DOI: https://doi.org/10.15276/aait.09.2026.27
UDC 004.45

Modelling security degradation of base Docker images using a rate-based metric

Bohdan S. Leshchenko[1)]
ORCID: https://orcid.org/0009-0001-5781-3518; zogyyy1@gmail.com. Scopus Author ID: 58736905200
Andrii A. Yefimenko[1)]
ORCID: https://orcid.org/0000-0003-2128-4797; yefimenko.andrii@gmail.com. Scopus Author ID: 57216846685
[1)] Zhytomyr Polytechnic State University, 103, Chudnivska Str. Zhytomyr, 10005, Ukraine

ABSTRACT

Topicality. Base Docker images are reused as long-lived software-supply-chain artifacts in pipelines. Unlike runtime-oriented software ageing, their degradation is primarily informational: the image may remain unchanged, while its scanner-observable security state changes as new vulnerabilities are disclosed and vulnerability databases evolve. Point-in-time scanner outputs identify known vulnerabilities at the scan date, but they do not show how quickly vulnerability burden has accumulated relative to image age, base operating-system family, and dependency footprint. This creates a scientific and technical contradiction: long-term base-image selection requires reproducible comparison of functionally suitable immutable artifacts, while existing indicators mainly describe current vulnerability state, static severity burden, package freshness, or short-term exploitability priority. Therefore, DevSecOps teams lack sufficient evidence for selecting candidates with a longer observable security horizon and lower expected security-maintenance burden. **Purpose and objectives.** The purpose of this study is to improve the scientific and practical grounding of long-term base Docker image selection in pipelines by formalizing observable security degradation as an age-normalized and base-family-contextualized accumulation process. To achieve this purpose, the study develops and empirically evaluates a layered comparative measurement model for identifying base-image candidates with longer observable security horizons. The objectives are to formalize severity-weighted vulnerability burden, normalize this burden by image age, contextualize degradation within base-image families, compare candidate variants at repository level, and evaluate whether the proposed metric provides information beyond raw vulnerability counts, static severity burden, vulnerability density, image age, and exploitability-oriented signals. **Methods.** The study uses a time-windowed dataset of Linux-based Docker images from popular Docker Hub repositories. Vulnerabilities are identified using a four-scanner ensemble consisting of Trivy, Grype, Docker Scout, and Snyk, deduplicated by vulnerability identifier, weighted by severity, and normalized by image age. The model is evaluated through family-level comparison, repository-level candidate comparison, and threshold-oriented security-horizon interpretation. **Results.** The proposed model links current vulnerability burden with age-normalized degradation intensity and interprets this signal within base-family, repository-level, and threshold-oriented contexts. Base operating-system families differ substantially. Repository-level comparisons show lower degradation intensity for reduced-footprint candidates than for higher-footprint candidates within same-ecosystem contexts, and lower degradation intensity for Alpine-based and Ubuntu-based candidates than for Debian-based candidates in repositories where direct comparison is available. **Conclusions.** The novelty of the study lies in formalizing base Docker image selection as a degradation-rate comparison problem. The proposed model differs from point-in-time vulnerability counts, static severity summaries, technical-lag measures, Exploit Prediction Scoring System, or Known Exploited Vulnerabilities, the proposed model provides an artifact-level signal for comparing candidate base images by the observed rate at which scanner-detected vulnerability burden accumulates over image age. The model is a reproducible comparative measurement procedure for long-term base-image selection. Practically, the model supports DevSecOps teams in selecting base-image candidates, prioritizing rebuilds, defining review policies, and governing pipelines by comparing images according to current vulnerability burden, observed degradation rate, base ecosystem, dependency-footprint context, and threshold-oriented security horizon.

Keywords: Software ageing; technical debt; container security; Docker images; CI/CD governance; base image selection; vulnerability accumulation

***For citation*: Leshchenko B. S., Yefimenko A. A. "Modelling security degradation of base Docker images using a rate-based metric". *Applied Aspects of Information Technology*. 2026; Vol.9 No.3: 396–414. DOI: https://doi.org/10.15276/aait.09.2026.27**

INTRODUCTION

Containerisation has become a foundational technology for modern software delivery, enabling reproducible builds, dependency encapsulation, and rapid deployment. However, container images differ from traditional software artifacts in a critical aspect: once published, they are immutable snapshots of operating systems, libraries, and application dependencies. Despite the efficiency and portability of containers, such environments expose complex security risks spanning image creation, distribution, and runtime execution.

Container images, registries, CI/CD pipelines, credentials, kernel interfaces, and image distribution mechanisms form a connected attack surface. In particular, Wong et al. [1] emphasize that Docker images may contain unsafe or insufficiently patched software and that, once deployed, a container image

is not patched in place but must be rebuilt and redeployed. This supports the interpretation of container image security as a lifecycle problem. Large-scale measurements on Docker Hub confirm that patching in images is often delayed or ignored, even for high-impact flaws [2]. Consequently, container image security is inherently a *time-dependent problem*.

In practice, images that are considered secure at build time may silently accumulate vulnerabilities as new Common Vulnerabilities and Exposures [3] (CVEs) are disclosed. This happens because the image embeds operating-system packages, runtime components, libraries, and inherited base layers whose known vulnerability status changes as vulnerability databases are updated. This lifecycle-oriented interpretation is consistent with cloud-security research, where adaptive security strategies, continuous monitoring, and rapid response are treated as necessary responses to evolving threats in cloud environments [4].

Although multiple quality attributes of Docker images exist, empirical evidence indicates that developers primarily respond to visible signals such as size and officiality, while deeper quality aspects have limited impact on actual adoption [5]. These approaches provide limited insights into how quickly an image accumulates vulnerability burden if left unchanged.

Despite being permanently unmaintained, many deprecated images continue to receive substantial pull traffic, in some cases exceeding hundreds of thousands or even millions of weekly pulls.

These images accumulate vulnerabilities irreversibly, as no upstream fixes are possible; however, they remain widely deployed in production environments, posing significant security risks to organisations that rely on them for critical applications.

Table 1 summarize frequently accessed deprecated Docker repositories and shows that several unmaintained images still receive substantial pull traffic. This example illustrates that container images may remain operationally relevant long after their maintenance horizon has ended, which strengthens the need for degradation-aware comparison of base-image candidates.

Related evidence also shows that Dockerfiles accumulate technical debt over time, increasing image size and build duration when refactoring is delayed [6]. This connects container maintenance with the broader problem of long-term artifact degradation.

Table 1. **Deprecated Docker Repositories**

Repository Name	Weekly Pulls	Last updated
google/cadvisor	4,464,388	Aug 16, 2021
portainer/portainer	211,620	Nov 21, 2022
centos	116,824	Dec 9, 2022
nats-streaming	47 116	Jun 12, 2024
sentry	17 322	Oct 19, 2019

***Source*: compiled by the authors**

The scientific and technical problem addressed in this study arises from a contradiction between the long-term role of base Docker images in DevSecOps pipelines and the short-term nature of commonly used security assessment indicators. A base image is selected as a reusable upstream software-supply-chain artifact and may influence many downstream builds. However, its observable security state changes over time because newly disclosed vulnerabilities become associated with embedded packages and inherited layers. Therefore, base-image selection is not only a compatibility, size, or current-vulnerability-count problem. It is also a long-term security-management problem. Security of such artifacts should be treated not as a late-stage verification activity but as a process integrated into design, validation, and vulnerability-prevention practices throughout software development [7].

For DevSecOps teams, this means that when several base-image candidates are functionally acceptable, the current vulnerability state alone is not enough for selection. A candidate with fewer current findings may still accumulate vulnerability burden faster than an alternative with a slightly higher present burden but a slower observed degradation rate.

The unresolved contradiction can be formulated as follows. On the one hand, DevSecOps practice requires stable, reproducible, and operationally interpretable criteria for selecting base Docker images that will remain maintainable and less burdensome over time. On the other hand, the dominant measurement signals describe the current vulnerability state, package freshness, or exploitability priority, but do not represent vulnerability accumulation as an artifact-level degradation process normalized by image age and interpreted within base-image family context.

Point-in-time vulnerability counts are therefore insufficient for this decision problem. They show how many findings are visible at scan time, but they do not distinguish between images that accumulated comparable burden over different lifecycle intervals. They also mix the effects of dependency footprint, scanner coverage, disclosure dynamics, base

operating-system family, and vulnerability-reporting practices.

Thus, the unresolved scientific and technical problem is the insufficient reproducible support for long-term selection and governance of immutable base Docker images whose scanner-observable security state changes over time without changes to the artifact itself. This problem arises because newly disclosed vulnerabilities and evolving vulnerability databases can increase the observed vulnerability burden of embedded packages and inherited layers, while commonly used assessment indicators mainly describe the current vulnerability state, package freshness, or short-term exploitability priority. As a result, DevSecOps teams cannot reliably distinguish between functionally suitable base-image candidates that differ not only in current vulnerability burden, but also in the rate at which this burden has accumulated relative to image age, base operating-system family, and dependency footprint.

This distinguishes container-image degradation from classical runtime-oriented software ageing. In classical software ageing, degradation is usually associated with resource exhaustion, memory leaks, state accumulation, or performance decay during execution. In contrast, an immutable Docker image may remain unchanged at the byte level while its observable security state deteriorates because vulnerability knowledge about its embedded components changes over time. Therefore, the degradation considered in this study is not runtime degradation, but informational security degradation of a software-supply-chain artifact.

