Why Johnny Can't Use Secure Docker Images: Investigating the Usability Challenges in Using Docker Image Vulnerability Scanners through Heuristic Evaluation

Taeyoung Kim*
tykim0402@skku.edu
Department of Electrical and Computer Engineering
Sungkyunkwan University
Suwon, Republic of Korea

Seonhye Park*
qkrtjsgp08@skku.edu
Department of Computer Science and Engineering
Sungkyunkwan University
Suwon, Republic of Korea

Hyoungshick Kim
hyoung@skku.edu
Department of Electrical and Computer Engineering
Sungkyunkwan University
Suwon, Republic of Korea

ABSTRACT

This paper explores the usability of Docker Image Vulnerability Scanners (DIVSes) through heuristic evaluations. Docker simplifies the process of software development, distribution, deployment, and execution by providing a container-based execution environment. However, vulnerabilities in Docker images can pose security risks to containers. To mitigate this, DIVSes are crucial in helping developers identify and address these vulnerabilities in the software packages and libraries within Docker images. Despite their importance, research on the usability of DIVSes has been limited. To address this gap, we developed 11 customized heuristics and applied them to three widely-used DIVSes (Grype, Trivy, and Snyk). Our evaluations revealed 239 usability issues within the tools evaluated. Our findings highlight that the evaluated DIVSes do not provide sufficient information to comprehend the risks associated with identified vulnerabilities, prioritize them, or effectively fix them. Our study offers valuable insights and practical recommendations for enhancing the usability of DIVSes, making it easier for developers to identify and address vulnerabilities in Docker images.

CCS CONCEPTS

• **Security and privacy → Usability in security and privacy**; **Software and application security**.

KEYWORDS

Container Images, Vulnerability Scanners, Heuristic Evaluation

ACM Reference Format:
Taeyoung Kim, Seonhye Park, and Hyoungshick Kim. 2023. Why Johnny Can't Use Secure Docker Images: Investigating the Usability Challenges in Using Docker Image Vulnerability Scanners through Heuristic Evaluation. In *The 26th International Symposium on Research in Attacks, Intrusions and Defenses (RAID '23), October 16–18, 2023, Hong Kong, China.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3607199.3607244

*Both authors contributed equally to this research.

RAID '23, October 16–18, 2023, Hong Kong, China

ACM ISBN 979-8-4007-0765-0/23/10…$15.00
https://doi.org/10.1145/3607199.3607244

1 INTRODUCTION

Docker is a platform that enables developers to build, ship, and run software applications using Docker Engine, a lightweight and portable packaging tool that utilizes container-based virtualization [25]. Docker creates container environments using Docker images, which bundle all necessary files, such as applications, libraries, and their runtime dependencies, into a single package. This approach streamlines application development by eliminating repetitive and time-consuming configuration tasks. Docker images can be uploaded to remote registry services like Docker Hub [4], allowing users to share and download Docker images easily.

The popularity of Docker images has grown significantly, with over 7 million application container image repositories on Docker Hub as of February 2023. Over 8 million developers use Docker images, resulting in over 13 billion monthly image downloads [3].

However, Docker images can become vulnerable to attacks (*e.g.*, privilege escalation, use-after-free, denial of service) due to the use of outdated packages with security vulnerabilities [34, 40]. In 2020, cybersecurity firm Prevasio discovered that 51% of the 4 million analyzed Docker images had vulnerabilities that could be exploited [32]. Even Docker images stored in reputable container registries are not entirely safe from vulnerabilities. This was evidenced by the presence of the "Heartbleed" vulnerability (CVE-2014-0160) in various Docker images stored on the well-known registry Red Hat Quay. A security report published by cloud-native security firm Sysdig revealed that 87% of container images still contain high or critical vulnerabilities, an increase from the previous year's 75%. Surprisingly, although 71% of these vulnerabilities have available fixes, they persist. This finding aligns with the results of *Liu et al.* [24], who discovered that Docker image vulnerabilities typically remain for a long time due to the lack of updates. These statistics highlight the importance of addressing vulnerabilities in Docker images to ensure the security of containerized applications.

To address these security concerns, Docker Image Vulnerability Scanners (DIVSes) have been developed [1, 2, 5–8, 10]. These tools allow developers to scan their images for known vulnerabilities in the packages and libraries. The DIVSes then generate a report that details any identified vulnerabilities, including their severity level and a comprehensive description. The report provides a clear understanding of the security posture of the Docker image, highlighting any outdated libraries, misconfigured settings, or potential code vulnerabilities. Many DIVSes are readily available and easy to install, making them accessible to all Docker image users.

Although DIVSes can enhance the security of Docker images, their effectiveness for developers remains uncertain. Patching vulnerable images is uncommon in Docker Hub, increasing the risk of exploitation by attackers [24]. Previous studies [14, 15, 22] have emphasized the critical role of software tool usability in ensuring software product security. Therefore, improving the usability of DIVSes is crucial for building a secure Docker image ecosystem by expediting vulnerable image updates. In this paper, we establish criteria for evaluating the usability of DIVSes and assess the usability of popularly used DIVSes (*i.e.*, Grype, Trivy, and Snyk). To the best of our knowledge, this is the first study to specifically examine the usability of DIVSes. The main objective of this work is to develop a suitable *heuristic evaluation* framework to assess the usability of DIVSes and identify potential usability challenges when using them. To achieve these goals, we formulate the following research questions.

- **RQ1:** What criteria can be used to assess the usability of Docker image vulnerability scanner (DIVSes)?
- **RQ2:** Do the existing DIVSes actually help developers discover and fix vulnerabilities in Docker images?

To address our research questions, we performed a two-phase evaluation with the help of ten computer science experts. The first phase involved assessing the suitability of existing heuristics used to evaluate the usability of static analysis tools for evaluating the usability of DIVSes. With the assistance of four experts, we refined the heuristics into a final set of 11 criteria for evaluating DIVS usability. In the second phase, six evaluators conducted a heuristic evaluation [30] to identify usability issues of Grype, Trivy, and Snyk based on these 11 criteria. To ensure a comprehensive evaluation, the evaluators were given four tasks related to vulnerability detection and fixing and were prompted to identify any usability issues they encountered during the tasks. All the usability issues identified were analyzed to understand their root causes. Based on these findings, we created actionable design recommendations for tool designers to enhance the usability of DIVSes, making it easier for users to detect and resolve vulnerabilities in Docker images. Our key contributions are summarized as follows.

- We developed 11 heuristic rules specifically tailored to evaluate the usability of DIVSes.
- We used our evaluation framework to conduct a heuristic evaluation of commonly used DIVSes (*i.e.*, Grype, Trivy, and Snyk) to assess their usability. Our evaluation uncovered 239 usability issues, including inadequate information for patching and a disorganized presentation of vulnerability records without considering their priority.
- Based on the results of our heuristic evaluation, we provide actionable design recommendations to enhance the usability of the DIVSes (*i.e.*, Grype, Trivy, and Snyk).

2 BACKGROUND

This section provides an overview of Docker and DIVSes and briefly introduces the DIVSes evaluated in this paper.

2.1 Docker

Docker is a containerization technology for building, distributing, and running software applications. Docker containers are created from images, which are composed of multiple layers. Each layer holds a segment of the container's file system, including the application's binaries, libraries, and data (Figure 1). Docker caches downloaded image layers and server configurations in a local directory, reducing the need for future downloads during subsequent deployments. If image layers are not cached, Docker fetches them from a repository when deploying a container. Image layers are read-only and cannot be modified during deployment. These read-only image layers are merged with a separate read/write layer to create the container and launch the application. Docker images can be stored in registries, such as Docker Hub [4], enabling the easy distribution and execution of applications in any environment with Docker installed [17].

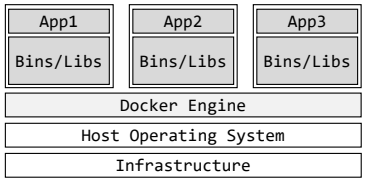


Figure 1: Docker architecture.

However, numerous software packages in a Docker image can present security challenges [24]. Addressing vulnerabilities in these packages can be difficult, leading to increased security risks within the Docker environment [24].

2.2 Docker Image Vulnerability Scanner

The Docker community offers automation tools to manage containers. Docker Image Vulnerability Scanners (DIVSes) have been developed to analyze the security vulnerabilities of Docker images. Developers can utilize a DIVS to assess the security of their Docker images by scanning for known vulnerabilities in the packages and libraries incorporated in the images. The DIVS then generates a *vulnerability report* that displays the information about any detected vulnerabilities. This report equips developers with a comprehensive understanding of their Docker image's security posture, highlighting outdated libraries, misconfigured settings, or security weaknesses. Figure 2 presents a vulnerability report produced by Grype, one of the DIVSes evaluated in this paper. The report enumerates the vulnerabilities identified by the tool and provides details for each. Specifically, each line in the report represents a *vulnerability record* that contains information such as the software name, installed software version, patched software version (if available), vulnerability type, vulnerability identifier, and severity level.

We conducted two studies to evaluate the usability of DIVSes. In the first study, we used three DIVSes (Anchore Engine, Trivy, and Snyk) to define the criteria for usability evaluation through heuristic evaluation. In the second study, we replaced Anchore Engine with Grype, assessing the usability of Grype, Trivy, and Snyk through heuristic evaluations, as Anchore Engine had reached the end of its support.

NAME	INSTALLED	FIXED-IN	TYPE	VULNERABILITY	SEVERITY
bash	5.1-6ubuntu1		deb	CVE-2022-3715	Low
coreutils	8.32-4.1ubuntu1		deb	CVE-2016-2781	Low
gpgv	2.2.27-3ubuntu2.1	Vulnerability Record	deb	CVE-2022-3219	Low
libc-bin	2.35-0ubuntu3.1		deb	CVE-2016-20013	Negligible
libc6	2.35-0ubuntu3.1		deb	CVE-2016-20013	Negligible
libgssapi-krb5-2	1.19.2-2	1.19.2-2ubuntu0.1	deb	CVE-2022-42898	Medium
libk5crypto3	1.19.2-2	1.19.2-2ubuntu0.1	deb	CVE-2022-42898	Medium
libkrb5-3	1.19.2-2	1.19.2-2ubuntu0.1	deb	CVE-2022-42898	Medium
libkrb5support0	1.19.2-2	1.19.2-2ubuntu0.1	deb	CVE-2022-42898	Medium
libncurses6	6.3-2		deb	CVE-2022-29458	Negligible
libncursesw6	6.3-2		deb	CVE-2022-29458	Negligible
libpam-modules	1.4.0-11ubuntu2	1.4.0-11ubuntu2.1	deb	CVE-2022-28321	Negligible
libpam-modules-bin	1.4.0-11ubuntu2	1.4.0-11ubuntu2.1	deb	CVE-2022-28321	Negligible

Figure 2: Example of a vulnerability report created by Grype, with each line representing a separate vulnerability record.

Anchore Engine. Anchore Engine [2], an open-source DIVS, offers a command-line interface (CLI) using its REST API. This CLI allows the scanning of Docker images on nine operating systems like Alpine, CentOS, Ubuntu, and Debian. It also supports four specific programming languages: GEM, Java Archive, NPM, and Python, enabling their package scans.

