SecCo: Automated Services to Secure Containers in the DevOps Paradigm

Luca Verderame*
DIBRIS - University of Genova
Genova, Italy
luca.verderame@dibris.unige.it

Luca Caviglione
Consiglio Nazionale delle Ricerche
Genova, Italy
luca.caviglione@cnr.it

Roberto Carbone
Fondazione Bruno Kessler
Trento, Italy
carbone@fbk.eu

Alessio Merlo
Centre for High Defense Studies
Rome, Italy
alessio.merlo@casd.difesa.it

ABSTRACT

Containers are core building blocks for creating applications based on the microservice paradigm. However, assessing their security is complex, especially when deployed in distributed and heterogeneous scenarios. Moreover, developers and IT operators should only focus on integration and delivery processes without dealing with tasks to guarantee securing requirements. To overcome such issues, in this paper, we introduce the ideas at the basis of Project SecCo (Securing Containers), i.e., an architecture for extending and improving current security assessment methodologies into the continuous integration and continuous delivery DevOps pipeline. To this end, SecCo proposes a framework able to orchestrate new automatic security services to prevent and reduce security vulnerabilities in the design, implementation, and deployment phases, and to identify and mitigate, at runtime, attempts to exploit them. The paper also showcases the main research challenges to be addressed for pursuing the vision of SecCo.

CCS CONCEPTS

• **Security and privacy → Virtualization and security**; *Security requirements*; *Distributed systems security*; *Intrusion detection systems*; *Vulnerability scanners.*

KEYWORDS

Container security, DevSecOps, CI/CD security

ACM Reference Format:
Luca Verderame, Luca Caviglione, Roberto Carbone, and Alessio Merlo. 2023. SecCo: Automated Services to Secure Containers in the DevOps Paradigm. In *International Conference on Research in Adaptive and Convergent Systems (RACS '23), August 6–10, 2023, Gdansk, Poland.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3599957.3606222

*Corresponding Author.

RACS '23, August 6–10, 2023, Gdansk, Poland

ACM ISBN 979-8-4007-0228-0/23/08.
https://doi.org/10.1145/3599957.3606222

1 INTRODUCTION

DevOps often refers to a software development paradigm that aims at improving the lifecycle of an application via a strict interaction between developers and IT operators. Such a process can also be used to support Continuous Integration and Continuous Delivery (CI/CD), where the idea is that the application is kept constantly under development and testing, and updated versions are released very often (e.g., daily). As a result, the application development lifecycle becomes a fast-paced, iterative process.

To pursue such a vision, functionalities of a full-featured application are broken down into a set of small, independent pieces of software, commonly defined as microservices [9]. Microservices naturally fit the DevOps model as they can be maintained, improved, and released far more rapidly than full-fledged and monolithic applications. Furthermore, such smaller components can be reused and constitute the standard architectural model for cloud-native applications. Deploying microservices in cloud scenarios (e.g., to reduce costs and enforce scalability properties) mainly requires containers. A container provides a lightweight execution environment and an interface to the "outside world", i.e., to other microservices or external applications. Containers also provide an abstraction layer to ease the migration between different physical servers (e.g., to enforce load balancing) as well as to support several architectural blueprints (e.g., cloud, edge, and mobile) [19].

Unfortunately, current heterogeneous and container-oriented scenarios exacerbate security issues and require a thorough understanding of the overall development process [27]. A large portion of the security space characterizing cloud-based microservice applications tightly depends on the properties of individual containers, including their interactions with the hosting environment [26]. In addition, an increasing number of publicly available containers have relevant security vulnerabilities that can be exploited quite straightforwardly [23], [15], thereby indicating that current security practices are far from being reliable. For example, a recent survey [28] showcased the 51% of ~4 million images hosted in Docker Hub exhibit exploitable vulnerabilities.

In this perspective, current limitations characterizing container security can be mitigated by decoupling the CI/CD phase from the securing one. Specifically, developers and IT operators should only focus on the CI/CD process without dealing with security-oriented tasks. For instance, security operations should be provided by a trusted third-party service that implements the best hardening

strategies to secure the containers hosting the application. Besides, specific security constraints related to the execution environment may entail static and runtime guarantees.

