Design of a Hardening Module for Automatically Securing Containers

Luca Verderame[1,*], Luca Caviglione[2]

[1]DIBRIS - University of Genova, Via Dodecaneso 35, Genova, I-16146, Italy
[2]National Research Council of Italy, Via de Marini, I-16149, Genova, Italy

Abstract

Modern microservices need to quickly face changes both in terms of technology and requirements of users. To address such a challenging scenario, developers and IT operators should be able to concentrate on integration and delivery tasks, possibly without having to deal with security aspects. In this vein, the creation of architectures for supporting the DevOps pipeline is an important goal. Yet, assessing the security of containers is a difficult task, especially when considering distributed or large-scale deployments.

To cope with such complexity, this paper presents the design of a hardening module for automatically securing containers. Such a mechanism is part of the framework envisioned in Project Securing Containers - SecCo, which aims at offloading the DevOps software development paradigm from security-related tasks.

Keywords

Container security, DevSecOps, CI/CD security

1. Introduction

Recent large-scale and adaptive Internet services require a paradigm shift in both the development and deployment phases. To meet performance goals and make the software more efficient and cost-effective, as well as to reduce issues and conflicts during the development stages, an emerging approach leverages microservices. In essence, the functionalities of a full-featured application are broken down into a set of small, independent components [1]. However, transforming a monolithic software into a set of compact entities poses many challenges. For instance, modern scenarios are often hybrid and contain serverful and serverless services, thus accounting for non-negligible planning and development efforts [2]. In this perspective, the DevOps model helps to balance the complexity-efficiency trade-off required by modern and fast-paced large-scale platforms. Specifically, microservices can be maintained, improved, and released in a quicker manner if compared to monolithic applications. Moreover, their smaller footprint allows for re-usability and makes possible the creation of cloud-native applications in an (almost) easy manner.

To pursue the vision of microservices, including the massive adoption of cloud platforms to cut costs and enjoy full scalability properties, containers play a critical role. Specifically, they offer lightweight execution environments that can be composed through simple interfaces or chained through the network [3]. Another important benefit of containers deals with the ability to migrate them across different machines, for instance, to implement load balancing or self healing policies [4]. Alas, properties such as flexibility and rapid deployment do not come for free. Container-based microservices have a security posture difficult to assess in a comprehensive manner and also require a non-negligible understanding of the entire development flow [5]. This can be mitigated by decoupling the Continuous Integration and Continuous Delivery (CI/CD) process from the counterpart devoted to security: developers and IT operators should then solely concentrate on the CI/CD phase without dealing with security-oriented tasks. As an example, specific security constraints may entail static and runtime guarantees built

ITASEC 2024: The Italian Conference on CyberSecurity, April 08–12, 2024, Salerno, Italy
*Corresponding author.
luca.verderame@unige.it (L. Verderame); luca.caviglione@cnr.it (L. Caviglione)
0000-0001-7155-7429 (L. Verderame); 0000-0001-6466-3354 (L. Caviglione)

into the execution environment. Similarly, developers could "offload" part of their pipeline by using a third-party service offering hardening strategies in a black box manner.

Therefore, this paper presents the design of the hardening component of the (Securing Containers) architecture [6]. In more detail, the goal of is to engineer a framework for securing containers in an automatic manner. The module responsible for hardening is critical, as it largely contributes to combining development (Dev), security (Sec), and operations (Ops) by providing developers with a set of container templates that match the functional requirements of the application and adhere to security best practices.

In this perspective, the contribution of this paper is twofold: it provides a thorough discussion of the architecture envisioned in SecCo, and it presents the preliminary assessment of the tools and technologies to build the hardening pipeline in terms of security liniting and container security scanning tools.

The rest of the paper is structured as follows. Section 2 introduces the architecture at the basis of the framework, Section 3 presents the design of the hardening module, and Section 4 discusses the implementation choices with emphasis on the tools that can be used to prepare hardened containers. Section 5 reviews the literature dealing with container security and Section 6 concludes the paper and outlines future research directions.