Once a vulnerability scan is complete, the Anchore Engine generates a report about all detected vulnerabilities, containing their corresponding CVE ID, software name, extension type, the link to the Debian vulnerability database, and software versions in which the vulnerability was resolved. The report also assigns a severity level to each vulnerability, categorized into one of five levels: Critical, High, Medium, Low, and Negligible.

Anchore Engine is no longer available since it was deprecated on January 10, 2022.

Grype. Grype [10] is designed to evaluate nine widely-used base images, including Alpine, Debian, and Ubuntu, as well as three language-specific packages such as GEM, Java Archive, and NPM.

Grype's vulnerability report is displayed in a tabular format, presenting all detected vulnerability records along with their corresponding software name, extension type, CVE ID, and the installed and resolved versions. By accessing the JSON-formatted report, developers can obtain additional information such as the data source (*e.g.*, the Ubuntu vulnerability database), vulnerability descriptions, and related vulnerabilities. As shown in Figure 2, each record contains the software name ("NAME"), extension type ("TYPE"), and corresponding CVE ID ("VULNERABILITY"). The installed and resolved versions are provided, labeled as "INSTALLED" and "FIXED-IN," respectively. The vulnerabilities are classified into six severity levels: Critical, High, Medium, Low, Negligible, and Unknown.

Trivy. Trivy [6] is an open-source security scanner designed to detect vulnerabilities in 16 operating systems, including Alpine Linux and Red Hat Enterprise Linux, as well as 10 language-specific packages, such as Bundler and NPM. Additionally, it can identify misconfigurations and sensitive information in container images based on custom rules defined by users.

Total: 36 (UNKNOWN: 0, LOW: 19, MEDIUM: 14, HIGH: 3, CRITICAL: 0)

Library	Vulnerability	Severity	Installed Version	Fixed Version	Title
bash	CVE-2022-3715	LOW	5.1-6ubuntu1		bash: a heap-buffer-overflow in valid_parameter_transform https://avd.aquasec.com/nvd/cve-2022-3715
coreutils	CVE-2016-2781		8.32-4.1ubuntu1		coreutils: Non-privileged session can escape to the parent session in chroot https://avd.aquasec.com/nvd/cve-2016-2781
gpgv	CVE-2022-3219		2.2.27-3ubuntu2.1		gnupg: denial of service issue (resource consumption) using compressed packets https://avd.aquasec.com/nvd/cve-2022-3219
libc-bin	CVE-2016-20013		2.35-0ubuntu3.1		sha256crypt and sha512crypt through 0.6 allow attackers to cause a denial of... https://avd.aquasec.com/nvd/cve-2016-20013
libc6					
libgnutls30	CVE-2023-0361	MEDIUM	3.7.3-4ubuntu1.1		timing sidechannel in RSA decryption https://avd.aquasec.com/nvd/cve-2023-0361
libgssapi-krb5-2	CVE-2022-42898		1.19.2-2	1.19.2-2ubuntu0.1	krb5: integer overflow vulnerabilities in PAC parsing https://avd.aquasec.com/nvd/cve-2022-42898

Figure 3: Example of the vulnerability report produced by Trivy.

Figure 3 displays an example of a vulnerability report generated by Trivy. The report organizes detected vulnerabilities by software packages and lists each record with the CVE ID ("Vulnerability"), description ("Title"), and a reference link to the Aqua vulnerability database for additional information. Moreover, the report shows the installed version and the patched version ("Fixed Version"). Trivy classifies vulnerabilities into five severity levels: Critical, High, Medium, Low, and Unknown. A summary of vulnerability statistics is also presented at the top of the report, indicating the number of detected vulnerabilities for each severity level.

Snyk. Snyk [5] offers two user interfaces for scanning security vulnerabilities: (1) an in-built command, "`docker scan`," and (2) a web interface integrated with the Docker Hub registry. This study concentrates on the usability and effectiveness of the Snyk web interface, which serves as a representative graphical user interface for DIVSes. Developers can create a project, import an image from the Docker Hub registry, and analyze its security risks using this web interface. Vulnerability information for each project is displayed on a separate page with a card-based design. Additionally, Snyk provides a time-series graph of vulnerability traces and a summary list of vulnerabilities.

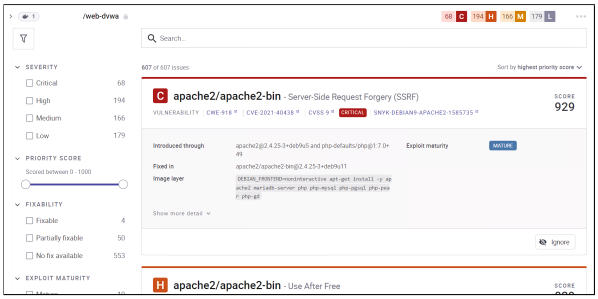


Figure 4: Example of the vulnerability report produced by Snyk.

As illustrated in Figure 4, Snyk's main project page presents the number of detected vulnerabilities, categorizing them into four severity levels (Critical, High, Medium, and Low) using distinct colors. The project page offers comprehensive information on each vulnerability, including identifiers (CWE ID, CVE ID, and vulnerability name), severity level, security score, and patched version. Furthermore, when available, Snyk provides additional information on related exploits through the "Exploit maturity" field.

3 STUDY 1: ESTABLISHING HEURISTICS

To address our first research question (**RQ1**), we developed heuristics for assessing the usability of DIVSes. To determine the effectiveness of our heuristics, we engaged four participants. The methodology used in the evaluation is outlined in Section 3.1. We present and discuss the 11 heuristics we developed in Section 3.2.

3.1 Methodology

In this study, we aim to establish a set of heuristics specifically designed to evaluate the usability of Docker image vulnerability scanners (DIVSes). While existing usability heuristics, such as Nielsen's heuristics [29], have been adapted for various studies, they are not suitable for evaluating security tools as they only consider

the user interface (UI) and user experience (UX), and do not take into account the guidance, functionality, and outputs provided by security tools. Previous research on security tool usability heuristics [27, 35, 36, 42] has primarily focused on source code analysis, making some of their heuristics inappropriate for evaluating DIVSes. For example, the heuristics proposed for source code static analysis tool usability [35, 36] include criteria such as control flow and end-user interaction, which are specific to source code analysis and not applicable to DIVSes. To address this challenge, we engaged four experts to develop domain-specific heuristics tailored to the unique features of DIVSes.

Originally, we planned to extend the heuristics proposed by *Smith et al.* (Appendix A) to assess the usability of static code analysis tools for detecting security vulnerabilities in source code. We assumed that most of these heuristics could be adapted for use in evaluating DIVSes, given the similarities between the two types of tools in presenting vulnerability results and providing information on how to address identified issues. However, we discovered that five heuristics (*i.e.*, "Control Flow & Call Information," "Data Storage & Flow," "Code Background & Functionality," "Application Context / Usage," and "End-User Interaction") that were specific to source code analysis were not applicable for evaluating DIVSes.

In addition, we created three new heuristics by modifying two existing heuristics, "Notification Text" and "Understanding Alternative Fixes & Approaches." The new heuristics are "Necessary & Sufficient Information about Vulnerabilities," "Consistency between Vulnerability Records," and "Correctness of the Fix." We changed the "Understanding Alternative Fixes & Approaches" heuristic to "Correctness of the Fix" to verify the correctness of the information provided by DIVSes in fixing reported vulnerabilities, rather than just assessing whether a solution has been proposed. We also split the "Notification Text" heuristic into two more specific heuristics: (1) "Necessary & Sufficient Information about Vulnerabilities," which evaluates whether the tool can provide adequate information about vulnerabilities, and (2) "Consistency between Vulnerability Records," which assesses the consistency of vulnerability information across multiple records.

We conducted Study 1 using 13 heuristics, referred to as our first heuristics (Appendix B). To validate their usefulness, we recruited four experienced evaluators. We asked them to apply these heuristics to assess the usability of DIVSes and share their feedback.

Evaluator Recruitment. To recruit participants experienced in using Docker or conducting heuristic evaluations, we posted a recruitment announcement on a university web bulletin board and sent emails to faculty members, staff, and computer science students. We selected four participants (P1—P4) with software development experience ranging from 2 to 10 years. Two of them had previously worked with Docker containers, and one was familiar with Docker usage. In addition, one participant had taken a Human-Computer Interaction (HCI) course. All participants were graduate students between the ages of 18 and 35 and received $100 as compensation for their participation in the study. We provided a brief tutorial video on Docker and heuristic evaluation to help participants familiarize themselves with the concepts five days before the study.

Tool Selection. To evaluate the effectiveness of the 13 candidate heuristics, we applied them to investigate Docker images using DIVSes. To select suitable DIVSes, we conducted a thorough search of academic research papers and the internet. We chose Anchore Engine (CLI) [2], Trivy (CLI) [6], and Snyk (web interface) [5] based on their popularity and accessibility. As of February 16, 2023, these tools had high usage rates with "50,217,163," "23,257,660," and "24,105,457" pull counts on Docker Hub. They are also freely available, with executable programs, documentation, and datasets easily accessible to anyone. To evaluate the effectiveness of the 13 candidate heuristics with DIVSes, we used two representative Docker images: an official Ubuntu image (version 22.04 LTS released on October 20, 2022) and a Docker image called `web-dvwa` that is a PHP/MySQL web application with multiple vulnerabilities designed for educational purposes. We carefully selected these images to provide representative samples of Docker images that contain vulnerable software packages commonly used in real-world scenarios. This allowed us to assess the effectiveness of our heuristics in identifying security issues in practical settings.

Procedure. To establish heuristic criteria for Docker image vulnerability scanners (DIVSes) evaluation, we performed heuristic assessments with four evaluators to gauge three DIVSes (Anchore Engine, Trivy, and Snyk) usability in detecting and fixing vulnerabilities in the official Ubuntu image and `web-dvwa` image through our 13 heuristics. Our study aimed not to pinpoint specific usability issues in DIVSes, but to solicit feedback about each heuristic's significance and necessity, confirm the evaluators' understanding, and seek recommendations for new criteria and changes. The heuristic evaluation completion time varied between approximately 1.5 and 2.5 hours. All interviews were transcribed and recorded.

3.2 Results

In Study 1, we established a set of 11 heuristics (Table 1). We modified five heuristics from the previous version (Appendix B) to create two new heuristics: "Necessary & Sufficient Information about Vulnerabilities," and "Essential Information for Patching." We also added four new heuristics: "Structure and Format of Vulnerability Reports," "Searching / Filtering Vulnerabilities," "Missing Fields in Vulnerability Records," and "Displaying Patching Status." The remaining five heuristics were evaluated without any issues, and participants' responses were aligned with our intention, so we decided to keep them.