The purpose of this study is to improve the scientific and practical grounding of long-term base Docker image selection in CI/CD by formalizing observable security degradation as an age-normalized and base-family-contextualized accumulation process, and by evaluating whether this representation supports the identification of functionally suitable candidates with a longer observable security horizon. To achieve this purpose, the study develops and empirically evaluates a comparative rate-based model for selecting base Docker images with longer observable security horizons.

The objectives of the study are as follows:

1) to formalize the vulnerability burden of a Docker image using deduplicated scanner-observable vulnerability identifiers and severity-based weighting;

2) to define the Docker Image Degradation Coefficient as an age-normalized measure of vulnerability burden accumulation;

3) to contextualize degradation intensity within base operating-system families in order to reduce over-interpretation of raw cross-family differences;

4) to compare candidate image variants at repository level;

5) to evaluate whether the proposed degradation coefficient provides information beyond raw vulnerability counts, static severity burden, vulnerability density, image age, and short-term exploitability-oriented indicators.

The scientific novelty of the study lies in formalizing base Docker image selection as a degradation-rate comparison problem. The proposed model treats scanner-observable vulnerability evidence as severity-weighted burden accumulated over the observable lifecycle of an immutable artifact.

The main contribution is a layered comparative measurement model that combines absolute vulnerability burden, age-normalized degradation intensity, family-normalized interpretation, and repository-level candidate comparison. The model provides an artifact-level signal for comparing functionally suitable base-image candidates by the observed rate at which vulnerability burden accumulates over image age.

The proposed Docker Image Degradation Coefficient evaluates vulnerability burden as a time-dependent accumulation process rather than as a scan-time count. This makes it possible to support DevSecOps decisions not only by asking how many vulnerabilities are visible in an image at scan time, but also by asking which candidate base image has accumulated scanner-visible vulnerability burden more slowly over its lifecycle. Practically, the model can support base-image selection, rebuild prioritization, review policies, and long-term CI/CD governance.

RELATED WORKS

A. Container Vulnerability Measurement

Research on container security has developed in several closely connected directions: general threat modelling of container ecosystems, empirical vulnerability measurement of Docker images, analysis of technical lag and outdated packages, investigation of base-image vulnerability propagation, and studies of Docker image quality and adoption. These works provide an important empirical and conceptual foundation for the present study. However, these approaches do not fully convert vulnerability evidence into a rate-based selection criterion for comparing immutable base-image candidates.

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

Empirical vulnerability measurement studies have shown that Docker images frequently contain vulnerabilities and that these vulnerabilities may persist for long periods. Kaur et al. [2] demonstrated that updating system packages removes a large share of detected vulnerabilities and reduces the vulnerability-to-package ratio, but updates do not eliminate all vulnerabilities and may be constrained by end-of-life distributions or broken package states. Wist et al. [8] analysed more than 2,500 Docker Hub images and showed that images frequently contain vulnerabilities of different severity levels, while official images tend to be less vulnerable than other image categories. Their results characterize vulnerability exposure across Docker Hub images, but they do not estimate how quickly this exposure accumulates relative to image age.

This research direction mainly answers which vulnerabilities are visible at scan time. For base-image selection, however, this is only the starting point: scanner-based measurement does not show whether the observed burden accumulated slowly or rapidly relative to image age. These measurements motivate the need for a rate-based view, but they do not themselves provide a criterion for comparing immutable base-image artifacts over their observable lifecycle.

B. Longitudinal Trends in Docker Image Vulnerabilities

Longitudinal studies have consistently shown that the number of vulnerabilities in Docker images tends to increase over time, even for images that receive regular updates [9]. Mills et al. [10] found that the number of vulnerabilities increased over time despite regular image updates and that the underlying operating system can strongly affect reported vulnerability counts, with Debian-based images often showing more vulnerabilities than other Linux distributions. Their work is relevant to the temporal dimension of container security because it shows that vulnerability exposure changes over time and that scanner output requires contextual interpretation.

Nevertheless, longitudinal studies often follow evolving image tags across several analysis periods. Therefore, the observed changes may combine image aging, maintenance actions, rebuilds, dependency changes, vulnerability disclosure dynamics, and scanner/database behaviour. The present study differs by treating Docker images as immutable artifacts observed under a controlled measurement design and by using age-normalized vulnerability burden as a comparative signal for base-image selection.

Zerouali et al. [11] applied the concept of technical lag to Docker images by measuring the difference between installed package versions and the most recent available versions. This perspective links container maintenance to dependency freshness and delayed updates. Their empirical analysis of Debian-based Docker images shows that containers often include outdated packages and that vulnerabilities and bugs may remain unresolved for extended periods. This study frames container maintenance as a dependency freshness and technical-lag problem. However, the scope of their work is limited to Debian-based images and package-level lag.

Technical lag and degradation rate describe related but different aspects of container security. Technical lag measures how outdated installed packages are compared with available package versions. The proposed degradation model measures how much severity-weighted vulnerability burden has accumulated relative to image age. Thus, technical-lag analysis explains dependency freshness, while the our analysis focuses on artifact-level degradation intensity and candidate comparison. This distinction is important because a package may be outdated without currently contributing to scanner-derived vulnerability burden, while a relatively recent image may still accumulate severe findings quickly due to its dependency footprint or base ecosystem.

C. Base-Image Selection and Inheritance

Docker image quality and adoption have been studied from a different angle by Rosa et al. [5] , who observe that developers often start Dockerfiles from pre-existing images and that many functionally equivalent base images may be available. Their study develops a taxonomy of externally observable and configuration-related quality features and evaluates which features influence developers' preferences and adoption behaviour. Rosa et al. show that developers' image choices are affected by observable properties such as officiality, size, and security-related signals, although these signals do not always translate into actual adoption decisions.

This work motivates the practical need for better decision-support metrics. If developers face multiple functionally suitable base-image alternatives, then a degradation-aware criterion can complement existing observable features by capturing the rate at which vulnerability burden accumulates over time. In this sense, base-image selection should not be reduced to a one-time comparison of image size, officiality, or current scanner output.

In their empirical study [12], Haque and Babar showed that a significant portion of vulnerabilities are inherited from the base images and propagate to all derived images. They show that vulnerabilities in a base image can propagate to derived images and, in an illustrative case, that most vulnerabilities in an application image may be inherited from its base-image chain. This provides strong evidence that base-image selection is a security-critical decision rather than a purely technical convenience.

However, Haque and Babar mainly investigate exploitability and impact characteristics of existing base-image vulnerabilities. They do not model how quickly different base-image candidates accumulate vulnerability burden over their lifecycle, nor do they propose a rate-based metric for long-horizon base-image selection. Therefore, existing base-image inheritance studies justify why base-image selection matters, but they do not fully formalize how candidate base images should be compared according to observable degradation intensity.

Image size and build practices form another relevant line of evidence. Durieux [13] demonstrated that inefficient image-build practices significantly affect image size, while Kaur et al. [2] demonstrated that package updates can remove a large share of detected vulnerabilities but do not eliminate all findings, especially under end-of-life or broken-package constraints. Related evidence also shows that image bloat can increase over time, particularly in community images [14]. These findings support treating dependency footprint and image size as contextual factors in degradation-aware base-image comparison.

A further unresolved issue concerns the distinction between dependency-footprint reduction and base-ecosystem substitution. Existing studies often discuss smaller images, slim variants, Alpine-based images, and image minification as related security-improvement strategies. However, these strategies are not equivalent. A slim image may reduce dependency footprint within a similar ecosystem, while an Alpine-based image may change the base operating-system ecosystem, package manager, libc implementation, and vulnerability-reporting context. For this reason, this work separates content-derived candidate classes and repository-level pairwise comparisons, rather than treating all smaller or minimized images as one homogeneous category.

D. Scanner Limitations

Vulnerability databases overlap, disagree on severity (CVSS vectors), and suffer from delays, inaccuracies, and duplication [15]. Existing research [16] highlights that poor usability of vulnerability scanners for Docker images contributes to ineffective vulnerability handling and the persistence of security issues in container images. Therefore, each image is analysed using a multi-scanner ensemble consisting of Trivy [17], Grype [18], Snyk and Docker Scout [19]. Related research on software-vulnerability identification also emphasizes that security-analysis methods may suffer from limited accuracy under uncertain or fuzzy input data, which supports treating scanner-derived vulnerability findings as approximate measurement evidence rather than definitive security truth [20].

Existing container image scanning tools significantly underestimate security risk due to their reliance on package manager metadata, failing to detect vulnerabilities in compiled or manually added components; in contrast, extended analysis approaches improve detection coverage by over 50% and reveal that up to 68 % of real-world container images contain known vulnerabilities [21].

Scanner limitations do not eliminate the need for a degradation model. Rather, they create a measurement-design requirement: the model must be reproducible, deterministic, and explicit about how vulnerability evidence is aggregated. For this reason, scanner/database limitations are treated as part of the measurement problem. The proposed model uses synchronized scanning, recorded scanner versions, database snapshot tracking, deterministic aggregation, and severity-weighted burden construction to reduce avoidable measurement ambiguity.

E. Unresolved Gaps

Taken together, the reviewed studies explain vulnerability presence, inheritance, package freshness, image quality, and scanner limitations.