2. Overall Architecture of the SecCo Framework

This section showcases the architectural components at the basis of the pipeline (see [6] for a thorough discussion). Specifically, the *hardening*, *compliance verification*, and *runtime monitoring* modules cooperate to implement the process needed to secure containers and make them more suitable for building production-quality microservices. We recall that aims at automatizing all the steps characterizing the DevOps paradigm, which are needed to successfully deploy a container. To this aim, all the needed techniques, mechanisms, and tools, are coherently grouped into the following functional components:

- **Hardening**: this functional component is devoted to make containers compliant with security best practices, functional requirements, as well as constraints characterizing the application under development. For instance, the CI/CD team should define supported/required network protocols [7], transport ports and services that could be remotely accessed, or the type of optional libraries required by the various containers. To properly work, the hardening module should be able to gain information on the requested containers (e.g., type and number) and a precise shortlist of security constraints. A first outcome is to route the CI/CD team to already-hardened containers. In case a pre-existing container exists, the developers can then directly retrieve the already-hardened image and prevent the pipeline from being triggered for producing a new object from scratch. On the contrary, the need to process a new container will trigger this module. In general, this requires querying external repositories, such as the Docker Hub, to search for images matching the criteria specified by the CI/CD team. Candidate containers are then subjected to threat and vulnerability assessments to recognize the main attack vectors, isolate security hazards/vulnerabilities, and define the optimal patching/mitigation techniques. Concerning the process used to patch a container, it can range from the modification of the container (e.g., the Dockerfile of a Docker container) to the injection of additional security services/libraries that can guarantee the required confidentiality, authentication, and authorization functionalities. To mention a possible example, the hardening module could patch a container to make it able to use a TLS service (e.g., a termination proxy), an Identity Provider, and an OAuth module. Despite the hardening procedure that has been enforced, the last step entails configuration/customization procedures to actually release the container in its final form.
- **Compliance Verification**: this functional component embraces all the mechanisms needed to outline how the container must act at runtime with respect to a set of desired security requirements. The module is then responsible for verifying security policies. In this vein, the preliminary step demands for checking the compliance of the various containers composing a microservice against

a list of security specifications that can be evaluated "offline", i.e., in a static environment. To be useful, the outcome must clearly state how to customize/instrument containers to support the runtime monitoring phase. In order to operate, the functional component for the compliance verification envisioned in should be fed by developers with two bits of information. The first is the set of hardened containers hosting the developed microservice application. The second is a thorough definition of the security policy characterizing the given application scenario. To make some examples, security requirements could prevent the adoption of non-root users, enforce restrictions of protocols or endpoints as well as specify runtime permissions such as impeding to switch APIs at runtime. To verify whether a container adheres to a well-defined security policy, a behavioral model should be used, e.g., to evaluate if the security-sensitive interactions of the container (e.g., with the container engine) are admissible. Concerning possible techniques for the compliance verification procedures, we mention automated security verification methods such as model-checking [8].

- **Runtime Monitoring**: this functional component groups all the mechanisms needed to evaluate security requirements in an "online" fashion, including unexpected runtime interactions between containers and the underlying engine. As a result, this module is responsible for both the verification of the compliance of dynamic and static security policies. In this perspective, techniques that can take advantage of kernel augmentation and code layering as well as the use of efficient tools like the extended Berkeley Packet Filter should be taken in high regard [9, 10]. A successful monitoring service should also be able to evaluate corner cases and unexpected runtime interactions among containers, including collusive attack templates or the presence of information-hiding-capable threats [11]. Lastly, runtime monitoring may also require suitable methodologies for automatic security enforcement and to interrupt interactions that violate (dynamic) security policies. We point out that, runtime monitoring can be also used to reveal the presence of threats and vulnerabilities not considered in the "offline" phase. Moreover, runtime monitoring may help to mitigate the impact of zero-days not known at the time of implementing the hardening procedure.

A detailed discussion of the design of the hardening module envisioned in is presented in the following.

3. Design of the Hardening Module

This section deals with the design of the *Hardening Module*, which is in charge of processing the set of requested containers and the security constraints of the CI/CD team to verify the availability of pre-hardened images from the *Hardened Templates DB*. This enables the CI/CD team to directly obtain pre-hardened, standardized templates, alleviating the need to construct and secure containers from scratch.