Figure 5 illustrates the evolution of the three versions of heuristics, namely *Smith et al.*'s heuristics, our first heuristics, and our second heuristics. When a heuristic is removed in the subsequent version, it is depicted within a dashed box. If a heuristic is updated or newly added compared to the previous version, it is shown in an orange or a blue box, respectively. For example, "Consistency between Vulnerability Records" in our initial heuristics is an update from "Notification Text" proposed in *Smith et al.*'s heuristic but is finally removed in our second heuristics. In the subsequent sections, we will describe our rationale for developing these heuristic criteria based on participants' feedback. Note that the heuristics are categorized into four themes.

3.2.1 Functionality and Understandability of Tools. All participants agreed that the "Confirming Expectations" and "Understanding & Interacting with Tools" heuristics are crucial for evaluating DIVSes usability. Users expect the tool to function as intended, and clear tool

Table 1: Our second heuristics.

Heuristic	Description
Confirming Expectations	Does the tool perform its function without unexpected or unintended outcomes?
Understanding & Interacting with Tools	Does the tool provide clear and user-friendly guidance on how to use its functionalities?
Structure and Format of Vulnerability Reports	Does the tool provide a vulnerability report that is properly formatted and structured?
Searching / Filtering Vulnerabilities	Does the tool provide searching or filtering functionality?
Vulnerability Severity & Rank	Does the tool clearly indicate the severity level of each vulnerability discovered, and prioritize them based on their severity levels?
Necessary & Sufficient Information about Vulnerabilities	Does the tool provide necessary and sufficient information to understand the risks associated with vulnerabilities and propose suitable countermeasures?
Resources & Documentation	Does the tool provide external references and documents that are necessary to gain a comprehensive understanding of those vulnerabilities?
Understanding Concepts	Does the tool provide an additional explanation about the unfamiliar concepts to make developers understandable?
Missing Fields in Vulnerability Records	There is no missing field in the vulnerability records and if the information cannot be provided, does the tool explain the reasons for the missing field?
Essential Information for Patching	Does the tool provide practical information about patches?
Displaying Patching Status	Once a fix has been selected and/or applied, does the tool provide information about patch status?

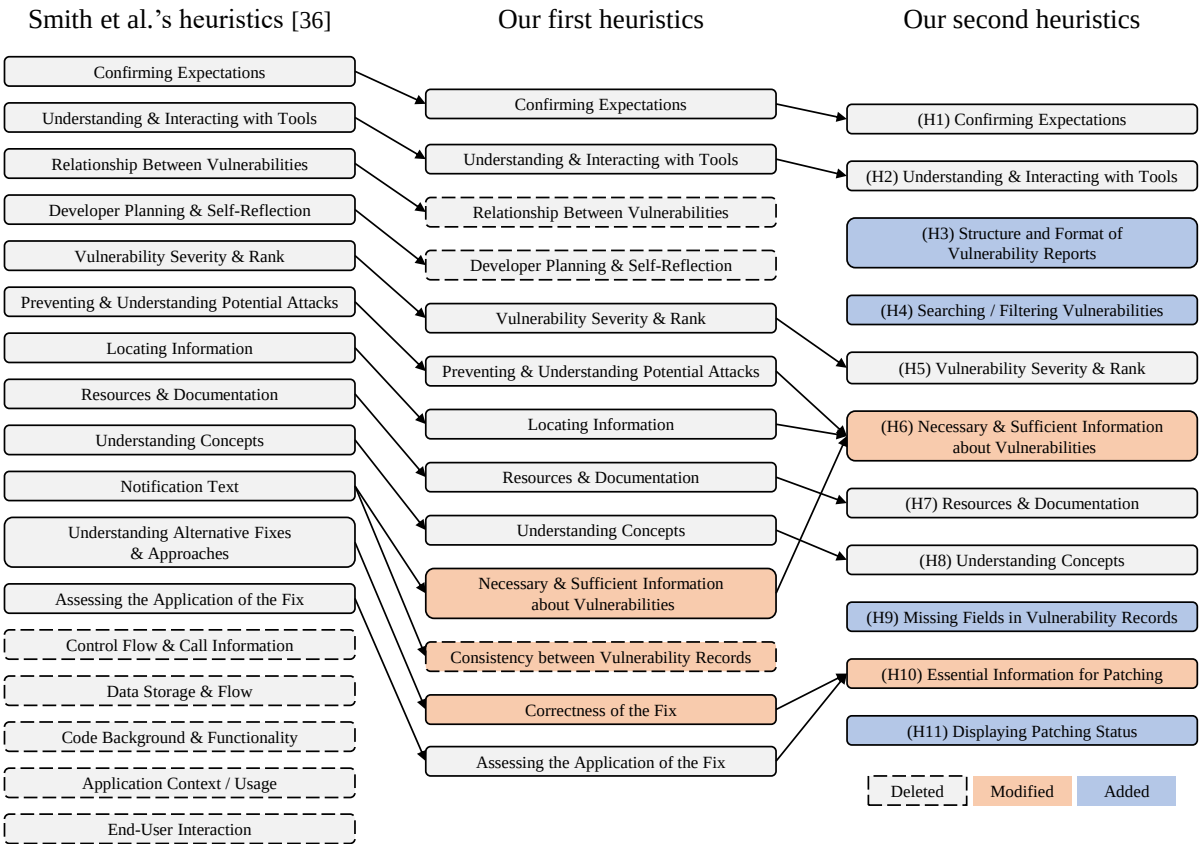


Figure 5: Relationships between the three versions of the heuristics. Heuristic(s) in dashed box are deleted, in orange box are modified, and in blue box are newly added.

usage guidance significantly reduces time and effort. For example, P1 noted that when using Anchore Engine, the tool does not provide any explanation of how to generate a report, except for the option to choose an inspection target (*i.e.*, operating system packages or language packages). This highlights the need for a heuristic offering proper guidance and tool usage information. Thus, to assess the usability of DIVSes, we consider the "Confirming Expectations" and "Understanding & Interacting with Tools" heuristics necessary. We have grouped these heuristics under the "Functionality and Understandability of Tools" category.

(H1) Confirming Expectations. This heuristic ensures that a tool performs its functions without unexpected or unintended outcomes.

(H2) Understanding & Interacting with Tools. This heuristic ensures that a tool provides clear and user-friendly guidance on how to use its functions. It is crucial that developers can leverage the tool's functions without relying on external resources or conducting additional web searches.

3.2.2 UI and UX of Vulnerability Reports. We need to establish heuristics for evaluating vulnerability reports produced by DIVSes since these reports are the final results that demonstrate the discovered vulnerabilities that need to be fixed.

During the study, participants P3 and P4 emphasized the need to evaluate the structure and format of vulnerability reports generated by DIVSes. They observed that in some cases, the formats of vulnerability reports from Anchore Engine and Trivy were not properly aligned on a CLI, making them difficult to understand even if the content was accurate. To address this issue, we introduced a new heuristic called "Structure and Format of Vulnerability Report" to assess how well DIVSes present vulnerability information to users.

P1 suggested a heuristic to evaluate a DIVS's capability to search for specific vulnerability information with a keyword or identifier. Given that a Docker image typically contains dozens of vulnerabilities[1], P1 argued that the lack of search and filtering features makes it hard to find specific vulnerability information when using a DIVS. Therefore, we added a new heuristic called "Searching/Filtering Vulnerabilities" to evaluate this aspect of DIVSes.

All participants agreed that "Developer Planning & Self-Reflection" is unnecessary or unimportant for evaluating DIVSes. However, they agreed that prioritizing the reported vulnerabilities is necessary to identify the ones that must be fixed with high priority. Therefore, we kept the "Vulnerability Severity & Rank" heuristic.

Finally, we excluded the existing "Relationship between Vulnerabilities" heuristic. P1 argued that the meaning of the relationship in this heuristic was too ambiguous depending on the context.

We have grouped these heuristics under the "UI and UX of Vulnerability Reports" category.

(H3) Structure and Format of Vulnerability Reports. This heuristic ensures that the vulnerability report generated by a tool is properly formatted and structured to effectively convey information about vulnerabilities.

(H4) Searching / Filtering Vulnerabilities. This heuristic evaluates a tool's search and filtering functionality in its vulnerability report. The search and filtering features are essential for developers to efficiently find specific vulnerability information, especially when navigating reports with numerous vulnerabilities.

(H5) Vulnerability Severity & Rank. This heuristic ensures that a tool clearly indicates the severity level of each vulnerability discovered, and prioritizes them based on their severity levels. By doing so, the tool can help developers quickly identify the most critical vulnerabilities that require immediate attention and fixing.

3.2.3 Understandability of Vulnerability Information. DIVSes should not only provide accurate vulnerability information but also offer developers the necessary details to understand associated risks and propose appropriate countermeasures. However, the required vulnerability information may vary depending on the individual. For example, participants P1, P3, and P4 highlighted the importance of knowing the exact locations of vulnerabilities to fix them, while P2 and P3 emphasized that assessing each vulnerability's risk requires information on its severity level, vulnerability type, and potential attack scenarios. Additionally, P3 mentioned that the ease of fixing a vulnerability could also be relevant in determining its severity level. Based on these findings, we consolidated the existing heuristics of "Preventing & Understanding Potential Attacks" and "Locating Information" into the "Necessary & Sufficient Information about Vulnerabilities" heuristic, recognizing that different types of information may be required depending on a developer's experience level and security knowledge.

All participants agreed that the existing heuristics of "Resources & Documentation" and "Understanding Concepts" were highly relevant for evaluating the usability of DIVSes. External references and detailed documents are necessary for developers to gain in-depth knowledge of vulnerabilities, and explaining unfamiliar concepts helps improve tool usability.

Regarding the "Consistency between Vulnerability Records" heuristic, P2 found it challenging to check for consistency and questioned its importance. Therefore, we removed this heuristic and added a new one called "Missing Fields in Vulnerability Records," based on feedback on the need to check for missing data.

We have grouped these heuristics under the "Understandability of Vulnerability Information" category.

(H6) Necessary & Sufficient Information about Vulnerabilities. This heuristic ensures that a tool provides developers with the critical information they need to understand the risks of vulnerabilities and suggest effective mitigation strategies.

(H7) Resources & Documentation. This heuristic ensures that a tool provides external references and detailed documents necessary for developers to gain in-depth knowledge of vulnerabilities. By including such references and documentation, developers can explore the root causes of the vulnerabilities and obtain a more thorough knowledge of their potential impact and severity.

(H8) Understanding Concepts. This heuristic ensures that a tool provides clear and concise explanations for unfamiliar concepts that developers may encounter in the vulnerability report or tool interface. By providing additional explanations, developers can quickly grasp the concept and continue to use the tool effectively.

(H9) Missing Fields in Vulnerability Records. This heuristic ensures that vulnerability records produced by a tool are complete

[1]A previous study [34] showed that a typical Docker image contains over 180 vulnerabilities.

and contain all the necessary information. If any required information is missing, the tool should provide a clear explanation to developers as to why that field is not available.

3.2.4 Easiness of Vulnerability Patching. During the evaluation, we received feedback from participants regarding the "Correctness of the Fix" and "Assessing the Application of the Fix" heuristics. All participants expressed confusion about the difference between the two heuristics. In response to this feedback, we consolidated them into a new heuristic called "Essential Information for Patching," which assesses whether DIVSes provide the necessary information to effectively fix identified vulnerabilities.

