

[14] P. Framework, Security, 2025. URL: https://phoenix.hexdocs.pm/security.html.

[15] FIRST, Common vulnerability scoring system version 4.0: Specification document, 2023. URL: https://www.first.org/cvss/specification-document.

[16] J. Wunder, A. Kurtz, C. Eichenmüller, F. Gassmann, Z. Benenson, Shedding light on cvss scoring inconsistencies: A user-centric study on evaluating widespread security vulnerabilities, in: 2024 IEEE Symposium on Security and Privacy, IEEE, 2024. doi:10.1109/SP54263.2024.00058.

[17] Mitre, Cwe documentation, 2026. URL: https://cwe.mitre.org/cwss/cwss_v1.0.1.html.

[18] MITRE, Common weakness scoring system (cwss), 2014. URL: https://cwe.mitre.org/cwss/cwss_v1.0.1.html.

[19] MITRE, Common weakness risk analysis framework (cwraf), 2013. URL: https://cwe.mitre.org/cwraf/introduction.html#howtousecwraf.

[20] Cybersecurity, I. S. Agency, Stakeholder-specific vulnerability categorization (ssvc), 2022. URL: https://www.cisa.gov/resources-tools/resources/stakeholder-specific-vulnerability-categorization-ssvc.

[21] FIRST, Exploit prediction scoring system (epss), 2026. URL: https://www.first.org/epss/.

[22] M. S. R. Center, Microsoft exploitability index, 2026. URL: https://www.microsoft.com/en-us/msrc/exploitability-index.

[23] CodeChecker, Codechecker documentation, 2025. URL: https://codechecker.readthedocs.io/en/latest/.

[24] G. Byatt, H. Oullette, Sobelow, 2026. URL: https://github.com/sobelow/sobelow.

[25] I. Semgrep, Semgrep, 2026. URL: https://semgrep.dev/.

[26] I. Semgrep, Semgrep documentation, 2026. URL: https://docs.semgrep.dev/kb/rules/understand-severities.

[27] I. Checkmarx, Checkmarx one, 2026. URL: https://https://checkmarx.com/product/application-security-platform/.

[28] I. Checkmarx, Checkmarx documentation, 2026. URL: https://docs.checkmarx.com/en/34965-68521-checkmarx-one-calculation-of-severity-level.html.

[29] I. Black Duck Software, Coverity scan, 2024. URL: https://scan.coverity.com.

[30] I. Black Duck Software, Coverity scan - static analysis, 2026. URL:

https://docs.blackduck.com/r/coverity/2026.3/coverity-documentation/severity-mapping-mapping-technical-defects-to-severity-levels.html.
[31] P. Software, Issue severity, 2025. URL: https://help.klocwork.com/2025.4/en-us/concepts/issueseverity.htm.
[32] SonarSource, Sonarqube rules, 2026. URL: https://docs.sonarsource.com/sonarqube-server/2026.1/quality-standards-administration/managing-rules/rules#mqr-severity-types.
[33] A. Security, Advanced sast, built for developers, 2026. URL: https://www.aikido.dev/code/sast-engine.
[34] A. Security, How is severity score calculated, 2025. URL: https://help.aikido.dev/getting-started/core-functionalities/how-is-severity-score-calculated.
[35] M. Corporation, Prioritizing common weaknesses based upon your environment, 2023. URL: https://cwe.mitre.org/community/swa/priority.html#detection_methods.