Figure 1 depicts the overall functional architecture. In essence, the hardening process happens between the planning (denoted as *PLAN*) and the development (denoted as *DEVELOP*) phases. In more detail, for any new container, the *Hardening Processor* initiates the *Container Template Builder* process by considering the container specification and the security requirements. As a first step, the service searches for suitable existing containers from external repositories, such as Docker Hub, to serve as a baseline. The selection is contingent upon both functional and security requirements. For instance, if the CI/CD team aims to deploy a Docker container hosting a Postgres database, the Container Template Builder identifies the official image[1] on Docker Hub as the foundational one. Subsequently, the Container Template Builder generates a minimal Dockerfile specification tailored to meet the requirements of the CI/CD team and initiates a security hardening procedure. This step endeavors to pinpoint potential security risks or vulnerabilities within the resulting container image and formulate optimal strategies to rectify or mitigate them, thereby minimizing the attack surface.

[1] https://hub.docker.com/_/postgres

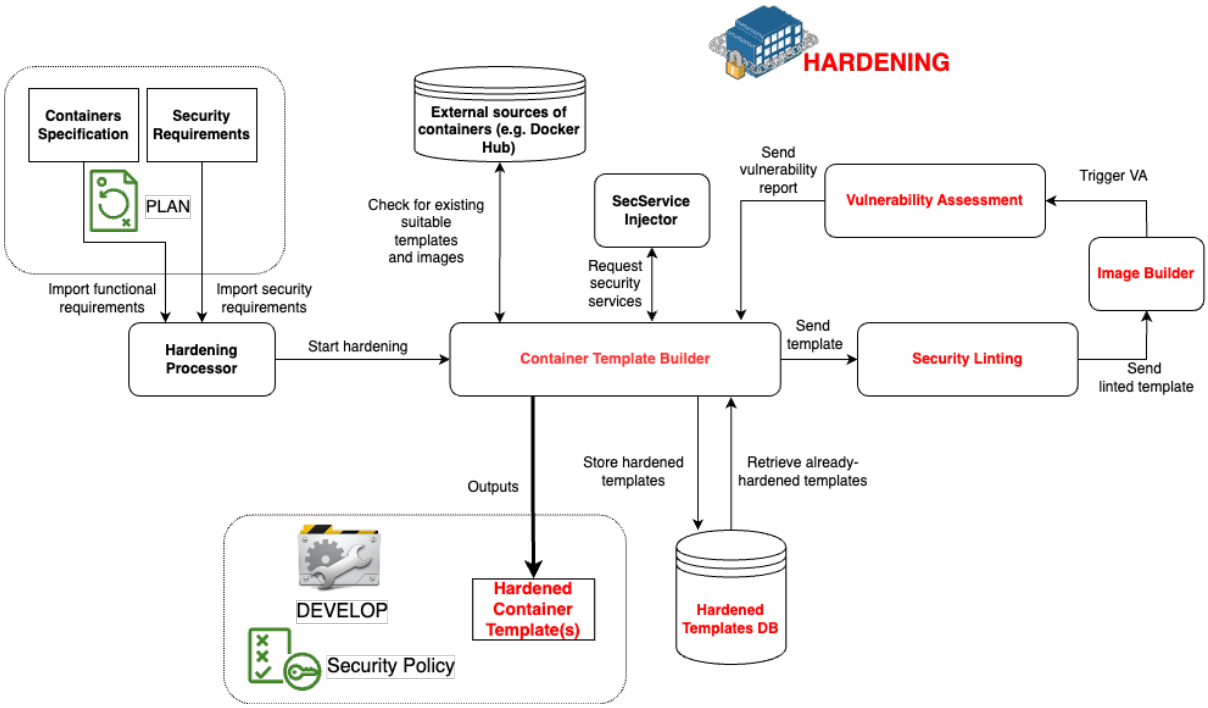

Figure 1: Architecture of the Hardening Module.

The initial phase of the security assessment involves *Security Linting* of the template. Linters are used to systematically scrutinize Dockerfiles for various security-related issues (often referred to as *smells*), such as the use of outdated base images with known vulnerabilities or the embedding of hard-coded secrets or sensitive information. Based on these findings, the Security Linting process can either automatically rectify the issues or alert the CI/CD team and suggest appropriate fixes.

The resulting Dockerfile is then utilized to build an image leveraging the *Image Builder* and subjected to a vulnerability assessment phase. This step generates a report detailing potential vulnerabilities present in the image. Depending on the nature and severity of the hazards, the Container Template Builder may either automatically address the vulnerability or report it to the developer.

The patching phase encompasses modifying the container specification, such as the Dockerfile of a Docker container, and the integration of on-demand security services and libraries via *SecService Injection*. These services furnish core functionalities such as confidentiality, authentication, and authorization to the container without their explicit implementation. Examples of such security services include a TLS service (e.g., termination proxy).