Another issue raised by P3 was the challenge of determining whether a patch has been actually applied when using DIVSes. To evaluate this aspect of DIVSes, we added a new heuristic called "Displaying Patching Status," which assesses whether DIVSes shows the status of the patch, distinguishable from the status before patching.

We grouped these heuristics under the "Easiness of Vulnerability Patching" category.

(H10) Essential Information for Patching. This heuristic ensures that a tool provides developers with the necessary information to patch the vulnerabilities detected in the analyzed Docker image.

(H11) Displaying Patching Status. This heuristic ensures that a tool properly displays the status of patching, making it easy for users to differentiate between the status before and after patching.

4 STUDY 2: EVALUATING THE DIVS TOOLS

To address our second research question (**RQ2**), we conducted Study 2 to evaluate the usability of commonly used DIVSes using the 11 heuristic criteria developed in Study 1 (Table 1). Section 4.1 describes the study methodology used for the heuristic evaluation. Section 4.2 presents our evaluation results.

4.1 Methodology

Our objective for conducting the heuristic evaluation was to identify potential usability issues in DIVSes using the heuristics outlined in Section 3. To accomplish this goal, we recruited computer security and HCI experts and asked them to conduct heuristic evaluations of three different DIVSes (Grype, Trivy, and Snyk). We used the two Docker images used in Study 1 (Section 3.1): an official Ubuntu image and a `web-dvwa` image created for educational purposes in cybersecurity. As Anchore Engine is no longer supported, we replaced it with Grype [10], another widely-used tool with a pull count of 1,506,117 on Docker Hub.

Evaluator Recruitment. We recruited a group of computer security and HCI experts who were experienced in heuristic evaluation or user studies. Table 2 provides detailed demographic information about the participants.

During the screening process, we asked all participants to complete a survey about their demographics, including their general computer skills, computer security knowledge, UI/UX design experience, heuristic evaluation experience, and experience with Docker. Based on their responses, we selected six evaluators (E1–E6) who demonstrated strong expertise in these areas. Four of the selected evaluators had prior experience with heuristic evaluations or had taken an HCI course. We compensated each participant with $100 for their time. To ensure that the evaluators were familiar with the materials and tools, we provided each expert with a short tutorial video on Docker and heuristic evaluation. In addition, we sent all necessary materials, including the heuristic evaluation form, to the participants in advance of the experiments.

Table 2: Demographic information of the evaluators in Study 2, including age range, education, and HCI/Security experience. The years of experience in HCI/Security are represented as Exp. Education denotes the evaluators' expertise level in Docker (Doc.), computer security (Sec.), and human-computer interaction (HCI). Docker experience is indicated by '++' for expert-level understanding and '+' for some experience. Expertise levels in computer security and HCI are indicated by '++' for expert level, '+' for some experience through seminar participation or self-study, and ' ' for no expertise.

			Experience		Education		
ID	Age	Gender	Major	Exp.	Doc.	Sec.	HCI
E1	24-35	Male	Sec.	6	+	++	+
E2	24-35	Female	HCI	5			++
E3	35-44	Male	Sec.	10	+	++	++
E4	35-44	Male	Sec.	8.6	+	++	
E5	24-35	Female	HCI	6	+		++
E6	35-44	Male	HCI &Sec.	10	++	++	++

Procedure. To simulate the experience of real-world users of DIVSes, we asked evaluators to complete four tasks: i) Finding the command or button to generate a vulnerability report, ii) Identifying the most severe vulnerability in the report and assessing its level of danger, iii) Explaining how to fix the vulnerability and fixing the vulnerability, and iv) Confirming that the vulnerability was fixed. During the tasks, evaluators were instructed to take note of any usability issues and rate each issue on a scale of 0 to 4 based on our 11 heuristics. We adopted the scoring method proposed by Nielsen [28], where 0 indicated no usability issue, 1 indicated a minor issue that requires no modification, 2 indicated a minor issue that can be modified with low priority, 3 indicated a usability issue that should be modified with high priority, and 4 indicated a serious usability issue that must be addressed immediately. To control for the learning effect of repeated testing across three tools, we employed a balanced *Latin square* [26] to determine the order in which each tool was used. Each evaluator spent approximately four hours completing the tasks for all DIVSes. All interviews were recorded and transcribed.

4.2 Results

In our heuristic evaluation of three DIVSes, Grype (G), Trivy (T), and Snyk (S), we identified 239 usability issues. We excluded minor concerns like output file format and font size to concentrate on important usability issues. The evaluation scores and distinct issue counts from six evaluators for each tool were grouped by theme and heuristic, as presented in Table 3. This evaluation revealed several usability problems in these DIVSes, potentially hindering developers' ability to detect and fix vulnerabilities in Docker images. We provide practical recommendations based on our findings.

The following subsections contain detailed descriptions of the identified usability issues and corresponding recommendations.

4.2.1 (H1) Confirming Expectations. Overall, Trivy performed as expected, and all evaluators agreed that it fulfilled its intended functions. However, Grype received mixed feedback, with four

Table 3: Usability issues and evaluation scores grouped by theme for three DIVSes: Grype (G), Trivy (T), and Snyk (S). Evaluation scores reflect the severity of usability issues, with higher scores indicating greater severity. The number in parentheses following the evaluation score represents the number of distinct usability issues identified for each tool.

Theme	Heuristics	G	T	S
Functionality and Understandability of Tools	(H1) Confirming Expectations	7 (1)	0 (0)	11 (3)
	(H2) Understanding & Interacting with Tools	11 (2)	9 (1)	10 (2)
UI and UX of Vulnerability Reports	(H3) Structure and Format of Vulnerability Reports	19 (3)	9 (1)	10 (2)
	(H4) Searching / Filtering Vulnerabilities	17 (2)	7 (2)	4 (1)
	(H5) Vulnerability Severity & Rank	19 (4)	16 (3)	7 (2)
Understandability of Vulnerability Information	(H6) Necessary & Sufficient Information about Vulnerabilities	18 (3)	10 (2)	7 (1)
	(H7) Resources & Documentation	19 (2)	10 (3)	7 (1)
	(H8) Understanding Concepts	12 (1)	10 (2)	12 (2)
	(H9) Missing Fields in Vulnerability Records	13 (1)	14 (1)	2 (1)
Easiness of Vulnerability Patching	(H10) Essential Information for Patching	21 (3)	21 (2)	13 (4)
	(H11) Displaying Patching Status	16 (2)	13 (1)	15 (2)

evaluators expressing satisfaction and two others (E1 and E3) noting discrepancies with the tool's inadequate explanations for its CLI flags (*i.e.*, options). Specifically, the "only-fixed" flag did not perform as described. Despite using this flag, Grype produced a vulnerability report that was identical to the report generated without the "only-fixed" flag, which is inconsistent with our expectations when using the "only-fixed" flag.

During the evaluation of Snyk, three usability issues were identified. Evaluators E1, E3, and E6 brought up the first two, while E1, E2, and E4 noted the third. Firstly, the loading time for vulnerability reports was problematic, especially for Docker images with many vulnerabilities. Snyk displayed all vulnerabilities on one page, requiring 2–3 seconds to scroll or refresh. Secondly, evaluators expected that links within the report would open a new tab or window, but they loaded in the same tab, against their expectations. This forced a disruption as they had to reload the original page to continue tasks. Thirdly, some buttons had misleading functions. For example, the "exploit maturity" button seemed to offer more exploit information, but it only showed the presence of corresponding exploits, potentially causing confusion for Snyk users.

Recommendation. Unit tests are crucial for ensuring that DIVSes perform as intended during functional testing. However, the three DIVSes (Grype, Trivy, and Snyk) analyzed are open-source projects that may be insufficiently tested, as is often the case with open-source software [44]. To address this issue, developers should be encouraged to promptly report any unexpected behavior or bugs they encounter when using the DIVSes. We recommend that each DIVS provides a bug-reporting feature to facilitate this process. Unfortunately, Grype and Trivy do not currently support this feature, so developers must report any discovered bugs via GitHub issues.

Our recommendation for Snyk is to improve the user experience by splitting the vulnerability results into multiple pages, displaying only a reasonable number of vulnerabilities on each page. This will reduce the web page loading time and provide a better browsing experience for developers. Additionally, we suggest implementing DIVSes that enable the option to open links in a new tab or pop-up window, allowing developers to seamlessly continue working on the vulnerability report without any interruption.

4.2.2 (H2) Understanding & Interacting with Tools. CLI tools (*i.e.*, Grype and Trivy) provide instructions on usage via the "help" flag (-h). In contrast, Snyk, which has a web user interface, offers a help page to guide users on how to use the tool.

While all evaluators except E6 found it easy to generate a default vulnerability report using CLI tools, they encountered difficulties using additional features due to the lack of descriptions. Specifically, when using Grype's CLI flags, they found the descriptions unclear and misleading. For example, the "fail-on string" flag was mistakenly perceived as a filtering function due to its description: "`set the return code to 1 if a vulnerability is found with a severity >= the given severity, options=[negligible low medium high critical]`." Although Grype provided an example of using this flag, E4 had trouble locating the example as it was not highlighted.

Trivy provided better flag explanations and examples than Grype, but E1, E4, and E5 found the instructions on flag usage insufficient. Although flags were divided into global and normal flags in Trivy, there were no usage instructions for global flags, and even the usage instructions for normal flags were not detailed enough.

Snyk's vulnerability report comprises two sections: the main project page and the view report page. The main project page provides detailed information about the identified vulnerabilities, whereas the view report page presents a summary of the vulnerabilities, including their statistical information and a brief description of each vulnerability. However, during our evaluation, E4, E5, and E6 raised a usability issue concerning this layout of Snyk's vulnerability report. While they found the view report page helpful for understanding the overall picture of the identified vulnerabilities, they noted that locating the "view report" button for loading the view report page was difficult. Additionally, Snyk does not explain the presence of the view report feature, which may cause new users to overlook it.

Recommendation. To improve the user experience of CLI DIVSes (*i.e.*, Grype and Trivy), it is crucial to provide a comprehensive manual document that includes clear descriptions of their functions and detailed examples of how to use them. Conducting further user studies can help identify and correct unclear and ambiguous expressions in the current documentation. In addition, it is essential to emphasize important descriptions and examples using appropriate indentation or color to make the documentation more accessible. A clear and logical documentation organization can help users locate the information they need quickly and easily.

To improve the user experience of DIVSes with web interfaces (*i.e.*, Snyk), it is essential to use visual cues like coloring, increasing the size, or adjusting the position up to the left to highlight important descriptions or UI components like buttons or tabs. Pop-up windows can be used to provide examples or instructions, which can make the features more intuitive to use. Additionally, it is crucial to ensure that multiple web pages are well-organized and easy to navigate, as this can significantly enhance the user experience.

4.2.3 (H3) Structure and Format of Vulnerability Reports. All evaluators identified a common usability issue in all DIVSes: displaying all vulnerabilities on a single page, which leads to poor readability and makes it challenging to obtain information on each vulnerability, especially when there are many vulnerabilities in the Docker image analyzed. This issue also affects the loading time for Snyk, as discussed in Section 4.2.1.