However, they mainly explain which vulnerabilities are present, how they are distributed or inherited, and how updates, package freshness, or minification affect current vulnerability counts. They do not fully address base-image selection as a long-term degradation-rate comparison problem.

Four limitations are especially relevant for long-term base-image selection.

First, existing scanner-based and longitudinal studies mainly characterise vulnerability presence or temporal change, but they do not formalize immutable Docker images as artifacts with an age-normalized degradation rate. Second, technical-lag and image-quality studies describe dependency freshness, image size, officiality, or adoption-related properties, but they do not integrate severity-weighted vulnerability burden, image age, and base-

family context into a single comparative selection model. Third, operating-system family effects are usually observed descriptively, but they are not consistently transformed into a family-normalized decision-support signal. Fourth, exploitability-oriented indicators such as EPSS and KEV support short-term remediation prioritization, but they do not answer the base-image selection question: which functionally suitable candidate has accumulated vulnerability evidence more slowly over its lifecycle?

The practical DevSecOps question addressed in the present study is different: when several functionally suitable base images are available, which candidate should be preferred from the perspective of longer-term observable security degradation? Answering this question requires age normalization, base-family context, and repository-level candidate comparison.

This study addresses these gaps by formalizing base Docker image selection as a comparative degradation-rate problem. The proposed model transforms scanner-observable vulnerability findings into a severity-weighted burden, normalizes this burden by image age, contextualizes the resulting degradation coefficient within base-image families, and supports pairwise comparison between content-derived candidate classes such as regular, slim, Alpine-based, Debian-based, and Ubuntu-based variants. In this way, the model connects empirical vulnerability measurement with the practical DevSecOps decision of selecting base images for long-term use in CI/CD pipelines.

RESEARCH PROBLEM AND MODEL REQUIREMENTS

Based on the gaps identified above, the model must transform heterogeneous scanner findings, image metadata, age, base-family context, and variant information into a reproducible comparison signal for candidate base images.

Let $I = \{i_1, i_2, \ldots, i_n\}$ denote a set of candidate base Docker images that may be used as upstream artifacts in CI/CD pipelines. The candidates may belong to the same repository, to different variants of the same image family, or to functionally comparable base-image alternatives. For each image $i \in I$, the following observable attributes are available or derived: scanner-reported vulnerability findings, image age, base operating-system family, image size or dependency footprint, and content-derived variant class such as regular, slim, Alpine-based, Debian-based, or Ubuntu-based.

The research problem is to transform these heterogeneous observations into a reproducible degradation signal that supports candidate comparison by both current burden and age-normalized accumulation intensity. The problem is not limited to vulnerability detection, it is a measurement and decision-support problem that integrates scanner evidence, temporal context, base-family context, and candidate structure.

For the proposed model to address this problem, it must satisfy the following requirements.

First, the model must be *reproducible*. Given the same image digest, scan timestamp, scanner outputs, vulnerability identifiers, severity information, and image metadata, the model should produce the same degradation values. This requires deterministic aggregation, explicit deduplication rules, documented scanner versions, and controlled scan timing.

Second, the model must operate at the *artifact level*, evaluating immutable Docker image digests rather than repository names, evolving tags, or package lists, because CI/CD base-image selection concerns concrete artifacts that enter downstream build chains.

Third, the model must be *age-normalized*, since static vulnerability counts or severity burden cannot distinguish between the same observed burden accumulated over short and long artifact lifecycles.

Fourth, the model must be *context-aware*, accounting for differences in package ecosystems, release cadence, vulnerability-reporting practices, and scanner coverage that make raw cross-family comparison unreliable.

Fifth, the model must support *repository-level and candidate-level comparison*. In practical DevSecOps work, engineers usually choose between candidate variants such as regular versus slim, Debian-based versus Alpine-based, or Ubuntu-based versus Debian-based images. The model should therefore support pairwise or ranked comparison of candidates within repository-level or functionally comparable groups.

Sixth, the model must produce an *interpretable* output for CI/CD decision support. The result should not be only a raw scanner report. It should provide a comparative estimate of degradation intensity that can be used for ranking candidate images, identifying unusually fast-degrading images, supporting base-image selection, and informing rebuild or review policies.

Accordingly, the expected output of the model is a structured comparative assessment of candidate base images. This assessment includes the absolute

vulnerability burden, the age-normalized degradation coefficient, the family-normalized degradation position, and repository-level candidate comparison. These outputs make it possible to rank or compare candidate images according to their observed security degradation, rather than relying only on point-in-time vulnerability counts or static severity summaries.

DEGRADATION MODEL

Based on this problem statement, the next section introduces the proposed degradation model as a layered measurement procedure that transforms scanner-observable vulnerability evidence into age-normalized and context-aware comparison signals.

A. General Structure and Functional Logic

The degradation model is designed as a layered comparative measurement procedure for immutable base Docker image artifacts. It transforms scanner-observable vulnerability evidence into a reproducible signal for comparing functionally suitable base-image candidates according to observed degradation intensity over the artifact lifecycle.

The functional logic follows a typical DevSecOps selection scenario: several candidate base images are acceptable for the same downstream task, but they differ by base operating-system family, dependency footprint, update history, image size, and variant class. Conventional scanner outputs describe their current vulnerability state, while the proposed signal expresses how quickly vulnerability burden has accumulated relative to image age.

The model operates in six consecutive stages.

In the initial stage, each Docker image is represented as an immutable artifact identified by its digest and described by artifact-level metadata. These metadata include the scan timestamp, last update timestamp, image age, base operating-system family, image size, and content-derived variant class, such as regular, slim, Alpine-based, Debian-based, or Ubuntu-based. This stage defines the unit of analysis and prevents the model from treating evolving tags or repositories as if they were stable artifacts.

Vulnerability evidence is then collected using a synchronized multi-scanner protocol. The model uses scanner outputs as observable measurement evidence rather than as complete security truth. This is necessary because scanners may differ in vulnerability databases, package-detection logic, severity reporting, and coverage of embedded components. Therefore, scanner versions, database snapshots, scan timestamps, and image metadata are recorded to support reproducibility.

The collected findings are aggregated and deduplicated by vulnerability identifier. Where the same vulnerability is reported by several scanners, the finding is treated as one vulnerability instance for the image. When severity values differ across scanners, the maximum reported severity is retained. The result of this stage is a deterministic set of unique vulnerability identifiers associated with the image.

The deduplicated set is subsequently transformed into a severity-weighted vulnerability burden. Each vulnerability contributes to the burden according to its severity class. This produces an absolute burden score that reflects both the number and the relative severity of scanner-observable findings. However, this burden is still a static measure and does not by itself express degradation over time.

This burden is normalized by image age. The resulting Docker Image Degradation Coefficient expresses the observed accumulation intensity of vulnerability burden in severity points per day. This stage is the central transformation of the model: it distinguishes between images that have the same current burden but accumulated it over different lifecycle intervals.

Finally, the degradation coefficient is contextualized and compared. At the family level, the coefficient is interpreted relative to the distribution of degradation values within the same base operating-system family. This reduces over-interpretation of raw cross-family differences caused by package ecosystems, release cadence, scanner coverage, and reporting practices. At the repository or candidate-group level, the model supports pairwise comparison between functionally comparable variants, such as regular versus slim images or Debian-based versus Alpine-based alternatives.

The output is not a single scanner score but a structured set of comparative signals: absolute vulnerability burden, age-normalized degradation intensity, family-normalized degradation position, and repository-level candidate comparison. These signals support base-image selection, review, and prioritization according to observed long-term degradation rather than only point-in-time vulnerability counts.

The procedure is original in how it connects these stages into one artifact-level comparison framework. Scanner outputs identify current findings, technical-lag measures describe

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

dependency freshness, and exploitability-oriented indicators support short-term prioritization. In this model, base-image selection is treated as a degradation-rate comparison task: vulnerability burden is normalized by image age, interpreted within base-family context, and applied to candidate-level comparison in CI/CD governance.

B. Vulnerability Aggregation and Weighting

Let V_i denote the set of unique CVEs detected in image i after deduplication across scanners, and let $s(c)$ be the severity weight assigned to CVE c, where $s(c) \in \{10, 5, 2, 1\}$ for CRITICAL, HIGH, MEDIUM, and LOW severities, respectively. The selected mapping is not intended to represent a universal risk scale. It is used as an operational severity-point scale that assigns substantially greater weight to HIGH and CRITICAL findings, reflecting the common security-management practice of treating severe findings as disproportionately more important than low-severity findings. For each CVE observed by multiple scanners, the maximum reported severity is retained.

The vulnerability burden of image i is then computed explicitly as:

$$WS_i = \sum_{c \in V_i} s\,(c) \tag{1}$$

where WS_i denotes the cumulative severity score of image i. This formulation makes the contribution of each individual vulnerability explicit and ensures that the metric scales linearly with the number and severity of distinct CVEs. We use a similar approach as other researchers [22]. Importantly, WS_i represents an absolute burden measure and does not encode temporal dynamics on its own.

C. Docker Image Degradation Coefficient

The Docker Image Degradation Coefficient (Cd) is defined by normalizing the cumulative vulnerability burden by the image age. Let A_i denote the age of image i in days, computed as the difference between the scan time and the image's last update timestamp.

