Environment-Aware Vulnerability Suppression Using Kubernetes Security Contexts and VEX

Alessio Greggi
alessio.greggi@suse.com
SUSE

Abstract

SBOM(Software Bill of Materials)-based vulnerability scanners frequently report vulnerabilities that are theoretically present but practically non-exploitable due to environmental constraints. This phenomenon contributes to alert fatigue and complicates vulnerability prioritization, particularly in large-scale Kubernetes environments.

This paper explores an environment-aware approach to vulnerability suppression based on Kubernetes Security Context [1] and the Vulnerability Exploitability eXchange (VEX)[2] specification. By leveraging Common Weakness Enumeration (CWE)[3] classifications and analyzing workload deployment manifests, a heuristic method is proposed to identify vulnerabilities. These vulnerabilities are mitigated by the configuration and by expressing this information through automatically generated VEX documents.

In conclusion, this exploratory approach demonstrates how security-platform controls can be incorporated into vulnerability assessment pipelines to improve signal quality and reduce noise.

1 Introduction

Vulnerability scanning has become a foundational practice in modern software supply chain security. Tools based on SBOM analysis enable the detection of known vulnerabilities across complex dependency graphs. However, these tools often lack contextual awareness of the runtime environment in which applications execute.

As a result, vulnerability reports frequently include findings that are not exploitable in practice. This disconnect increases the cognitive burden on security teams and shifts effort toward manual triage rather than remediation. Some research highlights a very low percentage of vulnerabilities that are actively exploited in the wild[4].

The Vulnerability Exploitability eXchange (VEX) specification addresses this issue by providing a standardized mechanism to communicate exploitability assessments. However, VEX documents are typically authored manually by security experts or developers with deep system knowledge.

[1] https://kubernetes.io/docs/tasks/configure-pod-container/security-context/
[2] https://www.ntia.gov/files/ntia/publications/vex_one-page_summary.pdf
[3] https://cwe.mitre.org/about/new_to_cwe.html
[4] https://www.theregister.com/2021/02/18/cve_exploitation_2_6pc_kenna_security/

This paper investigates whether exploitability assessments can be partially automated by using the declarative security constraints provided by Kubernetes.

2 Background

2.1 VEX

VEX is a machine-readable format for communicating the exploitability status of software vulnerabilities. It allows producers or operators to assert that a vulnerability is not exploitable due to specific conditions, such as unreachable code paths or mitigating controls.

VEX is supported by modern vulnerability scanners, including Trivy [5] and Grype[6], enabling suppression of findings when appropriate justification is available.

2.2 Kubernetes Security Context

Kubernetes Security Context allows operators to constrain container execution through declarative configuration. Examples include disabling privilege escalation, enforcing read-only filesystems, and restricting Linux capabilities.

These controls are commonly applied to reduce the attack surface but are not typically considered during vulnerability assessment.

2.3 CWE Classification

Each CVE[7] is commonly associated with one or more Common Weakness Enumerations (CWE), which describe abstract categories of software weaknesses. CWEs provide a potential bridge between individual vulnerabilities and generalized mitigation strategies.

3 Problem Statement

Current SBOM-based vulnerability scanning approaches treat exploitability primarily as a property of the software artifact itself. This ignores environmental constraints that may prevent exploitation partially or entirely.

The central research question explored in this paper is:

Can Kubernetes Security Context be used to systematically infer that certain classes of vulnerabilities are mitigated by configuration? In addition, can this information be expressed via automatically generated VEX documents?

4 Methodology

The proposed approach consists of three main steps:

[5] https://github.com/aquasecurity/trivy
[6] https://github.com/anchore/grype
[7] https://www.cve.org/about/overview

1. ML-Driven Exploitation Classification and Mitigation Mapping
2. Static Manifest Analysis
3. VEX Document Generation

4.1 ML-Driven Exploitation Classification and Mitigation Mapping

To enhance the granularity of vulnerability suppression, the proposed methodology combines the CWE classes together with a machine learning text classifier designed to predict specific *exploitation categories* associated with a given vulnerability description. A Multinomial Naive Bayes classification algorithm was employed for model training. The dataset consists of approximately 2,000 entries. The trained model is publicly available at: `https://github.com/alegrey91/vex8s-model`

This approach is based on the architecture proposed in (Yosifova et al., 2021) and moves beyond broad CWE categorizations. Instead, the model analyzes the vulnerability context to classify potential threats into $0 \leq x < N$ of the five distinct *exploitation categories* listed below:

- `arbitrary_file_write`
- `arbitrary_file_read`
- `system_privileges_escalation`
- `application_privileges_escalation`
- `resource_exhaustion`

These categories are defined according to the following rationale: *for each system-oriented exploitation category that can be blocked by the environment, a corresponding opposite category is defined to enable the model to distinguish it from its counterpart (if this exists).* For example, the `arbitrary_file_write` (mitigable using a read-only filesystem) and its counterpart `arbitrary_file_read` (that cannot be blocked).