Unlike the other tools, Grype presents the identified vulnerabilities in a list format (Figure 2). However, this simple format caused E3, E4, E5, and E6 to mention that it is not easy to distinguish each vulnerability's information from the others. E3, in fact, misread the severity level of a vulnerability as "Negligible" instead of "High" because there was no clear distinction between the preceding and following vulnerability descriptions. Additionally, Grype does not provide an overall summary of vulnerabilities in an image, making it challenging to grasp the overall status of the vulnerabilities.

Snyk's inconsistent presentation of vulnerability information can lead to usability issues. For example, the project page prioritizes displaying the CWE link, while the view report page features the CVE link first. This lack of uniformity in the report format can make it difficult for users to find the information they need.

Recommendation. To enhance the readability of vulnerability reports, we recommend that DIVSes consider breaking down information about identified vulnerabilities into multiple pages, with each page containing a manageable number of vulnerabilities, as discussed in Section 4.2.1. Additionally, a layered interface can be adopted to address the information overload issue. Previous studies have found that a layered interface is effective for security and privacy notifications [16, 19]. The interface could initially present users with information on the type and severity level of each vulnerability, and then offer more in-depth details, such as the affected software module, patch information, and reference links. Through this layered interface, developers can effortlessly bypass unnecessary information and selectively access more comprehensive data about vulnerabilities and patches.

CLI tools can implement a layered interface using CLI flags, or a user-interactive multi-window console user interface (CUI), such as the *gocui* [9] module in Go. To improve the report's readability, CLI DIVSes can present brief information about each vulnerability by default and provide additional details about each vulnerability when using a flag. For DIVSes with web interfaces, advanced user interfaces, such as tree views, can be utilized to display the vulnerability report hierarchically.

Lastly, we suggest incorporating a summary of all identified vulnerabilities, similar to the feature offered by Snyk. For instance, providing the number of vulnerabilities corresponding to each severity level would help users assess the overall security of a Docker image. This feature can help users quickly identify and prioritize the most critical vulnerabilities.

4.2.4 (H4) Searching / Filtering Vulnerabilities. All evaluators agreed that Grype's lack of filtering feature makes it challenging for developers to obtain necessary information, especially when dealing with a large number of vulnerabilities in a Docker image. E4 expressed dissatisfaction with Grype and suggested that the tool should export its results to an Excel file to make it easier to filter vulnerability information using Excel. In contrast, Snyk's filtering feature received positive feedback from the evaluators. Snyk allows developers to flexibly filter vulnerabilities using various attributes such as severity level, priority score, and exploit maturity. The evaluators, including E2, E3, E4, and E6, noted that Trivy supports filtering features based only on severity level or the presence of a patch. While all evaluators agreed that filtering by severity level is crucial, E2 and E3 pointed out that filtering by CVE ID is also necessary. Additionally, E4 and E6 highlighted the importance of filtering by the presence of a patch.

Regarding search features, all evaluators noted that both Grype and Trivy lack a search feature, while Snyk offers a satisfactory one, except for E4's concern that incomplete keywords should also be used for searching. For example, E4 suggested searching for vulnerabilities with the keyword "0286" to find the "CVE-2022-0286" vulnerability. However, Snyk currently does not provide search results using incomplete keywords, which E4 found limiting.

Recommendation. We recommend that search and filtering functionalities be included as default features in DIVSes, along with clear usage instructions to ensure ease of use. It is worth noting that implementing these features on the command line interface of CLI DIVSes may not be intuitive for some users. Therefore, we suggest that vulnerability reports be exportable to a file format that can be processed by popular spreadsheet programs such as Excel or OpenOffice. This would provide users with a more user-friendly way to search and filter vulnerabilities in DIVSes.

4.2.5 (H5) Vulnerability Severity & Rank. Although Grype and Trivy provide severity levels for discovered vulnerabilities, they do not offer corresponding priority scores, making it difficult for users to prioritize vulnerabilities with the same severity level. In contrast, Snyk's priority score feature received positive feedback from E2, E4, and E6 as it enables users to rank vulnerabilities easily based on their priority scores. However, E5 noted that some results showed inconsistencies between the severity level and priority score, which could lead to confusion in determining the severity of vulnerabilities. For example, in the Ubuntu image, the "CVE-2022-4450" vulnerability had a severity level of "Medium" and a priority score of 467, while a buffer overflow vulnerability in the same image had a severity level of "High" but a priority score of 411, as illustrated in Figure 6. This inconsistency could be confusing when determining which vulnerability is more critical and should be patched first. Additionally, E3 and E5 complained about the lack of sufficient explanation about the exact formula for calculating the priority score, which makes it difficult to understand this inconsistency. While Snyk shows some evidence for the computed priority score, such as the maturity of exploit, presence of available fix, and severity

level, as shown in Figure 7, E3 and E5 pointed out that they did not fully understand how these features contribute to the final score.

Figure 6: Inconsistency in severity level and priority score in Snyk. The "CVE-2023-0286" vulnerability has a higher severity level but a lower priority score than the "CVE-2022-4450" vulnerability.

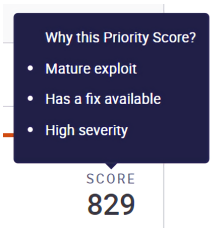


Figure 7: Evidence for the priority score of a vulnerability in Snyk.

All evaluators, except E1, expressed dissatisfaction with Grype lacking a severity-level sorting feature. E3 and E6 insisted this feature was vital for DIVSes. Despite Trivy also missing this feature, E1, E2, E4, and E5 found it acceptable due to default sorting by affected software module name and severity within each module. Conversely, all participants appreciated Snyk's sorting feature, allowing vulnerability sorting by severity level and priority score.

Another major concern with Grype was its missing information on the full range of severity levels, which all evaluators except E5 found unsatisfactory. Analyzing an Ubuntu image with a few vulnerabilities, E3, E4, and E6 observed that Grype's report could mislead developers into thinking "High" was the maximum severity level. Moreover, E4 noted that even when analyzing an image with vulnerabilities spanning all severity levels, the unsorted vulnerability results in the report made it difficult for developers to determine the full range of vulnerabilities.

Recommendation. We recommend that DIVSes provide both the severity level and priority score for each vulnerability, along with sufficient evidence of how the level and score are determined. The ability to sort the discovered vulnerabilities by severity level would also be a useful feature, which is currently only supported by Snyk. Furthermore, default sorting of vulnerabilities by severity level, from highest to lowest, would also be helpful in identifying the most critical and important vulnerabilities that need to be addressed first. Finally, the vulnerability report should specify the range of severity levels and priority scores at the beginning to help users better understand the findings.

4.2.6 (H6) Necessary & Sufficient Information about Vulnerabilities. Grype presents the CVE ID and severity level for each identified vulnerability but does not provide further details such as vulnerability type, security impact, or evidence for the severity level. E1 and E4 reported difficulty understanding the significance of vulnerabilities without additional information. E1 stated that searching for more details on each vulnerability on the internet using the CVE ID would be time-consuming when an image has many vulnerabilities. In contrast, Trivy provides more comprehensive information, including the names of the libraries containing the vulnerability, its CVE ID, and its description. Evaluators E2, E3, E4, and E5 found Trivy helpful in understanding the security risks and identifying critical vulnerabilities that require immediate attention. However, E1 and E3 mentioned that Trivy lacks explanations on the exploits related to each vulnerability and its security impact, which may be challenging for non-security experts. Snyk provides the most detailed vulnerability information, including the vulnerability type, the image layer where it resides, and external reference links. Snyk also categorizes the maturity of exploits into three categories: "Mature," "Proof of Concept," and "No known exploit." However, E4 found that details on known exploit concepts or code examples were not sufficiently provided.

Recommendation. We recommend that DIVSes provide a detailed description of each vulnerability, including its type information and identifier (*e.g.*, CVE ID), to reduce the burden on developers searching for information online. Providing more information on each vulnerability would help identify critical vulnerabilities that require immediate attention. Note that providing the source of vulnerability data (*e.g.*, Debian, NVD) and justification for each vulnerability's severity level would improve the tool's credibility.

4.2.7 (H7) Resources & Documentation. All evaluators agreed that Grype's vulnerability report missed external reference links, possibly leading developers to search for vulnerability information online and obtain incorrect details from wrong search results. Conversely, Trivy provides links to well-structured web pages on the Aqua Security vulnerability database for each vulnerability, as shown in Figure 3. However, E2, E4, and E5 raised concerns about Trivy's external references, as certain web pages lacked essential vulnerability details (*e.g.*, how to fix a specific vulnerability). Snyk's web page offers the NVD description of a vulnerability and links to related CVE, CWE, and CVSS web pages, as shown in Figure 4. While E1, E2, E3, and E4 were satisfied with links to various vulnerability databases as they provided complementary information, E5 and E6 suggested that considering multiple databases might be superfluous due to substantial overlap and redundancy.

Recommendation. A vulnerability report generated by a DIVS should contain external reference links to provide accurate and reliable information about vulnerabilities. It is recommended to include links to reputable databases managed by the DIVS vendor itself. A careful review of external references is necessary to avoid redundancy and overlapping information across multiple databases. Moreover, it is important to verify that external web pages contain essential information, such as how to fix vulnerabilities.

To improve usability, it is essential to label external reference links clearly using keyword tags. Providing a standard format for external reference links in vulnerability reports can prevent confusion and ensure that all external links provide essential information

about vulnerabilities. This will facilitate developers' access to the necessary information to address vulnerabilities effectively.

4.2.8 (H8) Understanding Concepts. The vulnerability reports generated by DIVSes contained some terminologies that were difficult to understand for evaluators who lacked experience in computer security, such as E2 and E5. For instance, they were unsure about the meaning of "Exploit maturity" in Synk's vulnerability reports. E4 and E5 found the "NAME" and "INSTALLED" fields in Grype's vulnerability report to be unclear. The term "fixed" in Trivy's vulnerability report and Snyk's "fixed" filtering option was found to be ambiguous by E2, E5, and E6. They cautioned that developers might not understand what was fixed since no explanation was provided. For example, E5 mistakenly believed that the "fixed" filter would only show the vulnerabilities that she had fixed rather than displaying all patched versions that address the vulnerabilities.

Another usability issue with Trivy was identified by E1 and E5, which was the abbreviation of vulnerability types (*e.g.*, SSRF) without providing their full names. This could confuse developers who are not familiar with security terminology, as the full names are only available on external web pages.

Recommendation. To improve the readability and comprehension of vulnerability reports, it is important to use consistent and descriptive labels for fields. Rather than using ambiguous terms like "fixed," it is advisable to use labels such as "patched versions" to indicate the software versions that address the vulnerability. Providing clear, concise descriptions that clarify the meanings of terms and fields is also essential. For example, a succinct explanation of the contents of the "fixed" filter for Snyk will enable users to better understand the information presented.