The degradation coefficient is then given by:

$$Cd_i = \frac{WS_i}{A_i}. \tag{2}$$

Cd expresses security debt accumulation as points per day, enabling direct comparison across images of different ages. Because vulnerability data exhibit heavy-tailed distributions, Cd values are summarized using median and interquartile range rather than mean and standard deviation.

D. Scan Protocol and Synchronization (Time-Stamped Scans)

All scans are executed under a fixed protocol. For each image digest, we record:

- *scan_start_utc* and *scan_end_utc* (UTC),
- scanner binary version for each scanner,
- vulnerability database snapshot identifier and/or database update timestamp,
- OS/package metadata extraction timestamp.

To minimize detection drift, multi-scanner scans are treated as synchronized if their *scan_start_utc* times fall within a fixed window Δ (e.g., $\Delta = 30$minutes). If scans cannot be completed within Δ, the image is re-queued so that all scanners run within Δ using pinned database snapshots.

E. Family-Normalized Degradation Scope

Base OS families differ in package ecosystem, release cadence, scanner coverage, and vulnerability reporting practices. To support within-family comparison, the family-normalized degradation score is:

$$Z_d(i,t) = \frac{C_d(i,t) - \mathrm{median}\left(C_d^f\right)}{\mathrm{IQR}\left(C_d^f\right)}, \tag{3}$$

where f is the base image family of image i, $\mathrm{median}\left(C_d^f\right)$ is the median degradation coefficient among images in family f, and $\mathrm{IQR}\left(C_d^f\right)$ is the interquartile range of degradation coefficients in family f.

The notation Z_d denotes a robust family-normalized degradation score based on median and interquartile range, rather than a conventional mean-standard-deviation $Z_d(i,t)$ compares an image to its ecosystem context. A positive Z_d means the image degrades faster than the family median. A negative Z_d means the image degrades slower than the family median. This avoids over-interpreting cross-family differences where Debian, Alpine, Ubuntu, scratch, and other families have different package ecosystems and reporting patterns.

F. Repository-Level Candidate Comparison

The final comparison layer of the proposed model operates at the repository or candidate-group level. This layer is necessary because base-image selection in CI/CD is rarely performed by comparing all images in the ecosystem against each other. In practice, engineers usually choose between functionally comparable candidates: for example, regular and slim variants of the same image, Debian-based and Alpine-based variants of the same

runtime, or Ubuntu-based and Debian-based alternatives that can support the same downstream application.

Let $G_r = \{i_1, i_2, \ldots, i_m\}$ denote a candidate group associated with repository r, where each image i_j is a functionally comparable base-image candidate. Each image in the group is represented by its degradation coefficient $C_d(i_j)$, family-normalized degradation score $Z_d(i_j)$, base operating-system family, and content-derived variant class. The purpose of the repository-level comparison is to determine which candidate shows lower observed degradation intensity under comparable functional or repository-level constraints.

For two candidate images i_a and i_b within the same group, the pairwise degradation difference is defined as:

$$\Delta C_d(i_a, i_b) = C_d(i_a) - C_d(i_b), \quad (4)$$

where a positive value indicates that candidate i_a has higher observed degradation intensity than candidate i_b, while a negative value indicates that candidate i_a has lower observed degradation intensity.

To express the relative improvement of one candidate over another, the pairwise improvement ratio can be written as:

$$PIR(i_a, i_b) = \frac{C_d(i_a) - C_d(i_b)}{C_d(i_a)}, \quad (5)$$

where i_a is treated as the reference candidate and i_b as the alternative candidate. A positive value of PIR means that the alternative candidate has a lower degradation coefficient than the reference candidate. A value close to zero means that the two candidates have similar observed degradation intensity. A negative value means that the alternative candidate degrades faster than the reference candidate.

The comparison can be applied to two types of candidate relations. The first type is dependency-footprint reduction, where the compared images belong to the same or similar base ecosystem but differ in the number of included packages and auxiliary components. Examples include regular versus slim variants. In this case, the comparison estimates whether reducing the dependency footprint is associated with lower observed degradation intensity.

The second type is base-ecosystem substitution, where the compared images provide a similar functional role but rely on different base operating-system ecosystems. Examples include Debian-based versus Alpine-based or Debian-based versus Ubuntu-based variants. In this case, the comparison estimates whether changing the base ecosystem is associated with lower observed degradation intensity. This distinction is important because footprint reduction and ecosystem substitution may produce similar reductions in vulnerability burden while relying on different technical mechanisms and compatibility assumptions.

Repository-level comparison prevents over-generalized ecosystem conclusions. A lower median degradation coefficient for one base family across the full dataset does not mean that every candidate from that family is preferable in every repository. Candidate choice is constrained by application compatibility, available tags, runtime requirements, package-manager behaviour, and operational policy; therefore, comparable pairs are evaluated within repository-level or functionally similar groups.

The output of this layer is a pairwise or ranked assessment of candidate images. For each candidate group, the model can identify the image with the lowest degradation coefficient, estimate the relative improvement compared with a reference candidate, and indicate whether the observed advantage is related primarily to dependency-footprint reduction or to base-ecosystem substitution. Thus, repository-level candidate comparison transforms the degradation coefficient from an individual artifact metric into a practical decision-support signal for base-image selection.

G. Threshold-Oriented Security-Horizon Interpretation

The final interpretive layer of the model translates the Docker Image Degradation Coefficient into a threshold-oriented security-horizon signal. Let τ denote an operational degradation threshold defined in severity points per day or converted into severity points per year. An image is interpreted as remaining within an acceptable observable security horizon while $C_d(i) \leq \tau$.

This threshold is an operational reference point and not a universal security boundary. It helps compare how image families or candidate groups remain below a selected degradation level and is therefore used descriptively.

In practical CI/CD governance, this layer can support review policies by identifying images whose degradation coefficient exceeds the selected threshold, prioritizing faster-degrading candidates for rebuild or replacement, and comparing candidate images by their observed ability to remain below the chosen degradation level.

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

DATASET AND EXPERIMENTAL DESIGN

A. Dataset Construction

The empirical analysis is based on Docker images collected from popular repositories on Docker Hub. We define a "popular repository" operationally using Docker Hub metadata captured at a fixed dataset-freeze time T_{freeze}(UTC).

Repositories are selected by the following deterministic criteria:

1) *Public accessibility:* the repository is publicly visible and pullable on Docker Hub at T_{freeze};

2) *Popularity ranking:* the repository is within the top-*K* repositories by *Total Pulls* according to Docker Hub's metadata field pull_count (or its equivalent) returned by the Hub API at T_{freeze};

3) *Linux scope:* the repository provides at least one Linux image tag whose last_updated timestamp lies in the 900-day observation window $[T_{\text{freeze}} - 900\text{d}, T_{\text{freeze}}]$.

Primary dataset exclusion: repositories flagged as deprecated by Docker Hub at $T_{\text{freeze}}$are excluded from the primary dataset and analysed separately.

The observation window is the last 900 days prior to T_{freeze}. For each selected repository, we consider all tags returned by the tag-listing endpoint, and we retain only tags who's *last_updated* lies within the window. Images are included in the analytic sample if at least one scanner produces a successful result.

After filtering, the final analytic sample consists of 6,073 images drawn from 171 repositories.

B. Docker Hub Retrieval and Temporal Sampling

All repository metadata and tag lists are retrieved deterministically from Docker Hub and logged with timestamps.

Tags are mapped to a normalised *variant* label (e.g., alpine, slim, bookworm) using tag-name rules and then to a *base OS family* using OS metadata extracted from the image (*SBOM* or */etc/os-release* where available).

Let $T_{\min} = T_{\text{freeze}} - 900\text{d}$ and $T_{\max} = T_{\text{freeze}}$. We partition $[T_{\min}, T_{\max}]$ into $B = 5$ equal-width temporal bins with edges:

$$E_b = T_{\min} + b \cdot \frac{(T_{\max} - T_{\min})}{B}, \quad b \in \{0,1,2,3,4,5\}. \tag{6}$$

For each (repository, variant), the target is up to $M = 5$ images (one per bin) subject to availability.

