Deciphering Discrepancies: A Comparative Analysis of Docker Image Security

Brittany Boles*, Eric O'Donoghue*, A. Redempta Manzi Muneza*, Garrett Perkins*
Clemente Izurieta*†‡, Ann Marie Reinhold*‡

*Gianforte School of Computing, Montana State University, Bozeman, Montana, USA
†Idaho National Laboratory, Idaho Falls, Idaho, USA
‡Pacific Northwest National Laboratory, Richland, Washington, USA

***Abstract*—As the use of microservices continues to grow and become a foundational approach to architecting software solutions, ensuring the security of microservices is paramount. Docker images have emerged as the predominant solution to containerize microservices–and thus, Docker images are becoming a large attack surface. Thus, reducing vulnerabilities in Docker images will reduce microservice cyberattacks. A common way to find vulnerabilities in Docker images employs static analysis tools like Trivy and Grype. However, these tools frequently generate disparate vulnerability reports when analyzing the same Docker image, thus causing uncertainty in tool selection. We collected 927 Docker images, analyzed them with Trivy and Grype, and compared the vulnerabilities reported in each image. Among the 865 images found to have vulnerabilities, Trivy and Grype disagreed on both the number of vulnerabilities and the vulnerability IDs found therein. Since both tools interface with external vulnerability databases, some discrepancies can be attributed to how the tools interface with these external resources. The external vulnerability databases partially overlap and frequently contradict one another, thereby creating challenges for static analysis tool developers and end users alike. This New Ideas and Emerging Results (NIER) study contains new and critical information that practitioners need for selecting and using static analysis tools–given that increases in the use of Docker technologies means increases in the size of the attack surfaces.**

I. Introduction

Microservices have emerged as an improved alternative to monolithic architectures and are commonplace in contemporary software solutions. Microservices offer many benefits, such as increased modularity, flexible configuration, simplified development, easier maintenance, and heightened productivity [1]. Prominent companies, including Netflix, Amazon, and Uber, have embraced the adoption of microservice architectures. As the use of microservice architectures grow, so to does the containerization technology, which provides details for the microservices. Docker has emerged as a front-runner containerization technology of microservices with reportedly over 75,000 company customers[3].

Docker images are used to initialize Docker containers, which in turn realize microservice solutions. Docker images layer-based architecture alleviates challenges associated with setting up development environments, making Docker highly desired. According to a survey conducted in 2023, Docker was reported as the most widely used tool with most developers expressing their intention to continue its usage in 2024[4].

As the use of Docker and microservices become the industry standard consequently security risks have emerged. Vulnerabilities within Docker images can allow bad actors to implement cyber attacks. A vulnerability is defined by the National Vulnerability Database (NVD[5]) as "A weakness in the computational logic (e.g., code) found in software and hardware components that, when exploited, results in a negative impact to confidentiality, integrity, or availability." Vulnerability databases such as the NVD and GitHub advisories[6] store hundreds of thousands of known vulnerability details. Many of these vulnerabilities reside in Docker images, including Docker images in active use. One study found that community and official Docker images had more than 180 vulnerabilities on average, with many of the vulnerabilities being persistent between versions [2]. These findings of widespread vulnerabilities in Docker images brings light to just how large the attack surface is.

As more developers adopt Docker technology, limiting vulnerabilities is critical. Static analysis of Docker images is a common strategy for assessing cybersecurity risks [3]. Static analysis tools report known vulnerabilities within Docker image packages and artifacts. The benefits of static analysis tools include their ability to scan software for vulnerabilities without executing the software, and fast processing speeds [4]. However, static analysis tools are not without fault; these tools only report known vulnerabilities and produce a concerning number of false positives—reducing trust [5]. Thus, end users face the challenge of picking the tool that fits their needs and has trustworthy results.