To prevent the CI/CD team from managing container security and to promote the diffusion of cloud-native applications, this work presents the main ideas at the basis of the Project SecCo (Securing Containers). In essence, SecCo aims to design and develop a novel service to secure containers used in microservice-based cloud applications automatically. At the same time, it also allows to truly combine development (Dev), security (Sec), and operations (Ops). To support such a vision, the proposed solution is built around three main modules, each in charge of a specific task, i.e., hardening, compliance verification, and runtime monitoring.

Summing up, the contribution of this paper is twofold: it provides a thorough discussion of the architecture envisioned in SecCo, and it presents the main challenges and research questions to be addressed for partially filling the gap in container security.

The rest of the paper is structured as follows. Section 2 reviews past works assessing and enforcing container security, while Section 3 discusses the reference scenario. Section 4 introduces the architecture proposed in SecCo, and Section 5 presents the main challenges and gaps to be addressed. Lastly, Section 6 concludes the paper and outlines future research directions.

2 RELATED WORK

Owing to the importance of the topic, a vast body of research has started to emerge in the field of container security. However, according to a recent survey [17], the main research trend is to solve hazards by focusing on the development phase. As possible examples, some works take advantage of vulnerability analysis mechanisms [10],[13], while others deploy mitigation strategies based on well-defined patterns or best practices [4],[8]. Another popular approach to improve container security is built around static analysis tools, which can be used to reveal the presence of known CVEs in container images (see, e.g., [5], [6]). Unfortunately, many de-facto standard mechanisms do not consider vulnerabilities of application packages or third-party libraries and have a limited detection rate [14]. In addition, many tools exploiting static analysis techniques only consider a single container image, thereby missing the evaluation of the possible interactions with the container engine or the other microservices (either on local or remote scales).

For the specific case of Docker containers, the work in [16] deals with an evaluation process taking advantage of the information on the relationship of vulnerable software packages and documented issues of Docker images. The outcome of the process can be further exploited to prevent the diffusion of insecure Docker images. Another possible mechanism relies upon the static analysis of Dockerfiles, especially to search for potential vulnerabilities of the used software components and libraries [7].

On the other hand, the literature also proposes frameworks for securing the CI/CD pipelines using the DevSecOps paradigm built around the Docker ecosystem. As an example, the work in [1] introduces a comprehensive solution that integrates existing security analysis tools (e.g., SonarQube) to improve global security properties. Dynamic testing techniques can also be used to further advance in performance. For instance, the work in [25] demonstrates how to

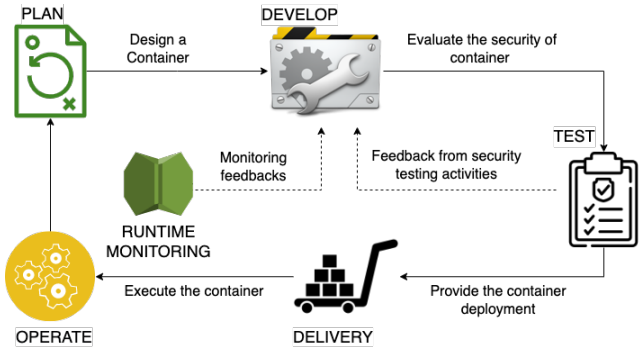

Figure 1: Typical DevSecOps scenario for cloud-native applications.

"orchestrate" three different automated dynamic testing techniques, i.e., Web Application Security Testing, Security API Scanning, and Behaviour Driven Security Testing. Another valuable approach is based on the creation of specific profiles. To this aim, the work in [18] exploits AppArmor to produce profiles to enforce access policies and mitigate risks caused by zero-day vulnerabilities. Access control has also been used in more general settings. For instance, appropriate access control rules are presented in [26], while the work in [24] suggests the use of suitable cryptographic protocols (e.g., TLS and OAuth) to enhance the security of a container ecosystem.