To make vulnerability reports more accessible and easily understood, it is recommended to use plain language when describing technical concepts, especially for users without computer security experience. Additionally, it is helpful to include the full name of technical terms such as "Server-Side Request Forgery (SSRF)" when first abbreviated in the report to avoid confusion.

4.2.9 (H9) Missing Fields in Vulnerability Records. All evaluators found that the field representing the patched software version in all three DIVSes (Grype's "`FIXED-IN`," Trivy's "`Fixed Version`," and Snyk's "`Fixed in`") can contain missing values. Moreover, these DIVSes did not provide sufficient explanations for the missing values, leaving developers to speculate about the reasons for the missing information. While Grype's vulnerability reports use the special string value "`won't fix`" to indicate that a software vendor has no plan to fix the vulnerability in a particular release version, as shown in Figure 8, this reason alone is insufficient. There may be many other reasons for missing values besides "`won't fix`."

E1 was the only evaluator to mention the empty value in "`Fixed in`" as a usability issue for Snyk. This is because Snyk's user documentation [37] explains the meaning of empty values in "`Fixed in`," which indicates that no patched version is available. To enhance the usability of vulnerability reports, it is recommended to provide clear reasons for missing field values in vulnerability reports.

Recommendation. To address usability issues related to missing field values in vulnerability reports, it is recommended to provide clear and consistent explanations for missing values. This can help

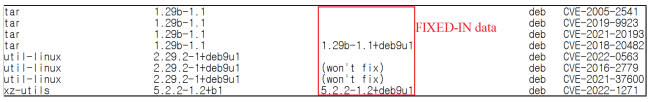


Figure 8: Grype's `FIXED-IN` column includes three types of values: (i) the software version where the vulnerability has been resolved, (ii) the term "`won't fix`" indicating that the vendor has no plan to fix the vulnerability in a particular release version, and (iii) a blank space indicating a missing value.

developers understand the reasons for the missing information and avoid making incorrect assumptions. Additionally, it can be helpful to provide links to external resources, such as vendor websites or security databases, for the field representing the patched software version so that users can directly check the patch status of each vulnerability if needed.

4.2.10 (H10) Essential Information for Patching. Four primary usability issues were identified across the evaluated DIVSes.

First, all evaluators reported that the DIVSes lacked the necessary information for patching Docker images, such as software update commands or software package types (*e.g.*, Ubuntu, Python, and Node). They only displayed higher software versions that resolved the identified vulnerability. Developers need to execute commands in the *Dockerfile*–a text file containing instructions for building a Docker image–to update the vulnerable image to a patched version. E1 and E5 noted that Snyk demonstrated not only the image layer containing vulnerable programs but also the Dockerfile commands (*e.g.*, "`apt-get install -y apache2` ") used to install them, enabling the deduction of required update commands.

Second, Grype and Trivy did not provide detailed update instructions for vulnerable software versions. They merely listed software versions that resolved the identified vulnerability. Consequently, E3, E4, and E5 faced difficulties updating vulnerable software versions when only a higher version was available instead of the recommended version in the repository. In contrast, E2 and E5 mentioned that Snyk offered detailed patch instructions in the form of "update [software name] to [version] or higher," instilling confidence when attempting updates with only higher versions available.

Third, Grype and Snyk displayed patch information for each vulnerability separately, not grouped by affected software modules. This resulted in lengthy reports and challenges in identifying software versions that address multiple vulnerabilities simultaneously. In contrast, Trivy consolidates patch information when a vulnerability impacts multiple software modules, simplifying the determination of appropriate patched versions to address affected modules simultaneously.

Lastly, E1, E2, E4, and E5 observed that Snyk occasionally displayed vulnerability records for only one software module among several dependent modules without any explanation when the affected software modules had dependencies. For instance, when a vulnerability impacted multiple dependent software packages, such as `php7.0` , `php7.0-xml` , and `php7.0-pgsql` , Snyk presented the vulnerability record for just one sub-package (*e.g.*, `php7.0-xml`), frustrating developers from identifying other vulnerable packages.

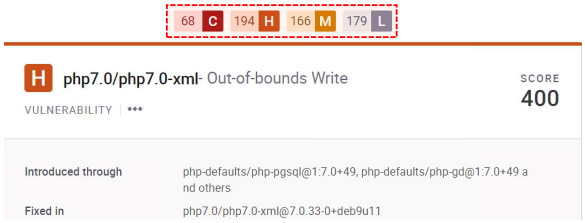
Recommendation. To help developers effectively patch vulnerable software versions in Docker images, DIVSes should provide

comprehensive information for updating vulnerable software to patched ones, including patch commands and package types. We recommend offering instructions that detail the steps for updating software. These instructions should include practical command examples (*e.g.*, "`apt-get install -y apache2=2.4.25` ") that can be easily adapted and used with the RUN instruction in a Dockerfile.

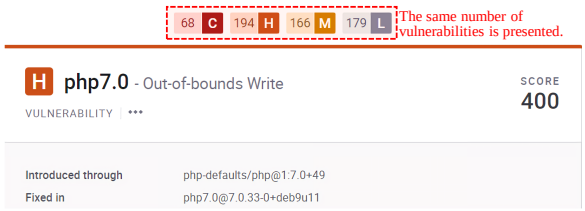
To enhance the readability of vulnerability reports, we suggest organizing patch information by affected software modules rather than displaying patch details for each vulnerability individually. This approach streamlines the identification of software versions that address multiple vulnerabilities simultaneously. Specifically, DIVSes should clearly present vulnerabilities affecting hierarchically related software packages. By displaying vulnerability records grouped by all relevant sub-packages (*e.g.*, `php7.0` , `php7.0-xml` , and `php7.0-pgsql`), developers can recognize all vulnerable packages at once and take appropriate actions to address them.

4.2.11 (H11) Displaying Patching Status. All evaluators mentioned that none of the evaluated DIVSes featured a method to check whether a vulnerable software version was properly updated. Consequently, after patching a Docker image, they had to scan it again and manually compare the vulnerability reports.

To simplify this verification process, E6 suggested monitoring the changes in the number of vulnerabilities before and after applying a patch. If the number of vulnerabilities decreases in the updated image, it can be assumed that the patch was effectively applied. This task would be relatively easy with Trivy's and Snyk's vulnerability reports, as the number of vulnerabilities is presented at the top of the report. However, E6 pointed out that Grype's report did not display these numbers, making it difficult to use this strategy to verify the patching status.



(a) Report for `php7.0-xml` before updating `php7.0-xml` .



(b) Report for `php7.0` after updating `php7.0-xml` .

Figure 9: Vulnerability scanning results generated by Snyk before and after updating `php7.0-xml` to address `CVE-2019-18218`.

As discussed in Section 4.2.10, Snyk can show identified vulnerability records for only one software module among several dependent modules when the affected modules had dependencies. E2, E4, and E5 warned that this feature could also complicate confirming the patching status. For instance, before the evaluators patched the "CVE-2019-18218" vulnerability in the `web-dvwa` image, Snyk showed that `php7.0-xml` had the vulnerabilities, as illustrated in Figure 9 (a). When the evaluators successfully patched the vulnerabilities in `php7.0-xml` , Snyk no longer displayed the vulnerabilities for `php7.0-xml` . However, it still showed the same number of vulnerabilities for `php7.0` , as seen in Figure 9 (b), which was not displayed before applying the patch. This interface causes confusion for developers trying to verify whether a patch was successfully applied.

Recommendation. To streamline the patching status verification process, we recommend that DIVSes incorporate a feature allowing developers to easily check if a vulnerable software version has been successfully updated. This feature could compare vulnerability reports before and after patching, highlighting the differences and eliminating the need for manual comparisons. Additionally, we suggest that DIVSes include a patch status indicator for each identified vulnerability across all previous versions, indicating whether a patch has been applied or not. For example, patched and unpatched vulnerabilities could be differentiated using distinct colors. Utilizing such an indicator would enable developers to verify the patching status without having to rescan the Docker image.

As a practical interim solution, DIVSes should consistently display the number of vulnerabilities at the top of vulnerability reports, making it easier for developers to track changes in vulnerability counts before and after applying a patch.

5 DISCUSSION

5.1 Summary of Usability Issues in DIVSes

The evaluation results from Study 2 (Section 4) revealed that the usability of the evaluated DIVSes requires improvement.

Regarding functionality and understandability, Trivy generally satisfied evaluators, while Grype received mixed feedback due to ambiguous explanations about CLI flags. Both Grype and Trivy had unclear usage descriptions, causing difficulties for evaluators. Snyk's vulnerability report provided a summary of identified vulnerabilities but lacked an explanation for the feature.

Regarding UI and UX for vulnerability reports, all tools displayed vulnerabilities on one page, resulting in poor readability. Snyk offered satisfactory search and filtering features, whereas Grype and Trivy did not. Grype and Trivy also lacked priority scores, hindering prioritization. While Snyk's priority score feature was helpful, it had inconsistencies and lacked explanations, leading to confusion.

Regarding the understandability of vulnerability information, Grype's report lacked details and references, making it difficult to grasp the significance of the vulnerabilities. Trivy provided references; however, some were missing critical details such as security impact. Snyk offered the most comprehensive information and references using multiple databases, but evaluators noted overlaps and redundancies. All three tools had instances of missing values in the patched software version fields without sufficient explanations.

Finally, regarding the ease of vulnerability patching, the DIVSes lacked the necessary information (*e.g.*, update instructions), only displaying software versions that resolved vulnerabilities. Grype

and Snyk provided patch information individually for each vulnerability, complicating the identification of versions with shared vulnerabilities. Furthermore, Snyk only showed vulnerability records for one software module among several dependent ones without explanation when the affected modules were hierarchically dependent, hindering developers from identifying other vulnerable packages. Evaluators found that none of the evaluated DIVSes offered a feature to verify if a vulnerable software version was updated correctly.

To address the identified usability issues, we created a visual mockup for CLI DIVSes, presented in Appendix C, which is based on our recommendations outlined in Section 4.

5.2 Ethical Considerations

In our studies, we adhered to ethical expectations and privacy standards by limiting our questions to tool usage and heuristic evaluation, avoiding any personal or sensitive information. The collected information is presented in Table 2. To ensure confidentiality and anonymity, we used pseudonyms for demographic data (P1–P4 in Study 1 and E1–E6 in Study 2).

Consequently, we did not seek IRB approval for our studies, believing that our studies would not harm participants significantly or violate their rights.

5.3 Limitations

First, our studies may have limited generalizability, as we evaluated only three DIVSes using two Docker images in each study. Although we carefully selected the DIVSes from Docker Hub based on their high pull counts and frequent updates as indicators of popularity, scanning other Docker images might uncover additional usability issues that we did not encounter.

Second, we did not consider the efficiency of generating vulnerability reports, which could significantly affect DIVSes' usability. Snyk takes up to 9 hours to scan a Docker image and generate a vulnerability report, while CLI DIVSes (*i.e.*, Grype and Trivy) require only a few seconds to download the latest vulnerability database. As waiting up to 9 hours for user studies would be impractical, we generated Snyk's report in advance and reused it for user studies.