The security assessment process may be iterated multiple times to evaluate incremental patches or fixes automatically applied by the Hardening Module or implemented by the CI/CD team, ensuring that the resultant template maintains a residual risk below a predefined threshold.

4. Towards Implementation of the Hardening Module

This section presents the preliminary steps performed to implement the Hardening Module. In more detail, we conducted an assessment focusing on security linting and container scanning tools tailored for the Docker ecosystem. The goal of this research/design activity is to identify the best technologies that could support the building of the Hardening Module. Specifically, we examined documentation and performed a series of prime hands-on assessments. To this aim, we relied upon several key parameters to ensure robust security practices as suggested in [12].

Firstly, we considered if a tool supports the SARIF format[2], i.e., a standardized method for sharing

[2]https://www.oasis-open.org/committees/sarif/

Table 1
Security Linters for the Docker ecosystem.

Tool	Last Release	CIS Docker Security Benchmark	OWASP Top 10	Output SARIF
Dockle	2023	✓	✓	✓
Docker-Bench-Security	2023	✓	✗	✗
Hadolint	2023	✗	✓	✓
Docker Cleaner	2023	✓	✓	✗
dockerfile-lint	2023	✗	✗	✗
Docker Scout	2024	✓	✓	✓
Kics	2023	✓	✓	✗

static analysis results. In fact, the compatibility with SARIF facilitates the seamless integration with various development and security tools, streamlining the overall security workflow. Furthermore, our evaluation encompassed the adherence of the tools to the CIS Security Docker Benchmark [13] and the OWASP TOP 10 Security risks [14]. Ensuring compliance with these benchmarks is crucial for mitigating known vulnerabilities and strengthening the overall security posture of microservices. In the following, we provide details on the obtained results.

4.1. Selection of Security Linters

Several security linters for Docker images exist, each one with its own focus and approach. Specifically, Dockle [15] and Docker Bench for Security [16] are explicitly tailored to evaluate the security of Dockerfiles and support checks aligned with the recommendations in the CIS Docker benchmark, ensuring adherence to industry standards. Hadolint [17] is a linter that parses the Dockerfile into an abstract syntax tree and performs rules on top of it, thereby ensuring a wider coverage compared to other open source tools [18]. Binnacle [18] is a tool that studies and detects Docker smells in Dockerfiles. Compared to other linters, this tool also analyzes the presence of smells inside GitHub and performs a comparison with a high-quality set of Dockerfiles. Additionally, Docker Cleaner [19] incorporates repair techniques based on CIS Docker Benchmark and OWASP Docker Security Cheat Sheet [20], leveraging language-docker for parsing Dockerfiles and suggesting patches. Recently, Docker developers have integrated their own security tool, namely Docker Scout [21], for searching and fixing vulnerabilities in the Docker images. Checkmarx KICS (Keeping Infrastructure as Code Secure) [22] is a robust static analysis tool to analyze configurations of Docker containers written in Dockerfiles. By leveraging its rule-based engine, Checkmarx KICS thoroughly examines Dockerfiles to identify potential security vulnerabilities, misconfigurations, and compliance issues. The tool can be integrated with vulnerability databases and compliance standards like CIS benchmarks. This ensures that Docker containers adhere to the highest security standards.

Table 1 provides a summary of our analysis. Specifically, the table reports the most relevant publicly-available security linters for the Docker ecosystem that we considered for our hardening module. As shown, the table details the year of the last release of each tool, as well as the evaluation of the compliance with respect to CIS Docker Security Benchmark, and the OWASP TOP 10 Security Risk. Lastly, the table also reports whether a tool supports the SARIF format.

4.2. Selection of Stating Application Security Testing Tools

In recent years, there has been a significant effort by both academia and industry to develop robust Docker image security scanners to address the growing need for securing containerized applications. Numerous tools have emerged, each offering unique features and capabilities for identifying and mitigating vulnerabilities of the Docker ecosystem. For the design of the Hardening Module, we have focused on some of the most widely used open-source Docker image security scanners available today.

Specifically, Trivy [23] is an open-source Docker image vulnerability scanner that supports the

Table 2
Container scanning tools for the Docker ecosystem.

