Poster: Death Is Not the End: A Longitudinal Study on the Impact of Automatic Updates on Container Vulnerability Lifespans

Simge Tekin*, Octavian Suciu†, Sungsu Kwag*, Yonghwi Kwon*, and Tudor Dumitras*
*Department of Computer Science, University of Maryland, College Park, MD, USA
†Google Research, New York City, NY, USA
*{stekin, sskwag, yongkwon, tudor}@umd.edu †osuciu@google.com

***Abstract*—Immutable infrastructures such as container ecosystems have changed how software is built, deployed, and patched. Instead of in-place updates, patches are delivered through image rebuilds and propagated across dependency hierarchies. However, the security implications of this patching model, such as vulnerability lifecycles, remain understudied.**

We study vulnerability remediation in the Docker ecosystem through a qualitative analysis of maintainer and user interactions and a longitudinal measurement study spanning more than six years. Across over 9,000 CVEs in 137 applications and 756,313 images, we find that 78% of inherited vulnerabilities remain unresolved 30 days after disclosure. A major cause of prolonged exposure is the breakdown of automated patch propagation after upstream end-of-life (EOL) events, which leaves 11% of inherited vulnerabilities unresolved, rising to 23% in deeper dependency layers. Our findings highlight the fragility of automated patch propagation in software supply chains and motivate clearer maintenance practices, better visibility into dependency status, and stronger accountability across image hierarchies.

1. Introduction

Cloud computing has transformed how software is developed, delivered, and deployed, enabling automated and continuous vulnerability patching in containerized ecosystems such as Docker. Unlike traditional environments, where dependencies are updated in place, container images are immutable and must be updated by releasing rebuilt image versions through standardized build–test–scan pipelines [1]. Container images are often derived from other images (i.e., base images) creating chains of multiple dependent images.[1]. In such chains, the contents of each upstream image—including inherited vulnerabilities—propagate to all downstream descendants.

This layered structure suggests that patching propagates transitively as well: once a vulnerability is fixed in an upstream base image, rebuilding dependent images should carry the fix downstream automatically. This automated patch flow represents an important advance in supply-chain security. However, existing work has largely focused on vulnerability prevalence at isolated snapshots, rather than on how vulnerabilities persist over time or how patch propagation behaves across dependency chains [2], [3], [4]. As a result, it remains unclear how long vulnerabilities survive in practice, when automated remediation fails, and how disruptions such as stalled rebuilds or end-of-life base images affect downstream exposure. Addressing these questions requires a longitudinal perspective that captures both the temporal dynamics of remediation and the dependency relationships through which patches are expected to flow.

1. For instance, a chain of `OS→Library→Application` such as `debian→openjdk→orientdb`.

2. Methods

We study this problem using a mixed-methods design. First, we qualitatively analyze 54 GitHub issue threads discussing persistent CVEs in the Dockerfile repositories of three popular Docker Official Image applications—Python, Node, and Nginx—and in the central CI/CD metadata repository, official-images [5]. Two researchers independently open-coded these issues and discussion threads to identify recurring causes and patterns behind patching delays.

Second, we conduct a large-scale longitudinal measurement study of Official Docker images published between January 2018 and July 2024. We obtain historical image digests from the archive repository repo-info [6], pull the corresponding images, and scan them using the Trivy scanner to extract CVEs. Our dataset includes 756,313 images from 137 applications, covering 21,598 Docker tags and 9,156 unique CVEs. To support this analysis, we introduce the notion of an image *lineage*, defined as the chronologically ordered set of images referenced by the same tag (e.g., `latest`), capturing how an image evolves over time. Using this abstraction, we define the lifespan of an inherited vulnerability from its first appearance in a lineage to its last. We distinguish five lifecycle events in total: two end-of-life events—application end-of-life and base-image end-of-life—that prevent further patching of the vulnerability, and three remediation modes for inherited vulnerabilities: *implicit updates*, acquired through rebuilds on patched base images; *explicit updates*, through manual migration to newer base-image tags (that is, updating the `FROM` statement in a Dockerfile); and *overwrites*, where derived images apply fixes independently while the base image remains vulnerable.

