CONSERVE: A framework for the selection of techniques for monitoring containers security

The Journal of Systems & Software 186 (2022) 111158

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

CONSERVE: A framework for the selection of techniques for monitoring containers security☆

Rodi Jolak [a,b,c,*], Thomas Rosenstatter [b,d], Mazen Mohamad [a,b], Kim Strandberg [b,c], Behrooz Sangchoolie [d], Nasser Nowdehi [c], Riccardo Scandariato [e]

[a] University of Gothenburg, Sweden
[b] Chalmers University of Technology, Sweden
[c] Volvo Car Corporation, Sweden
[d] RISE Research Institutes of Sweden, Sweden
[e] Hamburg University of Technology, Germany

ARTICLE INFO

Article history:
Received 5 March 2021
Received in revised form 21 November 2021
Accepted 23 November 2021
Available online 18 December 2021

Keywords:
Software and systems engineering
Virtualization
Security
Container monitoring
Intrusion detection
Attack analysis

ABSTRACT

Context: Container-based virtualization is gaining popularity in different domains, as it supports continuous development and improves the efficiency and reliability of run-time environments.
Problem: Different techniques are proposed for monitoring the security of containers. However, there are no guidelines supporting the selection of suitable techniques for the tasks at hand.
Objective: We aim to support the selection and design of techniques for monitoring container-based virtualization environments.
Approach : First, we review the literature and identify techniques for monitoring containerized environments. Second, we classify these techniques according to a set of categories, such as technical characteristic, applicability, effectiveness, and evaluation. We further detail the pros and cons that are associated with each of the identified techniques.
Result: As a result, we present CONSERVE, a multi-dimensional decision support framework for an informed and optimal selection of a suitable set of container monitoring techniques to be implemented in different application domains.
Evaluation: A mix of eighteen researchers and practitioners evaluated the ease of use, understandability, usefulness, efficiency, applicability, and completeness of the framework. The evaluation shows a high level of interest, and points out to potential benefits.

1. Introduction

Agility, flexibility, and ability to rapidly evolve are crucial for organizations to fulfill changes in the requirements of customers and markets. Accordingly, organizations are widely shifting towards the adoption of DevOps as well as continuous integration and deployment practices (Rodríguez et al., 2017). However, the adoption of these practices for the development of embedded systems is a challenging endeavor, since these systems are increasingly complex and intensively depend on hardware, sophisticated electronics, communication infrastructures, and real-time capabilities (Lwakatare et al., 2016).

Virtualization enables and simplifies continuous software development and deployment on virtual hardware, applications, or operative systems. Container-based virtualization is a software technology that enables software applications to run in virtual run-time environments on a single operating system. In cloud computing, container technology is emerging as an important part of the cloud computing infrastructure and is used in well-known public cloud platforms, such as Google and IBM/Softlayer (Bernstein, 2014). Furthermore, container-based virtualization is considered to have a great potential for significantly advancing Platform-as-a-Service technology towards distributed heterogeneous clouds through lightweightness and interoperability (Pahl, 2015).

Although the benefits of the container technology have been widely acknowledged in cloud computing; this technology is gaining popularity in different domains because of the advantages that it brings to the consistency of the software development and deployment process (Merkel, 2014), as well as to the efficiency, scalability, and reliability of the run-time virtual environment (Soltesz et al., 2007).

☆ Editor: Gabriele Bavota.
* Corresponding author at: University of Gothenburg, Sweden.
E-mail addresses: rodi.jolak@cse.gu.se (R. Jolak), thomas.rosenstatter@ri.se (T. Rosenstatter), mazen.mohamad@cse.gu.se (M. Mohamad), kim.strandberg@chalmers.se (K. Strandberg), behrooz.sangchoolie@ri.se (B. Sangchoolie), nasser.nowdehi@volvocars.com (N. Nowdehi), riccardo.scandariato@tuhh.de (R. Scandariato).

https://doi.org/10.1016/j.jss.2021.111158

For example, in the Internet of Things (IoT) domain there is a growing trend towards using the container technology, since it can be used on devices characterized by fewer computational resources to efficiently execute complementary software (Morabito, 2017).

In the automotive domain, modern vehicles are equipped with sophisticated software applications to serve different purposes, such as supporting autonomous driving and vehicle-to-everything (V2X) communication. The increasing complexity of automotive systems, and the proliferation of software functionalities that they provide, demands the adoption of more efficient solutions for continuous development and deployment in the future. Indeed, container-based virtualization has shown to be effective in several ways. For instance, Berger et al. (2017) use a containerized software development and deployment approach for self-driving vehicles. They show the effectiveness of the approach in enabling a continuous deployment of software components as well as ensuring traceability between software sources and binaries. Moreover, Morabito et al. (2017) indicate that container-based virtualization is an efficient solution that offers high flexibility in the management of the processes running on the On Board Unit (OBU), and allows overcoming the complex software updating procedures required by OBUs.

Motivation. While the use of container-based virtualization brings many advantages including the facilitation of cloud-based deployment and networking, there still exist concerns about the security and safety of this technology (Bernstein, 2014; Combe et al., 2016; Chandramouli and Chandramouli, 2017; Martin et al., 2018). Furthermore, the ever increasing connectivity between systems paves the road for more security attacks. Thus, sophisticated attack analysis techniques for protecting these connected systems are required.

Different techniques for monitoring containers security are proposed. However, there are no guidelines for supporting the selection of suitable techniques for the tasks at hand. The lack of such guidelines might thus influence the decisions of architects and developers which, in turn, might unintentionally undermine the overall security and safety of the developed systems.

Contribution. To deal with the aforementioned issues, we address the following research questions:

- **RQ.1** *What techniques are available for monitoring container-based virtualization environments?*
  First, we review the literature to identify relevant techniques for monitoring container-based virtualization environments with the goal to provide a comprehensive overview of these techniques. In particular, we review 99 studies and identify 15 monitoring techniques. The results are reported in Section 3. Second, we ensure the reliability of the review by performing a quality control on 15% of the data (15 studies).
- **RQ.2** *How can we support the selection of these monitoring techniques?*
  First, we categorize the identified techniques to help developers understand their purpose, technical characteristics, applicability, and effectiveness. Yet, combining these techniques can be beneficial to achieve multiple monitoring-layers for securing containers.
  Second, we further elaborate on the *trade-offs* (i.e., pros and cons) that are associated with each of the techniques, e.g., with respect to efficiency, accuracy, and other qualities. As a result, we present CONSERVE in Section 4, a framework for supporting the selection of techniques for monitoring container-based virtualization environments in different application domains, such as cloud computing, cyber–physical, and automotive.

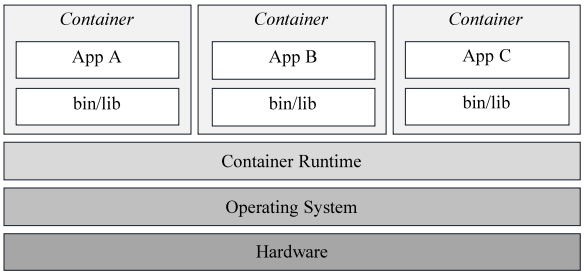


Fig. 1. Architecture of container-based environments (Huang and Wu, 2018).

- **RQ.3** *What is the perception of researchers and practitioners on the understandability, usefulness, efficiency, applicability, and completeness of the framework?*
  We plan and conduct an evaluation of the CONSERVE framework involving a mix of eighteen researchers and practitioners from both academia and industry. We report the results of the evaluation in Section 5.

In summary, we provide a multi-dimensional multi-domain decision support framework that is built based on an comprehensive literature analysis. This framework helps developers in having the required knowledge about container monitoring techniques that are applicable to their problem. Ultimately, it leads developers to the informed and optimal selection of a suitable set of techniques to be implemented for monitoring container-based virtualization environments.

2. Background

Virtualization techniques are beneficial in several aspects, such as the ability to optimize the use of resources, separation by design (i.e., sandboxing), migration to other hardware, and increased monitoring capabilities (Huang and Wu, 2018). Hence virtualization techniques can be used to improve the resilience of systems. However, by adding complexity in the system architecture, the attack surface increases and, thus, it is crucial that applications are secured.

There are various methods for running applications securely and isolated, such as (i) container-based virtualization, (ii) hypervisor-based virtualization and (iii) the use of trusted execution environments (TEEs).

In container-based virtualization, also called application virtualization, the host operating system is shared between the applications. Fig. 1 illustrates the basic structure of containerized environments. The container runtime, e.g., Docker (2020), is responsible for deploying the applications into containers and guarantees that the containers run in isolation. The runtime therefore ensures that the operating system provides means for (i) namespace isolation, to control the resources a container has access to; (ii) resource allocation, to control the consumption of resources, e.g., memory; and (iii) file system virtualization, to efficiently use physical storage across several containers. In addition to the application itself, a container may also include specific libraries (Souppaya et al., 2017; Huang and Wu, 2018).

Early work already showed that container-based virtualization is more resource efficient and scales better than hypervisor-based solutions, which virtualize an entire system including its operating system (Soltesz et al., 2007). Huang and Wu (2018) provide an overview of the classification and aspects of virtualization. Two types of hypervisor-based virtualization exist, namely Type 1 and Type 2 hypervisors. The former hypervisor, also called bare-metal, runs directly on the physical hardware whereas the latter runs as

application on top of the operating system. Huang and Wu further compare hypervisor-based virtualization to container-based virtualization and identify four advantages of container-based virtualization: *faster deployment*, containers start in seconds whereas hypervisor-based solutions are slower since they need to launch the VM with its own kernel; *less resource requirement*, containers commonly share the operating system; *flexibility*, freezing and resuming containers requires less resources compared to virtual machine states of hypervisor-based virtualization ; and *forensics*, the state of containers can be more easily accessed and monitored by the host system (Huang and Wu, 2018).

Approaches to combine the benefits known from hypervisor-based solutions, i.e., isolation and separation of applications, with containers led to so-called *lightweight virtualization* or *micro VMs*. These solutions use a hypervisor such as KVM/QEMU, deploy each container in a separate VM with its own lightweight kernel and use an optimized Virtual Machine Monitor (VMM). Kata Containers (Randazzo and Tinnirello, 2019) and Firecracker (Agache et al., 2020) are representatives for this type of virtualization and are compliant to the specifications defined by the Open Container Initiative (OCI) (OCI, 2021).