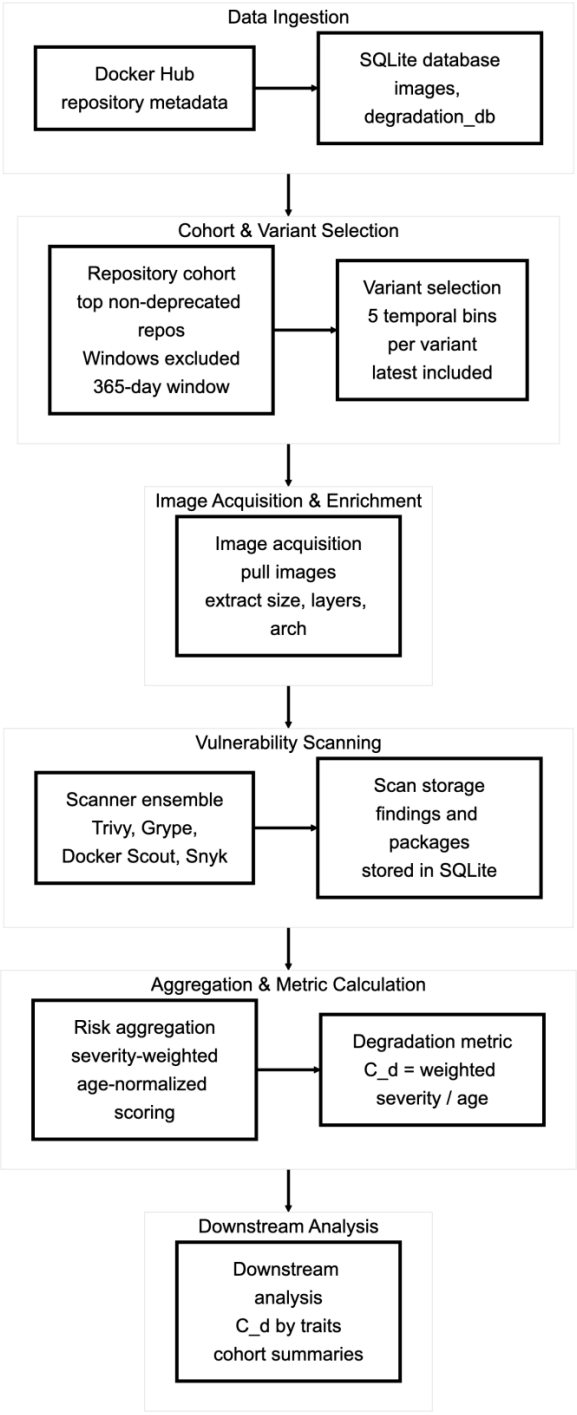

Fig 1. **Docker Image Degradation Measurement Pipeline**
***Source*: compiled by the authors**

If multiple candidate tags fall inside a bin, we select one tag by:

1) sort candidates by *last_updated* ascending;

2) select the *median* element;

If a bin contains no candidate tags for a (repo, variant), we select the nearest available bin by temporal distance:

1) search outward to the nearest non-empty bin on the left and/or right;

2) if both sides exist at equal distance, choose the older side (left) to preserve age coverage;

3) apply the within-bin selection rule to that bin's candidates.

If the fallback selects a digest already selected for another bin within the same (repo, variant), select the next nearest candidate by time (stable tie-break by digest).

Fig. 1 shows how repository selection, image acquisition, scanner-based vulnerability collection, aggregation, metric calculation, and downstream analysis are connected in the measurement pipeline.

The primary procedure is deterministic and uses no randomness.

EMPIRICAL RESULTS

A. Global Degradation Characteristics

Across the full dataset, Cd exhibits a strongly skewed distribution with a long right tail. While many images cluster at low degradation rates, a small number of images accumulate security debt at extremely high rates. This distributional shape motivates the use of robust statistics throughout the analysis.

B. Base Operating System Family Effects

Base operating system family is a primary modulator of degradation rate. Median Cd ranges from 0.72 for Alpine (N = 1,432) to 2.71 for Debian (N = 3,031). Alpine-based images exhibit the lowest median Cd, while Debian-based images show higher median values and substantially wider dispersion.

Fig. 2 supports two observations. First, family medians are visibly separated, indicating that ecosystem membership affects degradation intensity. Second, within-family dispersion is also large, especially for Debian, which means that a family label alone is insufficient for prioritization.

However, when vulnerability counts are normalized by package inventory and exposure time, the distributions overlap considerably, indicating that observed differences largely reflect ecosystem maintenance practices and dependency breadth rather than intrinsic operating system security.

C. Image Size and Dependency Inventory

Image size shows a strong positive association with Cd, indicating that larger images generally degrade faster. Image size and Cd show a Spearman correlation [23] of $\rho = 0.65$ with a 95% CI of [0.63, 0.67]. When normalized by package count, this association weakens but remains detectable, suggesting that residual image bloat contributes modestly to degradation risk beyond dependency count alone.

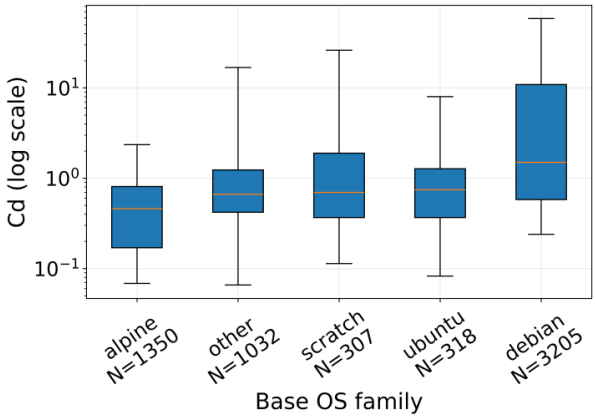

Fig 2. **Distribution of Docker Image Degradation Coefficient by base operating system family**

***Source*: compiled by the authors**

Fig. 3 shows that larger images tend to occupy higher parts of the degradation distribution, which is consistent with broader dependency footprint increasing scanner-observable exposure. However, the relationship is not deterministic, so image size should be treated as an informative correlate rather than as a substitute for the degradation coefficient.

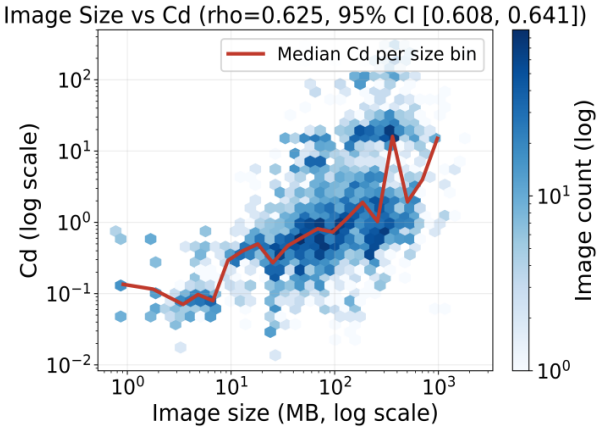

Fig 3. **Relationship between image size and Docker Image Degradation Coefficient on a log-log scale**

***Source*: compiled by the authors**

This result should not be interpreted as a causal statement. A likely explanation is that larger images often contain more operating system packages, language dependencies, build tools, or auxiliary utilities. These components increase the observable dependency inventory and therefore the number of potential vulnerability matches.

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

D. Variant Minification

The median repository-level Cd is 4.3 for regular variants versus 1.7 for slim/alpine variants across 72 paired repositories. Slim [24] and Alpine variants generally exhibit lower median Cd values than their regular counterparts. Paired comparisons at the repository level show that most repositories benefit from variant minification. However, severity composition analysis indicates that minification primarily reduces low- and medium-severity findings, while the relative proportion of critical vulnerabilities remains comparable.

Fig. 4 shows the share of images in each base-image family whose family-normalized degradation score exceeds two outlier thresholds. Values above $Z_d > 1$ identify images degrading substantially faster than the median image in the same family, while values above $Z_d > 2$ identify stronger within-family outliers.

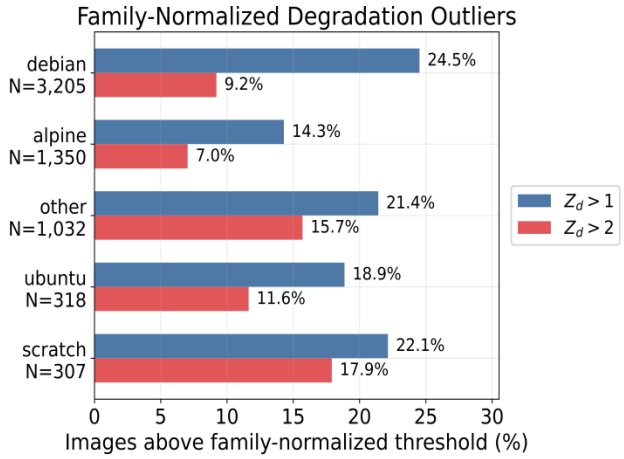

Fig 4. **Family-normalized degradation outliers by base operating system family**
***Source*: compiled by the authors**

The plot shows that high degradation is not only a cross-family effect: every major family contains a visible high-degradation tail. Debian has the largest share of moderate outliers, with 24.5 % of images above $Z_d > 1$, which is important because Debian also dominates the cohort by sample size. This supports a more conservative operational interpretation: a Debian image with high C_d may reflect both ecosystem-level package and reporting characteristics and image-specific accumulation, whereas a high Z_d image is elevated relative to images in the same family. In practice, Z_d is therefore useful for prioritizing family-specific review queues and for avoiding over-interpretation of raw cross-family differences.

E. Maintenance Velocity

As a secondary descriptive check on whether simple repository activity can replace the proposed model, repository update frequency was compared with repository median C_d from the same frozen primary cohort. Update frequency was operationalized as retained cohort images per month of observed tag-release span within the 900-day window, restricted to repositories with at least 20 images and at least 30 observed span days. Under this executable core path, the association is weak and negative: Spearman $\rho = -0.171$ with 95% bootstrap CI $[-0.350, 0.018]$ across 107 repositories.

Because the confidence interval includes zero, this result should be interpreted as secondary descriptive evidence rather than as a stable standalone predictor. The practical conclusion is still that update frequency alone is not a reliable proxy for long-term image security. Rebuild quality, dependency changes, base image refreshes, and whether vulnerable components are actually removed all matter. A frequently updated repository may still retain a high degradation coefficient if updates introduce or preserve vulnerable dependencies.

This auxiliary result supports the broader interpretation that neither age alone nor update activity alone can replace rate-based degradation measurement.