Two popular static analysis tools for Docker images are Trivy[7] and Grype[8]. These open-source static analysis tools have become an industry standard for analyzing Docker images. Both tools leverage vulnerability databases to report

[3]www.docker.com/trust/

[4]https://survey.stackoverflow.co/2023/
[5]https://nvd.nist.gov/vuln
[6]https://github.com/github/advisory-database
[7]https://github.com/aquasecurity/trivy
[8]https://github.com/anchore/grype

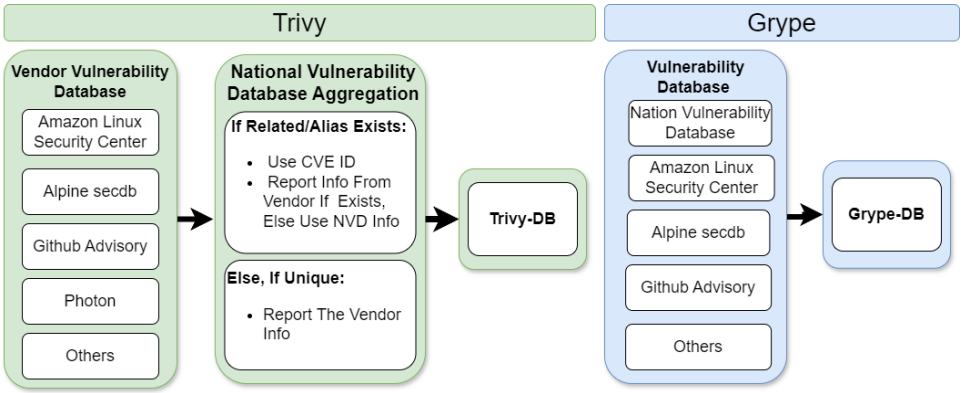

Fig. 1. Schematic representing the process Grype (blue) and Trivy (green) use to create the Grype-DB and Trivy-DB, respectively. Both tools use the NVD but in different ways. Trivy aggregates information between databases into one source. Note Grype-DB uses Vunnel[2], an Anchore tool, to pull the different vulnerability databases.

vulnerabilities in Docker images, software bill of materials (SBOMs), and file systems. Trivy and Grype purportedly provide the same information: reports of all known vulnerabilities in a software artifact. Despite these similarities, we found they frequently yield different results [6] [7]. For instance, in a corpus of 1,151 Software Bills of Materials, Trivy reported 309,022 vulnerabilities whereas Grype reported 43,553 vulnerabilities [7]. Yet, the reasons why Trivy and Grype produce such different reports has never been investigated systematically. The absence of a systematic evaluation is problematic. Without a systematic evaluation of each tool and comparison of their results, end users of these tools are faced with numerous uncertainties. *Which tool should I trust? Why are the results so different?* These uncertainties are impediments for researchers and practitioners who rely on static analysis tools for assessing the security of Docker images. Here, we evaluate and compare the results of Trivy and Grype on a common set of targets and investigate the reasons underpinning the differences.

Our study addresses the following research goal: to systematically evaluate the differences in the vulnerabilities reported by Trivy and Grype in a large corpus of Docker images.

II. Background

Trivy and Grype rely on external databases to give them trustworthy vulnerability information. Transitively, the users of Trivy and Grype are also reliant on these databases. These databases contain important information about each vulnerability that is codified using a vulnerability ID. Each entry also contains metadata for each vulnerability; examples of metadata include the severity of the vulnerability and where the vulnerability is found. Knowing which set of external databases each tool uses and how each tool aggregates all vulnerability information from each database is critical.

A. External Databases

Trivy and Grype pull vulnerability details from external databases to create Trivy-DB and Grype-DB, their respective internal vulnerability databases1. Trivy and Grype use many of the same external databases, including the largest, most widely used databases, GitHub Advisories, with 211,900 vulnerabilities, and the NVD, with 243,544. Though the tools share many external databases, Trivy uses nine additional databases, referred to as 'vendor vulnerability databases'.

The databases often use their own labeling conventions for vulnerabilities, creating unique ID types. Common Vulnerabilities and Exposures (CVE) IDs are the most used ID type for vulnerabilities. CVEs are primarily assigned by CVE-numbering authorities, and enriched by the NVD. Other vulnerability labels include GitHub Security Advisories (GHSA) IDs or Amazon Linux AMI Security Advisory (ALAS) IDs, created by GitHub Advisories[9] and Amazon[10] respectively. When two databases contain the same vulnerability, they are referred to as a 'related' vulnerability or an alias. These related vulnerabilities can either share the same ID or be labeled with different IDs. Taking the infamous Log4j vulnerability for example, the NVD labels it CVE-2021-44228, whereas the GitHub advisories label it GHSA-jfh8-c2jp-5v3q. However, it is the same underlying vulnerability. Thus, related vulnerabilities are the same issue documented in different databases.