Trusted execution environments (TEEs) are designed to provide a secure environment for applications to perform security relevant computations that in most cases require accessing cryptographic key material through, e.g., data authentication, device/system authentication, and encryption/decryption. These trusted applications are typically not standalone; they execute requests from the normal execution environment via a secure communication channel, i.e., secure monitor. Two commercially used TEE solutions are ARM TrustZone and Intel Software Guard Extensions (SGX). Pinto and Santos (2019) highlight that the code base of trusted applications needs to be kept lean to avoid complex applications that are potentially more prone to code vulnerabilities. Since, there is only one TEE per device, a vulnerability in one trusted application can affect the security of another.

Attempts to combine virtualization techniques with TEEs have been made. For instance, Li et al. (2019) propose to isolate trusted applications using virtualization. A thin hypervisor (TEE-visor) running in the TEE executes multiple virtualized TEE instances (vTEEs) which are better separated from each other. SCONE (Arnautov et al., 2016) utilizes a TEE, namely Intel SGX, to increase the isolation of (Docker, 2020) containers. These research efforts (Li et al., 2019; Arnautov et al., 2016) highlight the potential of combining TEEs with virtualization techniques, although more research needs to be conducted to explore how TEEs and container monitoring can be combined.

2.1. Related work

NIST provides guidelines for the secure use of container-based virtualization in SP 800-190 (Souppaya et al., 2017). These guidelines cover security aspects for the entire software life cycle. Overall, NIST SP 800-190 results in six recommendations for securing containers, among others the advice of using OSs that have been specifically designed for the use with containers in order to minimize the attack surface; grouping of containers with the same properties on a single OS kernel; and the use of container-aware runtime defense mechanisms. The document further details the major risks for each container component, namely the container image, registry, orchestrator, container and host OS; and identifies corresponding countermeasures described on a system level. With the CONSERVE framework we support the NIST recommendation to *use container-aware runtime defense tools*.

Casalicchio and Iannucci (2020) present an overview of commonly used container technologies grouped in container type, container manager and orchestration framework. Performance, orchestration and cyber-security are identified as the main challenges in containerization, where cyber-security is further categorized in isolation, encryption of image layers, and network security. The authors describe for each sub-category relevant literature and conclude that the reviewed works focus mainly on improvements in the container isolation, encryption of images at rest and run-time as well as solutions making use of Intel SGX.

Bélair et al. (2019) set the focus of their survey on kernel security mechanisms for improving container security and propose a taxonomy based on how the data to enforce security is transmitted, i.e., configuration-based, code-based and rule-based, where solutions belonging to the latter two allow the container to require certain security demands. Moreover, the authors evaluate the seven reviewed solutions in terms of (i) the granularity one is able to define the policies; (ii) the level of customization; (iii) the need for software modifications for enforcing policies; (iv) usability in real life scenarios; and (5) the extent to which security concerns can be addressed.

Another review of security solutions for containers is provided by Sultan et al. (2019). The authors focus on security issues and challenges; and review existing security solutions and map them to four defined use cases: (i) inter-container protection; (ii) protection of the host from their containers; (iii) protection of the containers from the applications in it; and (iv) protection of the containers from the host. Additionally, each study is also linked to the major risks for containers identified in NIST SP 800-190.

In comparison to the existing work focusing on reviewing security measures for containerized environments, with CONSERVE we provide a framework for categorizing and choosing suitable secure monitoring techniques for containers allowing designers to make an informed decision for selecting techniques for the task at hand. Moreover, we provide a detailed analysis of each technique including a trade-off analysis (i.e., pros and cons).

3. Approach

To create the CONSERVE framework, we employ the *design science* research methodology (Wieringa, 2014). Design science is an iterative process that involves problem space exploration, solution implementation, and solution evaluation. We first conduct a literature review to get a comprehensive understanding of the characteristics of container monitoring techniques. These characteristics are then used to create a multi-dimensional framework with a goal to support developers in making design decisions and ultimately selecting the most appropriate monitoring techniques for the task at hand.

In this section, we first provide details on the literature review and snowballing approaches that we used to identify techniques for monitoring containerized environments (Sections 3.1 and 3.2). After that, we describe how we create the CONSERVE framework based on the studies that are identified via the literature review (Section 3.3). Fig. 2 provides an illustration of the approach which is further detailed in the following subsections.

3.1. Literature review

First, we perform a literature review using the *Scopus* database. Scopus is a well curated database covering more than 77 million records making it one of largest databases of abstracts and citations (Elsevier, 2021). We first use *Scopus* and collect a set of 99 studies in December 2020 using the following search terms: *monitor(ing), security, software or Linux, and container or Docker*.

To check whether or not the studies are relevant for this work, we perform a screening process by reading the title, abstract, and

Table 1
Considered studies (Monitoring Techniques MTs) in the CONSERVE framework.

ID	Reference	Publication	Publication Venue
MT1	De Benedictis and Lioy (2019)	Journal	Future Generation Computer Systems (FGCS)
MT2	Lei et al. (2017)	Conference	Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)
MT3	Khalimov et al. (2019)	Conference	Utility and Cloud Computing (UCC)
MT4	Mattetti et al. (2015)	Conference	Communication and Network Security (ICCNS)
MT5	Chen et al. (2019)	Conference	Design, Automation, and Test in Europe Conference (DATE)
MT6	Zou et al. (2019)	Journal	IEEE Transactions on Cloud Computing (TCC)
MT7	Du et al. (2018)	Conference	Algorithms and Architectures for Parallel Processing (ICA3PP)
MT8	Fourati et al. (2019)	Conference	Parallel and Distributed Computing, Applications and Technologies (PDCAT)
MT9	Gantikow et al. (2020)	Conference	Parallel, Distributed and Network-Based Processing (PDP)
MT10	Abed et al. (2015)	Workshop	Security and Trust Management (STM)
MT11	Srinivasan et al. (2018)	Conference	Security in Computing and Communication (SSCC)
MT12	Sayed and Azab (2019)	Conference	Information Technology, Electronics, and Mobile Communication (IEMCON)
MT13	Abed et al. (2020)	Journal	Communication Networks and Distributed Systems
MT14	Gantikow et al. (2019)	Conference	Cloud Computing and Services Science (CLOSER)
MT15	Kamthania (2019)	Conference	Electrical and Computer Engineering (WIECON-ECE)

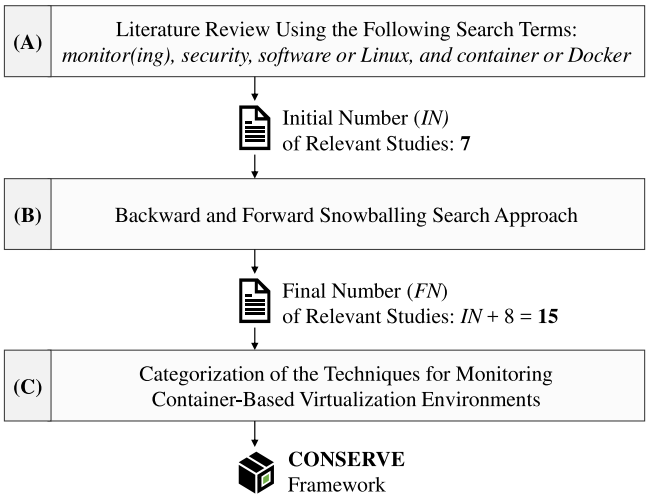


Fig. 2. Approach.

keywords of the studies. During the screening process, we include studies that present techniques or approaches for monitoring the security of containerized environments. We exclude studies that:

1. are not written in English,
2. describe high-level recommendations for securing containers, without providing a comprehensive description of a technique that can be applied in practice , and
3. do not report an evaluation of the proposed techniques.

In total, we find 7 studies following the literature review approach. These studies are included in Table 1.

3.2. Snowballing approach

To complement our review and cover more relevant work, we use the snowballing approach (Wohlin, 2014). This search approach essentially involves repeating the screening process on the papers in the reference list (i.e., backward snowballing) and papers that cite the study under inspection (i.e., forward snowballing). *Google Scholar* is used to perform this inspection to avoid publisher bias (e.g. searching in one publisher's database) (Wohlin, 2014). As a result, we identify additional 8 relevant studies which ultimately increases the number of studies considered in this work to 15 studies (see Table 1). These studies are ordered by the publication date; from September 2015 to March 2020. Out of the fifteen identified studies, eleven are conference publications, three are journal publications, and one study is a workshop publication.

3.3. Categorization of the techniques

The aim of this step is categorize the monitoring techniques to provide a decision-support framework for an informed selection of the techniques.

After collecting the relevant studies, two authors of this manuscript discussed different aspects that can be used for categorizing the monitoring techniques. The aim of this discussion was to prepare and plan for the process of data extraction. The discussed aspects are the following:

- What are the detection and analysis strategies adopted by the monitoring technique? What are the required input, activities, and outcome of the analysis?
- Which domains are the monitoring techniques applicable to?
- What is the purpose of the monitoring techniques and what are the consequences?
- How is the technique evaluated and what are the evaluation results?

To systematically extract relevant details for the categorization of the techniques, we read the relevant papers identified by the literature review and snowballing search. By doing so, different details and aspects have emerged and noted. These details are then organized in themes by conducting a thematic analysis at the explicit level (Boyatzis, 1998). As a result, high-level categories and sub-categories are distinguished. The results are shown in Table 2. Four main categories have emerged: Technical Characteristics, Applicability, Effect, and Evaluation. Each one of these categories include a number of sub-categories providing specific details of the monitoring techniques. Based on these categories, we made the following design decisions for building of the CONSERVE framework (presented in Section 4):

- Technical Characteristics and Applicability, including their corresponding sub-categories, will be used for supporting the selection process and assisting the identification of candidate monitoring techniques applicable to the problem or task at hand.
- Effect and Evaluation, including their corresponding sub-categories, will be used for supporting an objective decision-making process for an optimal selection based on comparing the performance and consequences of applying the monitoring techniques.

In the following, we provide details on the emerged categories and their corresponding sub-categories.

Table 2
Categorization of the techniques for monitoring the security of containerized environments.