A relevant case to consider to fully assess container security deals with the orchestration phase, significantly improving cloud-native applications. However, most cloud-oriented security platforms still need a comprehensive process to enforce security within the standard operation flow. To this extent, the work in [12] showcases a Secure Container Orchestrator engine exploiting hardware-based trusted execution environment technologies for data protection, i.e., Intel SGX. Instead, for the specific case of Kubernetes, the work in [29] takes again advantage of AppArmor policies to secure cloud-native deployments.

Alas, most efforts dealing with container security mainly aim at reducing (part of) the attack surface or focusing on a specific vulnerability, threat, use case, or subsystem. Consequently, to our knowledge, we advocate for a comprehensive security analysis of the entire container ecosystem, especially from image creation to distribution processes.

3 REFERENCE SCENARIO

A typical DevSecOps scenario considered in Project SecCo is depicted in Figure 1. In more detail, the design of a cloud-native application involves creating a set of containers, each hosting a different component or functionality. The CI/CD team preliminary plans the requirements for each container and starts the development phase. The security properties of a container are tested by considering best practices and guidelines. To this aim, suitable static and dynamic methodologies are used to identify security issues, which should be addressed before the delivery phase. The next step is the deployment of the containers implementing the cloud-native application. This phase can be completed with a runtime monitoring activity,

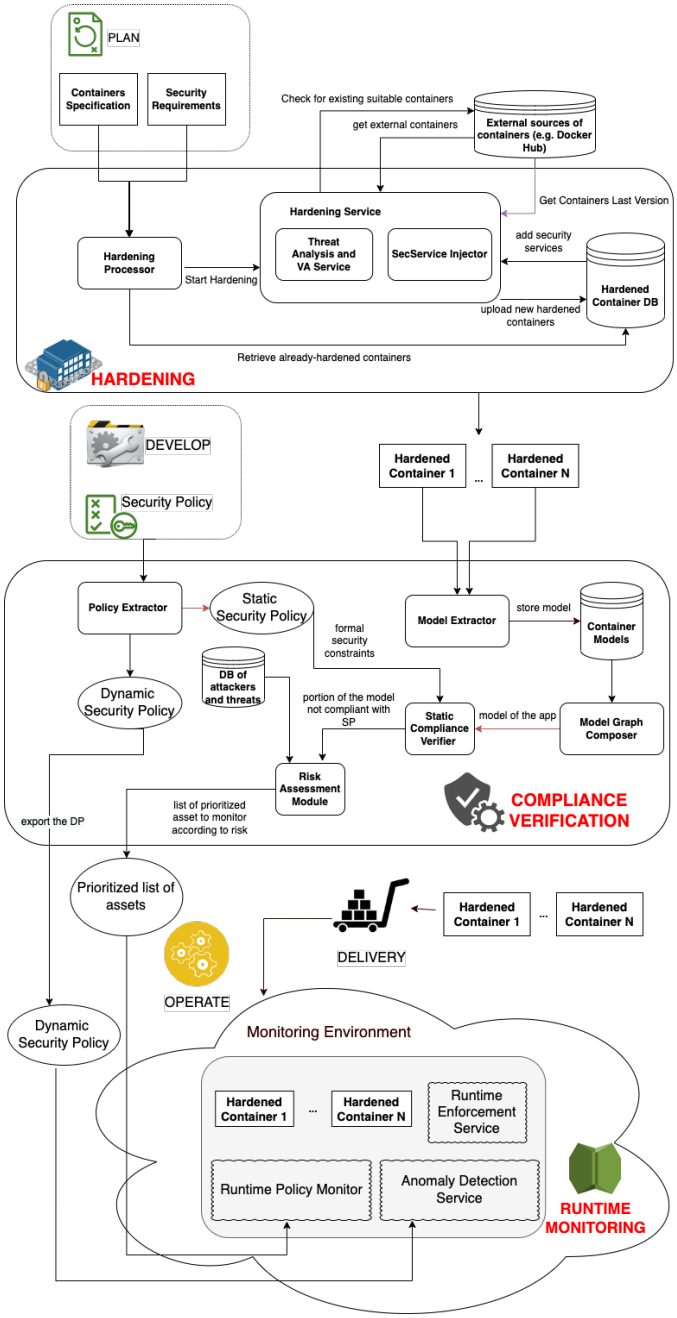

Figure 2: Overall architecture for SecCo and interfaces with the DevOps pipeline.

providing a vast array of feedback information. It is worth noticing that, both test and runtime monitoring phases can further include functional testing activities, e.g., static test cases and adherence to performance indexes. Those feedback information can help the CI/CD team to fix the code or re-organize part of the microservice architecture.

