SecCO-OC: Securing microservice-based apps

Valentina Casola*, Vincenzo Riccio†, Giuseppe Tricomi‡, Giovanni Merlino‡,
Pietro Di Gianantonio†, Bruno Crispo‖ Massimiliano Rak¶, Antonio Puliafito‡,
† Department of Mathematics, Computer Science and Physics, Università di Udine, Udine, Italy
* Department of Electrical Engineering and Information Technologies, University of Naples Federico II, Naples, Italy
‖ Department of Information Engineering and Computer Science - Università di Trento, Trento, Italy
¶ Department of Engineering, University of Campania Luigi Vanvitelli, Aversa, Italy
‡ Department of Engineering - *Università di Messina,* Messina, Italy

***Abstract*—Containers are essential elements for developing applications based on the microservices paradigm. Evaluating their security is a complex challenge, especially in distributed and heterogeneous contexts. Additionally, developers should focus exclusively on integration and deployment processes, not caring about the security of microservices/containers produced for the application, platform, or framework where it is implemented.**

Smart Cities are environments belonging to computing continuum that enables the distribution of computation among resources spanning from Cloud to Edge Resources. The microservice-based applications distributed on Smart Cities present further security breaches due to their distributed nature. The SecCo-OC project aims to create an architecture capable of integrating within the CI/CD process, typical of the DevOps development paradigm, a series of workflows dedicated to the security of the container and the developed microservice. This involves static and dynamic threat analysis, their resolution, also through runtime enforcement, and the publication of the secure container in a dedicated repository.

The architecture of SecCo-OC is designed to meet the requirements of scalability, flexibility, and reliability. This is achieved by adopting Cloud/Edge and I/O cloud-based solutions, enabling the extension of the DevOps paradigm to security even in environments related to pervasive computing.

***Index Terms*—Container Security; CI/CD; Edge Computing;**

I. Introduction and Related Works

In the context of Smart Cities, multiple scenarios such as remote monitoring of critical infrastructures, emergency responses, traffic management, and urban planning, benefit from moving functionality towards the edges. Container technology supports this by developing container-orchestration systems specifically for edge devices (e.g., microk8s and k3s). These benefits are even more emphasized when applications running on containers are developed as a combination of microservices. Scalability, portability, and fault tolerance are just a few of the attributes that can be better exploited by running microservices at the edge. However, ensuring security is more challenging due to the lack of a single enforcement point, the distributed data and computations, and the weaker isolation properties of containers compared to traditional virtual machines [1]. In this work, we introduce the methodology and tools developed for securing containers as part of the SecCo-OC project. We also demonstrate how it will be applied to a smart city case study, i.e., an Urban Intelligent application. One of the main challenges we encountered consists in integrating our architecture within the CI/CD paradigm, in alignment with the security requirements and best practices.

According to McGraw [2], security should be addressed early at the requirements level. Software requirements should include both functional and security requisites. Additionally, McGraw suggests using abuse cases to describe the system's behavior under attack and to cover the security space. Russell et al. [3] identify most software security problems in the use of unsophisticated development techniques, the lack of security-focused quality assurance, and limited training for software developers and project managers. In [4]–[6] authors are proposing secdevops methodologies and a model that outlines all the security activities needed during software development [6]. In this context, are defined the enabling methodologies and tools to build a Hardening module capable of outputting a set of secured containers—ready to host a microservice application—that comply with security best practices and the requirements of the CI/CD team. The hardening process includes activities, such as threat modeling and vulnerability assessment (VA), applied to the case of containers [6]. Although the container security research is an hot topic of research, there is a notable absence of behavioral models and dedicated languages for specifying container security policies. Temporal logics like LTL and modal μ-calculus [7] are used for process behavior, supporting static model-checking [8] and dynamic monitoring [9]. Satoh and Tokuda [10] proposed a mechanism for composing security policies in Service-oriented Architectures, but its applicability to the less-structured container context is unclear. Burco et al. [11] introduced a graph-based model for containers using Bigraphical Reactive Systems (BRSs), enabling the formalization and verification of container properties through graph theory techniques. An open research challenge is designing solutions that simultaneously guarantee enforcement of runtime policies (e.g., control flow integrity) and acceptable overheads to run on a wide range of edge devices [12], [13].

II. A methodology to design and test container-based applications

We have defined a general enabling methodology and a suite of tools to secure containers in compliance with security best practices and the requirements of the CI/CD team. The hardening process for containers includes various activities

Threat	Description
Injection Flaws	Injection Flaws occur when data not validated are sent as part of a command or query to their interpreter. The data can deceive the interpreter running commands not provided or accessing data for which you have no authorization.
Online Guessing	An attacker may try to guess valid username/password combinations
Unauthorized Entry	An adversary can entry to a server or account without authorization
Spoofing External IPv6	An attacker in a container can craft IPv6 router advertisements and consequently, spoof external IPv6 hosts, obtain sensitive information or cause a denial of service
Container Escape	An attacker can perform full container escape, gaining control over the host system

TABLE I
EXAMPLE OF THREATS FROM THE CATALOGUE

[4], such as threat modeling, risk evaluation, security control identification, and both static and dynamic security testing techniques, including VA and penetration testing. Figure 1 illustrates the main steps of the adopted methodology, which are mapped onto the typical stages of the DevOps process from planning to testing.