Category	Sub-Category	Description or Example
Technical Characteristics	Detection Strategy	Misuse-based, Anomaly-based.
	Monitored Object	Host OS and/or Containerized Application.
	Intrusiveness	External, Internal, or Both.
	Required Resources	Required Resources for analysis: Software, Hardware, or Both.
	Analysis Strategy	Remote Attestation, Filtering and Introspection, Rule-based, or ML-based.
	Analysis Time	Real-time or Forensically.
	Analysis Input	E.g., System Calls, An Application, Sensor or Traffic Data, or Configuration Files.
	Collected Measurements	E.g., Interactions with the OS, as well as Resource Usage (CPU, Memory, Network).
	Analysis Procedure	Detailed Analysis Steps
	Response/Reaction	E.g., Logging, Alarm Activation, or Attack Mitigation.
Applicability	Domain	General-Purpose, Cloud Computing, Cyber–Physical, High Performance Computing, IoT.
	Software Type	E.g., Linux, Docker, Kubernetes.
Effect	Targeted Threats	STRIDE: spoofing, tampering, repudiation, information disclosure, DoS, elevation of privilege.
	Targeted Attacks	E.g., Malware, Unauthorized Access, Kernel Exploit, Binaries Modification.
	Targeted Faults	E.g., Memory Leak, Log Explosion, and Network Latency.
	Pros	E.g., Enhanced Verification.
	Cons	E.g., Inaccurate Prediction of Anomalies.
	Side Effect	E.g., Impact on Performance and CPU Utilization.
Evaluation	Type	Case Study, Experimental study, or testing study.
	Approach	Detailed Evaluation Approach
	Metrics	Evaluation Metrics, e.g., Performance, Detection Rate, Precision, Recall.
	Result	Detailed Evaluation Results

3.3.1. Technical characteristics

What are the detection and analysis strategies adopted by the monitoring technique? What are the required input, activities, and outcome of the analysis? This category includes a number of sub-categories:

- *Detection strategy*: To distinguish between two different strategies for intrusion detection; misuse-based vs. anomaly based detection strategy.
- *Monitored object*: The object in the container-based virtualization environment that is monitored by the technique. This can be the host operating system (OS), container engine, and/or the containerized applications.
- *Intrusiveness*: It indicates whether the monitoring is done internally or externally with respect to the environment in concern.
- *Required resources*: The resources that the monitoring technique requires to operate. These resources can be software, hardware, or both.
- *Analysis strategy*: It specifies the strategy that is adopted by the technique for monitoring and analyzing the activities in the virtualization environment. Analysis strategies can be remote attestation, filtering and introspection, rule-based, ML-based, or statistical analysis.
- *Analysis time*: It specifies whether the monitoring is done in real-time or forensically (i.e., offline mode).
- *Analysis input*: The resources that the technique uses as an input for doing the analysis. These resources can be, e.g., a set of system calls within a specific period of time, a suspected application, sensor or traffic data, or configuration details.
- *Collected measurements*: Such as the frequency of interaction with the OS, as well as resource usage (e.g., CPU, memory and network).
- *Analysis procedure*: This provides a detailed description (i.e., step by step) of the analysis procedure.
- *Response or Reaction*: The action that the monitoring technique performs once an anomaly or misuse is detected. This can be, e.g., activating an alarm, logging details about the anomaly, or attack mitigation. Additionally, monitoring can be either active or passive, where the former is event-driven, monitoring occurrences of events, and therefor not time dependent. The latter, on the other hand, is based on state inspection polling during intervals and is vulnerable to transient attacks, i.e. attacks that occur between the polling and is thus time dependent. However, active monitoring is vulnerable to attacks which are outside of the specified events, and therefore circumvent detection mechanisms (Pham et al., 2014). Event based monitoring is used in a majority of the considered papers, thus we do not further put any emphasis on this distinction.

3.3.2. Applicability

Which domains are the monitoring techniques applicable to? This category is further broken down into the following two sub-categories:

- *Domain*: This subcategory specifies the domain in which the monitoring technique can be applied. The domain can be of general purpose, cloud computing, cyber–physical systems, high performance computing, or IoT.
- *Software type*: It specifies the type of software on which the monitoring technique can be applied, e.g., Linux containers (2021), Docker (2020), or Kubernetes (2020).

3.3.3. Effect

What is the purpose of the monitoring techniques and what are the consequences? This category includes four sub-categories:

- *Targeted threats*: The security threats that are targeted by the monitoring technique. The threats can be spoofing, tampering, repudiation, information disclosure, denial of service, and/or elevation of privilege. These security threats are known as STRIDE (Howard and Lipner, 2006).
- *Targeted attacks*: The type of the security attack that is targeted by the monitoring technique. The attack type can be, e.g., malware, unauthorized access, kernel exploit, network congestion and binaries modification.
- *Targeted faults*: The faults that are targeted by the technique, e.g., memory leak, CPU consumption, network latency, and log explosion.
- *Pros and cons*: A description of the advantages (e.g., enhanced verification) and disadvantages (e.g., inaccurate anomaly prediction) of the monitoring technique.

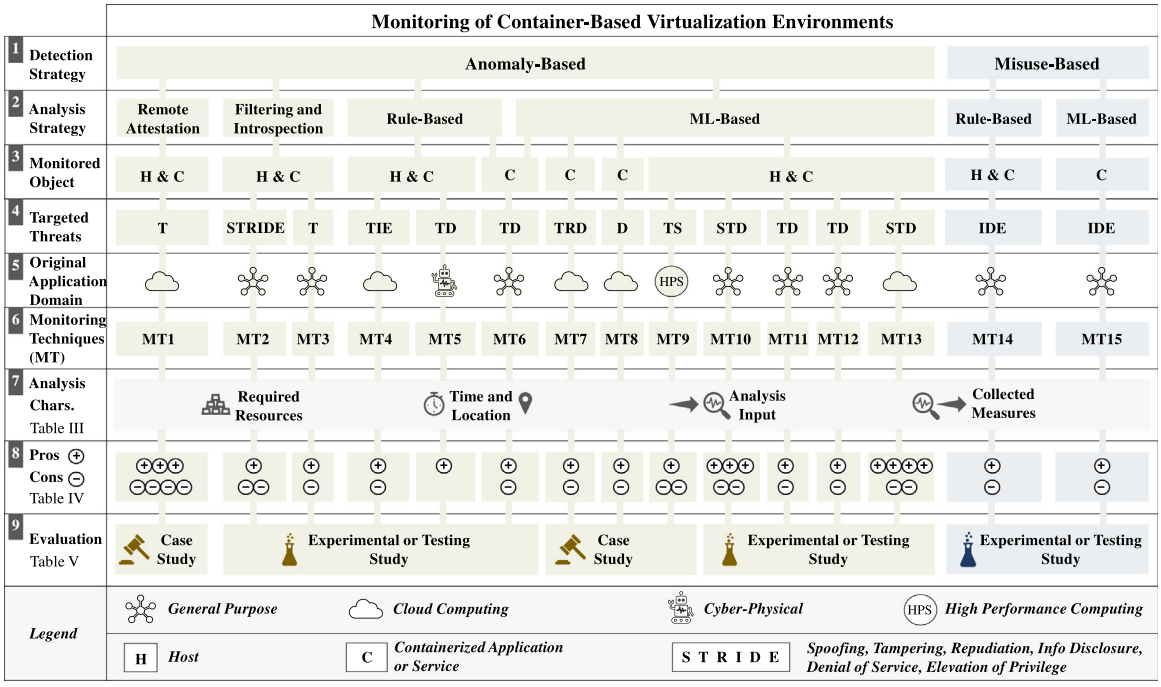


Fig. 3. CONSERVE Framework for the Selection of container monitoring techniques.

- *Side effect*: It specifies the impact of the monitoring technique on some quality aspects or resources, such as performance and CPU utilization.

3.3.4. Evaluation

How is the technique evaluated and what are the evaluation results?

This category describes the type of evaluation that is performed on the monitoring technique, which can be a case study, experimental, or testing study. Moreover, it details the evaluation approach together with the evaluation metrics (e.g., performance or detection rate) and results.

4. The framework: CONSERVE

In this section we present CONSERVE (see Fig. 3), a top-down selection framework for supporting the selection and implementation of monitoring techniques of container-based virtualization environments.

First, the framework supports the selection process by assisting the identification of candidate monitoring techniques applicable to the problem or task at hand (level *1 Detection Strategy* to level *5 Original Application Domain* in Fig. 3). In particular, to find out which monitoring techniques are applicable to a given problem, the framework questions whether the needed detection strategy is misuse- or anomaly-based. After that, for each detection strategy the framework details the analysis strategies that can be used and the objects in the container-based virtualization environment that can be monitored. The analysis strategies can be remote attestation, filtering and introspection, rule-based, or ML-based. The monitored objects can be the host, containerized application or service, or both. The framework further lists the security threats and attacks that need to be targeted, and describes the domains for which the monitoring techniques were originally designed and applied. By going through these level, a set of candidate monitoring techniques can be identified and selected for the problem at hand (level 6 *Monitoring Techniques MT*).

Second, the framework supports an objective decision-making process for an optimal selection based on comparing the following details (level *7 Analysis Characteristics* to level *9 Evaluation* in Fig. 3):

- characteristics of the analysis strategy, including the required analysis resources, analysis location and time, analysis input, and collected measures (see Table 3),
- pros and cons that arise from the adoption of the monitoring techniques (see Table 4), and
- type, metrics, and result of the evaluation that is conducted to assess the performance of the monitoring techniques (see Table 5).

In the following, we first provide details on the detection and analysis strategies in Section 4.1. After that, we describe the monitoring techniques in Section 4.2.

Finally, we describe how the CONSERVE framework can be used, and provide an example scenario in Section 4.3.

4.1. Detection and analysis strategies

Misuse-based detection identifies defined suspicious patterns, or signatures, within the analyzed data of the system to be protected. These techniques can identify known attacks with an acceptable accuracy and they tend to produce few false alarms (Garcia-Teodoro et al., 2009). However, this detection strategy does not provide means for detecting unknown intrusions (Garcia-Teodoro et al., 2009).

Anomaly-based detection involves the estimation of the normal behavior of the system that needs to be protected, and the identification of unexpected events by checking whenever a given observation deviates from the normal behavior (Garcia-Teodoro et al., 2009). Anomaly-based detection enables the detection of previously unknown intrusions. However, the rate of false positive alarms is usually higher compared to misuse-based detection systems (Garcia-Teodoro et al., 2009).

The following analysis strategies can be used when adopting the anomaly- or misuse-based detection strategy: remote

attestation, filtering and introspection, rule-based, and ML-based analysis.

- *Remote attestation* is a process by which a software application certifies its verification of certain security criteria to remote parties (Haldar et al., 2004). It is used to attest different properties of a software and verify its integrity and behavior. The application that is required to certify itself sends a unique hash of its executable created and signed by the Trusted Platform Module to the remote party. The remote party then verifies the signature of the application's hash prior to approval of the software. Eventual verification failures are further treated as anomalies.
- *Filtering and Introspection* involves filtering and analyzing the interaction between the software application and the host OS. Every application or program specifies a set of system call sequences that it can produce. Analyzing the number and sequence of system calls can provide an indication of normal or abnormal system behavior (Forrest et al., 1996).
- *Rule-based analysis* involves using a set of rules which capture the normal behavior and expected activities of applications. This set of rules is then used to identify anomalous behaviors (Lunt et al., 1989). The rules can be defined by an administrator or learnt via the use of an algorithm.
- *ML-based analysis* involves the use of machine learning techniques for identifying malicious activities. In supervised learning, a ML model is trained to learn the malicious behavior (classification). In unsupervised learning, a ML model is used to identify patterns and eventually anomalies in data through clustering. An analysis using supervised learning significantly outperforms an analysis using unsupervised learning if the test data contain no unknown attacks (Laskov et al., 2005).