B. Internal Aggregation of Vulnerabilities

Developers of Trivy and Grype face a formidable challenge. They need to aggregate vulnerability information from multiple databases- some overlapping or unique to each database- in the creation of their internal vulnerability databases. Both tools compile lists of all the Docker image components, imported packages, operating systems, etc., all with specific versions. The tools look for matches between the components of each Docker image and known vulnerabilities contained in each tool's respective internal database.

Trivy aggregates information from the NVD and the other databases to create Trivy-DB[11] as follows. Trivy combines vulnerabilities related to entries in the NVD into one local

[9]https://github.com/advisories
[10]https://alas.aws.amazon.com/
[11]https://github.com/aquasecurity/trivy-db

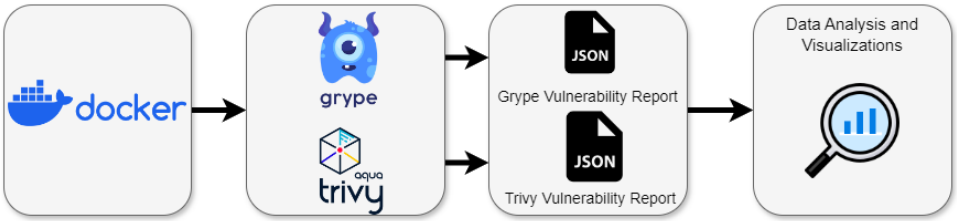

Fig. 2. Process Diagram of the data pipeline. Docker images are pulled; then desired versions of Trivy and Grype are downloaded. Each image is run through each Trivy and Grype and the results are aggregated. Finally, those results are analyzed and data visuals of the reported vulnerabilities are built.

TABLE I
THE DIFFERENT VULNERABILITY DATABASES THAT TRIVY AND GRYPE USE. TRIVY USES MORE VULNERABILITY DATABASES THAN GRYPE

Data Source	Grype	Trivy
NVD	x	x
Alpine	x	x
Amazon	x	x
Debian	x	x
GitHub Advisories	x	x
Oracle	x	x
Redhat	x	x
SUSE	x	x
Ubuntu CVE Tracker	x	x
Photon Security Advisory	-	x
Arch Linux	-	x
CBL Mariner	-	x
Node js Security	-	x
GitLab Advisories Community	-	x
AlmaLinux	-	x
RubySec	-	x
PhP Security	-	x
Rocky Linux	-	x

source, i.e., Trivy-DB. Trivy uses the CVE labelling from the NVD but uses the vendor database information to populate other data fields such as severity. This relabeling process means most entries in Trivy-DB will have CVE IDs.

The aggregation process for Grype differs than for Trivy. Grype combines the vulnerability databases listed in Table I to create Grype-DB[12]. Each vulnerability ID in the set of external database becomes an entry in Grype-DB. Grype-DB contains a related vulnerability field that informs users when a reported ID has a related vulnerability in the NVD. The related vulnerability field is *only* populated when vendor databases or GitHub Advisories have a related vulnerability in the NVD. These relationships are either provided by the databases, or are obvious because the related IDs are identical to the CVE ID. In the case of a vendor ID being related to multiple vulnerabilities in the NVD Grype fills the related vulnerability field with multiple vulnerability IDs.

The NVD presents vulnerability information using common platform enumeration (CPE)[13]. Historically, Grype used CPE to match components of Docker images to known vulnerabilities in the NVD. In an effort to reduce the reporting of false positives, Grype now limits the use of CPE matching[14]. Grype relies on other databases like Github Advisories, and states some ecosystems could see up to an 80% reduction in false positives with this change.

III. METHODS

We began by creating a corpus of official Docker images (Fig. 2). We built a data pipeline in Python (https://doi.org/10.5281/zenodo.13380592). We evaluated ten versions (evenly spaced through all the available versions) of the top 97 most pulled Docker Official images[15] on Docker Hub as of February 1, 2024. Only versions compatible with a Linux system were included. If fewer than ten versions of a Docker image were released, we used all available versions.

We selected Trivy and Grype because of their popularity. As of August 2024, the GitHub repository for Trivy had been forked 2,200 times and Grype had been forked 542 times. In addition, Trivy and Grype were recommended by SECL (https://www.montana.edu/cyber/) industry and government subject matter experts. We selected the most recent versions of Grype (v0.73.0) and Trivy (v0.49.0) for tool comparison on February 1, 2024. With Trivy, we used the configuration "-timeout 30m" because several images failed with an "analyze error: timeout: context deadline exceeded". We found a timeout of 30 minutes was more than satisfactory to prevent this error.

