SBSeg 2026 Armação dos Búzios - RJ
VEJA OS SELOS DE ARTEFATOS

A Uniform Random-Sample Security Measurement of Docker Hub Images

Cristhian Kapelinski[1], **Diego Kreutz**[1]

[1]AI Horizon Labs, Federal University of Pampa (UNIPAMPA)
cristhianavila.aluno@unipampa.edu.br
diegokreutz@unipampa.edu.br

Abstract. *Registry-scale security measurements of Docker Hub define their samples by repository class, category, popularity, or estimated reach. Their findings therefore describe selected subsets rather than the registry-wide posture of a typical scannable* `latest` *image. We estimate that posture from* ***2,879*** *such images obtained by uniformly sampling 12.7 million repositories. Six open-source scanners match the pipeline used on our exposure-ranked sample, where exposure is the number of pulls reached directly or through reused layers.* ***94.4%*** *of random images carry a critical vulnerability, compared with 93.4% of exposure-ranked images. Tool choice strongly affects the reported posture:* ***70.5%*** *of image–CVE pairs appear in only one of three scanners. TruffleHog reports secrets in* ***82.4%*** *of images, but hand-labeling shows that 99.5% of these detections are false positives. Exposure-based prioritization identifies vulnerabilities likely to affect more users, but the selected images are not more vulnerable than typical scannable* `latest` *images.*

1. Introduction

Docker Hub hosts **15 million** repositories and has served over **318 billion** pulls, a 145% increase in a single year.[1][2] Users can pull and run an image as published, so registry content can become running software directly. That scale also creates a security burden. In one month in 2025, a scan revealed over **10,000** images leaking live secrets. These included credentials from a national bank and a Fortune 500 company, one of the largest U.S. corporations by revenue.[3]

Docker Hub studies have selected images by repository class or popularity [Shu et al. 2017, Liu et al. 2020] and by pulls and position in the dependency graph [Shi et al. 2025]. Our exposure-ranked sample defines an image's exposure as the pulls its layers reach directly or through derived images [Kapelinski et al. 2026]. None of these designs gives every repository in an enumerated frame the same selection probability. Uniform sampling does, allowing population-wide estimates for repositories with a scannable `latest` image. If those images are no safer than prioritized ones, prioritization ranks impact while leaving a widespread baseline risk unmeasured.

This paper estimates the posture of scannable `latest` images with the same six scanners [Kapelinski et al. 2026]. To our knowledge, this is the first uniformly random multi-scanner measurement of Docker Hub (Sections 3–4.1). We show that the critical-vulnerability rates of the random and exposure-ranked samples differ by one percentage point (Section 4.2). We also reproduce prior analyses across sampling frames: vulnerability rates, scanner disagreement, and spurious secret detections (Sections 4.3–4.5). We release the code and random-sample dataset.

1. JFrog, malicious Docker Hub repositories, 2024 2. Docker, Docker Index, 2021 3. Flare, Docker Hub secrets exposed, 2025

2. Related Work

Table 1 compares studies by sampling frame, which determines the population described, and scanner count, which affects tool dependence.

Table 1. Study corpora; scanner count covers security-analysis tools.

Work	Images	Scanners	Sampling frame
[Shu et al. 2017]	356,218	1	community + official
[Liu et al. 2020]	2,227,244	1	top repositories
[Wist et al. 2021]	2,500	1	repository categories
[Dahlmanns et al. 2023]	337,171	1	keywords + address ranges
[Mills et al. 2023]	380	6	longitudinal sample
[Shi et al. 2025]	33,952	1	influence-ranked
[Kapelinski and Kreutz 2026b]	5,606	14	operating-system bases
[Kapelinski et al. 2026]	52,895	6	exposure-ranked
This work	**2,879**	**6**	**uniform random**

Prior measurements. Docker Hub has been measured periodically since 2017. Early crawls mapped parent-to-child vulnerability propagation [Shu et al. 2017] and analyzed run commands, malicious images, and delayed patching [Liu et al. 2020]. Later work measured Debian packages lagging behind the newest version in the same release [Zerouali et al. 2019]. Dr. Docker selected influential images by pull count and dependency-graph weight [Shi et al. 2025]. Other studies compared application variants [Ibrahim et al. 2020] and assessed the exploitability of base-image vulnerabilities [Haque and Babar 2022].