4.2. Selection of container monitoring techniques

We use the CONSERVE framework to support the selection process of monitoring techniques based on the previously described detection and analysis strategies.

4.2.1. Anomaly-based detection and remote attestation analysis

- **MT1**: De Benedictis and Lioy (2019) propose DIVE, a technique based on anomaly-based detection and remote attestation analysis for Docker containers. DIVE exploits remote attestation to verify software integrity and correct the behavior of nodes in cloud applications at run-time. It monitors the host and software running in the containers.
  DIVE targets tampering threats and attacks e.g., by launching malicious scripts and code, service configurations and binary modifications, and starting new processes. The characteristics (i.e., required resources, analysis location and time, analysis input, and collected measures) of the analysis strategy are presented in Table 3.
  DIVE consists of three main components, namely verifier, attester, and infrastructure manager. The infrastructure manager starts the remote attestation process and sends a list of containers and hosts to the verifier.
  Whereafter the verifier contacts the attester and asks for integrity reports. After that, the verifier checks the measures belonging to the containers of interests against a whitelist. Finally, the verifier returns the integrity verification result to the infrastructure manager which is continuously keeping track of the containers and hosts. The infrastructure manager can terminate a compromised container or reboot the whole system if the host OS is compromised.

4.2.2. Anomaly-based detection and filtering & introspection analysis

Adopting a monitoring technique based on these strategies leads to two options, which depend on the STRIDE security threats that need to be targeted.

There are two techniques that support the monitoring of both the containerized applications (or services) and the host OS, as detailed below:

- **MT2**: SPEAKER Lei et al. (2017) is a general-purpose non-intrusive technique for enhancing the efficiency of the monitoring and analysis of containerized applications based on Linux containers. SPEAKER can effectively reduce the attack surface by removing unnecessary system calls that may be exploited by malicious processes in the container.
  For a given application container, SPEAKER uses a tracing module for profiling the available system calls in a booting phase and a running phase. The tracing module shares the system call lists with a slimming module, which is responsible for constraining the available system calls when the container boots up and runs.
- **MT3**: A general-purpose non-intrusive malware detection and analysis technique for Docker containers is proposed by Khalimov et al. (2019). It employs containers as sandboxes for tracking and introspecting system calls (syscalls) to the host kernel.
  This sand-boxing technique targets tampering threats and attacks, such as malicious code and network intrusion. This technique records and logs information about the malware behavior by using SystemTap (SystemTap, 2020). SystemTap is an open source software infrastructure that simplifies the gathering of information about the running activities in a Linux system. It also enables altering the values of syscall parameters, which enables this techniques to effectively hide the artifacts of the environment from the malware. In summary, this technique guarantees that the containerized solution deceives sandbox evasion by artifact obfuscation, network restructuring, and system call introspection.

4.2.3. Anomaly-based detection and rule-based analysis

There are two techniques that support the monitoring of both the containerized applications (or services) and the host OS.

- **MT4**: Mattetti et al. (2015) present LiCShield for monitoring and securing Linux containers in cloud computing systems. LiCShield targets tampering, information disclosure, and elevation of privilege threats. Examples of targeted attacks are: kernel exploits, attacks on shared kernel resources, misconfigurations, malicious modules, and data leakage. LiCShield automatically creates security profiles protecting the execution of a container on the host and in the container. This technique observes the execution of the activities and operations in a training environment to automatically define rules describing the expected activities of containers. LiCShield monitors the security of containerized applications by tracking their execution and generating profiles of kernel security modules which can restrict the capabilities of the containers when anomalies are detected. As a result, the execution and propagation of illegal operations and activities is blocked.
- **MT5**: Chen et al. (2019) present ContainerDrone, a container-based DoS attack resilience and monitoring technique for realtime cyber–physical systems using Docker application containers. This technique targets tampering and DoS security threats. Moreover, it targets safety violations caused by DoS attacks.

ContainerDrone uses the Simplex architecture (Sha, 2001) to provide attack-resilience. In particular, a container-controller and a host-controller are provided. A safety-monitor runs on the host control environment and keeps monitoring the output from both controllers. When a security violation is detected, the monitor switches to a safety controller to mitigate the attack and prevent further damage. The safety controller is robust and runs a limited number of modules that are critical to the correct functioning of the system. To protect the CPU from DoS attacks, the technique restricts the access of the container control environment to the CPU using cgroups and Docker capabilities. To protect the memory, the technique uses MemGuard (Yun et al., 2013). MemGuard ensures that each CPU core has a reserved memory bandwidth and does not access the memory exceeding a certain rate. Finally, the technique protects the communication between the component of the system from DoS attacks. In particular, sensor data and user inputs are reviewed by the container and host controllers. Moreover the network stack of the two controllers are separated. The container controller is located in a sandboxed network space where it does not have access to Internet, and can only communicate with the host controller through a specified interface.

There is one technique that supports the monitoring of containerized applications or services. This technique adopts two analysis strategies; rule-based and ML-based.

- **MT6**: Zou et al. (2019) propose a non-intrusive online container anomaly monitoring technique for Docker containers. This technique uses the optimized isolation forest algorithm (iForest) to calculate an anomaly value of the resource usage rates of each container on the host machine. The technique can automatically set the monitoring time in order to reduce monitoring delay and system overhead. Moreover, it can locate the cause of an anomaly by analyzing the log of the container. This technique targets tampering and DoS security threats and is designed to defeat network congestion and attacks. This technique also targets several faults such as endless CPU loop or spin lock, memory leak, memory overflow, improper disk scheduling, and log explosion.
  In the host machine is a monitoring agent that collects data on the resource utilization rate of the monitored container. The collected data are stored in a monitoring storage module which sends the data to an anomaly detection module. The anomaly detection module checks the received data for abnormality using an iForest algorithm. It calculates the anomaly value and identifies the anomalous resource metric when the anomaly value exceeds a certain threshold. The abnormal container information are then sent to an anomaly analysis module which firstly obtains the log of the abnormal container from the host, and secondly analyzes the log for locating the cause of the anomaly in the container.

4.2.4. Anomaly-based detection and ML-based analysis

In addition to the technique of (Zou et al., 2019) that we described in the previous paragraph, there are two techniques that support the monitoring of containerized applications or services.

- **MT7**: An anomaly detection technique that is used to detect and diagnose anomalies in container-based microservices is proposed by Du et al. (2018). This technique monitors and analyzes real-time performance data of microservices running in a cloud environment. The targeted security threats are tampering, repudiation, and DoS. Moreover, this technique addresses service level agreement violations and fault injections. The targeted faults are CPU consumption, memory leak, network package loss, and network latency increase.
  This technique includes three modules. First, a monitoring module that obtains performance data of the monitored container including CPU, memory, and network metrics. Second, a data processing module based on a set of ML classification models is used for the detection of anomalies. Third, there is a fault injection module which simulates service faults and collects performance data representing both normal and abnormal conditions. When an anomaly is detected in a microservice, then the data of all the containers running in this microservice are diagnosed using a dynamic time warping algorithm to locate the anomalous container. The prototype of the proposed detection technique is deployed on the orchestration system (Kubernetes, 2020).
- **MT8**: DockerAnalyzer is a technique for monitoring software executions in microservices-based applications (Fourati et al., 2019). This technique also identifies of the root cause of an abnormal behavior. It targets DoS and resource saturation attacks.
  This technique first collects data related to resource utilization and application performance such as CPU and memory usage, and application response time. A (Sysdig, 2020) monitoring component is used to filter the collected data and identify violations that need to be analyzed. An anomaly detector is invoked to check these violations and find out whether they are caused by normal resource saturation based on an increase of the number of requests or by an anomaly. In order to do so, the anomaly detector uses an outlier detection algorithm. In case of an anomaly, an anomaly detector based on a ML decision tree model is used to search for and identify the root cause of the anomaly which can be a specific request consuming the resources, virtual machine problem, or container problem.

There are five techniques that support the monitoring of both the containerized applications (or services) and the host OS.

- **MT9**: Gantikow et al. (2020) propose a container monitoring technique using neural networks for high performance computing environments. This techniques targets tampering and spoofing threats as well as mimicry attacks.
  A one layer Long Short Term Memory (LSTM) neural network is used to detect anomalies in system calls distribution. The neural network is trained to predict the distribution of system calls at time $t + 1$ based on the distribution at time t. In particular, the deviation between the prediction and the actual value is measured using the Root Mean Square Error (RMSE). After calculating the RMSE, the distribution of system calls is classified into either a normal or anomalous based on a predefined threshold value. Furthermore, the LSTM neuronal network can predict the next file system path that will be used by a system call based on the path of the currently used file system. This is done by determining the deviation between the expected and the actual file system path. Consequently, the file system path is classified into normal or anomalous based on a predefined threshold value. As this technique analyzes system calls using (Sysdig, 2020) it provides native container support.
- **MT10**: Abed et al. (2015) propose a non-intrusive technique for realtime monitoring of applications within Linux containers running in a standalone or cloud-based environment. This techniques targets spoofing, tampering, and DoS attacks via e.g., malware injections, OS compromise, file system

Table 3
Detailed characteristics of the analysis strategy of each Monitoring Technique (MT).