Third, not all of our recruited evaluators were proficient in using Docker. We aimed to recruit experts in security, software engineering, or HCI fields. As a result, some evaluators (*e.g.*, E2 in Study 2), with an HCI background, lacked prior Docker experience or experience with DIVS tools. To address this limitation, we provided a Docker tutorial video before the evaluation and offered guidance during the evaluation.

Fourth, a learning effect may arise during the study involving the three different DIVSes. To minimize this learning effect and ensure fairness, we used the Latin Square method [26] to assign participants to the order of the tools evaluated.

6 RELATED WORK

Evaluation on Docker Image Vulnerability Scanners. Docker simplifies the development, deployment, and execution of software applications. Previous studies examining vulnerabilities in images hosted on the official Docker Hub registry [34, 43] revealed that approximately 82% of certified images contain at least one high or critical severity vulnerability. To tackle this problem, various strategies have been proposed [13, 18], and Docker image scanners [1, 2, 5–8, 10] have been developed to analyze Docker images and identify vulnerabilities within them. The majority of these approaches rely heavily on vulnerability databases sourced from security bug repositories (*i.e.*, CVE, CWE, and CVSS), focusing on known vulnerabilities in packages and libraries installed in each image layer. However, a prior study [20] has demonstrated that the detection accuracy of Docker image vulnerability scanners is not high, with the highest accuracy reaching only 65.7% for Anchore Engine. Our study is unique in that it focuses on evaluating the usability of DIVSes rather than delving into their security aspects. To the best of our knowledge, our work is the first to explore the usability of Docker security tools.

Usability Evaluation on Static Analysis Tools. Several recent studies have evaluated the usability of security-oriented static analysis tools [22, 35, 38, 39]. *Johnson et al.* [22] conducted a user study with 20 developers to determine why software developers do not use static analysis tools for bug detection. Their findings indicated that false positives and developer overload are significant factors contributing to dissatisfaction, which ultimately leads to the abandonment of adopting these tools. *Smith et al.* [35] conducted a user study using 17 heuristics developed in [33] to evaluate the usability of four static analysis tools, identifying 334 usability issues. Our study employs 11 heuristics, derived and adapted from their work, to specifically focus on evaluating DIVSes. Both studies discovered common usability issues, such as the disorganized presentation of security vulnerabilities without considering their severity. However, while *Smith et al.*'s research [35] pinpointed problems related to code examples, our study specifically emphasized usability issues associated with patching. *Tahaei et al.* [38] conducted an online survey to assess the effectiveness of security notifications produced by static analysis tools based on their content. Efforts to enhance the usability of static analysis tools using statistical learning techniques [39] have also been explored.

Heuristic Evaluation. Heuristic evaluation [30] is a usability analysis methodology where evaluators assess an interface to determine if it satisfies predefined heuristic rules. This approach has been applied across various domains, including video games [23, 31], medical devices [12, 45], and code analysis tools [35, 42]. Numerous studies have validated the effectiveness of heuristic evaluation [21, 41], demonstrating its value as a usability analysis technique.

7 CONCLUSION

We conducted a two-phase evaluation involving ten computer science experts. In the first phase, we developed 11 DIVS-specific heuristics with four experts using three DIVSes (Anchore Engine, Trivy, and Snyk). In the second phase, we conducted a heuristic evaluation with six evaluators, identifying 239 usability issues in three DIVSes (Grype, Trivy, and Snyk). From these identified issues, we suggest actionable design recommendations to improve the usability of DIVSes.

The proposed heuristics can be applied to not only DIVSes but also other security analysis tools which scan for known security vulnerabilities in software. As part of our future work, we plan to investigate usability issues in open-source vulnerability scanners.

ACKNOWLEDGMENTS

The authors would thank the anonymous reviewers and study participants for their valuable feedback and insights. Hyoungshick Kim is the corresponding author. This work was supported by Institute for Information & communication Technology Planning & Evaluation (IITP) grant funded by the Korea government (NIST) (No.2018-0-00532, Development of High-Assurance (>=EAL6) Secure Microkernel, 50%), (No.2022-0-00995, Automated Reliable Source Code Generation from Natural Language Descriptions, 30%), (No.2019-0-01343, Regional Strategic Industry Convergence Security Core Talent Training Business, 10%), and (No.2022-0-00688, AI Platform to Fully Adapt and Reflect Privacy-Policy Changes, 10%).

REFERENCES

[1] 2021. *Dagda.* Retrieved Feburary 15, 2023 from https://github.com/eliasgranderubio/dagda.git
[2] 2022. *Anchore Engine.* Retrieved December 11, 2022 from https://github.com/anchore/anchore-engine.git
[3] 2022. *Docker.* Retrieved February 27, 2023 from https://www.docker.com/
[4] 2022. *Docker hub.* Retrieved December 10, 2022 from https://hub.docker.com/
[5] 2022. *Snyk.* Retrieved December 11, 2022 from https://snyk.io/
[6] 2022. *Trivy.* Retrieved December 11, 2022 from https://github.com/aquasecurity/trivy.git
[7] 2023. *Clair.* Retrieved February 14, 2023 from https://github.com/quay/clair
[8] 2023. *Docker Bench security.* Retrieved February 14, 2023 from https://github.com/docker/docker-bench-security
[9] 2023. *gocui.* Retrieved march 20, 2023 from https://github.com/jroimartin/gocui
[10] 2023. *Grype.* Retrieved Feburary 15, 2023 from https://github.com/anchore/grype.git
[11] 2023. *pyCUI.* Retrieved March 29, 2023 from https://github.com/jwlodek/py_cui
[12] Raniah N Aldekhyyel, Jwaher A Almulhem, and Samar Binkheder. 2021. Usability of telemedicine mobile applications during COVID-19 in Saudi Arabia: A heuristic evaluation of patient user interfaces. In *Healthcare*, Vol. 9. MDPI, 1574.
[13] Kelly Brady, Seung Moon, Tuan Nguyen, and Joel Coffman. 2020. Docker container security in cloud computing. In *IEEE Annual Computing and Communication Workshop and Conference (CCWC)*. 0975–0980. https://doi.org/10.1109/CCWC47524.2020.9031195
[14] Chess Brian and McGraw Gary. 2004. Static analysis for security. *IEEE Security & Privacy Magazine* 2, 6 (2004), 76–79. https://doi.org/10.1109/MSP.2004.111
[15] Lisa Nguyen Quang Do, James R. Wright, and Karim Ali. 2022. Why do software developers use static analysis tools? A user-centered study of developer needs and motivations. *IEEE Transactions on Software Engineering* 48, 3 (2022), 835–847. https://doi.org/10.1109/TSE.2020.3004525
[16] Pardis Emami-Naeini, Henry Dixon, Yuvraj Agarwal, and Lorrie Faith Cranor. 2019. Exploring how privacy and security factor into IoT device purchase behavior. In *CHI Conference on Human Factors in Computing Systems (CHI)*. 1–12.
[17] Sara Gholami, Hamzeh Khazaei, and Cor-Paul Bezemer. 2021. Should you upgrade official Docker Hub images in production environments?. In *IEEE/ACM International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE, 101–105.
[18] Delu Huang, Handong Cui, Shihao Wen, and Cheng Huang. 2019. Security analysis and threats detection techniques on Docker container. In *IEEE International Conference on Computer and Communications (ICCC)*. 1214–1220. https://doi.org/10.1109/ICCC47050.2019.9064441
[19] Yue Huang, Borke Obada-Obieh, and Konstantin Beznosov. 2022. Users' Perceptions of Chrome Compromised Credential Notification. In *Symposium on Usable Privacy and Security (SOUPS)*. 155–174.
[20] Omar Javed and Salman Toor. 2021. Understanding the quality of container security vulnerability detection tools. https://doi.org/10.48550/ARXIV.2101.03844
[21] Robin Jeffries, James R Miller, Cathleen Wharton, and Kathy Uyeda. 1991. User interface evaluation in the real world: a comparison of four techniques. In *SIGCHI Conference on Human Factors in Computing Systems (CHI)*. 119–124.
[22] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. 2013. Why don't software developers use static analysis tools to find bugs?. In *International Conference on Software Engineering (ICSE)*. 672–681. https://doi.org/10.1109/ICSE.2013.6606613
[23] Christina Koeffel, Wolfgang Hochleitner, Jakob Leitner, Michael Haller, Arjan Geven, and Manfred Tscheligi. 2010. Using heuristics to evaluate the overall user experience of video games and advanced interaction games. (2010). https://doi.org/10.1007/978-1-84882-963-3_13
[24] Peiyu Liu, Shouling Ji, Lirong Fu, Kangjie Lu, Xuhong Zhang, Wei-Han Lee, Tao Lu, Wenzhi Chen, and Raheem Beyah. 2020. Understanding the security risks of Docker Hub. In *European Symposium on Research in Computer Security (ESORICS)*. 257–276.
[25] Dirk Merkel. 2014. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal* 2014, 239 (2014).
[26] Douglas C Montgomery. 2017. *Design and analysis of experiments.* John wiley & sons.
[27] Marcus Nachtigall, Michael Schlichtig, and Eric Bodden. 2022. A large-scale study of usability criteria addressed by static analysis tools. In *ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. 532–543. https://doi.org/10.1145/3533767.3534374
[28] Jakob Nielsen. 1994. *Severity Ratings for Usability Problems.* Retrieved Feburary 15, 2023 from https://www.nngroup.com/articles/how-to-rate-the-severity-of-usability-problems/
[29] Jakob Nielsen. 2005. Ten usability heuristics.
[30] Jakob Nielsen and Rolf Molich. 1990. Heuristic evaluation of user interfaces. In *SIGCHI conference on Human factors in computing systems (CHI)*. 249–256.
[31] David Pinelle, Nelson Wong, and Tadeusz Stach. 2008. Heuristic evaluation for games: usability principles for video game design. In *SIGCHI conference on human factors in computing systems (CHI)*. 1453–1462.
[32] Prevasio. 2020. *Operation Red Kangaroo.* Retrieved May 24, 2022 from https://knowledge-base.prevasio.io/pdf.html?file=Red_Kangaroo.pdf
[33] Andrew Sears. 1997. Heuristic walkthroughs: Finding the problems without the noise. *International journal of human-computer interaction (IJHCI)* 9, 3 (1997), 213–234.
[34] Rui Shu, Xiaohui Gu, and William Enck. 2017. A study of security vulnerabilities on Docker Hub. In *ACM on Conference on Data and Application Security and Privacy (CODASPY)*. 269–280. https://doi.org/10.1145/3029806.3029832
[35] Justin Smith, Lisa Nguyen Quang Do, and Emerson Murphy-Hill. 2020. Why can't johnny fix vulnerabilities: A usability evaluation of static analysis tools for security. In *Symposium on Usable Privacy and Security (SOUPS)*. 221–238.
[36] Justin Smith, Brittany Johnson, Emerson Murphy-Hill, Bill Chu, and Heather Richter Lipford. 2015. Questions developers ask while diagnosing potential security vulnerabilities with static analysis. In *Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. 248–259. https://doi.org/10.1145/2786805.2786812
[37] Snyk. 2023. *Fixed in version vs. fixable attribute in vulnerabilities.* Retrieved Feb 12, 2023 from https://docs.snyk.io/manage-issues/starting-to-fix-vulnerabilities/fixed-in-version-vs.-fixable-attribute-in-vulnerabilities
[38] Mohammad Tahaei, Kami Vaniea, Konstantin Beznosov, and Maria K Wolters. 2021. Security notifications in static analysis tools: Developers' attitudes, comprehension, and ability to act on them. In *CHI Conference on Human Factors in Computing Systems (CHI)*. 1–17.
[39] Omer Tripp, Salvatore Guarnieri, Marco Pistoia, and Aleksandr Aravkin. 2014. ALETHEIA: Improving the usability of static security analysis. In *ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. 762–774. https://doi.org/10.1145/2660267.2660339
[40] Olufogorehan Tunde-Onadele, Jingzhu He, Ting Dai, and Xiaohui Gu. 2019. A Study on Container Vulnerability Exploit Detection. In *IEEE International Conference on Cloud Engineering (IC2E)*. 121–127. https://doi.org/10.1109/IC2E.2019.00026
[41] Karel Vredenburg, Ji-Ye Mao, Paul W Smith, and Tom Carey. 2002. A survey of user-centered design practice. In *SIGCHI conference on Human factors in computing systems (CHI)*. 471–478.
[42] Michael S. Ware and Christopher J. Fox. 2008. Securing Java code: heuristics and an evaluation of static analysis tools. In *Workshop on Static Analysis (SAW)*. 12–21. https://doi.org/10.1145/1394504.1394506
[43] Katrine Wist, Malene Helsem, and Danilo Gligoroski. 2021. Vulnerability analysis of 2500 Docker Hub images. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 307–327.
[44] Stan Zajdel, Diego Elias Costa, and Hafedh Mili. 2022. Open source software: an approach to controlling usage and risk in application ecosystems. In *ACM International Systems and Software Product Line Conference-Volume A (SPLC)*. 154–163.
[45] Jiajie Zhang, Todd R Johnson, Vimla L Patel, Danielle L Paige, and Tate Kubose. 2003. Using usability heuristics to evaluate patient safety of medical devices. *Journal of biomedical informatics* 36, 1-2 (2003), 23–30.