We ran every Docker image in our collection through Trivy and Grype. Some Docker images could not be analyzed by both tools. For instance, Grype returned empty JSON files for all versions of the Docker image *Mono* as well as the images alpine:3.17.1, alpine3.18.5, and alpine 3.18.2, so these images were removed from our corpus. Trivy did not process *golang:1.4rc1*, and we removed this image from the corpus.

We implemented two primary controls for repeatability and validity. First, we downloaded the tools' databases on November 11, 2023. Thus, both tools used static databases to avoid updates throughout the study. The databases used for both tools are available at the DOI provided above. Second, both tools analyzed the exact same corpus of Docker images.

The reports from Trivy and Grype are depicted in the third step of our process diagram (Fig. 2). In the fourth step, we processed the tool reports and analyzed them.

We computed the difference in counts reported by Grype and Trivy in each version of each Docker image. We visualized the difference in distributions with a density plot (Fig. 3). We

[12]https://github.com/anchore/grype-db
[13]https://cpe.mitre.org/about/index.html
[14]https://anchore.com/blog/say-goodbye-to-false-positives/
[15]https://hub.docker.com/search?image_filter=official

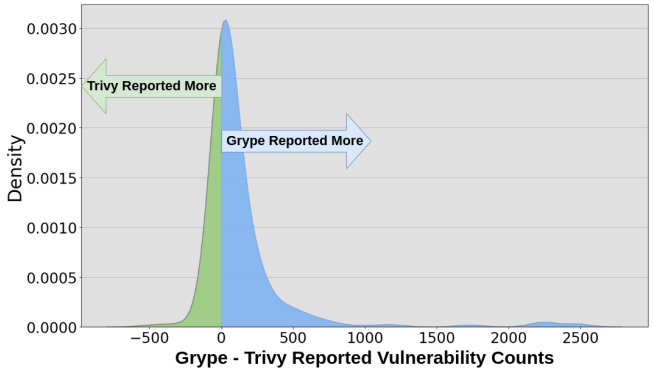

Fig. 3. A density plot of the differences in Trivy and Grypes' reported vulnerabilities per image. The x-axis represents the difference between Trivy and Grypes' total count for each image. Both tools were analyzed over the corpus of 927 Docker images. In total, Grype reported 603,259 vulnerabilities, and Trivy reported 473,661.

also calculated descriptive statistics, i.e., the average count of vulnerabilities in each image and the associated standard deviations.

IV. Results

Results from Trivy and Grype rarely agreed (Table II; Fig. 3). Grype found more vulnerabilities than Trivy in 84.6% of the Docker images. Over the 927 images in the corpus, Trivy found 473,661 vulnerabilities, whereas Grype found 603,259.

Of the 603,259 vulnerabilities Grype reported in the corpus, 577,307 had a related vulnerability in the NVD. With respect to these related vulnerabilities in the NVD, Grype only double-counted vulnerabilities 675 times. These instances of double-counting are indicative of redundant reports for the same issue. However, there were relatively few instances. Thus, double-counting is not a primary reason that Grype generally reported more vulnerabilities than Trivy.

The magnitude of the differences between the tool findings was often large. On average, Trivy reported ∼140 fewer vulnerabilities than Grype (S.D. = 357.0). Furthermore, in 6.7% of the images, Trivy and Grype reported differences exceeding 500 vulnerabilities. The largest difference was found in image python: 3.7.6-stretch, with a difference of 2,516 vulnerabilities. That is, Trivy found 3,208 vulnerabilities, whereas Grype found 5,724.

Trivy and Grype only agreed on the number of vulnerabilities in 15.3% of the images (142 of 927 images). In 62 of these 142 images, Trivy and Grype found zero vulnerabilities. Of the remaining 865 images found to contain vulnerabilities, Trivy and Grype found the same number of findings in 9.2% of those images. That is, only 80 images were found to have the same number of non-zero findings. However, even when Trivy and Grype reported the same number of vulnerabilities, the identities of the vulnerabilities (e.g., CVE IDs) *never* agreed. The only instance where the tools were unanimous in their findings was when they both found nothing.

TABLE II
Counts of the different of vulnerability IDs reported by Trivy and Grype in the corpus of Docker images. The "Other" column is comprised of ID types DLA, DSA and NSWG.