The predicted labels serve as the basis for a heuristic mapping engine, which links specific exploitation vectors to countervailing Kubernetes Security Context constraints. The idea is that if a deployment environment structurally prevents the required exploitation category, the vulnerability can be considered neutralized.

Case Study: CVE-2007-4559 This vulnerability is classified under CWE-22[8] (Improper Limitation of a Pathname to a Restricted Directory). Vulnerabilities in this class are commonly associated with directory traversal flaws and typically require the ability to read from or write to sensitive areas of the filesystem. In particular, CVE-2007-4559[9] allows user-assisted remote attackers to overwrite arbitrary files by exploiting directory traversal sequences (e.g., "..").

Because CWE-22 is identified as a class that may be potentially mitigable depending on the exploitation modality, the description of CVE-2007-4559 is analyzed using the machine learning model to refine the understanding of its root-cause exploitation characteristics.

The official CVE description is reported below:

Directory traversal vulnerability in the (1) extract and (2) extractall functions in the tarfile module in Python allows user-assisted remote attackers to overwrite arbitrary files via a .. (dot dot) sequence in filenames in a TAR archive, a related issue to CVE-2001-1267.

[8] https://cwe.mitre.org/data/definitions/22.html
[9] https://nvd.nist.gov/vuln/detail/CVE-2007-4559

After processing this description, the machine learning model classifies the exploitation vector as `arbitrary_file_write`, indicating that successful exploitation requires the ability to write to the filesystem.

4.2 Static Manifest Analysis

Kubernetes workload manifests are statically parsed to extract relevant Security Context settings at both Pod and container levels. These settings are evaluated against the mitigation rules associated with each CWE.

If all required conditions are satisfied, the corresponding CVE is marked as mitigated.

Recalling the example above, if the deployment enforces filesystem immutability through the following configuration parameters:

- `readOnlyRootFilesystem: true`
- `volumeMounts[*].readOnly: true`

The system concludes that the environment renders the `arbitrary_file_write` exploitation categorinoperable. Therefore, the vulnerability is determined to be mitigated with respect to that specific deployment context.

4.3 VEX Document Generation

For each mitigated CVE, a VEX statement is generated indicating that the vulnerability is not exploitable due to configuration-based mitigations. Vulnerability scanners can then consume these statements to suppress findings in the next vulnerability reports.

5 Discussion

This approach treats deployment configuration as first-class security metadata. Rather than relying solely on software composition, it incorporates platform-level guarantees into exploitability assessment.

The method is particularly well-suited for Kubernetes environments, where security controls are declarative and lend themselves to static analysis. By operating at the CWE level, the approach generalizes across individual CVEs while maintaining a reasonable level of abstraction.

However, this generalization is also a source of imprecision, as discussed in the following section.

6 Limitations

The proposed method has some limitations:

1. **Heuristic Nature**: CWE classes are broad, and not all vulnerabilities within a class share identical exploitation requirements. This may result in false positives or false negatives.
2. **Static Analysis Only**: The approach does not account for runtime behavior, kernel-level vulnerabilities, or external attack vectors that bypass container-level constraints.

3. **Non-Substitutive**: This method does not replace expert vulnerability analysis but rather aims to reduce noise and improve prioritization.

7 Conclusion

This paper presents an exploratory approach to environment-aware vulnerability suppression by combining Kubernetes Security Context analysis with CWE-based reasoning and VEX generation.

By encoding platform-level mitigations into structured exploitability metadata, vulnerability reports can more accurately reflect real-world risk. While the approach is inherently heuristic and configuration-dependent, it demonstrates the potential value of integrating orchestration semantics into software supply chain security workflows.

An implementation of what was discussed in this paper can be found in the following repositories:

- `github.com/alegrey91/vex8s`
- `github.com/alegrey91/vex8s-model`

References

Yosifova, V., Tasheva, A., and Trifonov, R. (2021). Predicting vulnerability type in common vulnerabilities and exposures (cve) database with machine learning classifiers. *IEEE Xplore*.