However, automatizing security procedures in production-quality or realistic scenarios requires facing several challenges simultaneously. First, guidelines, requirements, and tools should be clearly outlined and built around some consistency properties. Second, the identification and definition of vulnerabilities could be fragmented and prone to the "viewpoints" of the teams involved in the CI/CD process. For instance, a team could give more value to business support constraints, whereas another could prefer to focus on data.

Third, the security analysis is traditionally a fine-grained procedure, which could be difficult to abstract. Specifically, the security of virtualized environments is primarily enforced on a per-container or per-application basis. As a consequence, when in the presence of large-scale or complex applications, ordinary "per-app" security assessments exhibit severe scalability issues[1]. Moreover, modern applications could require to deploy and orchestrate a vast amount of microservices, which could be also organized in different functional tiers [11].

In the following, we will introduce the functional components envisioned in SecCo to support automatically the pipeline depicted in Figure 1 and to cope with the aforementioned limitations.

4 ARCHITECTURE

This section presents the most relevant architectural components envisaged in SecCo and the main interfaces for interacting with the DevOps pipeline. Figure 2 depicts the overall blueprint. As shown, the main idea is to deploy new services to "automatize" the process of securing containers at the basis of the modern microservice paradigm. In more detail, the SecCo architecture implements the following modules:

- **Hardening**: This module provides a set of containers hardened according to security best practices, CI/CD team requirements, and application-specific constraints.
- **Compliance Verification**: this module supports the CI/CD teams in defining several security requirements (i.e., a security policy) that the single container or the set of containers must comply with at rest and runtime. First, SecCo must verify the compliance of the containers with the set of security requirements that can be evaluated at rest (hereafter, the static security policy). The compliance and verification process must also clearly indicate how to customize/instrument containers to support their runtime evaluation.
- **Runtime Monitoring**: this module provides a monitoring service for the "online" evaluation of security requirements and to reveal unexpected runtime interactions among containers (e.g., using anomaly detection capabilities).

In the following, the three components at the basis of SecCo will be briefly discussed.

4.1 Hardening

The Hardening module (cf. Hardening in Figure 2) supports the CI/CD team in exploiting hardened containers to develop the microservice application.

During the planning phase, the CI/CD team designs the application's architecture and identifies a set of containers that will be used to host the required logic. Furthermore, the team can also define constraints that insist on one or more containers. Examples of constraints could include the supported and verified [3] network protocols, exposed transport ports and services, or the type of third-party libraries installed on the containers.

The module processes the set of requested containers and the security constraints of the CI/CD team to check the availability of already-hardened containers from the Hardened Container database. In this way, the CI/CD team can directly download already-hardened, standard containers to avoid the burden of building a container and securing it from scratch.

Instead, for any new container, the module triggers the Hardening Service. First, the service checks for existing suitable containers from external repositories (e.g., Docker Hub). Then, it performs the threat and vulnerability assessments of the candidate containers to evaluate the primary attack vectors, identify their security hazards/vulnerabilities, and determine the best strategies to patch or mitigate the detected security issues. The patching phase can include the modification of the container specification (e.g., the Dockerfile of a Docker container) and the injection (cf. SecService Injection) of on-demand security services and libraries that can provide the proper confidentiality, authentication, and authorization functionality to the container, without the need for the container to implement them on its own. Examples of security services include a TLS service (e.g., termination proxy), an Identity Provider, and an OAuth module. Finally, the service executes the configuration/customization procedures to release the hardened container.