Tool	CVE	GHSA	ALAS	ELSA	Other	Total
Trivy	466,592	965	0	0	6104	437,661
Grype	573,953	26,028	1,462	1,816	0	603,259

We also observe differences in the types of IDs reported (Table II). The external vulnerability databases have different labeling schemes, such as GHSA IDs from GitHub Advisories or CVE IDs from the NVD. Different counts of label types reported by Trivy and Grype arise because they do not use the same set of databases, and they use those databases differently (Fig. 1 and Table I). Consequently, Trivy reported ID types Debian Linux Advisory (DLA), Debian Security Advisory (DSA), and Node.js Security Working Group (NSWG), all of which Grype never reported in our corpus. Conversely, Grype reported ID types Enterprise Linux Security Advisor (ELSA) and ALAS, which Trivy never reported. Trivy also reported a greater percentage of vulnerabilities as CVEs compared to Grype (Trivy = 98.5%, n_{Trivy} = 466,592; Grype = 95.1%, n_{Grype} = 573,953 vulnerabilities).

The discrepancies between Trivy and Grype extended to the metadata associated with vulnerabilities. The severity of the vulnerabilities is an important example of this discord. While we did not systematically evaluate severities, we found extensive anecdotal evidence of differences between the tool reports. In total, the tools disagreed 60,799 times on the severity of vulnerability IDs that were identical. Taking CVE-2019-17594 as an example, Grype reported a 'medium' severity for this CVE, whereas Trivy reported a 'low' severity. We even found instances (e.g., CVE-2019-8457) where Grype labeled a vulnerability as 'negligible' whereas Trivy reported the same vulnerability as 'critical'.

V. Discussion

Trivy and Grype disagreed on the count, IDs, and severity of the vulnerabilities in the corpus, begging the question, *What is causing these differences?* Here, we unpack the reasons for the differences in the vulnerabilities reported by Trivy and Grype. We consider how the tools interact with external databases (see Subsection II-A) and how they aggregate information from those databases internally (see Subsection II-B).

A. Different Sets of External Vulnerability Databases

The external databases used by Trivy and Grype impact the IDs reported by the tools (Table II). However, differences in external databases are not primary factors driving the differences in vulnerability counts. Evidence for this observation includes that Trivy found fewer vulnerabilities despite pulling information from nine more external databases than Grype. This observation was surprising because we expected having more vulnerability information would result in finding more vulnerabilities.

In contrast, the differences in the sets of external databases used to create Trivy-DB and Grype-DB lead to different types of IDs being reported. Trivy, for instance, reports NSWG vulnerability IDs sourced from Node js Security[16], a database not utilized by Grype (Table II). On the other hand, Grype reports the ID type ELSA, which Trivy does not. The documentation from Trivy indicates that ELSA IDs were likely relabelled to their related CVE IDs, causing Trivy to never report ELSA IDs. This data aggregation leads to the high percentage of CVE IDs reported by Trivy. Thus, a primary driver of the differences reported by Trivy and Grype is *how the tools aggregate information from external databases* to create their internal vulnerability databases.

B. Different Internal Aggregation Processes

How the tools aggregated information from the external vulnerability databases, specifically handling related vulnerabilities, greatly impacted their results. Their distinct processes affected vulnerability IDs reported, their severities, and the total number of vulnerabilities reported.

Grype reported 95.7% vulnerabilities had a related vulnerability ID in the NVD, illustrating the extent to which the vendor vulnerability databases overlap with the NVD. Further, external vulnerability databases may also overlap with each other. We speculate that Grype reported more vulnerabilities (Fig. 3) because of how they handles related IDs.

Further research is needed to understand if Grype is reporting multiple IDs for the same underlying vulnerability and the degree to which redundancies may impact overall count. Validation into the aggregation processes used by Trivy is also needed; it remains unclear how Trivy handles multiple related vulnerabilities. While the aggregation process used by Grype can cause redundancies, it is transparent.

Vulnerability databases often disagree on the severity of vulnerabilities [8]. Disagreement between the databases caused Trivy and Grype to report different severities for the same vulnerability IDs. This disagreement arose because the tools can pull from different databases to populate metadata for the same vulnerability ID (Fig. 1).

Disagreement in severities is particularly problematic. End users of Trivy and Grype often filter the reported vulnerabilities by severity to tackle the most severe issues first. Thus, disagreement in severities creates a problem for the developers of Trivy and Grype, as well as for their end users who rely on these tools for vulnerability assessment.