MT	Required Resources (*SW*: software, *HW*: hardware)	Location & Timing	Analysis Input	Collected Measures
MT1	*SW*: Linux capabilities and Integrity Measurement Architecture (IMA) (Sailer et al., 2004) *HW*: Trusted Platform Module	Offboard and Realtime	Platform configuration registers and IMA measures	Integrity verification results of the input data
MT2	*SW*: Linux capabilities, Secure Computing Mode (Seccomp)	Offboard and Realtime	A list of system calls issued by the running processes of a container to the host kernel	the ID of the originating processes, arguments, and return values
MT3	*SW*: Linux capabilities and SystemTap	Offboard and Forensically	A list of system calls issued by the running processes of a container to the host kernel	the ID of the originating processes, arguments, and return values, input/output activity, network traffic, and memory dumps
MT4	*SW*: SELinux (2020), SystemTap (2020) and AppArmor (2021)	Offboard and Realtime	A Dockerfile which is built into a Docker image and then run in a Docker container	A list of performed kernel operations during container creation and execution together with their resources and required permissions
MT5	*SW*: Linux capabilities and MemGuard (Yun et al., 2013)	Onboard and Realtime	Sensor data and user inputs. Also, network traffic and received threads	CPU core number, CPU utilization, Memory bandwidth, interval between two consecutive output received by the host controller
MT6	*SW*: (InfluxDB, 2021), and Apriori algorithm (Agarwal et al., 1994)	Offboard and Realtime	Running processes of a containerized application or service	Resource utilization: Container's ID, time, CPU usage, memory usage, disk read/write speed, and network speed
MT7	*SW*: InfluxDB (2021), cAdvisor (2020), Heapster (2020), and dynamic time warping algorithm	Offboard and Realtime	Running processes of containerized services	Resource usage (CPU, Memory, Network) and performance data of a specific micro-service
MT8	*SW*: Sysdig (2020)	Offboard and Realtime	Running processes of containerized services	Data related to number of service requests. CPU usage, memory usage, and application performance (i.e., response time).
MT9	*SW*: Sysdig (2020), Long Short Term Memory (LSTM) neural network	Offboard and Realtime	A list of system calls issued by the running processes of a container to the host kernel	the ID of the originating processes, arguments, and return values. Also, the used file and directory paths
MT10	*SW*: Linux capabilities (Strace)	Offboard and Realtime	A list of system calls issued by the running processes of a container to the host kernel	the ID of the originating processes, arguments, and return values
MT11	*SW*: Linux capabilities (Strace)	Offboard and Realtime	A list of system calls issued by the running processes of a container to the host kernel	the ID of the originating processes, arguments, and return values
MT12	*SW*: CRIU (2021), Sysdig (2020)	Onboard and Realtime	A list of system calls issued by the running processes of a container to the host kernel	the ID of the originating processes, arguments, and return values
MT13	*SW*: Linux capabilities (Strace) and Sysdig	Offboard and Realtime	A list of system calls issued by the running processes of a container to the host kernel	the ID of the originating processes, arguments, and return values
MT14	*SW*: Linux Capabilities, Sysdig (2020) and Falco (2020)	Offboard and Realtime	A list of system calls issued by the running processes of a container to the host kernel	the ID of the originating processes, arguments, and return values. CPU load, memory usage, network traffic, and block I/O

(*continued on next page*)

Table 3 (*continued*).

MT	Required Resources (*SW*: software, *HW*: hardware)	Location & Timing	Analysis Input	Collected Measures
MT15	*SW*: Restricted Boltzmann Machine (RBM) algorithm	Offboard and Realtime	Configuration details of application workloads running in containers: name of the container instance, invoker strategy, container interceptor settings, container instance cache, locking approach, security domain, and cluster configuration	Container run-time statistics: Number of container volumes mapped to the container, user access privileges, and authentication mechanisms. Also, number and sorts of network interfaces, network access, inter-process communications, and resource allocation limits.

access, and brute force attacks. The used analysis strategy is a sliding window and frequency-based bags of system calls (BoSC) analysis which keeps track of the frequencies of the system calls in a specific window of size k, where k is the number of monitored system calls at each epoch. This technique uses Linux capabilities, such as *strace* which reports system calls including information about their originating process ID, arguments, and return values. By tracking these system calls, the technique learns the behavior of the containerized applications and determines potential anomalies in the environment.

The proposed technique does not require any prior knowledge of the running application in the container, neither does it require any change to the container nor the host kernel. The technique first runs in a training mode where a classifier adds the new BoSC to the normal-behavior database. If the current BoSC already exists in the normal-behavior database, its frequency is incremented by one, otherwise, the new BoSC is added to the database with the initial frequency of one. A continuous training is applied during detection mode to further improve the accuracy of the technique.

Once trained, the technique runs in a detection mode where a sliding window is used to check if the current BoSC is present in the normal behavior database, if not a mismatch is declared. Moreover, a trace is declared anomalous if the number of mismatches exceeds a certain threshold.

- **MT11**: Srinivasan et al. (2018) propose a general-purpose technique for real-time monitoring of applications running in Docker containers. This technique targets tampering and denial of service security threats, such as malware, Trojan attacks, and SQL injection.

  Given an application running within a container, this technique uses a system call-based approach for the detection of anomalies and hence reports intrusions when they occur in real-time. The technique works in two modes: a normal and a detection mode. Based on these two modes, safe and unsafe sequences of system calls are maintained as n-grams and the probabilities of occurrences of these n-grams are calculated. The overall relative n-gram probabilities of safe sequences are stored in a database. The technique checks each n-gram probability against the probabilities stored in the database. If the n-gram is not present in the database, or if the difference between the probabilities of the observed and stored n-grams is beyond a certain threshold, the technique flags for a possible anomaly. When the number of the flags reaches a specific threshold, the technique considers the monitored activity as malicious and provides options to either continue running the container or stop the execution. The threshold is calculated by assessing the highest difference between the observed n-grams probabilities during the normal mode and detection mode.

- **MT12**: Time Machine (TM) (Sayed and Azab, 2019) is a general-purpose container monitoring technique that can be used to identify abnormal behaviors and malicious activities in Linux containers, in a way that keeps containerized stateful applications up and running in a safe state. This technique targets both tampering and DoS threats. In particular, it avoids logic bomb activation in mission critical systems. Logic bombs are a hidden code snippets that are added to the source code on purpose to enable input-triggered activation of a list of malicious features.

  Given a container of interest, the TM creates a shadow container (i.e., clone) of that container. Then, TM runs these two containers in two parallel environments; realtime running the original container and delayed-time running the shadow container. The incoming system-call traffic to realtime container is monitored. The monitoring strategy is based on the bag of system call (BoSC) and sliding window classification analysis, which checks whether or not a BoSC is present in a normal-behavior database. If the traffic results in a normal behavior, then the traffic is forwarded to the shadow (i.e., delayed) container and a green flag is declared. Otherwise, if the traffic results in a malicious behavior, then the TM blocks the calls from reaching the shadow container, clones the shadow container to the real container, saves the calls that resulted an anomalous behavior in anomaly profile to skip these calls in the future, and declares a red flag to alert the admin of the system.

- **MT13**: Abed et al. (2020) propose a resilient intrusion detection and resolution technique for cloud-based systems. This technique targets spoofing, tampering, and DoS attacks via e.g., malware injections, OS compromise, file system access, and brute force attacks. This technique extends the real-time behavior monitoring mechanism of Abed et al. (2015). In particular, it uses a Moving Target Defense (MTD) mechanism based on run-time container migration to quarantine malicious containers and reduce attack propagation. Moreover, to avoid zero-day attacks, the technique supports random live migrations between running containers to obfuscate its execution behavior.

  The system uses Sysdig tool (Sysdig, 2020) to trace system calls from the containers to the host kernel. When the system detects a misbehavior, the moving-target defense reacts as follows: in case of a stateful application, the system immediately migrates the affected container running the stateful application to a quarantine zone for further inspection. In case of a stateless application, the system rolls-back the misbehaving container to a previous safe state. The proposed technique was tested on an Apache Hadoop (Apache, 2021) cluster running Docker containers.

4.2.5. Misuse-based detection and rule-based analysis

There is one technique that supports the monitoring of both the containerized application and the host OS.

Table 4
Pros, Cons, and Side effect of each Monitoring Technique (MT).

MT	Trade-off		Reported Side effect
	Pros	Cons	
MT1	• Enables integrity verification of the host, container engine, and running containers. • Provides the possibility to distinguish which container is compromised. • Improves the remote attestation efficiency.	• Does not cover the full range of software attacks, such as detecting in-memory manipulations of code or data. • The white-list should be updated each time the host system is updated, or a false positive will be triggered. • It introduces a lock-in on the "Device Mapper" storage driver for Docker. • Its dependency on OpenAttestation (2021) makes it non-portable on hosts equipped with Trusted Platform Module (TPM) 2.0.	• Performance: Low impact • CPU: impact is proportional to the number of running containers. • Memory: low impact.
MT2	• Reduces the system call interface and incurs almost no performance overhead.	• Incomplete system call tracing. • Risk of system call misuse attacks e.g., mimicry attack.	• Performance: Low impact
MT3	• Uses sandboxing to analyze malware and rigorously record its behavior.	• Risk of fingerprinting through different sources that malware can easily be found.	• N/A
MT4	• Low overhead on the production environment.	• Risk of blocking legitimate operations that were never observed in the training environment.	• Performance: Low impact
MT5	• Protects from Denial of Service (DoS) attacks on CPU, memory, and communication.	• N/A	• CPU: Low impact.
MT6	• Low performance overhead.	• Inaccurate prediction: false positives and false negatives.	• N/A
MT7	• Includes performance monitoring.	• Inaccurate prediction: false positives and false negatives.	• N/A
MT8	• Identifies the root cause of the abnormal behavior.	• Inaccurate prediction: missing outliers detection.	• N/A
MT9	• Helps preventing mimicry attacks.	• Cannot be used alone since the reduction to file system paths leads to too much data not being analyzed. • Inaccurate predictions: false positives.	• Performance: high impact when the processed dataset is large
MT10	• Requires less storage space compared to sequence-based approaches while providing better accuracy. • Computationally manageable and does not require limiting the application programming interfaces. • No information disclosure about the nature of the application running in the container.	• Inaccurate prediction: false positives. • Risk of mimicry attacks.	• Performance: Low impact
MT11	• Scalability: the ability to run multiple Docker containers and a single monitoring IDS.	• Inaccurate prediction: false positives and false negatives.	• CPU: Low impact
MT12	• Helps keeping the stateful containers up and running in a safe state.	• Requires manual checks for the updates to the applications within the container.	• Performance: Low impact
MT13	• Limits intrusion dispersion by enabling container live migration between different managed cloud-hosts or rolling back to a safe state. • Requires less storage space compared to sequence-based approaches while providing better accuracy. • Computationally manageable and does not require limiting the application programming interfaces. • No information disclosure about the nature of the application running in the container.	• Inaccurate prediction: false positives. • Risk of mimicry attacks.	• Performance: Low impact
MT14	• Low performance overhead.	• The detection of a Buffer Overflow is not possible.	• N/A
MT15	• Increases the possibility of identifying zero-day vulnerabilities.	• Inaccurate prediction: false positives and false negatives	• N/A

Table 5
Evaluation of each Monitoring Techniques (MT) as reported by the studies.