4.2 Compliance Verification

The Compliance Verification module receives two inputs from the CI/CD team: *i*) the set of hardened containers hosting the developed microservice application, and *ii*) the security policy of the application scenario. Possible examples of security requirements include the use of non-root users, restrictions of network traffic, and limitations in gaining additional permissions at runtime[2]. The module first analyzes the security policy to derive the security requirements to be evaluated at rest (i.e., the Static security Policy - SP) and runtime (Dynamic security Policy - DP).

To verify whether the containers comply with the SP, the Compliance Verification module includes a service to infer the behavioral model of a specific hardened container. The output of this phase is expressed as a model describing all the (at least security-sensitive) interactions of the container among its inner layers, with other containers (either local or remote) and the container engine.

Depending on the SP, the behavioral models of containers can be combined by the Model Graph Composer to obtain a model describing the interactions of a set of containers. Then, the Static Compliance Verifier exploits automated security verification techniques (e.g., model-checking [2]) for verifying the model's compliance concerning the SP. If the verification is not conclusive or if the model is not compliant (either partially or entirely) with the static security policy, the module use risk-based methodologies (cf. Risk Assessment Module in Figure 2) to automatically analyze — when doable statically — the risk exposure of assets (e.g., container or security sensitive operations in the graph) to the set of attackers and threats in container-based applications. The objective is to identify assets exposed to high levels of risk to allow prioritizing the runtime monitoring activities.

[1] https://netflixtechblog.medium.com/scaling-appsec-at-netflix-6a13d7ab6043 [Last Accessed: May 2023]

[2] https://success.myshn.net/Skyhigh_Cloud_Infrastructure_(CNAPP)/CSPM/Configuration_Audit/Configuration_Audit_Templates_and_Policies/Policy_Templates_for_CSPM [Last Accessed: May 2023]

4.3 Runtime Monitoring

Concerning Runtime Monitoring, this module verifies the compliance of dynamic and static security policies. To this aim, it takes advantage of a set of techniques that allow the runtime monitoring of the various container interactions in a non-invasive way while granting the minimum overhead in terms of performance decay of the microservices.

This monitoring phase then aims at the definition of an approach able to recognize and monitor the most relevant security-sensitive interactions of containers, possibly in a transparent way. To this end, techniques like code layering, kernel augmentation, and monitor synthesis based on temporal logic can be adopted. In more detail, the Runtime Monitoring module exploits a set of probes to verify the behavior of the running containers against the DP and the part of SP requiring a runtime evaluation. Also, the module includes an anomaly detection service to evaluate "live" interactions among containers and the underlying environment (e.g., hardware and network components). The monitoring service is also expected to support evaluating corner cases and unexpected runtime interactions among containers, such as collusive attack templates. Finally, the Runtime Monitoring module will include proper mechanisms for automatic security enforcement, i.e., to tame an interaction that violates some dynamic security policies while guaranteeing a minimum impact on the application's functioning.

5 CHALLENGES

To effectively implement all the services and techniques envisaged in SecCo, a wide range of research and engineering challenges must be faced.

The first question to be addressed concerns the definition of a detailed architecture specification, mainly to find functional relationships. The complex blueprint depicted in Figure 2 accounts for the interplay of many services and technologies, which could be difficult to have in real deployments or production-quality scenarios. Hence, precise design and engineering efforts are mandatory.

In parallel, creating the advanced mechanisms envisioned in SecCo accounts for an appropriate amount of research.

First, the design of an automated hardening pipeline requires sound methodologies to ensure the identification of the main security issues of the involved containers and the definition of the appropriate security hardening strategies. In more detail, this phase requires a study of the primary attack vectors of containers and security hazards, explicitly focusing on the interaction among the containers composing cloud-native applications to define a comprehensive security knowledge base. Then, it must involve the design of a vulnerability assessment methodology for containers exploiting novel verification techniques and state-of-the-art approaches to cope with the analysis of the composition of containers and their mutual interactions.

Also, a prominent challenge of the compliance verification phase consists in determining the proper formalisms to model the runtime behavior of containers along with the definition — and automatic identification — of relevant assets to protect, such as sensitive resources and services. Similarly, a suitable specification language shall be designed for expressing security policies on containers and their interactions as well as the corresponding analysis methodologies — following a risk-based approach — to automatically check the compliance of the containers with the policy and, if necessary, prioritize the protection of the considered assets based on risk levels. Consequently, it is paramount to define how to protect the assets previously considered through the runtime monitoring and identification of anomalous behaviors (e.g., interactions among containers and between a container and its host through system calls).