Tooling and sampling. Five of the eight measurements in Table 1 use one scanner. Jaccard similarity is the fraction of distinct findings two scanners share out of all findings either reports, from 0 (none shared) to 1 (identical). Across three scanners, it ranged from 0.59 to 0.80 [Kaur et al. 2021]. The reported vulnerability rate can therefore reflect the tool. We run six and quantify disagreement (Section 4.4). Samples defined by class, category, popularity, or influence do not alone support registry-wide estimates [Baltes and Ralph 2022]; a uniform draw does. Our later census of the same frame measured its cryptographic posture [Kapelinski and Kreutz 2026a]. A uniform draw also serves as a population-level control for studies based on selection criteria.

3. Methodology

Sampling frame and draw. The sampling frame is the **12,716,568** public repositories found by a crawl that enumerated all discoverable Docker Hub repository names [Kapelinski et al. 2026]. We draw uniformly at random from this frame, so every public repository has the same selection probability regardless of pulls, stars, or age. The unit of measurement is each repository's `latest` image, the tag a user obtains by default. We overdraw to **4,800** candidates because many repositories do not resolve through the default tag (Section 4.1). The **2,879** images that scanned completely form the corpus. With this sample size, an estimated vulnerability prevalence of 95% has a margin of error below one percentage point at 95% confidence. The estimate covers repositories with a scannable `latest` image in that frame and uses Wilson's method [Wilson 1927].

Reachability outcomes. For each repository, we pull and scan `latest`. We record *scanned* or *without* `latest` when no default tag resolves. *Unpullable* covers non-image artifacts, obsolete image descriptions (legacy manifests), and other pull errors. *Denied* is the authorization error shared by private and deleted repositories. *Other-architecture* means no `linux/amd64` build exists; *did not finish* means a pulled image's scan stalled.

Scanners. We apply the same six-scanner configuration to both samples [Kapelinski et al. 2026]. Syft inventories the software bill of materials (SBOM); Trivy, Grype, and the Open Source Vulnerabilities scanner (OSV-Scanner) report package vulnerabilities; Dockle checks container hardening; and TruffleHog detects hard-coded secrets. Both samples use the same image references and settings. Because several tags float and vulnerability databases are fetched at execution time, results describe the tool builds and data resolved during the campaigns, not an immutable software environment. Adapters convert vendor labels and Common Vulnerability Scoring System (CVSS) scores into six ordered levels: critical (score ≥ 9.0), high (≥ 7.0), medium (≥ 4.0), low (> 0), informational, and unknown. Findings are merged but not deduplicated across scanners, so we also report distinct Common Vulnerabilities and Exposures (CVE) identifiers (Section 4.1).

Comparison design. We compare against the exposure-ranked corpus from the same crawl and pipeline [Kapelinski et al. 2026] (Table 2). A two-proportion z-test asks whether an observed difference between rates is larger than random sampling alone would usually produce. Its p-value is the probability of a difference at least this large if the population rates were equal, and values below 0.05 indicate statistically distinguishable rates. Section 4.3 also reproduces prior analyses.

4. Results

4.1. Repository reachability and vulnerability rates

Reachability. Of the **4,800** repositories drawn, **60.0%** (2,879) scanned. Another **34.9%** (1,673) have no `latest` tag. These repositories are typically dormant, with a median last push in 2023. The remaining 248 are unpullable (113), denied (42), available only for another architecture (36), or did not finish (57). Unpullable repositories comprise 48 legacy schemas, 28 non-runnable Open Container Initiative artifacts, and 37 other pull errors.