MT	Type	Evaluation metric	Result
MT1		• Performance	It enables run-time verification of container applications at the cost of limited overhead. The integrity verification time increases with the number of active containers. Also, container integrity verification significantly impacts the CPU utilization at the verifier side, if compared to host-only attestation. Memory consumption at the increase of running containers is not affected as much.
MT2		• Effectiveness • Performance	It can successfully reduce more than 50% and 35% system calls in the running phase for the data store containers and the web server containers, respectively, with negligible performance overhead.
MT3		• Effectiveness	Docker containers are a promising option for a sandbox. Some artifacts can be hidden such as network artifacts, the Linux capabilities profile and proc file. However, this option comes at the cost of new detection artifacts which make containers subject to fingerprinting through different sources that malware can easily find.
MT4		• Effectiveness • Performance	It is efficient to prevent the targeted attacks, while having almost no overhead on the production environment. It is recommended to deploy the approach with technologies like Host-based Intrusion Detection Systems (HIDS) to achieve increased protection while optimizing the performance.
MT5		• Effectiveness	(1) In the case without MemGuard, the drone starts to drift right after the Bandwidth task is launched by the attacker and results in a crash shortly after. When the MemGuard is enabled, the drone oscillates for a short time but then managed to stabilize itself. (2) in case of a UDP attack, the attitude error control kills the received thread on host controller. (3) in case of security monitoring of safety attacks on the complex controller, the security monitor detects that the output from the container controller has not received for some time and kills the received thread and switches to the output from the safety controller
MT6		• Detection rate • False alarm rate	It can accurately detect anomalies in the container with small performance overheads: The optimized iForest has an acceptable small false alarm rate and high detection rate. The optimized iForest has a better performance than the compared detection methods. Overall, optimized iForest has better anomaly detection results compared to other two compared methods.
MT7		• Precision/Recall • F1 score	The detection performance of the anomalous service is excellent for most of the classifiers with measure values above 0,9.
MT8		• Resource usage • performance	The results demonstrate the effectiveness of the proposed technique in reducing resource usage compared to Kubernetes. Also, the technique helps also to improve the performance of the application comparing to Kubernetes by improving the response time to resource scaling requests.
MT9		• Effectiveness • Performance	File system path analysis using neural networks has been shown to be a good complement to a general approach such as system call distribution analysis, because by evaluating the system call parameters mimicry attacks can be prevented. This method might cause performance problems due to the larger number of processing steps compared to the system call distribution analysis for workloads with many file system accesses.
MT10		• True positive rate • False positive rate • Complexity	Results show a high detection rate of 100% is easily achievable using a low detection threshold of 10 mismatches per epoch. The false positive rate is 2%. The algorithm used is linear in the size of the input trace. The Time Complexity (TC) for looking up an index for a given system call is O(1) operation. The TC for updating the database with a new Bag of System call is O(1) operations. The TC for comparing the database before and after an epoch k, and computing the similarity metric, is O(nk), where nk is the size of the database after epoch k. Hence, the algorithm used is linear in the size of the input trace. The TC of running an epoch of size S is O(S + nk).
MT11		• Sensitivity • False positive rate	The IDS system boasts of high values of sensitivity for all the datasets tested in the range of 96%–100%, and the False Positive Rate is very low, ranging from 0%–14%.
MT12		• Performance	The Time Machine added time overhead about 17 milli-second per request (SQL query).
MT13		• Effectiveness • Performance	Attack dispersion without the Moving Target Defense (MTD) is massive. With the increase of the MTD response-rate, the attack dispersion is much less and the impact is limited. The impact on the application performance is negligible. The migration process takes less than 0,05 sec in most cases. The impact of intrusions is extremely limited and the intrusion dispersion rate is minimum.
MT14		• Effectiveness • Performance	It is effective in many attack scenarios and comes at a low performance overhead cost (Sysdig with full capture overhead is 5,45% and for Sysdig with filter is 2,02%).
MT15		• True positive rate • False positive rate	A relatively high-classification rate ($>$ 0,929) for some of the container security issues identified by the algorithm for the test records generated based on container profile. The classification rate can be improved further if more records are used to train the algorithm.

- **MT14**: Gantikow et al. (2019) propose a general-purpose rule-based security monitoring technique for enhancing the security of containerized environments, such as Docker, rkt and LXC. This technique targets information disclosure, DoS,

and elevation of privilege security threats. In particular, it detects a variety of misuse and attacks, such as unauthorized access, unauthorized launch of applications, container breakout, unexpected network connections, and loading of kernel module.
This technique is based on two open source tools; (Sysdig, 2020) which is a universal system visibility tool with native support for containers, and (Falco, 2020) which is a behavioral activity monitoring support for containers. First, Sysdig is used to capture and process the captured system call events. After that, Falco is used to detect suspicious behavior based on a predefined set of rules. In case of a suspicious behavior, an incident notification is issued using logging frameworks, messengers, or e-mail.

4.2.6. Misuse-based detection and ML-based analysis

There is one technique that supports the monitoring of containerized applications or services.

- **MT15**: Kamthania (2019) proposes a general-purpose ML technique for the identification of malicious patterns and zero-day vulnerabilities in container production environments which can be applied to any container platform complying to the Open Container Initiative specifications (OCI, 2021). This technique uses the deep learning restricted Boltzmann machine (RBM) algorithm (Ackley et al., 1985). This technique targets information disclosure, DoS, and elevation of privilege security threats. In particular, it deals with unbounded network access from containers, insecure run-time configurations, rogue containers, improper user access rights, and embedded clear texts.
  Given an application running in a container, the RBM derives the profile and configuration details of the monitored container. Next, it creates a behavioral knowledge map of the container by collecting container statistics at run-time (e.g., the number and types of network interfaces, network access, inter-process communications, and resource allocation limits). After that, the machine learning algorithm builds the complete security profile of the container under test which, in turn, is used to identify container security threat patterns. The identified malicious patterns can be used by container orchestration tools (e.g., Kubernetes) to make decisions on the current state of the production containers.

4.3. Using the CONSERVE framework

In this section, we first describe how CONSERVE can be used to perform the selection of container monitoring techniques. Second, we provide an application scenario. Fig. 4 contains a flow diagram and pseudo-code showing how the CONSERVE framework can be used. As input, a container-based environment (CBE) has to be monitored in order to enhance its resiliency to security threats. CONSERVE supports the selection of a monitoring technique for CBE through two steps:

Step A: Identification of candidate monitoring techniques.

Step B: Making of an objective and optimal selection (i.e., identifying the most suitable technique).

Step A. To identify a set of candidate monitoring techniques for CBE, the detection strategy can be selected first. There exist two detection strategies to select from: anomaly-based or misuse-based. Based on the selected detection strategy, we get a list of candidate monitoring techniques out of the available fifteen (see Fig. 4, *flow chart*: level 1, *pseudo-code*: lines 12–14).

After selecting the detection strategy, the analysis strategy can be selected. If anomaly-based detection was selected previously, then as analysis strategy there exist four options to select from: remote attestation, filtering and introspection, rule-based, or ML-based. On the other hand, if misuse-based detection was selected, then as analysis strategy there exist two options to select from: rule-based or ML-based. Based on the selected analysis strategy, we identify an updated (i.e., filtered) list of candidateMTs (*flow chart*: level 2, *pseudo-code*: lines 16–19).

Next, the monitored object can be selected. There are three options to select from: monitoring the host, containerized application/service, or both of them. Once we select the monitored object(s), we update the list of candidateMTs (*flow chart*: level 3, *pseudo-code*: lines 21–23). After that, the targeted security threats can be selected. The targeted threats can be up to six threats according to the STRIDE model (Howard and Lipner, 2006). Based on the selected targeted threats, the list of candidateMTs gets further updated (*flow chart*: level 4, *pseudo-code*: lines 25–27).

Next, the original application domain can be selected. There are four domains to select from: general purpose, cloud computing, cyber–physical systems, or high performance computing. Based on the selected domain, we update the list of candidateMTs (*flow chart*: level 5, *pseudo-code*: lines 29–31). As a result of *Step A*, a list of candidateMTs is identified (*flow chart*: level 6, *pseudo-code*: line 34).

Step B. In this step, CONSERVE supports an objective decision-making for an optimal selection. In particular, for each candidate monitoring technique identified in the previous step (i.e., *Step A*), CONSERVE reports its detailed analysis characteristics, pros and cons, and evaluation results (*flow chart*: level 7, 8, and 9, *pseudo-code*: lines 41–48). Based on the details provided in *Step B*, the most suitable monitoring technique for CBE can be identified and hence selected (*flow chart*: output, *pseudo-code*: line 52).

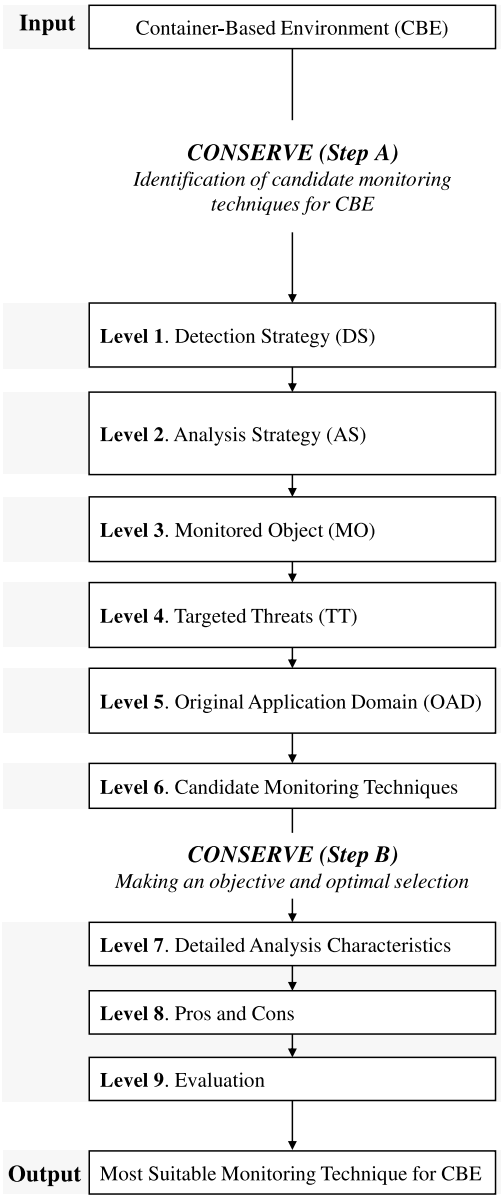
4.3.1. An application scenario

Bob, a system architect, needs to use a container monitoring technique to secure containerized applications running on a Vehicle Computational Unit (VCU). A VCU is a powerful piece of hardware that typically comes with multiple processors to enable computationally demanding autonomous drive functions. Moreover, it provides core vehicle functionality and services such as mode management, on-board diagnostics, software download, settings management, and connectivity. A VCU can run multiple applications on the same hardware and/or operating system. Bob has the following two requirements that he needs to fulfill:

- the monitoring technique should monitor the interaction between the host OS and the container, and
- the monitoring technique should target unknown attacks that belong to the following security threats: spoofing, and denial of service (DoS).