Tool	Last Release	CIS Docker Security Benchmark	OWASP Top 10	Output SARIF
Trivy	2023	✓	✗	✓
Clair	2023	✗	✓	✗
Snyk	2023	✗	✓	✓
Grype	2023	✗	✗	✓
Docker Scout	2023	✓	✓	✓
Dagda	2021	✗	✓	✗
OpenSCAP	2023	✓	✓	✗

analysis of Docker images, OCI[3] images, and container filesystems. Trivy leverages vulnerability databases like the National Vulnerability Database and the Red Hat Security Data API to identify known vulnerabilities in the software packages and dependencies installed within Docker images. Its fast scanning capabilities and direct integration with CI/CD pipelines make it a popular choice among DevOps teams for continuously assessing the security posture of containerized applications. Clair [24] is a powerful open-source vulnerability scanner designed specifically for Docker containers. It performs static analysis on Docker images to identify known vulnerabilities in the software packages and additional libraries. The tool can seamlessly integrate with container registries such as Docker Hub, allowing for automatic scanning of images as they are pushed or pulled. A major benefit of Claris is rooted in its modular architecture and adoption of a RESTful API, which make it highly extensible and suitable for integration into various container security workflows. Snyk [25] is a comprehensive security platform that offers vulnerability scanning and monitoring for Docker containers. It can be integrated into the software development lifecycle, providing developers with tools to find, fix, and prevent vulnerabilities in their code and dependencies, including Docker images. As regards the vulnerability database used, the tool can rely on information from public sources and proprietary research, ensuring comprehensive coverage of known vulnerabilities. Nevertheless, Snyk offers both static analysis for scanning Docker images before deployment and runtime monitoring for detecting vulnerabilities in running containers. Another software for fast and accurate scanning of container images for known vulnerabilities is Grype [26]. In essence, it offers a command-line interface that allows users to scan Docker images and generate detailed vulnerability reports. It can be used to identify vulnerabilities in various package ecosystems, including Linux distributions and programming language libraries.

In addition to the security linting functionalities, Docker Scout can also be used to evaluate the vulnerabilities inside images by creating a full inventory of the packages called a Software Bill of Materials. It then correlates this inventory with a continuously updated vulnerability database to identify vulnerabilities of images. Concerning static analysis, Dagda [27] can perform both pre-deployment scanning and runtime monitoring. For the case of pre-deployment scanning, it analyzes Docker images for known vulnerabilities and misconfigurations, Instead, when used for runtime monitoring, it checks the behavior of running containers and detects suspicious activities or potential security breaches. Despite being mainly a tool for security compliance, OpenSCAP [28] also offers capabilities for scanning container images. It leverages the Security Content Automation Protocol to assess the security of Docker images against predefined security profiles and benchmarks. As an output, OpenSCAP provides detailed reports highlighting compliance with security standards such as the DISA STIG and NIST SP 800-53.

To summarize, Table 2 reports the selection of container scanning tools that we have assessed for integration into the Hardening Module of the pipeline of . Specifically, the table outlines the most recent release year of the tool and evaluates the compliance with the CIS Docker Security Benchmark, coverage of OWASP TOP 10 Security Risks, and support for the SARIF format.

[3] https://opencontainers.org/

4.3. Discussion

As a result of the analysis performed on the various security linters and container scanning tools, we have identified the software components that are most suited for meeting the requirements of the Hardening Module to be deployed within the framework of . In more detail, among the security linters discussed in Section 4.1, Docker Scout and Docker Cleaner exhibit full compliance with CIS and OWASP standards. Additionally, they incorporate built-in heuristics to automatically patch analyzed Dockerfiles, thereby contributing to the hardening phase at the basis of the Container Template Builder. As regards container scanning tools presented in Section 4.2, both Tryvy and OpenSCAP emerge as open-source solutions capable of supporting both the OWASP TOP 10 and the CIS Docker Security Benchmark. Moreover, they offer native support for the SARIF format, which surely facilitates the seamless integration and interoperability with other components that populate the Hardening Module.