F. Threshold-Oriented Security-Horizon Analysis

After burden, context, structure, and pairwise comparison, the final model layer is security-horizon interpretation. Fig. 5 shows the empirical probability of remaining below the selected degradation threshold as images age under a single-observation design. The curve estimates should be interpreted descriptively: the event is threshold status at observation, defined by $C_d \geq \theta_d$, not system compromise or exploitation. The threshold is $\theta_d = 10/365$ severity points per day, equivalent to 10 severity points per year.

Fig. 5 translates the degradation coefficient into a governance-oriented timescale. What can be understood from the figure is not when a specific image truly first crossed the threshold, but how long images from different families tend to remain below the operational threshold under the current age-at-observation construction.

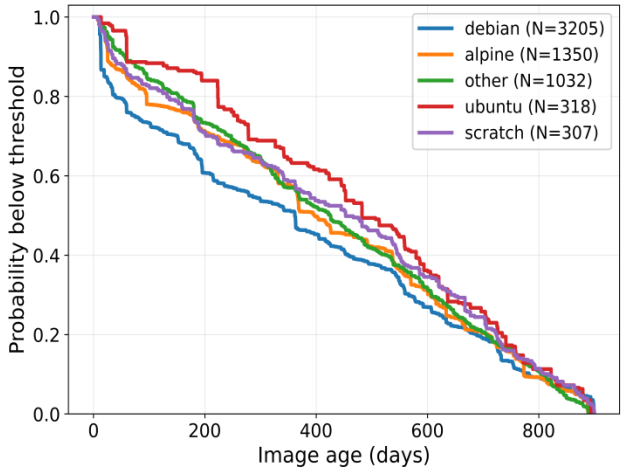

Fig 5. **Threshold-retention curves by base OS family**

***Source*: compiled by the authors**

Under this cross-sectional construction, the estimated median threshold-retention ages are ~352 days for Debian images and ~408 days for Alpine images.

The restricted mean time below threshold over 0-900 days is 360.8 days for Debian, 424.9 days for Alpine, 471.4 days for Ubuntu, and 433.2 days for scratch images.

As a descriptive comparison, the Debian-versus-Alpine log-rank statistic is z=8.108 with p<0.001. The log-rank statistic is reported only as a descriptive comparison of threshold-retention curves constructed from age-at-observation data.

G. Comparison with Alternative Metrics

To assess whether the degradation coefficient adds information beyond conventional scanner summaries, the analysis compares it with raw vulnerability count, severity burden, vulnerability density, image age, family-normalized degradation, and an optional exploitability-weighted variant.

A raw vulnerability identifier count captures how many vulnerability identifiers are detected in an image at a given scan time. Its main limitation is that it ignores both severity and image age. A severity burden measure improves on raw counts by weighting vulnerabilities according to severity [10], but it remains a static snapshot: older images naturally have more time to accumulate vulnerability burden. Vulnerability density relates vulnerability count to the size of the package inventory, but it still ignores time and depends on the quality of package detection. An age-only metric captures lifecycle exposure, but it does not account for the actual number or severity of vulnerabilities.

The proposed degradation coefficient combines severity-weighted vulnerability burden with image age. It therefore measures degradation as accumulation intensity rather than as a static burden. The family-normalized score places this degradation intensity in the context of comparable images from the same family. This makes it possible to distinguish images that are unusually degraded relative to their ecosystem from images that merely belong to generally higher-risk families.

These results indicate that C_d is related to existing static assessment methods but is not reducible to them. The moderate overlap with raw vulnerability identifier count and static severity burden suggests that a substantial share of high-degradation images would be missed or ranked differently by conventional point-in-time metrics. The weak association with vulnerability density indicates that dependency inventory alone does not explain degradation intensity. The near-zero top-decile overlap with age-only ranking confirms that age by itself is not a sufficient proxy for degradation. In practical terms, policies based only on raw vulnerability count, static severity burden, vulnerability density, or age-only ranking would answer different selection or prioritization questions.

EPSS and KEV express a related but different exploitability-prioritization problem and are discussed here as conceptual complements rather than as artifact-level joined metrics in the executable analysis path.

The model complements established container image security assessment approaches by reorganizing their outputs into a layered comparison framework for immutable image artifacts that differ in age, ecosystem context, and candidate composition.

The proposed model uses vulnerability identifiers from multiple scanners, severity weights, image age, family labels, and content-derived candidate classes to produce a layered set of comparative signals. The burden component remains compatible with conventional severity aggregation. The degradation coefficient adds age normalization and distinguishes between the same vulnerability burden accumulated over different lifecycle intervals. The family-normalized score provides ecosystem-relative interpretation, which is usually absent from static scanner summaries.

DISCUSSION

A. Interpretation of the Results

The model should be interpreted as a comparative measurement procedure rather than as a

causal explanation of vulnerability emergence. Its role is to transform scanner findings into a rate-based signal that makes candidate base images comparable across age, family, and content-derived context. The family-normalized score reduces over-interpretation of raw cross-family differences, while the pairwise improvement ratio supports practical candidate comparison under imperfect vulnerability data.

The empirical results show that degradation intensity is not distributed uniformly across the analysed Docker image ecosystem. Base operating-system family, dependency footprint, image size, and content-derived candidate class all shape how vulnerability burden appears in scanner outputs. Debian-based images show higher median degradation intensity than Alpine-based images in the analysed cohort, while reduced-footprint variants generally show lower degradation coefficients than higher-footprint variants in repository-level comparisons. These findings reflect scanner-visible vulnerability accumulation under the dataset, scanners, and protocol used in this study, not a universal ranking of operating-system security.

The threshold-oriented security-horizon analysis should also be interpreted descriptively. It does not estimate the real moment when a particular image becomes unsafe, nor does it model exploitation or compromise. Instead, it translates the degradation coefficient into an operational reference frame that helps compare how image families or candidate groups remain below a selected degradation threshold. Therefore, this layer is useful for governance and review policies, but it should not be treated as a survival or hazard model.

B. Originality of the Proposed Model

The proposed approach treats base Docker image selection as a measurable degradation-rate comparison problem. Existing scanner outputs, technical-lag measures, EPSS, and KEV remain useful, but they address different assessment tasks from the long-term base-image selection problem considered in this study.

The main practical role of the model is base-image selection under long-term security considerations. Conventional scanner outputs answer the question: “How many vulnerabilities are visible in this image now?” EPSS and KEV help answer: “Which disclosed vulnerabilities should be prioritized for remediation now?” The degradation coefficient answers a different question: “Which candidate base image has accumulated scanner-observable vulnerability burden more slowly over its lifecycle?”

The contribution is broader than the coefficient itself. The contribution lies in the layered structure of the model: scanner-observable findings are aggregated into severity-weighted burden, this burden is normalized by image age, the resulting degradation coefficient is interpreted within base-family context, and candidate variants are compared at repository level. This sequence transforms scanner data into a context-aware comparative signal rather than another point-in-time vulnerability summary.

The family-normalized score is important because base operating-system families differ in package ecosystems, release cadence, vulnerability disclosure patterns, and scanner coverage. Therefore, raw cross-family comparison may lead to over-interpretation. Family-normalized interpretation allows engineers to identify images that degrade unusually fast relative to comparable images from the same family.

Repository-level candidate comparison further strengthens the practical value of the model. It separates dependency-footprint reduction from base-ecosystem substitution. A slim variant and an Alpine-based variant may both reduce observed vulnerability burden, but they do so through different technical mechanisms and may have different compatibility implications. The proposed model makes this distinction explicit and therefore supports more realistic DevSecOps decision-making.

C. Practical Implications

The distinction matters because a base image is not selected only for the current scan result. Once adopted, it becomes part of the downstream build chain and can influence future rebuild frequency, review burden, and security maintenance effort. A candidate with a lower current vulnerability count but a high degradation coefficient may be less attractive than a candidate with a slightly higher current burden but slower observed degradation.

The proposed model can be used as a decision-support component in CI/CD governance. Conventional scanner reports remain necessary for detecting known vulnerabilities and supporting remediation. However, scanner reports alone usually describe the current state of an image. The degradation coefficient adds a long-term artifact-level signal by showing how quickly vulnerability burden has accumulated relative to image age.

In practical base-image selection, the model can help compare candidates that are functionally acceptable for the same downstream task. Images with high absolute burden, high degradation coefficient, or high family-normalized score can be prioritized for review. Repository-level pairwise comparisons can be used to justify migration from regular to slim variants or from one base ecosystem to another, provided that compatibility constraints are satisfied.

The results suggest concrete guidance for practitioners. First, image age should be treated as a security signal, not merely metadata. Second, base image choice has long-term security implications that persist across downstream artifacts. Third, deprecated images represent an unpatchable risk surface that warrants explicit policy controls. The proposed degradation coefficient can be integrated into CI/CD pipelines to prioritize rebuilds, enforce lifecycle policies, or inform base-image selection decisions.

Threshold-oriented interpretation can support operational rules, such as flagging images that exceed a selected degradation level expressed in severity points per day or per year. In this sense, the model does not automate the entire security decision, but it provides a measurable signal that can be incorporated into CI/CD dashboards, rebuild prioritization, base-image approval workflows, and periodic security reviews.

D. Limitations and Threats to Validity

Several limitations should be considered when interpreting the results.