Posture. The scanners collected **9.9 million** findings, including 8.5 million package vulnerabilities. Figure 1(a) shows the latter per image: the median is **947** merged package vulnerabilities (mean 2,939; maximum 26,511). **94.4%** carry a critical vulnerability, **96.1%** a high-severity one, and **96.8%** one of any severity. Deduplication reduces this median to **260** distinct CVE identifiers (mean 857; maximum 8,265) without changing the share of affected images. Medium and high severities dominate the finding-level distribution in Figure 1(b). Figure 1(c) shows that at least five of the six scanners completed for 96.1% of images, including 62.7% for which all six completed. The remaining 3.9% have results from four or fewer scanners; Appendix A analyzes the failure modes.

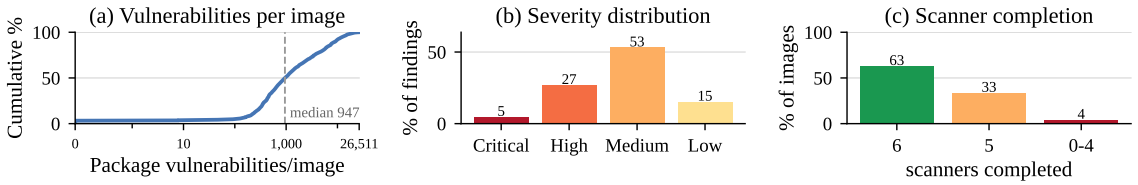


Figure 1. Random-sample findings: (a) package vulnerabilities per image (log scale); (b) severity; (c) scanner completion.

4.2. Random versus exposure-ranked samples

The two samples have nearly identical vulnerability rates: each pair differs by at most one percentage point (Table 2) [Kapelinski et al. 2026]. The tests do not distinguish the high- or any-severity rates ($p = 0.15$ and $p = 0.23$). The one-point critical-vulnerability gap is statistically distinguishable ($p = 0.03$), but small; the random median

is also higher (947 vs. 885). Raw secret detections are more frequent in the random sample ($p < 0.001$), although Section 4.5 shows that nearly all are non-credentials in both samples. Thus, exposure ranking selects images whose vulnerabilities reach more users, not unusually vulnerable images.

Table 2. Random vs. exposure-ranked samples [Kapelinski et al. 2026], same pipeline.

	Random	Exposure-ranked	Statistically different?
Images scanned	2,879	52,895	not applicable
≥ 1 critical vulnerability	94.4%	93.4%	yes ($p = 0.03$)
≥ 1 high vulnerability	96.1%	95.5%	no ($p = 0.15$)
≥ 1 vulnerability	96.8%	96.3%	no ($p = 0.23$)
Secret detections (raw, TruffleHog)	82.4%	76.9%	yes ($p < 0.001$)
of which non-credentials	99.5%†	99.7%	no ($p = 0.48$)
Repositories without `latest`	34.9%	not observable	not comparable

† false-positive rate from hand-labeled detections (Section 4.5).

4.3. Comparison with prior measurements

Table 3 and Figure 2 reproduce selected analyses, paired with exposure-ranked values where available. Comparisons are indicative: scanners, samples, and units differ.

The vulnerability rates persist across designs. Figure 2(a) shows CVE identifiers reaching back to 1999 [Mills et al. 2023]. Panel (b) classifies the 2.31 million severe (critical or high) findings: **83.4%** affect operating-system packages (e.g. those managed by apt or apk) and **15.7%** affect application-language packages (e.g. npm or the Python Package Index). This contrasts with Wist et al.'s five most represented critical vulnerabilities, which came from JavaScript and Python [Wist et al. 2021]. Panel (c) places per-image counts in historical context. A Debian-focused 2018 corpus reported a median of 601 vulnerabilities [Zerouali et al. 2019]; our merged-scanner medians are 885 in the exposure-ranked sample and 947 in the random sample. Under comparable definitions, our rates meet or exceed the earlier measurements in panel (d). The highest severity per image is now most often *critical*, whereas Shu et al. reported *high* under an earlier three-level scale (Table 3) [Shu et al. 2017]. Finally, panel (e) shows that the most widespread vulnerable packages are operating-system libraries.