5. Related Work

According to a recent survey, the majority of works dealing with container security proposes to address hazards by focusing on the development phase [29]. In this perspective, static analysis tools demonstrated to be effective, especially for revealing the presence of known CVEs in container images [30], [24]. However, many tools used in production-quality environments have the limit of not considering vulnerabilities of third-party libraries or additional software packages. As a result, they are characterized by a very modest detection rate and should be used "with care" when handling sensitive or mission-critical containers [31]. Another limitation of static analysis tools is due to their scope. Specifically, many packages only consider a single container image, hence missing the evaluation of possible interactions with the container engine or local/remote microservices [29]. Unfortunately, this reduces the effectiveness of the approach when used against a modern microservice architecture, which could be highly mutable, heterogeneous and orchestrated in a complex manner, especially to guarantee scalability and performances of monolithic counterparts [32]. Therefore, a common approach to face challenges in container security is to take advantage of two strategies at once. The first strategy exploits mechanisms devoted to vulnerability analysis [33],[34]. The second strategy exploits various mitigation workarounds based on well-defined patterns or best practices [35],[36]. Another relevant case considered by the literature deals with the orchestration phase, which is a core building block of cloud-native applications. Alas, the majority of cloud-oriented security platforms still requires a comprehensive process to enforce security within the standard operation flow. A possible idea is to deploy a Secure Container Orchestrator engine based on hardware-based trusted execution environment technologies for data protection [37]. Another viable approach is to take advantage of the peculiarities of the software architecture. For instance, when Kubernetes is used, AppArmor policies for secure cloud-native deployments are demonstrated to be effective [38].

For the specific case of Docker containers, the work in [39] introduces a comprehensive solution that integrates existing security analysis tools (e.g., SonarQube) for enhancing the global security posture of a deployment. Dynamic testing techniques can also be adopted to further advance in performance. As an example, they can be employed to "coordinate" three different automated dynamic testing techniques: Web Application Security Testing, Security API Scanning, and Behaviour Driven Security Testing [40]. When generalization is not possible, an effective solution entails the definition and creation of specific security profiles. For instance, AppArmor can be used to produce profiles to enforce access policies and mitigate risks caused by zero-day vulnerabilities [41]. Access control has also been used too: appropriate rules can be considered effective [42]. Yet, suitable cryptographic protocols (e.g., TLS and OAuth) to enhance the security should be considered in the most critical container ecosystems [43]. Indeed, more holistic approaches should be also considered as envisioned in [6] and the references therein.

Lastly, a relevant amount of work has been done for the specific case of Docker containers, mainly owing to their widespread diffusion. An increasing number of publicly available containers have relevant security vulnerabilities that can be exploited quite straightforwardly [44], [45], thereby indicating that current security practices are far from being reliable. For example, a recent research paper [46]

showcased that the 51% of ~ 4 million images hosted in Docker Hub have exploitable vulnerabilities. As regards securing Docker containers, we mention the work in [47], which considers exploiting the information on the relationship of vulnerable software packages and documented issues of Docker images. Another idea relies upon the static analysis of Dockerfiles, especially to search for potential vulnerabilities of the used software components and libraries [48]. Besides, the increasing degree of sophistication of attacks, for instance, threats aiming at increasing the energy footprint of a datacenter, accounts for new techniques to reveal potential hidden communication paths that can be used to orchestrate offensive campaigns [49]. As regards online detection through syscalls or fine-grained behaviors, a possible approach is to use kernel augmentation to inspect agentless applications or prevent the need for sidecar containers [10].

6. Conclusions and Future Work

This paper presented the overall architecture at the basis of Project , which can be used as a reference template to automatize security operations needed in modern DevOps pipelines. A major benefit of having a comprehensive framework is to let developers and IT operators to solely concentrate on the delivery process. The framework is composed of three core modules, i.e., hardening, compliance verification, and runtime monitoring. As discussed, the hardening module should process the set of requested containers and the security constraints of the CI/CD team to build container templates that have a minimal footprint in terms of security risks. The preliminary assessment made in this paper lays the foundation for the hardening pipeline by analyzing and discussing the relevant tools to support the security linting and container scanning tasks for the Docker ecosystem.

Future works aim at advancing the development of the overall framework and integrating all the required software components into a prototype to test performance on real container images collected "in the wild". For the specific case of the hardening module, the main mid-term research questions to be addressed range from the suitability of pre-existent tools (e.g., security linters) to an efficient solution to integrate the various output to fully assess the security of containers. A possible approach that we are considering is to use Large Language Models to process logs.

Acknowledgments

This work was partially supported by project SERICS (PE00000014) under the NRRP MUR program funded by the EU - NGEU.

References

[1] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, L. Safina, Microservices: yesterday, today, and tomorrow, Present and ulterior software engineering (2017) 195–216.