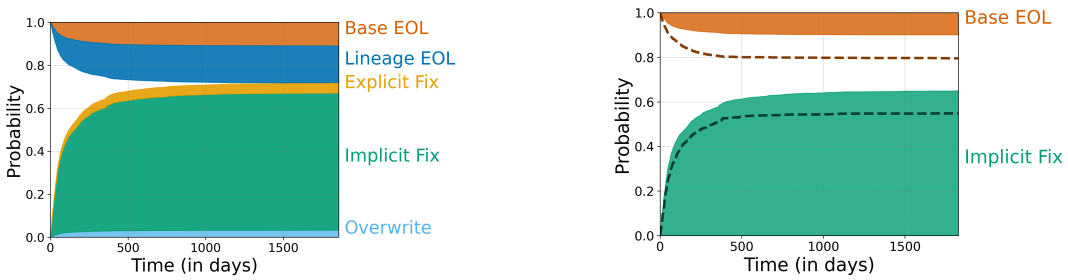

Figure 1: Cumulative incidence function (CIF) for events in vulnerability lifespan, throughout the 6-year, 7-month observation period. (Left: (a) the CIF for all inherited vulnerabilities. Right: (b) CIF stratified by vulnerability origin: depth 1 (i.e., immediate base image; filled curve) versus depth $\geq$3 (dashed curve) in the base image chain.

We then use a competing-risks framework to model these events by estimating the Cumulative Incidence Function (CIF) [7] for each lifecycle event. Our dataset includes approximately two million lifecycle events.

3. Results

The qualitative analysis reveals three recurring explanations for persistent vulnerabilities. First, maintainers often expect fixes to arrive through automated image rebuilds triggered by base image updates, even when manual remediation is possible (e.g., via overwrites). Second, users often expect maintainers to fix all vulnerabilities, while maintainers view inherited vulnerabilities as the responsibility of the upstream base image maintainers. Third, users are often unaware that affected tags have reached end-of-life and continue to expect security fixes to arrive automatically.

Quantitative results show that, implicit updates dominate remediation: 63.8% of inherited vulnerabilities are fixed through upstream rebuilds, compared with 4.5% through explicit updates and 3.3% through overwrites as depicted on Figure 1. This shows a strong reliance on automation, but limited manual intervention when automation fails.

Exposure remains substantial: 30 days after disclosure, only 22% of inherited vulnerabilities are remediated. Base-image end-of-life is a major structural barrier, leaving 11% of inherited vulnerabilities unresolved overall and 23% unresolved when they originate from deeper dependency layers. After implicit updates become impossible (i.e., due to base end-of-life or lineage end-of-life), these vulnerabilities persist for an average of 303 (median:168) additional days.

4. Discussion

Our results show that inherited vulnerabilities in container ecosystems often persist because automated patch propagation depends heavily on base lineages that may silently reach end-of-life (EOL). This problem becomes more severe as dependency chains grow: when a deeper base image reaches EOL, patch propagation can stop for all derived images in the chain. These findings suggest that better visibility into EOL status is essential for reducing long-term vulnerability exposure.

Overall, the security benefits of immutable infrastructures depend not only on automation, but also on clear maintenance practices and accountability across dependency hierarchies. Improving transparency around lineage status, integrating stronger CI/CD testing to facilitate smooth migration to maintained base images in case of EOL, and clarifying downstream remediation strategies are therefore important steps toward reducing vulnerability exposure in container supply chains [8].

References

[1] D. Lorenc and M. Kaczorowski, "Exploring container security: How containers enable passive patching and a better model for supply chain security | Google Cloud Blog — cloud.google.com," https://cloud.google.com/blog/products/containers-kubernetes/exploring-container-security-how-containers-enable-passive-patching-and-a-better-model-for-supply-chain-security, 2018, [Accessed 21-01-2024].

[2] R. Shu, X. Gu, and W. Enck, "A study of security vulnerabilities on docker hub," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, 2017, pp. 269–280.

[3] A. Zerouali, T. Mens, G. Robles, and J. M. Gonzalez-Barahona, "On the relation between outdated docker containers, severity vulnerabilities, and bugs," in *2019 ieee 26th international conference on software analysis, evolution and reengineering (saner)*. IEEE, 2019, pp. 491–501.

[4] M. U. Haque and M. A. Babar, "Well begun is half done: An empirical study of exploitability & impact of base-image vulnerabilities," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022, pp. 1066–1077.

[5] I. Docker, "Docker Official Images," https://github.com/docker-library/official-images, 2024, available at https://github.com/docker-library/official-images. [Online]. Available: https://github.com/docker-library/official-images

[6] D. Library, "Docker Official Images: Repo-info," https://github.com/docker-library/repo-info, 2024, available at https://github.com/docker-library/repo-info. [Online]. Available: https://github.com/docker-library/repo-info

[7] P. C. Austin, D. S. Lee, and J. P. Fine, "Introduction to the analysis of survival data in the presence of competing risks," *Circulation*, vol. 133, no. 6, pp. 601–609, 2016. [Online]. Available: https://www.ahajournals.org/doi/abs/10.1161/CIRCULATIONAHA.115.017719

[8] S. Tekin, O. Suciu, S. Kwag, Y. Kwon, and T. Dumitras, "Death is not the end: A longitudinal study on the impact of automatic updates on container vulnerability lifespans," in *2026 IEEE Symposium on Security and Privacy (SP)*, 2026.