Vulnerability detection depends on scanner coverage and database freshness. Some vulnerabilities may be missed or misclassified, particularly in minimal or distroless images. These effects introduce measurement noise but do not invalidate the rate-based formulation. The model measures scanner-observable vulnerability burden, not the complete security state of a Docker image.

Vulnerability databases differ in coverage, severity values, update timing, and identifier handling. The use of a multi-scanner ensemble, deduplication, maximum severity retention, scan synchronization, and database snapshot tracking reduces avoidable measurement ambiguity, but it does not eliminate scanner-related uncertainty. Absolute values may change when scanner databases are updated.

The dataset is restricted to Linux-based images from Docker Hub and may not generalize to private registries or Windows containers. However, Docker Hub represents the dominant public ecosystem and is therefore appropriate for ecosystem-level analysis.

Relationships between degradation coefficient, base family, image size, update activity, or candidate variant should not be interpreted as causal effects. A higher degradation coefficient does not prove that a particular base family causes vulnerabilities. It indicates that, under the selected measurement protocol, the image has accumulated scanner-visible vulnerability burden more rapidly relative to its age.

Severity weights are discrete and intentionally simple. While alternative weighting schemes are possible, the primary contribution of this work lies in rate normalization rather than the specific choice of weights. Sensitivity analysis of alternative weight mappings is left for future work.

Future work could extend the proposed Docker image degradation model by integrating it into probabilistic SDLC frameworks, such as GERT-based hybrid DevOps models with AI-assisted stages [25]. In this setting, degradation could be modelled as a time-dependent factor influencing rework probabilities, quality gates, and release success. Further research could focus on telemetry-driven calibration and the development of adaptive, potentially autonomous, mechanisms for degradation-aware security control in CI/CD pipelines.

CONCLUSIONS

This paper proposed and evaluated a comparative rate-based degradation model for selecting base Docker images with longer observable security horizons. The model addresses the DevSecOps selection problem in which several candidates may be functionally suitable, while conventional scanner outputs mainly describe the current vulnerability state. The proposed approach adds an age-normalized and context-aware signal for comparing how vulnerability burden accumulates across image lifecycles, base-image families, dependency footprints, and candidate variants.

The purpose of the study was achieved through the development and empirical evaluation of a layered comparative measurement model. The first objective, formalizing the vulnerability burden of a Docker image, was addressed by aggregating deduplicated scanner-reported vulnerability identifiers and weighting them by severity. The

ISSN 2617-4316 (Print)
ISSN 2663-7723 (Online)

second objective, normalizing vulnerability burden by image age, was addressed through the Docker Image Degradation Coefficient, which expresses observed vulnerability accumulation intensity in severity points per day. The third objective, contextualizing degradation within base-image families, was addressed through the family-normalized degradation score, which compares an image with the distribution of degradation values within the same base operating-system family. The fourth objective, comparing candidate variants at repository level, was addressed through pairwise comparison of content-derived candidate classes such as regular, slim, Alpine-based, Debian-based, and Ubuntu-based variants. The fifth objective, evaluating the proposed metric against alternative indicators, was addressed by comparing it with raw vulnerability counts, static severity burden, vulnerability density, image age, and exploitability-oriented signals such as EPSS and KEV.

The main contribution is the formalization of base-image selection as a degradation-rate comparison problem. The model transforms scanner-observable findings into severity-weighted burden, normalizes this burden by image age, contextualizes the resulting coefficient within base operating-system families, and supports repository-level candidate comparison. The degradation coefficient is therefore an artifact-level comparative signal for long-term base-image selection, rebuild prioritization, and CI/CD governance.

The empirical results show that degradation intensity differs across base operating-system families and content-derived candidate classes. Debian-based images have higher median degradation intensity than Alpine-based images in the analysed cohort, image size is positively associated with degradation intensity, and reduced-footprint candidates tend to show lower repository-level degradation than higher-footprint candidates. In subsets where direct comparison is available, Alpine-based and Ubuntu-based candidates also show lower degradation intensity than Debian-based candidates. These results indicate that dependency-footprint reduction and base-ecosystem substitution should be analysed separately rather than treated as a single minimized-image effect. This distinction is important for practical base-image selection because detected base ecosystem, dependency footprint, libc behavior, package availability, and compatibility constraints may all differ across candidate classes.

The model complements existing vulnerability-management signals. Raw counts, severity burden, vulnerability density, EPSS, and KEV remain useful for detection, static assessment, dependency-aware comparison, and short-term remediation prioritization. The degradation coefficient answers a different question: which candidate base image has accumulated vulnerability burden more slowly over its observable lifecycle? This distinction matters because base-image choice affects downstream software supply chains, rebuild pressure, review effort, and long-term maintenance cost.

Practically, the model gives DevSecOps teams a structured basis for comparing base-image candidates beyond point-in-time scanner outputs. It can support candidate selection, rebuild prioritization, review policy design, and long-term CI/CD governance by combining current burden, degradation rate, ecosystem context, candidate structure, and threshold-oriented security-horizon interpretation.

The model should also be interpreted within its limitations. It is based on scanner-observable evidence from public Linux-based Docker Hub images and does not represent complete security risk, zero-day exposure, private registry behaviour, or causal exploitability. Future research should evaluate the approach in private CI/CD environments, test alternative severity-weighting schemes, integrate exploitability-aware extensions, and develop adaptive mechanisms for degradation-aware base-image governance.

REFERENCES

1. Wong, A. Y., Chekole, E. G., Ochoa, M. & Zhou, J. “On the security of containers: Threat modeling, attack analysis, and mitigation strategies”. *Computers & Security*. 2023; 128: 103140, https://www.scopus.com/pages/publications/85150419071. DOI: https://doi.org/10.1016/j.cose.2023.103140.

2. Kaur, B., Dugré, M., Hanna, A. & Glatard, T. “An analysis of security vulnerabilities in container images for scientific data analysis*”. GigaScience*. 2021; 10 (6): giab025, https://www.scopus.com/pages/publications/85107985867. DOI: https://doi.org/10.1093/gigascience/giab025.

3. “CVE: Common Vulnerabilities and Exposures”. – Available from: https://www.cve.org. – [Accessed: Jan 06, 2026].

4. Liubchenko, V. V. & Volkov, D. V. “Cyber-aware threats and management strategies in cloud environments”. *Herald of Advanced Information Technology*. 2024; 7 (2): 158–170, https://www.scopus.com/pages/publications/105024789300. DOI: https://doi.org/10.15276/hait.07.2024.11.

5. Rosa, G., Scalabrino, S., Bavota, G. & Oliveto, R. “What quality aspects influence the adoption of Docker images?”. *ACM Transactions on Software Engineering and Methodology*. 2023; 32 (6): 1–30, https://www.scopus.com/pages/publications/85174680157. DOI: https://doi.org/10.1145/3603111.

6. Ksontini, E., Mastouri, M., Khalsi, R. & Kessentini, W. “Refactoring for Dockerfile quality: A dive into developer practices and automation potential”. *arXiv*. 2025. DOI: https://doi.org/10.48550/arXiv.2501.14131.

7. Revniuk, O. A., Zagorodna, N. V., Kozak, R. O., Karpinski, M. P. & Flud, L. O. “The improvement of web-application SDL process to prevent Insecure Design vulnerabilities”. *Applied Aspects of Information Technology*. 2024; 7 (2): 162–174. DOI: https://doi.org/10.15276/aait.07.2024.12.

8. Wist, K., Helsem, M. & Gligoroski, D. “Vulnerability analysis of 2500 Docker hub images”. *arXiv*. 2020. DOI: https://doi.org/10.48550/arXiv.2006.02932.

9. Shi, H., Ying, L., Chen, L., Duan, H., Liu, M. & Xue, Z. “Dr. Docker: A large-scale security measurement of Docker image ecosystem”. *Proceedings of the ACM Web Conference*. 2025. p. 2813–2823, https://www.scopus.com/pages/publications/105005141422. DOI: https://doi.org/10.1145/3696410.3714653.

10. Mills, A., White, J. & Legg, P. “Longitudinal risk-based security assessment of Docker software container images”. *Computers & Security*. 2023; 135: 103478, https://www.scopus.com/pages/publications/85172381983. DOI: https://doi.org/10.1016/j.cose.2023.103478.

11. Zerouali, A., Mens, T., Robles, G. & Gonzalez-Barahona, J. “On the relation between outdated Docker containers, severity vulnerabilities and bugs”. *arXiv*. 2018. DOI: https://doi.org/10.48550/arXiv.1811.12874.

12. Haque, M. U. & Babar, M. A. “Well begun is half done: An empirical study of exploitability & impact of base-image vulnerabilities”. *IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2022. p. 1066–1077, https://www.scopus.com/pages/publications/85135859108. DOI: https://doi.org/10.1109/SANER53432.2022.00124.

13. Durieux, T. “Empirical study of the Docker smells impact on the image size”. *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 2024. p. 1–12, https://www.scopus.com/pages/publications/85196809274. DOI: https://doi.org/10.1145/3597503.3639143.

14. Opdebeeck, R., Lesy, J., Zerouali, A. & De Roover, C. “The Docker hub image inheritance network: construction and empirical insights”. *IEEE 23rd International Working Conference on Source Code Analysis and Manipulation (SCAM)*. 2023. p. 198–208, https://www.scopus.com/pages/publications/85181447834. DOI: https://doi.org/10.1109/SCAM59687.2023.00029.