Finally, the creation of the runtime monitoring module needs suitable approaches to monitor the behavior of containers reliably. In more detail, the monitoring mechanism must comply with the strict performance requirements of the DevOps paradigm, i.e., little or negligible overheads on the containerized application should be experienced. At the same time, the monitoring techniques must be sound and complete from a security standpoint, i.e., the monitoring activity must cover all security-sensitive operations that the containers can carry out. Even if runtime monitoring of containers and software environment is an emerging and open problem, a promising approach concerns the extended Berkeley Packet Filter (eBPF) [20]. A successful runtime monitoring service should also be able to check the adherence of a running container against a set of security policies. In parallel, information gathered at runtime can be used to implement anomaly detection functionalities. To this aim, artificial intelligence can be considered a technological enabler to identify whether the container deployment is the target of external attacks or if used to implement collusive or offensive schemes [30].

6 CONCLUSIONS AND FUTURE WORK

In this paper, we presented the founding ideas of the SecCo, which is aimed at providing a complete pipeline for securing containers at the basis of many modern microservice applications. The main goal of our framework is to allow developers and IT operators to only focus on the integration and delivery processes without considering tasks related to cybersecurity aspects. The three main modules envisioned in SecCo, i.e., hardening, compliance verification, and runtime monitoring, can be "composed" to fully support the lifecycle of containerized applications.

Future works aim to perform various research activities, allowing full architecture development. For instance, suitable threat modeling is needed to enable the hardening and configuration of containers. Besides, efficient data gathering techniques [22],[21] are needed to not impact run-time on containers' performances while feeding AI-based models to spot anomalous working conditions or potential attacks.

ACKNOWLEDGMENTS

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

REFERENCES

[1] Manish Kumar Abhishek and D. Rajeswara Rao. 2021. Framework to Secure Docker Containers. In *2021 Fifth World Conference on Smart Trends in Systems Security and Sustainability (WorldS4)*. 152–156.

[2] Alessio Merlo Luca Verderame Alessandro Armando, Gabriele Costa. 2014. Enabling BYOD through secure meta-market. 219–230. https://doi.org/10.1145/2627393.2627410

[3] Roberto Carbone Alessio Merlo Davide Balzarotti Alessandro Armando, Giancarlo Pellegrino. 2012. From model-checking to automated testing of security protocols: Bridging the gap. 7305 LNCS (2012), 3–18. https://doi.org/10.1007/978-3-642-30473-6_3

[4] Nera Besic. 2021. Microservices Security: Challenges and Best Practices. https://brightsec.com/blog/microservices-security/. (2021). Accessed: July 12, 2023.

[5] Nera Besic. 2023. Container Vulnerability Scanning. https://anchore.com/container-vulnerability-scanning/. (2023). Accessed: July 12, 2023.

[6] Clair. 2023. Vulnerability static analysis for containers. https://github.com/quay/clair. (2023). Accessed: July 12, 2023.

[7] Thien-Phuc Doan and Souhwan Jung. 2022. DAVS: Dockerfile Analysis for Container Image Vulnerability Scanning. *CMC-COMPUTERS MATERIALS & CONTINUA* 72, 1 (2022), 1699–1711.

[8] Docker. 2019. Security best practices. https://docs.docker.com/develop/security-best-practices/. (2019). Accessed: July 12, 2023.

[9] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. 2017. Microservices: yesterday, today, and tomorrow. *Present and ulterior software engineering* (2017), 195–216.

[10] Ana Duarte and Nuno Antunes. 2018. An empirical study of docker vulnerabilities and of static code analysis applicability. In *2018 Eighth Latin-American Symposium on Dependable Computing (LADC)*. IEEE, 27–36.