[2] N. C. Mendonça, P. Jamshidi, D. Garlan, C. Pahl, Developing self-adaptive microservice systems: Challenges and directions, IEEE Software 38 (2019) 70–79.

[3] E. Casalicchio, Container orchestration: A survey, Systems Modeling: Methodologies and Tools (2019) 221–235.

[4] L. Ma, S. Yi, Q. Li, Efficient service handoff across edge servers via docker container migration, in: Proceedings of the Second ACM/IEEE Symposium on Edge Computing, 2017, pp. 1–13.

[5] J. Watada, A. Roy, R. Kadikar, H. Pham, B. Xu, Emerging trends, techniques and open issues of containerization: a review, IEEE Access 7 (2019) 152443–152472.

[6] L. Verderame, L. Caviglione, R. Carbone, A. Merlo, Secco: Automated services to secure containers in the devops paradigm, in: Proceedings of the 2023 International Conference on Research in Adaptive and Convergent Systems, 2023, pp. 1–6.

[7] A. Armando, G. Pellegrino, R. Carbone, A. Merlo, D. Balzarotti, From model-checking to automated testing of security protocols: Bridging the gap 7305 LNCS (2012) 3–18. doi:10.1007/978-3-642-30473-6_3.
[8] A. Armando, G. Costa, A. Merlo, L. Verderame, Enabling byod through secure meta-market, 2014, pp. 219–230. doi:10.1145/2627393.2627410.
[9] L. Caviglione, W. Mazurczyk, M. Repetto, A. Schaffhauser, M. Zuppelli, Kernel-level tracing for detecting stegomalware and covert channels in linux environments, Computer Networks 191 (2021) 108010.
[10] M. Zuppelli, M. Repetto, A. Schaffhauser, W. Mazurczyk, L. Caviglione, Code layering for the detection of network covert channels in agentless systems, IEEE Transactions on Network and Service Management 19 (2022) 2282–2294.
[11] W. Mazurczyk, P. Szary, S. Wendzel, L. Caviglione, Towards reversible storage network covert channels, in: Proceedings of the 14th International Conference on Availability, Reliability and Security, 2019, pp. 1–8.
[12] D. B. Cruz, J. R. Almeida, J. L. Oliveira, Open source solutions for vulnerability assessment: A comparative analysis, IEEE Access (2023).
[13] C. for Internet Security, Cis docker benchmark, https://www.cisecurity.org/benchmark/docker, 2024. Accessed: April 21, 2024.
[14] O. Foundation, Owasp top 10 ci/cd security risks, https://owasp.org/www-project-top-10-ci-cd-security-risks/, 2022. Accessed: April 21, 2024.
[15] G. LLC, Dockle, https://github.com/goodwithtech/dockle/, 2022. Accessed: April 21, 2024.
[16] Docker, Docker bench for security, https://github.com/docker/docker-bench-security/, 2019. Accessed: April 21, 2024.
[17] Hadolint, Hadolint, https://github.com/hadolint/hadolint, 2022. Accessed: April 21, 2024.
[18] J. Henkel, C. Bird, S. K. Lahiri, T. Reps, Learning from, understanding, and supporting devops artifacts for docker, in: Proceedings of the ACM/IEEE 42nd international conference on software engineering, 2020, pp. 38–49.
[19] Q.-C. Bui, M. Laukötter, R. Scandariato, Dockercleaner: Automatic repair of security smells in dockerfiles, in: 2023 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2023, pp. 160–170.
[20] O. Foundation, Docker security cheat sheet, https://cheatsheetseries.owasp.org/cheatsheets/Docker_Security_Cheat_Sheet.html, 2022. Accessed: April 21, 2024.
[21] Docker, Docker scout, https://docs.docker.com/scout/, 2023. Accessed: April 21, 2024.
[22] Checkmarx, Keeping infrastructure as code secure (kics), https://github.com/Checkmarx/kics, 2021. Accessed: April 21, 2024.
[23] Aquasecurity, Trivy, https://github.com/aquasecurity/trivy, 2023. Accessed: April 21, 2024.
[24] Clair, Vulnerability static analysis for containers, https://github.com/quay/clair, 2023. Accessed: April 21, 2024.
[25] Snyk, Snyk cli, https://github.com/snyk/cli/, 2023. Accessed: April 21, 2024.
[26] Anchore, Grype, https://github.com/anchore/grype, 2023. Accessed: April 21, 2024.
[27] E. Grande, Dagda, https://github.com/eliasgranderubio/dagda, 2021. Accessed: April 21, 2024.
[28] M. Preisler, M. Haicman, Security automation for containers and {VMs} with {OpenSCAP} (2017).
[29] T. Leppänen, A. Honkaranta, A. Costin, Trends for the devops security. a systematic literature review, in: International Symposium on Business Modeling and Software Design, Springer, 2022, pp. 200–217.
[30] Anchore, Anchore engine, https://github.com/anchore/anchore-engine, 2022. Accessed: April 21, 2024.
[31] O. Javed, S. Toor, An evaluation of container security vulnerability detection tools, in: Proceedings of the 2021 5th International Conference on Cloud and Big Data Computing, 2021, pp. 95–101.
[32] G. Mazlami, J. Cito, P. Leitner, Extraction of microservices from monolithic software architectures, in: 2017 IEEE International Conference on Web Services (ICWS), IEEE, 2017, pp. 524–531.
[33] A. Duarte, N. Antunes, An empirical study of docker vulnerabilities and of static code analysis