Findings from studies of selected image groups also hold in the random sample. Within Debian, vulnerability counts vary widely among images of the same application [Ibrahim et al. 2020]; across distributions, our base-image census [Kapelinski and Kreutz 2026b] and random sample show sharply different postures. Minimal images are not uniformly safer, echoing [Haque and Babar 2022]: package count barely predicts vulnerability count, and several language-only images still carry more than a thousand CVEs. Tool choice also shifts counts, as observed for SBOM generators [O'Donoghue et al. 2024] and, in Section 4.4, for vulnerability scanners. Even the most-agreeing pair shares only **43%** of its combined distinct findings, a Jaccard index of 0.43; a mixed 48-image dataset reported 0.694 [Churakova et al. 2026]. Two analyses are not computable on a uniform draw. The sample contains **zero** official repositories, preventing the official-versus-community split [Liu et al. 2020]. Tracking staleness over time requires registry timestamps that our snapshot lacks.

4.4. Tooling, base images, and hardening

Figure 3 links the findings to scanner choice, base distribution, and configuration. Tool dependence remains strong (Figure 3(a)): **70.5%** of image–CVE pairs appear in

Table 3. Prior analyses repeated on our random sample.

Study	Analysis	Reported	Exposure-ranked	Random (ours)
[Shu et al. 2017]	worst severity, modal	*high*	*critical*	*critical*, 94.4%
[Liu et al. 2020]	severe, official images	≈30%	93.8%	**n/a**[§]
[Liu et al. 2020]	severe, community images	>64%	95.6%	96.4%
[Haque and Babar 2022]	minimal images safer?	no	–	lower median, mixed
[Dahlmanns et al. 2023]	secret hit rate	8.5%[‡]	76.9%	82.4%
[Churakova et al. 2026]	Jaccard	0.69	–	0.43 (best pair)

– marks an analysis unavailable for that sample; n/a means not applicable. [§] No official repository occurred in our sample. [‡] Filtered pipeline rate; 76.9% and 82.4% are raw detector rates (Section 4.5).

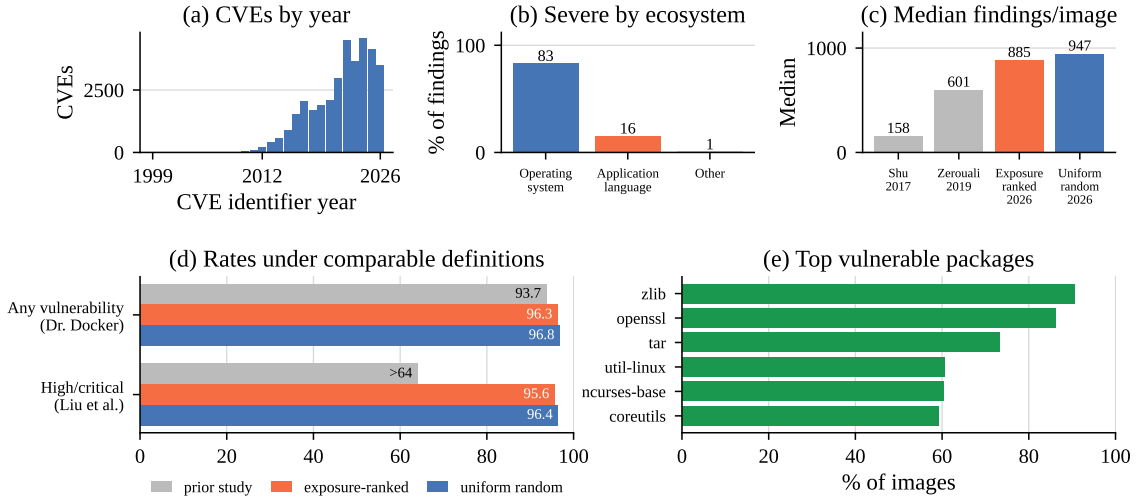


Figure 2. Prior analyses repeated on the random sample: panels (a)–(e).