15. Boles, B. M. “Static analysis tool discrepancies and the pursuit of a unified vulnerability database”. *IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)*. 2024, https://www.scopus.com/pages/publications/85215295876.
DOI: https://doi.org/10.1109/SCAM63643.2024.00034.

16. Kim, T., Park, S. & Kim, H. “Why Johnny can’t use secure Docker images: Investigating the usability challenges in using Docker image vulnerability scanners through heuristic evaluation”. *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*. 2023. p. 669–685, https://www.scopus.com/pages/publications/85175720682. DOI: https://doi.org/10.1145/3607199.3607244.

17. “Trivy”. – Available from: https://aquasecurity.github.io/trivy/v0.59. – [Accessed: Mar 3, 2025].

18. “anchore/grype [Go]”. *Anchore, Inc*. 2026. – Available from: https://github.com/anchore/grype. – [Accessed: Jan 16, 2026].

19. “Docker Scout. Docker Documentation”. – Available from: https://docs.docker.com/scout. – [Accessed: Jan 05, 2026].

20. Mozhaiev, M., Davydov, V. & Liqiang, Z. “Analysis and comparative researches of methods for improving the software”. *Advanced Information Systems*. 2020; 4 (3): 124–132. DOI: https://doi.org/10.20998/2522-9052.2020.3.18.

21. Doan, T. P. & Jung, S. “DAVS: Dockerfile analysis for container image vulnerability scanning”. *Computers, Materials & Continua*. 2022; 72 (1): 1699–1711, https://www.scopus.com/pages/publications/85125381482. DOI: https://doi.org/10.32604/cmc.2022.025096.

22. Mills, A., White, J. & Legg, P. “OGMA: Visualisation for software container security analysis and automated remediation”. *2022 IEEE International Conference on Cyber Security and Resilience (CSR).* 2022: 76–81, https://www.scopus.com/pages/publications/85137365963.
DOI: https://doi.org/10.1109/CSR54599.2022.9850335.

23. Schober, P., Boer, C. & Schwarte, L. A. “Correlation coefficients: appropriate use and interpretation”. *Anesthesia & Analgesia.* 2018; 126 (5): 1763–1768, https://www.scopus.com/pages/publications/85048115048. DOI: https://doi.org/10.1213/ANE.0000000000002864.

24. Han, J., Huang, C., Liu, J. & Zhang, T. “An Effective Docker Image Slimming Approach Based on Source Code Data Dependency Analysis”. *arXiv*. 2025. DOI: https://doi.org/10.48550/arXiv.2501.03736.

25. Semenov, S., Tsukur, V., Molokanova, V., Muchacki, M., Litawa, G., Mozhaiev, M., et al. “Mathematical model of the software development process with hybrid management elements”. *Applied Sciences.* 2025; 15 (21): 11667, https://www.scopus.com/pages/publications/105021457617. DOI: https://doi.org/10.3390/app152111667.

Conflicts of Interest: The authors declare that they have no conflict of interest regarding this study, including financial, personal, authorship or other, which could influence the research and its results presented in this article

Received 23.03.2026
Received after revision 11.06.2026
Accepted 18.06.2026

DOI: https://doi.org/10.15276/aait.09.2026.27
УДК 004.45

Моделювання безпекової деградації базових Docker-образів на основі швидкісної метрики

Лещенко Богдан Сергійович[1)]
ORCID: https://orcid.org/0009-0001-5781-3518; zogyyy1@gmail.com. Scopus Author ID: 58736905200
Єфіменко Андрій Анатолійович[1)]
ORCID: https://orcid.org/0000-0003-2128-4797; yefimenko.andrii@gmail.com. Scopus Author ID: 57216846685
[1)] Державний університет «Житомирська Політехніка», вул. Чуднівська, 103. Житомир, 10005, Україна

АНОТАЦІЯ

Актуальність. Базові Docker-образи повторно використовуються як довготривалі артефакти ланцюга постачання програмного забезпечення в конвеєрах розроблення, однак їхній безпековий стан зазвичай оцінюється за результатами сканування в певний момент часу. Такі результати дають змогу виявити відомі вразливості на дату сканування, проте не показують, з якою швидкістю навантаження вразливостей накопичувалося відносно віку образу, родини базової операційної системи та обсягу залежностей. Це формує науково-технічну суперечність: вибір базового образу потребує довготривалого й відтворюваного порівняння функціонально придатних кандидатів, тоді як наявні метрики переважно описують поточний стан вразливостей, статичне навантаження критичності, актуальність пакетів або короткостроковий пріоритет експлуатації вразливостей. Отже, командам DevSecOps бракує достатньої доказової основи для вибору кандидата з довшим спостережуваним горизонтом безпеки. **Мета і завдання.** Метою цього дослідження є підвищення наукової та практичної обґрунтованості довгострокового вибору базових Docker-образів у конвеєрах шляхом формалізації спостережуваної безпекової деградації як нормалізованого за віком і контекстуалізованого за сімейством базової системи процесу накопичення. Для досягнення цієї мети в роботі розроблено та емпірично оцінено багаторівневу порівняльну вимірювальну модель для виявлення кандидатів базових образів із довшими спостережуваними безпековими горизонтами. Завдання дослідження полягають у формалізації тяжкісно зваженого вразливісного навантаження, нормалізації цього навантаження за віком образу. **Методи.** У дослідженні використано часово обмежений набір даних Linux-базованих Docker-образів із популярних репозиторіїв Docker Hub. Вразливості ідентифіковано за допомогою набору з чотирьох сканерів, до якого входять Trivy, Grype, Docker Scout і Snyk. Отримані результати дедубліковано за ідентифікатором вразливості, зважено за рівнем критичності та нормалізовано за віком образу. Ідентифікатори Common Vulnerabilities and Exposures зберігаються за наявності, однак основне агрегування здійснюється відповідно до ідентифікатора вразливості, повідомленого конкретним інструментом сканування. Модель оцінено шляхом порівняння на рівні родин базових образів, порівняння кандидатів на рівні репозиторіїв і порогової інтерпретації горизонту безпеки. **Результати.** Запропонована модель поєднує абсолютне навантаження вразливостей, нормалізовану за віком інтенсивність деградації, нормалізоване порівняння в межах родини, порівняння кандидатів на рівні репозиторіїв і порогову інтерпретацію горизонту безпеки. Родини базових операційних систем істотно відрізняються між собою. Порівняння на рівні репозиторіїв показують нижчу інтенсивність деградації для кандидатів зі зменшеним обсягом залежностей порівняно з кандидатами з більшим обсягом залежностей у межах тієї самої екосистеми, а також нижчу інтенсивність деградації для

Alpine-базованих та Ubuntu-базованих кандидатів порівняно з Debian-базованими кандидатами в репозиторіях, де доступне пряме порівняння. **Висновки.** Наукова новизна дослідження полягає у формалізації вибору базових Docker-образів як задачі порівняння швидкості безпекової деградації. На відміну від підрахунку вразливостей у певний момент часу, статичних зведень критичності, показників технічного відставання, Exploit Prediction Scoring System або Known Exploited Vulnerabilities, запропонована модель формує сигнал на рівні програмного артефакту, що дає змогу порівнювати кандидатні базові образи за спостережуваною швидкістю накопичення виявленого сканерами навантаження вразливостей відносно віку образу. Модель є відтворюваною порівняльною вимірювальною процедурою для довготривалого вибору базових образів. Практично модель підтримує команди DevSecOps у виборі кандидатних базових образів, пріоритизації перебудови, визначенні політик перегляду та керуванні конвеєрами розроблення шляхом порівняння образів за поточним навантаженням вразливостей, спостережуваною швидкістю деградації, базовою екосистемою, контекстом обсягу залежностей і пороговим горизонтом безпеки.

Ключові слова: старіння програмного забезпечення; технічний борг; безпека контейнерів; Docker-образи; керування CI/CD; вибір базового образу; накопичення вразливостей

ABOUT THE AUTHORS

Bohdan S. Leshchenko - PhD student, Software Engineering Department. Zhytomyr Polytechnic State University, 103, Chudnivska Str. Zhytomyr, 10005, Ukraine
ORCID: https://orcid.org/0009-0001-5781-3518; zogyyy1@gmail.com. Scopus Author ID: 58736905200
***Research field*:** Software Engineering; DevSecOps; Container Security

Лещенко Богдан Сергійович - аспірант кафедри Інженерії програмного забезпечення. Державний університет "Житомирська політехніка", вул. Чуднівська, 103. Житомир, 10005, Україна

Andrii A. Yefimenko - Head of Department of Computer Engineering and Cybersecurity. Zhytomyr Polytechnic State University, 103, Chudnivska Str. Zhytomyr, 10005, Ukraine
ORCID: https://orcid.org/0000-0003-2128-4797; yefimenko.andrii@gmail.com. Scopus Author ID: 57216846685
Research field: design, development, and operation of computer networks; information security in computer networks; network and cloud security; technologies and tools for virtualization of computer networks

Єфіменко Андрій Анатолійович - кандидат технічних наук, доцент, завідувач кафедри Комп'ютерної інженерії та кібербезпеки. Державний університет "Житомирська політехніка", вул. Чуднівська, 103. Житомир, 10005, Україна