By using CONSERVE, Bob explores the different monitoring techniques applicable to his problem. First, Bob visits CONSERVE's level 1 (Detection Strategy) and selects *anomaly-based* as a detection strategy since the monitoring technique must target unknown attacks. Accordingly, techniques from MT1 to MT13 are considered as candidates. Next, by looking at level 3 (Monitored Objects), Bob excludes techniques that do not monitor the interaction between the host OS and the container. The excluded techniques are MT6, MT7, and MT8. Last, by exploring level 4 (Targeted Threats) Bob explores the monitoring techniques that target spoofing and DoS threats, and identifies three candidate monitoring techniques. These techniques are MT2, MT10, and MT13.

To get further details on these techniques, Bob explores CONSERVE's level 7 (Detailed Analysis Characteristics). Moreover, to make an objective selection, Bob explores the trade-offs and evaluation results (levels 8 and 9). He finds out that MT13 has a low impact on performance and provides a mechanism that enables




```
AB = Anomaly-Based;
MB = Misuse-Based;
RA = Remote Attestation;
FI = Filtering and Introspection;
RB = Rule-Based;
ML = Machine Learning;
CAS= Containerized App or Service;
GP = General Purpose;
HPC= High Performance Computing;
candidateMTs= 15 Candidate Monitoring Techniques;

if interestedIn(DS) then
 select DS = (AB || MB);
candidateMTs = candidateMTs basedOn selected(DS);

if interestedIn(AS) then
 if DS = AB then select AS = (RA || FI || RB || ML);
 if DS = MB then select AS = (RB || ML);
candidateMTs = candidateMTs basedOn selected(AS);

if interestedIn(MO) then
 select MO = (Host || CAS || Both);
candidateMTs = candidateMTs basedOn selected(MO);

if interestedIn(TT) then
 select TT = targetedThreats <STRIDE>;
candidateMTs = candidateMTs basedOn selected(TT);

if interestedIn(OAD) then
 select OAD = (GP || Cloud || CyberPhys || HPC);
candidateMTs = candidateMTs basedOn selected(OAD);


get candidateMTs;




foreach MT in candidateMTs
{

info = check detailedAnalysisChars(MT);
info = info + check prosNcons(MT);
info = info + check evaluation(MT);

}


get mostSuitableMT basedOn info;
```

Fig. 4. A flow chart (*left*) and pseudo-code (*right*) showing how to use the CONSERVE Framework.

rolling back a container to a safe-state. Bob considers this mechanism as important, since the system he works on is safety-critical. Hence, Bob considers MT13 as the most suitable monitoring technique for addressing the problem and requirements he has at hand.

5. Framework evaluation

To evaluate the CONSERVE framework, we use a mixed methods approach employing both quantitative and qualitative data collection and analysis (Creswell and Creswell, 2017). We use this approach because it enables the investigation of the strengths and weaknesses of the developed framework including the social and cognitive processes surrounding it. For data collection, we follow a design comprising a try-out task and semi-structured questionnaires (see Section 5.1). Moreover, we adopt a *sequential explanatory* strategy for data analysis (Creswell and Creswell, 2017). Using this strategy, we use the qualitative data to complement as well as assist the explanation and interpretation of the quantitative data.

5.1. Try-out task and questionnaires

We create a pre-evaluation questionnaire to collect data on participants' highest educational degree, domain of expertise, and current occupation as well as experience in systems security and in development or engineering of computing systems. Moreover, we design a try-out task to engage the participants in using the framework, which is necessary to assess its strengths and weaknesses. The task challenges the participants to select a set of container monitoring techniques by using CONSERVE and based on a set of specific requirements. Last, we create a post-task questionnaire to collect perceptions on the ease of use, understandability, usefulness, efficiency, applicability, and completeness of the framework. The evaluation material, including the pre-evaluation and post-task questionnaires, is available online.[1]

[1] Evaluation Material: http://rodijolak.com/conserve.

5.2. Participants

The target group is the entire population of people who have a basic knowledge or expertise in engineering or development of computing systems. The accessible population for this study is a subset of the targeted group and identified opportunistically via networks of collaborators and contacts (i.e., through convenience sample). In particular, a mix of 18 researchers and practitioners are involved in the evaluation of the framework.

Table 6 provides details about the participants and is created based on the answers collected by the pre-evaluation questionnaire. Seven of the participants are researchers working in academia or research institutes, namely Chalmers University of Technology, University of Gothenburg, Vietnam National University, University of Paderborn, TU Hamburg, and RISE Research Institutes of Sweden. The remaining eleven participants are practitioners working in industrial organizations, namely AB Volvo Group Trucks Technology, Volvo Cars, and Nvidia.

5.3. Evaluation procedure

Before running the evaluation, we conducted a pilot study which helped in shaping the design of the evaluation protocol and assessing whether it is realistic and workable. The pilot study also helped in identifying issues with the design of evaluation protocol as well as in training the supervisor of the evaluation. Based on the pilot, we concluded that the participants would require around 90 minutes to go through the try-out task and the questionnaires. After running the pilot, we collected date and time preferences of the participants for conducting the evaluation. The evaluations are conducted in January and June 2021 over Zoom, Microsoft Teams, or other online meeting tools that are preferred by the participants. The first author of this article supervised all the evaluation sessions to avoid eventual unbalance in the assistance that the participants might have otherwise received by the appointment of different supervisors.

For each evaluation meeting, the supervisor sent the evaluation material to the participant and explained that the evaluation material includes two files:

1. *Evaluation Steps*, which provides a step by step guideline for performing the evaluation.
   Beside presenting the task and questionnaires, this file provides an introduction to containers and the threats that targets the containers and the objects in the virtualization environment.
2. *CONSERVE*, which is used for performing the evaluation task described in the *Evaluation Steps* file.

The supervisor was present in the meetings to assist and guide the participants in case they have questions regarding the evaluation process or technical problems e.g., link to pre-evaluation questions is not working. The supervisor asked the participants to start the evaluation by going through the evaluation steps. On average, the evaluation sessions lasted 75±15 minutes.

5.4. Evaluation results

To analyze the collected *quantitative* data, we generate descriptive statistics including (i) percent stacked-bar charts which display comparisons between categories of data and highlight the relationship of constituent categories/parts to the whole bar, (ii) precision and recall to investigate the accuracy of the answers of the participants to the questions of the try-out task, and (iii) Mean±SD (standard deviation) to analyze central tendencies and dispersion. Furthermore, we conduct an inductive thematic analysis (Boyatzis, 1998) which helps to identify, analyze, and report patterns (i.e., themes) within the collected *qualitative* data. In particular, we code the qualitative data as well as identify and analyze themes at the explicit level (Boyatzis, 1998).

5.4.1. Task results

The evaluation task comprises of three steps. In the first step, the participants are asked to select a set of candidate monitoring techniques that can be used for monitoring the security of a containerized application of a Vehicle's Computational Unit (more details are provided in evaluation material). The selection should be done using CONSERVE and based on two requirements, namely, (i) the monitoring techniques should monitor the interaction between the host OS and the container, and (ii) the monitoring techniques should target unknown attacks that belong to Spoofing and Denial of Service security threats. Moreover, we ask the participants to describe the reasoning behind their selection.

The set from which the participants can select the monitoring techniques is provided by CONSERVE and it includes fifteen techniques in total. Three monitoring techniques (i.e., Abed et al. (2015), Lei et al. (2017), and Abed et al. (2020)) fulfill the specified requirements and are therefore the correct ones. Table 7 shows the mean and standard deviation values of the precision and recall related to the first step. The mean precision of all the participants is 0.82±0.29 and the mean recall is 0.89±0.26. To explain these numbers, we analyze the reasoning behind the selections and observe the following:

- *Human error*: one participant included one false positive and one false negative in the set of candidates; and another participant included two false positives in the set of candidates.
  However, both of these participants have correctly reasoned for their selections, which should have consequently helped them in the selection of the correct techniques.
  Accordingly, we consider these cases as consequences of a human error.
- *Misunderstanding*: there are three cases connected to some sort of misunderstanding: First, three participants misunderstood the evaluation task.
  These participants included false positives in their solutions. After checking their reasoning, we find that they considered techniques targeting either Spoofing or Denial of Service security threats, as opposed to the task's requirement which was on the selection of both of these threats.
  Second, one participant removed one true positive from the candidates set after discussing the pros and cons of the technique, which is not required at this step.
  Thus, we consider this case as a misunderstanding of the task.
  Third, the set of candidates of one participant has no true positives.
  After checking the reasoning, we find that the participant did not take into account the requirements defined for this step. Thus, we consider this case as a misunderstanding of either the evaluation task or the framework.

In the second step, the participants are asked to report the required software and hardware components for the proper functioning of each candidate monitoring technique in the list of candidate monitoring techniques. Table 7 shows the mean and standard deviation values of the precision and recall related to the second step. The mean precision and recall of all the participants are the same, 0.94±0.24. Indeed, only one participant could not fetch the required software and hardware components for the proper functioning of the selected techniques.

In the third step of the evaluation task, we ask the participates to select only one monitoring technique from their list or set of candidates and provide a reasoning on their selection. We find that most of the participants (17 out of 18) objectively and correctly reasoned about their final selection based on the

Table 6
Participants in the Evaluation of CONSERVE.