only one vulnerability scanner, and **11.1%** in all three. A result from one scanner therefore does not represent their combined findings. Most images use common Linux bases (Figure 3(b)). An identifiable operating-system package set occurs in **93.9%**; Debian, Alpine, and Ubuntu together cover 89.3%. Their individual shares are 38.4%, 32.8%, and 18.1%. The remaining 6.1% are distroless (no operating-system package manager) or language-only. Hardening is largely absent. Excluding the near-universal content-trust check, Dockle reports a misconfiguration in **99.4%** of images, most often a missing `HEALTHCHECK` container-health probe (97.4%) or non-root user (89.0%). It flags potentially sensitive values in environment variables or files in **31.7%**. Without a health probe, platforms cannot automatically detect an unhealthy container; running as root increases the damage after compromise.

4.5. False positives in secret detection

TruffleHog flags secrets in **82.4%** of random images, whereas a filtered multi-registry pipeline reported 8.5% [Dahlmanns et al. 2023]. We manually classified **1,100** randomly sampled detections (Appendix A). Only **5** of them were genuine, from a population of 169,528 raw detections. The estimated false-positive rate is therefore **99.5%**, with a 95% confidence interval of 98.9–99.8%. The estimated genuine rate among detections is **0.45%** (0.2–1.1%). Package hashes (60%), placeholders (18%), binaries, lock files, and test fixtures explain nearly all detections; bare hashes and identifiers provide no credential context (Figure 3(c)). The exposure-ranked study found 99.7% false positives [Kapelinski et al. 2026]; the rates are not statistically distinguishable ($p = 0.48$). This problem follows the detector, not the sampling frame.

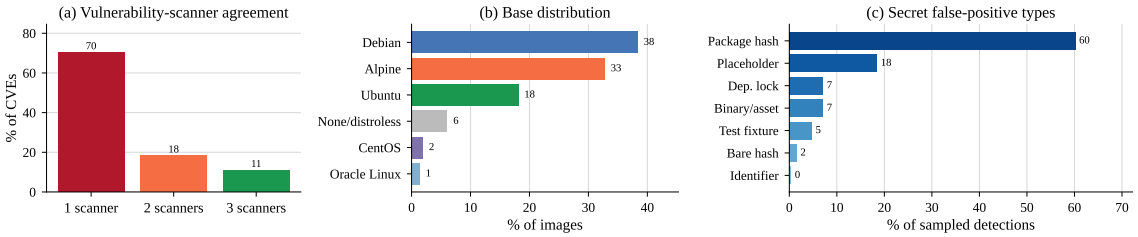


Figure 3. Tool and image profiles: (a) scanner agreement; (b) base distribution; (c) secret-detection categories.

5. Discussion

Toward mitigation. Docker Hub security controls should not use popularity as a proxy for vulnerability because rates differ little between samples. Publishers should update base operating-system distributions and run multiple scanners because 70.5% of image–CVE pairs are tool-specific; credentials should be provisioned at deployment rather than baked into images. Consumers should pin the scanned image's immutable identifier (digest), prioritize findings that are reachable in the deployed configuration [Haque and Babar 2022], and separate secrets from package hashes and test fixtures before incident response.

6. Limitations and Conclusion

Limitations. Findings can overstate practical risk because vulnerabilities may not be reachable, and base-image inheritance propagates them [Haque and Babar 2022]. A shared pipeline reduces comparison bias, but floating tags and database updates permit tool drift. Reachability covers all 4,800 draws; vulnerability rates cover the 60.0% whose `latest` image completed scanning. We analyze `linux/amd64` only, merge scanner counts (Section 4.1), and report 82.4% as a raw secret-detector rate (Section 4.5). The uniform sample measures security characteristics, not ecosystem-wide impact, which requires exposure information [Kapelinski et al. 2026].

Conclusion. A typical scannable `latest` image is about as likely as an exposure-ranked one to carry a critical vulnerability (94.4% vs. 93.4%), so exposure-based selection alone does not explain the high rates. Scanner disagreement and the secret detector's false-positive rate also remain high across samples. Excluding repositories outside a prioritized set from security controls shifts their risk to consumers. Mitigation must therefore reach the whole registry; our code and dataset are public.[4][5]

References

Baltes, S. and Ralph, P. (2022). Sampling in software engineering research: A critical review and guidelines. *EMSE*, 27(4):94.