Death Is Not the End: A Longitudinal Study on the Impact of Automatic Updates on Container Vulnerability Lifespans

Simge Tekin[1], Octavian Suciu[2], Sungsu Kwag[1], Yonghwi Kwon[1], Tudor Dumitras[1]
University of Maryland, College Park[1] Google Research[2]

Pitfalls of Automated Patching

- **Container images facilitate automated patching:** when a base image is updated, fixes propagate to derived images through image rebuilds.
- **This automation is fragile:** when a base image reaches end-of-life, patch propagation stops, leaving inherited vulnerabilities unresolved.
- Our analysis of 54 GitHub issues shows that **users are often unaware that affected tags have reached end-of-life** and are **confused about who is responsible for patching** once automation stalls.

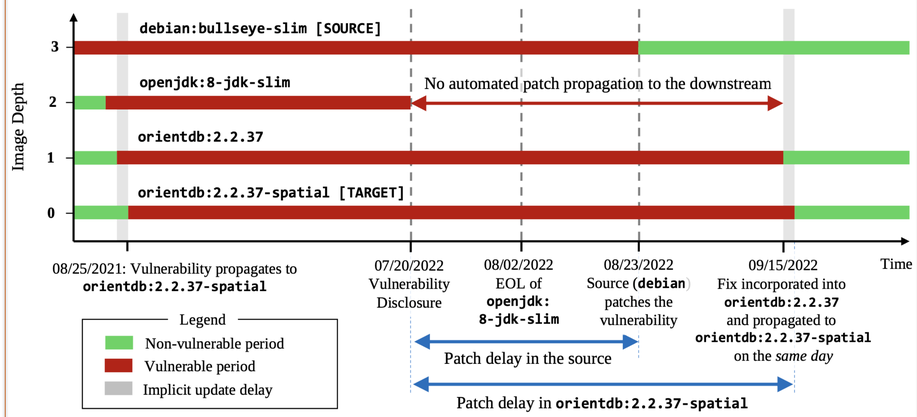

Figure 1: Lifespan of CVE-2021-46828, propagating through the base image chain. An intermediate end-of-life (EOL) at depth 2 (openjdk:8-jdk-slim) interrupts patch flow, extending vulnerability exposure downstream.

Methodology

- We introduce lineage as the chronologically ordered set of images referenced by the same tag. A vulnerability's lifespan is defined from the first image in the lineage that contains it to the last.
- We model vulnerability lifecycles using a multistate survival framework that captures interdependent events—such as base-image end-of-life and multiple patching modes—across more than 9,000 CVEs in 756,313 Docker images spanning 137 applications over 6+ years.

Results

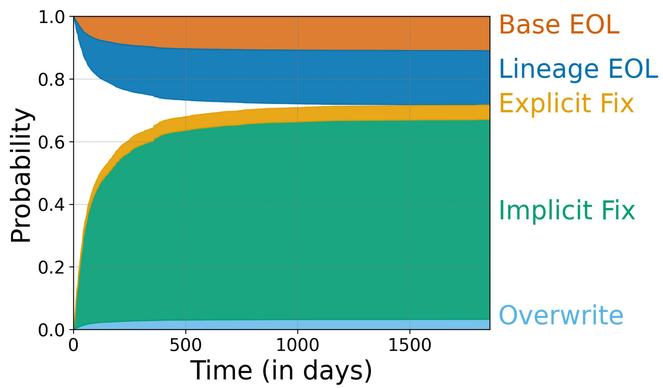

Figure 2: Cumulative incidence function (CIF) for events in inherited vulnerability lifespan, throughout the 6-year, 7-month observation period.

- **Most vulnerabilities persist:** Only 22% of inherited vulnerabilities are fixed within 30 days after disclosure.
- **Automation dominates:** 63.8% of fixes occur via automated (implicit) updates.
- **End-of-life breaks patch propagation:** 11% of vulnerabilities (up to 23% in deeper layers) remain unresolved due to base-image EOL.
- **Manual intervention is rare:** Explicit updates (4.5%) and overwrites (3.3%) are infrequently used, even when automation fails.

Actionable Recommendations

Based on our findings, we provide the following recommendations for container image developers and maintainers:

- **Integrate end-of-life tracking into CI/CD,** so developers know when a base lineage stops receiving updates.
- **Choose base lineages using lineage relationships** rather than relying on semantic tags alone, since structural lineage relationships are a more reliable indicator of continued maintenance.
- **Integrate comprehensive testing into CI/CD**, so maintainers can make migrate to new base tags more safely when automatic patch flow breaks.
- **Utilize overwriting for fixing inherited vulnerabilities** when the base image is already EOL or upstream fixes cannot arrive fast enough.