It is also important to note that many vulnerability databases contain inaccuracies and duplications [9]. These inaccuracies and duplications present an additional source of uncertainty. Determining which source of vulnerability information static analysis tools should use–and how that information should be aggregated–is challenging when databases disagree.

C. NIER Considerations & Challenges

While we found variation in the vulnerability reports produced by Trivy and Grype, this research does not imply that these tools should not be used. Instead, we highlight challenges associated with their use–in particular, the challenges that arise from discrepancies between the tools and inaccuracies in their underlying vulnerability databases. Ultimately, if Trivy and Grype are pulling inaccurate data from the vulnerability databases, their results will also be inaccurate.

Some research shows that NVD has robust and trustworthy severity ratings [10]. Using the NVD is mandated for federal contractors to use as an authoritative source of threats by many federal government programs such as the Federal Risk and Authorization Management[17] program. However, the NVD is far from perfect.

While both Trivy and Grype use the NVD, these tools favor information from vendor databases and GitHub Advisories. Grype limits matching with the NVD. Trivy specifies choosing vendor severity ratings over the NVD. Trivy documentation states that vendor severity ratings are more accurate than the NVD[18] and Red Hat agrees [19]. We also suspect that the developers of Trivy and Grype minimize the use of the NVD because of notoriously long lag times and pervasive inaccuracies [11]. These lags and inaccuracies can contribute to the tools reporting false positives, one of the most significant complaints of end users of static analysis tools [12].

Lags and inaccuracies are not unique to the NVD. However, the impact of these flaws is exacerbated by the NVD's prominence, large size, and processing procedure. A comprehensive database loses value if it cannot be trusted.

Each vulnerability database compiles, updates, and validates a rapidly increasing number of vulnerabilities. However, keeping pace with the volume and velocity of this data is a formidable challenge. Simplifying and aggregating the information across these databases is crucial, as manually gathering and verifying vulnerability information is both time-consuming and tedious. Our current work takes aim at surmounting these challenges directly.

We are currently consolidating the vulnerability information in these diverse databases into a unified, searchable graph database that links shared vulnerabilities. Our graph database will (1) streamline the vulnerability detection process by facilitating collaboration among database creators/maintainers, (2) aid static analysis developers by presenting comprehensive information in aggregate, and (3) enable end users and static analysis tool developers to make informed decisions when sources conflict. Our goal is to make the secure choice, the easy choice.

D. Threats to Validity

Our study was highly controlled – using stringent data processing steps to ensure replicability. Moreover, our study design minimizes threats to internal validity. However, we examine three other potential threats to validity: construct validity, content validity, and external validity using the classification scheme of Cook et al. [13] and Campbell et al. [14].

[16]https://github.com/nodejs/security-wg

[17]www.fedramp.gov/2024-02-16-rev-5-additional-documents-released/

[18]https://aquasecurity.github.io/trivy/v0.49/docs/scanner/vulnerability/

[19]www.redhat.com/en/blog/security-flaws-and-cvss-rescore-process-nvd

We consider potential threats to the construct and content validity of our study. In particular, our study assumes that a single version of Trivy and Grype produces a meaningful and comprehensive assessment of the vulnerabilities in each Docker image in the corpus. However, other studies have found that different versions of the same static analysis tool can produce different results even when holding the corpus of software artifacts constant [6], [15]. Although this "version variation" is known to exist for Trivy and Grype [6], Trivy and Grype have proven to produce more consistent and reliable results across versions than some binary analysis tools [15]. Thus, we acknowledge these potential threats but do not perceive them to undermine the overall conclusions of our study. Moreover, our systematic design can easily be extended to include multiple versions of Trivy and Grype.

The external validity of our study primarily hinges on the breadth of the corpus of Docker images. We analyzed 10 version of the 97 most frequently downloaded Official Docker images that run on Linux. Therefore, our study is representative of Official Docker images that run on Linux. Trivy and Grype may produce different results when analyzing Docker images that are not Official or that run on other operating systems. Thus, a modicum of caution is warranted when extrapolating these results beyond the scope of the corpus.

VI. Conclusions & Future Directions

Our work here introduces several challenges. The disagreement between results from Trivy and Grype bring the following questions to the forefront: *How reliable and accurate are Trivy and Grype? Additionally, how can we improve the reliability and accuracy of the vulnerability databases on which these tools depend?* These questions point to critical research challenges. Surmounting these challenges will reduce the attack surfaces of microservices. The high number of vulnerabilities we found in the corpus of popular, Official Docker images suggests that microservices are far from secure; this statement would hold true even if half of the reported vulnerabilities were false positives. Therefore, microservices are–and will continue to be–prime targets for malicious actors.