Churakova, Y., Ekstedt, M., and Schmid, L. (2026). Vexed by VEX tools: Consistency evaluation of container vulnerability scanners. In *FPS 2025*, volume 16402, pages 139–156.

Dahlmanns, M. et al. (2023). Secrets revealed in container images: An internet-wide study on occurrence and impact. In *ASIA CCS '23*, pages 797–811.

4. `https://doi.org/10.5281/zenodo.22664930`

5. This work was partially supported by Capes, CNPq (Grant No. 409743/2025-9), and FAPERGS (Grant Nos. 24/2551-0001368-7 and 24/2551-0000726-1). We thank the anonymous reviewers of SBSeg 2026 for their suggestions and comments. The authors also acknowledge the use of Generative AI as a supporting tool for reviewing and improving the code and manuscript.

Haque, M. U. and Babar, M. A. (2022). Well begun is half done: An empirical study of exploitability and impact of base-image vulnerabilities. In *SANER 2022*, pages 1066–1077.

Ibrahim, M. H., Sayagh, M., and Hassan, A. E. (2020). Too many images on Docker Hub! How different are images for the same system? *EMSE*, 25(5):4250–4281.

Kapelinski, C. and Kreutz, D. (2026a). CryptoCensus: Cryptographic posture and post-quantum readiness of Docker Hub. In *WTICG/SBSeg*.

Kapelinski, C. and Kreutz, D. (2026b). A multi-scanner census of the Linux operating-system base images of Docker Hub. In *SBSeg*.

Kapelinski, C., Machado, B., and Kreutz, D. (2026). Vulnerabilities, secrets and misconfiguration in the highest-exposure Docker Hub images. *arXiv preprint arXiv:2608.02669*.

Kaur, B. et al. (2021). An analysis of security vulnerabilities in container images for scientific data analysis. *GigaScience*, 10(6):giab025.

Liu, P. et al. (2020). Understanding the security risks of Docker Hub. In *ESORICS 2020*, volume 12308, pages 257–276.

Mills, A., White, J., and Legg, P. (2023). Longitudinal risk-based security assessment of docker software container images. *Computers & Security*, 135:103478.

O'Donoghue, E. et al. (2024). Impacts of software bill of materials (SBOM) generation on vulnerability detection. In *SCORED '24*, pages 67–76.

Shi, H. et al. (2025). Dr. Docker: A large-scale security measurement of Docker image ecosystem. In *WWW '25*, pages 2813–2823. ACM.

Shu, R., Gu, X., and Enck, W. (2017). A study of security vulnerabilities on Docker Hub. In *CODASPY*, pages 269–280.

Wilson, E. B. (1927). Probable inference, the law of succession, and statistical inference. *JASA*, 22(158):209–212.

Wist, K., Helsem, M., and Gligoroski, D. (2021). Vulnerability analysis of 2500 Docker Hub images. In *Advances in Security, Networks, and Internet of Things*, pages 307–327.

Zerouali, A. et al. (2019). On the relation between outdated Docker containers, severity vulnerabilities, and bugs. In *SANER 2019*, pages 491–501.

A. Supporting Methodology

Scanner failures. Trivy accounts for 971 failures (33.7% of images), including 915 cache-contention errors; OSV fails on 5.1%, Grype on 2.5%, Syft on 2.2%, and Dockle and TruffleHog on less than 0.3% each. Missing output can make merged counts conservative.

Manual secret validation. Using seed 20260522, one reviewer inspected the detector, raw value, and file path of 1,100 of the 169,528 detections available when labeling began. Package metadata, examples, tests, binaries, dependency caches, and context-free values were false positives; five were plausible credentials, and one uncertain client key was conservatively counted as a false positive. TruffleHog lacked active verification, so labels assess plausibility, not liveness.