applicability, in: 2018 Eighth Latin-American Symposium on Dependable Computing (LADC), IEEE, 2018, pp. 27–36.
[34] O. Flauzac, F. Mauhourat, F. Nolot, A review of native container security for running applications, Procedia Computer Science 175 (2020) 157–164.
[35] N. Besic, Microservices security: Challenges and best practices, https://brightsec.com/blog/microservices-security/, 2021. Accessed: April 21, 2024.
[36] Docker, Security best practices, https://docs.docker.com/develop/security-best-practices/, 2019. Accessed: April 21, 2024.
[37] G. P. Fernandez, A. Brito, Secure container orchestration in the cloud: Policies and implementation, in: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 138–145. URL: https://doi.org/10.1145/3297280.3297296. doi:10.1145/3297280.3297296.
[38] H. Zhu, C. Gehrmann, Kub-sec, an automatic kubernetes cluster apparmor profile generation engine, in: 2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS), IEEE, 2022, pp. 129–137.
[39] M. K. Abhishek, D. Rajeswara Rao, Framework to secure docker containers, in: 2021 Fifth World Conference on Smart Trends in Systems Security and Sustainability (WorldS4), 2021, pp. 152–156.
[40] T. Rangnau, R. v. Buijtenen, F. Fransen, F. Turkmen, Continuous security testing: A case study on integrating dynamic security testing tools in ci/cd pipelines, in: 2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC), IEEE, 2020, pp. 145–154.
[41] F. Loukidis-Andreou, I. Giannakopoulos, K. Doka, N. Koziris, Docker-sec: A fully automated container security enhancement mechanism, in: 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), IEEE, 2018, pp. 1561–1564.
[42] S. Sultan, I. Ahmad, T. Dimitriou, Container security: Issues, challenges, and the road ahead, IEEE access 7 (2019) 52976–52996.
[43] C. Munoz, F. Montoto, F. Cifuentes, J. Bustos-Jiménez, Building a threshold cryptographic distributed hsm with docker containers, in: 2017 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON), IEEE, 2017, pp. 1–5.
[44] R. Millman, Most docker container images have critical flaws, https://www.itpro.com/development/containers/357984/most-docker-container-images-have-critical-flaws, 2020. Accessed: April 21, 2024.
[45] B. Kaur, M. Dugré, A. Hanna, T. Glatard, An analysis of security vulnerabilities in container images for scientific data analysis, GigaScience 10 (2021) giab025.
[46] A. Y. Wong, E. G. Chekole, M. Ochoa, J. Zhou, Threat modeling and security analysis of containers: A survey, arXiv preprint arXiv:2111.11475 (2021).
[47] S. Kwon, J.-H. Lee, Divds: Docker image vulnerability diagnostic system, IEEE Access 8 (2020) 42666–42673. doi:10.1109/ACCESS.2020.2976874.
[48] T.-P. Doan, S. Jung, Davs: Dockerfile analysis for container image vulnerability scanning, CMC-COMPUTERS MATERIALS & CONTINUA 72 (2022) 1699–1711.
[49] E. Cambiaso, L. Caviglione, M. Zuppelli, Dockerchannel: A framework for evaluating information leakages of docker containers, SoftwareX 24 (2023) 101576.