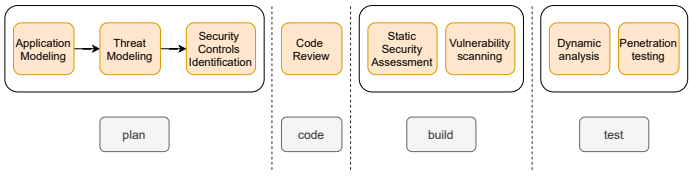

Fig. 1. Main stages of the SecDevOps methodology for Containers.

A catalogue of container-specific threats has been identified, leading the pipeline to support the hardening of general-purpose containers to meet the technical and security features of the application provided by the CI/CD team and security best practices. Table I lists some threats specific to the Smart Cities case study along with their descriptions.

This will applied to the initial architecture of the SecCo case study, represented by an Urban Intelligent Application, as represented in Figure 2.

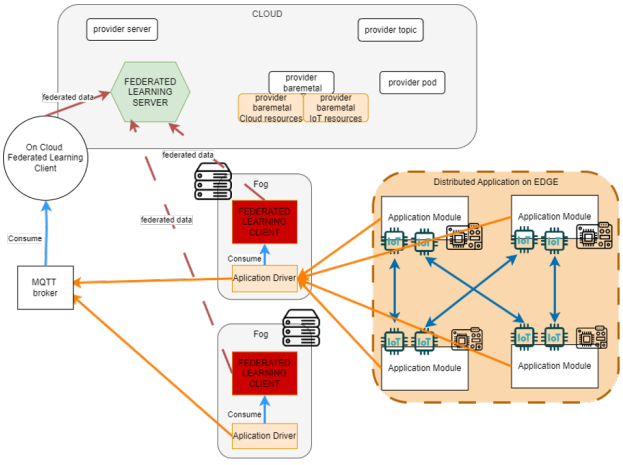

Fig. 2. An Urban Intelligence application Architecture

The architecture includes *vehicles Edge nodes*, which execute a Microservices Application, and *Smart City ITS nodes* with 3rd party Application Collectors or cooperative applications that gather data from vehicles. Applications running on *Edge Devices* include microservices for controlling vehicle routes and predictive maintenance. A core application on Fog, the *Urban mobility application*, gathers information from edge nodes (microservices injected on Edge to collect data and receive information from the Smart City) for the preemptive identification of congestion and shaping vehicular flow to minimize emissions in areas with high pollution levels (via route advices).

III. ACKNOWLEDGMENTS

This work is partially supported by the Project SecCo-OC CUP N. D33C22001300002 PNRR M4 C2 I1.3 “SEcurity and RIghts in the CyberSpace (SERICS)” PE0000014 PE7 funded by Next-Generation EU.

REFERENCES

[1] S. Sultan, I. Ahmad, and T. Dimitriou, “Container security: Issues, challenges, and the road ahead,” *IEEE access*, vol. 7, pp. 52 976–52 996, 2019.

[2] G. Mcgraw, “Software security,” *IEEE Security amp; Privacy Magazine*, vol. 2, no. 2, p. 80–83, Mar. 2004. [Online]. Available: http://dx.doi.org/10.1109/MSECP.2004.1281254

[3] R. L. Jones and A. Rastogi, “Secure coding: Building security into the software development life cycle,” *Information Systems Security*, p. 29–39, Nov. 2004. [Online]. Available: http://dx.doi.org/10.1201/1086/44797.13.5.20041101/84907.5

[4] V. Casola, A. De Benedictis, M. Rak, and U. Villano, “A novel security-by-design methodology: Modeling and assessing security by slas with a quantitative approach,” *Journal of Systems and Software*, vol. 163, p. 110537, 2020.

[5] V. Casola, A. De Benedictis, M. Rak, and G. Salzillo, “A cloud secdevops methodology: from design to testing,” in *Quality of Information and Communications Technology QUATIC 2020, Faro, Portugal, September 9–11, 2020*. Springer, 2020, pp. 317–331.

[6] D. Granata, M. Rak, and G. Salzillo, “Metasend: a security enabled development life cycle meta-model,” in *Proceedings of the 17th International Conference on Availability, Reliability and Security*, 2022, pp. 1–10.

[7] J. Bradfield and C. Stirling, “12 modal mu-calculi,” *Studies in logic and practical reasoning*, vol. 3, pp. 721–756, 2007.

[8] C. Stirling and D. Walker, “Local model checking in the modal mu-calculus,” *Theoretical Computer Science*, vol. 89, no. 1, pp. 161–177, 1991.

[9] F. Martinell and I. Matteucci, “Through modeling to synthesis of security automata,” *Electronic Notes in Theoretical Computer Science*, vol. 179, pp. 31–46, 2007.

[10] F. Satoh and T. Tokuda, “Security policy composition for composite web services,” *IEEE Transactions on Services Computing*, vol. 4, no. 4, pp. 314–327, 2010.

[11] F. Burco, M. Miculan, and M. Peressotti, “Towards a formal model for composable container systems,” in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 2020, pp. 173–175.

[12] M. Grisafi, M. Ammar, M. Roveri, and B. Crispo, “{PISTIS}: Trusted computing architecture for low-end embedded systems,” in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 3843–3860.

[13] L. Degani, M. Salehi, F. Martinelli, and B. Crispo, “μ ips: Software-based intrusion prevention for bare-metal embedded systems,” in *European Symposium on Research in Computer Security*. Springer, 2023, pp. 311–331.