As with all wicked problems, there are no silver bullets. However, we need to come together as a community to discuss how to surmount these challenges. Moreover, we advocate for building connections among the creators and managers of vulnerability databases, the developers of cybersecurity static analysis tools, and the researchers and practitioners who depend on these databases and tools.

Acknowledgments

This research was conducted with support from the U.S. Department of Homeland Security (DHS) Science and Technology Directorate (S&T) under contract 70RSAT22CB0000005. Any opinions contained herein are those of the author and do not necessarily reflect those of DHS S&T. Thanks to Gabe Cowley, Sabrina Hendricks, Madie Munro, Russell Conti, Yvette Hastings, Zach Wadhams, Ally Buhr, Ashley Boles, Emma Sheppard, Tom McElroy, and Jimmy Boles for their contributions to this work.

References

[1] M. Söylemez, B. Tekinerdogan, and A. Kolukısa Tarhan, "Challenges and solution directions of microservice architectures: A systematic literature review," *Applied Sciences*, vol. 12, no. 11, 2022. [Online]. Available: https://www.mdpi.com/2076-3417/12/11/5507

[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*, ser. CODASPY '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 269–280. [Online]. Available: https://doi.org/10.1145/3029806.3029832

[3] K. Brady, S. Moon, T. Nguyen, and J. Coffman, "Docker container security in cloud computing," in *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, 2020, pp. 0975–0980.

[4] A. Aggarwal and P. Jalote, "Integrating static and dynamic analysis for detecting vulnerabilities," in *30th Annual International Computer Software and Applications Conference (COMPSAC'06)*, vol. 1, 2006, pp. 343–350.

[5] F. Cheirdari and G. Karabatis, "Analyzing false positive source code vulnerabilities using static analysis tools," in *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 4782–4788.

[6] A. M. Reinhold, B. Boles, A. R. M. Muneza, T. McElroy, and C. Izurieta, "Surmounting challenges in aggregating results from static analysis tools," *Military Cyber Affairs*, vol. 7, 2024. [Online]. Available: https://digitalcommons.usf.edu/cgi/viewcontent.cgi?article=1101&context=mca

[7] E. O'Donoghue, A. M. Reinhold, and C. Izurieta, "Assessing security risks of software supply chains using software bill of materials," in *2nd International Workshop on Mining Software Repositories for Privacy and Security*. IEEE International Conference on Software Analysis, Evolution and Reengineering, 2024, [In-press].

[8] R. Croft, M. A. Babar, and L. Li, "An investigation into inconsistency of software vulnerability severity across data sources," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2022, pp. 338–348.

[9] R. Croft, M. A. Babar, and M. M. Kholoosi, "Data quality for software vulnerability datasets," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2023, pp. 121–133.

[10] P. Johnson, R. Lagerström, M. Ekstedt, and U. Franke, "Can the common vulnerability scoring system be trusted? a bayesian analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 6, pp. 1002–1015, 2018.

[11] A. Anwar, A. Abusnaina, S. Chen, F. Li, and D. Mohaisen, "Cleaning the nvd: Comprehensive quality assessment, improvements, and analyses," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 6, pp. 4255–4269, 2022.

[12] Z. Wadhams, A. M. Reinhold, and C. Izurieta, "Automating static code anlaysis through ci/cd pipeline integration," in *2nd International Workshop on Mining Software Repositories for Privacy and Security*. IEEE International Conference on Software Analysis, Evolution and Reengineering, 2024, [In-press].

[13] T. Cook and D. Campbell, *Quasi-experimentation: Design & Analysis Issues for Field Settings*. Houghton Mifflin, 1979. [Online]. Available: https://books.google.com/books?id=BFNqAAAAMAAJ

[14] D. Campbell and J. Stanley, *Experimental and Quasi-experimental Designs for Research*. R. McNally, 1966. [Online]. Available: https://books.google.com/books?id=kFtqAAAAMAAJ

[15] A. M. Reinhold, T. Weber, C. Lemak, D. Reimanis, and C. Izurieta, "New version, new answer: Investigating cybersecurity static-analysis tool findings," in *2023 IEEE International Conference on Cyber Security and Resilience (CSR)*, 2023, pp. 28–35.