[11] Christof Ebert and Lorin Hochstein. 2023. DevOps in Practice. *IEEE Software* 40, 1 (2023), 29–36. https://doi.org/10.1109/MS.2022.3213285

[12] Gabriel P. Fernandez and Andrey Brito. 2019. 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, 138–145. https://doi.org/10.1145/3297280.3297296

[13] Olivier Flauzac, Fabien Mauhourat, and Florent Nolot. 2020. A review of native container security for running applications. *Procedia Computer Science* 175 (2020), 157–164.

[14] Omar Javed and Salman Toor. 2021. An evaluation of container security vulnerability detection tools. In *Proceedings of the 2021 5th International Conference on Cloud and Big Data Computing*. 95–101.

[15] Bhupinder Kaur, Mathieu Dugré, Aiman Hanna, and Tristan Glatard. 2021. An analysis of security vulnerabilities in container images for scientific data analysis. *GigaScience* 10, 6 (2021), giab025.

[16] Soonhong Kwon and Jong-Hyouk Lee. 2020. DIVDS: Docker Image Vulnerability Diagnostic System. *IEEE Access* 8 (2020), 42666–42673. https://doi.org/10.1109/ACCESS.2020.2976874

[17] Tiina Leppänen, Anne Honkaranta, and Andrei Costin. 2022. Trends for the DevOps Security. A Systematic Literature Review. In *International Symposium on Business Modeling and Software Design*. Springer, 200–217.

[18] Fotis Loukidis-Andreou, Ioannis Giannakopoulos, Katerina Doka, and Nectarios Koziris. 2018. Docker-sec: A fully automated container security enhancement mechanism. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 1561–1564.

[19] Lele Ma, Shanhe Yi, and Qun Li. 2017. Efficient service handoff across edge servers via docker container migration. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. 1–13.

[20] Sebastiano Miano, Fulvio Risso, Mauricio Vásquez Bernal, Matteo Bertrone, and Yunsong Lu. 2021. A framework for eBPF-based network functions in an era of microservices. *IEEE Transactions on Network and Service Management* 18, 1 (2021), 133–151.

[21] M. Migliardi and A. Merlo. 2011. Modeling the energy consumption of distributed IDS: A step towards Green security. 1452–1457.

[22] M. Migliardi and A. Merlo. 2013. Improving energy efficiency in distributed intrusion detection systems. 19, 3 (2013), 251–264. https://doi.org/10.3233/JHS-130476

[23] Rene Millman. 2020. Most Docker container images have critical flaws. https://www.itpro.com/development/containers/357984/most-docker-container-images-have-critical-flaws. (2020). Accessed: July 12, 2023.

[24] Caterina Munoz, Francisco Montoto, Francisco Cifuentes, and Javier Bustos-Jiménez. 2017. Building a threshold cryptographic distributed HSM with docker containers. In *2017 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON)*. IEEE, 1–5.

[25] Thorsten Rangnau, Remco v Buijtenen, Frank Fransen, and Fatih Turkmen. 2020. 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, 145–154.

[26] Sari Sultan, Imtiaz Ahmad, and Tassos Dimitriou. 2019. Container security: Issues, challenges, and the road ahead. *IEEE access* 7 (2019), 52976–52996.

[27] Junzo Watada, Arunava Roy, Ruturaj Kadikar, Hoang Pham, and Bing Xu. 2019. Emerging trends, techniques and open issues of containerization: a review. *IEEE Access* 7 (2019), 152443–152472.

[28] Ann Yi Wong, Eyasu Getahun Chekole, Martín Ochoa, and Jianying Zhou. 2021. Threat modeling and security analysis of containers: A survey. *arXiv preprint arXiv:2111.11475* (2021).

[29] Hui Zhu and Christian Gehrmann. 2022. Kub-Sec, an automatic Kubernetes cluster AppArmor profile generation engine. In *2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS)*. IEEE, 129–137.

[30] Marco Zuppelli, Matteo Repetto, Andreas Schaffhauser, Wojciech Mazurczyk, and Luca Caviglione. 2022. Code Layering for the Detection of Network Covert Channels in Agentless Systems. *IEEE Transactions on Network and Service Management* 19, 3 (2022), 2282–2294.