A SMITH ET AL.'S HEURISTICS

To develop customized heuristics tailored for evaluating the usability of Docker image vulnerability scanners, we draw upon the heuristics proposed by *Smith et al.* [35, 36]. *Smith et al.* proposed 17 heuristics specifically designed for assessing the usability of static analysis tools. Table 4 presents the list of heuristics along with their descriptions.

Table 4: *Smith et al.*'s heuristics adapted from [35, 36].

Heuristic	Description
Confirming Expectations	Does the tool behave as expected?
Understanding & Interacting with Tools	Does the tool provide information about accessing and making sense of tools available?
Relationship between Vulnerabilities	Does the tool provide information about how co-occurring vulnerabilities relate to each other?
Developer Planning & Self-Reflection	Does the tool provide information about the tool user reflecting on or organizing their work?
Vulnerability Severity & Rank	Does the tool provide information about the potential impact of vulnerabilities, including which vulnerabilities are potentially most impactful?
Preventing & Understanding Potential Attacks	Does the tool provide information about how an attack would exploit this vulnerability or what types of attacks are possible in this scenario?
Locating Information	Does the tool provide information that satisfies "where" questions, searching for vulnerabilities in the code?
Resources & Documentation	Does the tool provide additional information about help resources and documentation?
Understanding Concepts	Does the tool provide information about unfamiliar concepts that appear in the code or in the tool?
Notification Text	Does the tool provide textual information that an analysis tool provides and how that text relates to the potentially vulnerable code?
Understanding Alternative Fixes & Approaches	Does the tool provide information about alternative ways to achieve the same functionality securely?
Assessing the Application of the Fix	Once a fix has been selected and/or applied, does the tool provide information about the application of that fix or assessing the quality of the fix?
Control Flow & Call Information	Does the tool provide information about the callers and callees of potentially vulnerable methods?
Data Storage & Flow	Does the tool provide information about data collection, storage, its origins, and its destinations?
Code Background & Functionality	Does the tool provide information about the history and the functionality of the potentially vulnerable code?
Application Context / Usage	Does the tool provide information about how a piece of potentially vulnerable code fits into the larger application context (*e.g.*, test code)?
End-User Interaction	Does the tool provide information about sanitization/validation and input coming from users? Does the tool show where input to the application is coming from?

B OUR FIRST HEURISTICS

By removing five source code analysis-specific heuristics and modifying two from the 17 heuristics (Appendix A) proposed by *Smith et al.*, we developed an initial version of 13 heuristics specifically designed to assess the usability of DIVSes. Table 5 presents the list of heuristics along with their descriptions.

Table 5: Our first heuristics.

Heuristic	Description
Confirming Expectation	Does the tool perform its designed function without unexpected or unintended outcomes?
Understanding & Interacting with Tools	Does the tool provide clear and user-friendly guidance on how to use its functionalities?
Relationship between Vulnerabilities	Does the tool provide information about how co-occurring vulnerabilities relate to each other?
Developer Planning & Self-Reflection	Does the tool provide information about the tool user reflecting on or organizing their work?
Vulnerability Severity & Rank	Does the tool clearly indicate the severity level of each vulnerability discovered, and prioritize them based on their severity levels?
Preventing & Understanding Potential Attacks	Does the tool provide information about how an attacker would exploit this vulnerability or what types of attacks are possible in this scenario?
Locating Information	Does the tool provide information that satisfies "where" the vulnerability locates in the Docker image?
Resources & Documentation	Does the tool provide external references and documents that are necessary to gain a comprehensive understanding of those vulnerabilities?
Understanding Concepts	Does the tool provide an additional explanation about the unfamiliar concepts to make developers understandable?
Necessary & Sufficient Information about Vulnerabilities	Does the tool provide necessary and sufficient information about vulnerabilities?
Consistency between Vulnerability Records	Does the tool provide same amount of information for all individual vulnerabilities?
Correctness of the Fix	Does the tool provide information about alternative ways to achieve the same functionality securely? Is that information correct?
Assessing the Application of the Fix	Does the tool provide practical information about fix? Once a fix has been selected and/or applied, does the tool provide information about the application of that fix?

C VISUAL MOCKUP FOR CLI DIVSES

Based on our recommendations Section 4, we developed a visual DIVS mockup for CLI environments, as shown in Figure 10. We demonstrated the feasibility of the proposed CLI DIVS interface by implementing it using the Python module PyCUI [11] without developing a core engine for vulnerability scanning.

In Figure 10, the DIVS interface comprises three text boxes, one search box, and four buttons. The vulnerability report is divided into three sections: (1) a summary of the scanning results (<Summary> box), (2) a list of identified vulnerabilities (<Vulnerabilities> box), and (3) detailed information about the selected vulnerability (<More Information> box).

The <Summary> box provides an overview of vulnerability statistics for the analyzed Docker image, making it easier for developers to understand (satisfying H3). Each vulnerability's severity level and priority score are clarified, helping developers trust the results and prioritize identified vulnerabilities (satisfying H5).

The <Vulnerabilities> box displays four essential details for all vulnerabilities, such as severity level, priority score, affected software module, and CVE ID. Vulnerabilities are sorted by priority scores to highlight which ones need fixing first (satisfying H5). The search box enables developers to search by vulnerability number, severity level, affected software module, or CVE ID (satisfying H4). Developers can select vulnerabilities from the <Vulnerabilities> box or use the search box to view more information.

The <More Information> box provides a detailed description (description, patch information, related vulnerabilities, references, and details) for the selected vulnerability. The "Description" and "Details" sections offer brief and detailed descriptions, respectively (satisfying H6). The "Patch information" section not only displays the installed and patched versions of the affected software module but also provides update instructions such as the command for patching (satisfying H10). The "References" section offers reference links, including URLs for the DIVS vendor's database and other reliable public vulnerability databases (satisfying H6 and H7). Clear and concise terms are used throughout the document (satisfying H8), and the word "None" is employed when information is unavailable (satisfying H9).

We added four buttons at the bottom to provide additional functionality. The first button ("Manual") loads a manual page about using the tool (satisfying H2). The second button ("Export to Other Formats") enables developers to save the vulnerability report and leverage other formats for in-depth analysis (satisfying H4). The third button ("Compare Reports") allows developers to compare patched vulnerabilities (satisfying H11). The last button ("Report Issues") lets users report issues with the tool, ensuring its functionality (satisfying H1).

This mockup design demonstrates how our recommendations can be applied to CLI DIVSes.

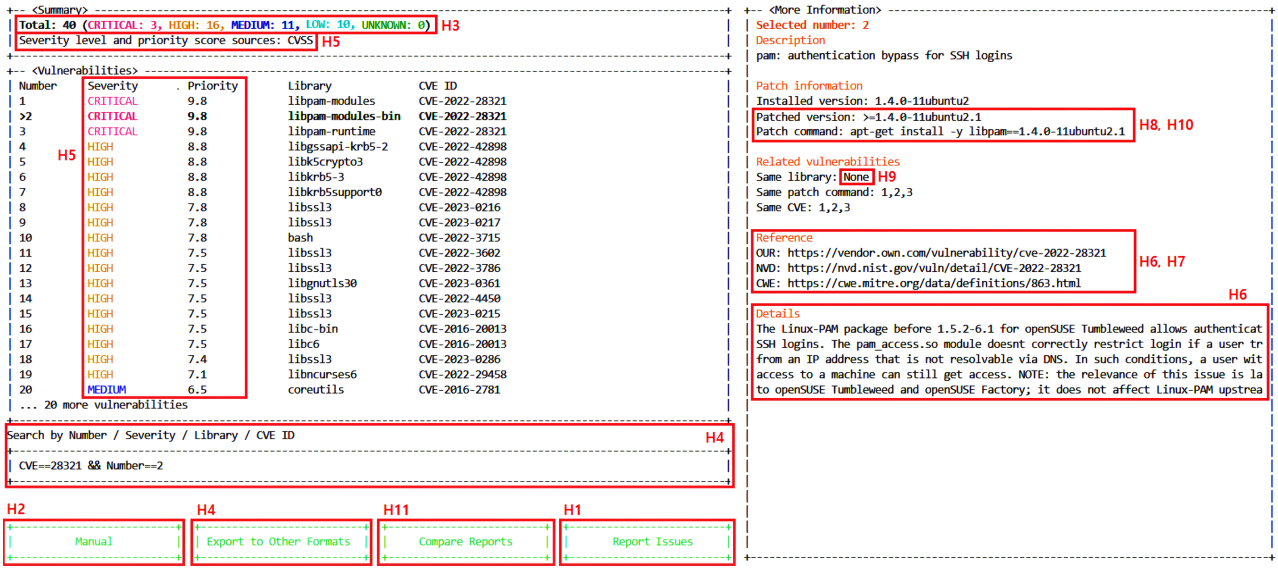


Figure 10: Visual mockup for CLI DIVSes, designed with our recommendations.