P#	*Degree*	*Domain*	*Occupation*	Systems Security		Systems Development/Engineering		Virtualization and Containers	
				Expertise[b]	*Experience (years)*	*Expertise*[b]	*Experience (years)*	*Expertise*[b]	*Experience (years)*
01	Ph.D.	Software Engineering	Ph.D. Student	4.00	4.00	4.00	9.00	1.00	0.17
02	M.Sc.	Computer Science	Research Assistant	3.00	0.00	4.00	3.00	3.00	2.00
03	M.Sc.	Computer Science and Engineering	Ph.D. Student	2.00	1.00	3.00	6.00	3.00	0.42
04	Ph.D.	Computer Engineering	Technology and Strategy Leader	3.00	0.00	4.00	10.00	3.00	5.00
05	Ph.D.	Computer Science and Engineering	Researcher	5.00	7.00	5.00	27.00	4.00	1.00
06	Ph.D.	Networks and Systems	Ph.D. Student	4.00	0.00	3.00	0.00	3.00	0.00
07	Ph.D.	Computer Engineering	Principle Engineer within Cybersecurity	4.00	2.00	4.00	15.00	3.00	0.50
08	M.Sc.	Computer Science	Ph.D. Student	2.00	0.00	4.00	4.00	2.00	0.50
09	Lic.Eng.[a]	Networks and Systems	Security Engineer	5.00	8.00	4.00	13.00	3.00	4.00
10	B.Sc.	Mechatronic Engineering	Logical Architect	2.00	0.00	5.00	34.00	2.00	0.00
11	M.Sc.	Computer Science and Engineering	Systems Security Engineer	4.00	10.00	4.00	5.00	3.00	2.00
12	M.Sc.	Communication Engineering	Ph.D. Student	2.00	0.34	4.00	4.00	2.00	0.00
13	M.Sc.	Computer Science and Engineering	Security System Designer	4.00	10.00	4.00	10.00	3.00	0.50
14	B.Sc.	Computer Science	Security Architect	5.00	23.00	4.00	32.00	4.00	10.00
15	Ph.D.	Software Engineering	System Architect	3.00	3.00	5.00	10.00	3.00	3.00
16	Ph.D.	Network and Systems	Software Engineer	2.00	2.00	4.00	5.00	3.00	1.00
17	M.Sc.	Software Engineering	Software Developer	3.00	0.00	3.00	12.00	2.00	1.00
18	B.Sc.	Computer Science	Software Developer	3.00	5.00	3.00	17.00	2.00	3.00
			Median (ordinal data)	3.00		4.00		3.00	
			First Quartile (Q1)	2.25		4.00		2.00	
			Third Quartile (Q3)	4.00		4.00		3.00	
			Inter-Quartile Range (IQR)	1.75		0.00		1.00	
			Mean (interval data)		4.19		12.00		1.89
			Standard Deviation		5.87		9.86		2.50

[a]Licentiate of Engineering is a pre-doctoral degree common in Sweden.
[b]Expertise: 1 very low, 2 low, 3 average, 4 high, 5 very high.

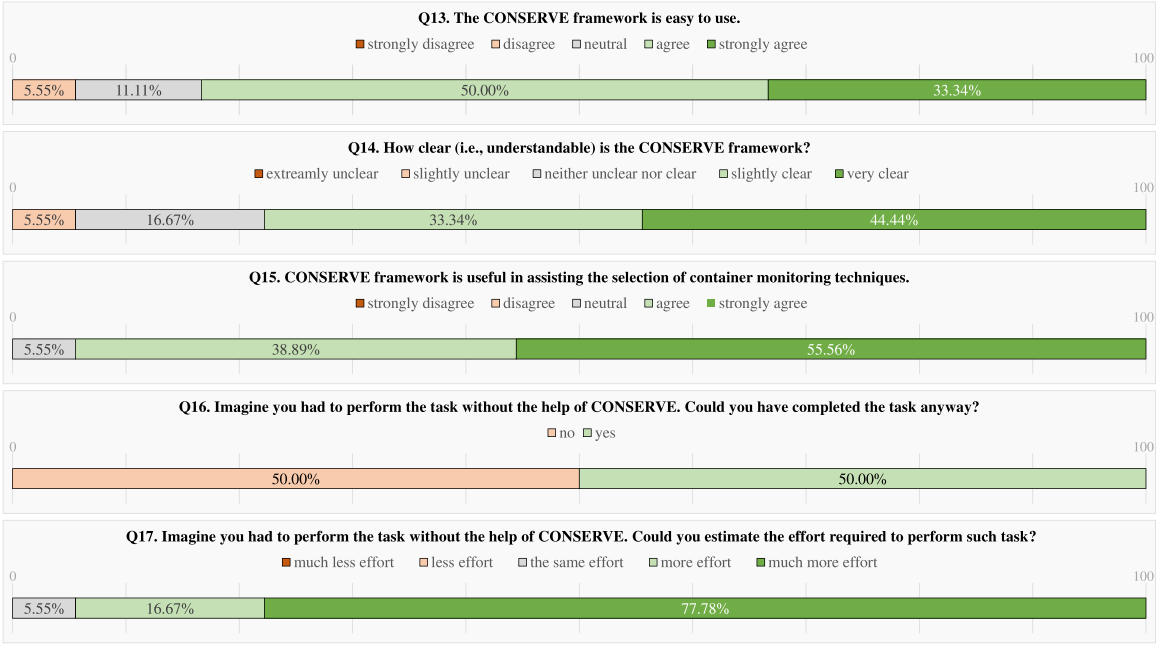


Fig. 5. Evaluation: Results of Post-Task Questions 13-17.

Table 7
Precision and Recall of (Step 1) the Selected Candidate Techniques and (Step 2) the Required Resources.

Step 1 (n=18)	Precision	Recall	Step 2 ($n = 18$)	Precision	Recall
Mean	0.82	0.89	Mean	0.94	0.94
Std. Dev.	0.29	0.26	Std. Dev.	0.24	0.24

details provided by Table 4 (pros and cons of the techniques) and Table 5 (evaluation of the techniques). This result indicates that the provided details on trade-offs and performance evaluation by CONSERVE helped in assisting an objective decision-making process of the monitoring techniques.

5.4.2. Post-task results

Fig. 5 presents the answers of the participants on the closed-ended questions of the post-task questionnaire. This figure shows that the perceptions of the participants on the ease of use, clarity, and usefulness of the framework are positive (Q13-Q15). In fact, the majority of participants agree that the framework is easy-to-use (83%), clear (78%) and useful (94%). Moreover, this figure shows that 50.00% of the participants consider the framework as essential for performing the selection of container monitoring techniques. In contrast, 50.00% of the participants state that they could perform and complete the task without the help of CONSERVE (Q16). However, this 50.00% and the rest of participants estimate that performing the selection task requires the same effort (5.55%), more effort (16.67%), and much more effort (77.78%) without the help of CONSERVE (Q17).

The answers to the open-ended questions (Q18-Q22) of the post-task questionnaire are analyzed using thematic analysis. The results are presented in the following paragraphs.

Benefits. According to the participants, there are three main perceived benefits *(b)* of CONSERVE:

(b1) It provides a *systematic decision support* for navigating through the applicable monitoring techniques for the task at hand.

(b2) It is a *simple and easy to use* framework for selecting container monitoring techniques.

(b3) It supports an *efficient* selection by saving time and effort in matching the techniques with the requirements.

Applicability. CONSERVE is considered as a valuable *light-weight* framework that can be easily adopted in practice without additional training. Moreover, the participants perceived the potential applicability of the framework as an essential means for understanding the features of the monitoring techniques and supporting the selection thereof in practice. A continuous support and maintenance of the framework (e.g., by updating the considered techniques or adding new ones) is perceived as an important step for enduring and increasing the applicability of the framework in practice.

Challenges. The main perceived challenges are related to the presentation and understandability of the framework and the evaluation task. For instance, the used publication references as identifiers of the techniques in the framework is perceived as confusing. On this regard, it is suggested to enhance the presentation of the framework by using identifiers such as Tech 1, Tech 2, etc. The reported evaluations in Table 5 are perceived to be somehow difficult to use for comparing the techniques as not all the evaluations use the same metrics. The lack of sufficient domain knowledge of the participants who have a low experience and experience in system security is considered as a challenge to understand the framework and the evaluation task. Two participants also reported that the understanding of the framework was challenging at the beginning, but they were able to fully understand it after reading the evaluation tasks.

Completeness. To complete the framework and assist its understandability and applicability, it is suggested to describe the use of the framework by means of examples and flow diagrams. Moreover, a connection to other quality attributes such as safety, maintainability, reusability, and variability is considered as good to have. To better assist the selection process, it is suggested to add more details about the metrics used in the evaluation of the monitoring techniques such as the number of evaluated objects, domain of evaluation, performance costs, and false positive rates.

Suggestions. It is suggested to create a web application or GUI for an interactive presentation of the framework. This would indeed provide a means for filtering, grouping, and highlighting the techniques with a click of a button, which in turn would simplify the comparison and selection process.

6. Threats to validity

Our study is subject to threats to its construct validity, internal validity, external validity, and reliability.

6.1. Construct validity

Construct validity refers to how well captured data and measures represent what researchers intended them to represent in the study in question. The literature review is conducted by involving one search database, namely Scopus, which is claimed to be one of the largest databases of abstracts and citations (Keele et al., 2007). Still, using one database in the literature review raises the risk of considering a non-representative set of relevant studies. To complement the review and cover more relevant work, we use the snowballing search approach. In particular, Google Scholar is used to perform the snowballing search in order to avoid publisher bias (e.g., searching in one publisher's database) (Wohlin, 2014).

Discretizing the measurement of continuous properties, such as the level of expertise and experience of the participants in systems security, could have lead to a threat to construct validity. This threat is investigated for balanced *Likert* scales and identified as not compromising the measures (Ray, 1982).

6.2. Internal validity

Internal validity concerns efforts made to ensure that possible confounding factors are identified and alleviated. The inclusion and exclusion of studies in the literature review is done by one researcher. Hence, there is a threat of subjectivity (i.e., literature review reliability). To mitigate this threat, a second researcher performed a quality control of the review. Using the same search terms, a search on Scopus database is performed and 15 studies are randomly selected which account to approximately 15% of the total number of the search results. The inclusion as well as exclusion criteria and the snowballing search approach are performed on these studies. To assess the reliability, we calculate the inter-rater agreement coefficient (Cohen's kappa (Cohen, 1960)) between the two researchers (i.e., raters). We obtain a kappa value of 0.75 which indicates a substantial inter-rater agreement according to Landis and Koch (1977). The difference in the considered studies (one study not included by the first researcher) is discussed and resulted in a clarification of the inclusion and exclusion criteria.

The level of expertise and experience in virtualization and systems security of the participants might influence the understanding of the framework and, thus, the evaluation thereof. We contacted participants from both academia (Ph.D. students and researchers) and industry (e.g., systems security experts) to make sure that our participants have different levels (i.e., high, medium, and low) of expertise and experience. The main reason is that we wanted to increase realism, since not all developers working in industry have a high expertise and experience in virtualization and systems security. Furthermore, considering people with different levels of expertise and experience helps in obtaining diverse and significant feedback – that might not otherwise manifest by involving only experts and highly experienced people – on the applicability, usefulness, and understand-ability of the framework.

The presentation of the framework (Fig. 3 and the three complementing tables) might have influenced the understanding and evaluation thereof. A pilot study was conducted to ensure that the framework is clear and well presented. Moreover, the perceptions on the ease of use and clarity were positive, thus we consider the effect of this issue as minimal.

Based on the publication type (e.g., conference vs. journal), venue name, publication date, and number of current citations, we notice that not all the included studies are published in well-known venues. Moreover, four out of the fifteen included studies still have a low number of citations. This might be considered as a threat to the credibility of these studies. The four studies that have a low number of citations are recently published, and this might be the reason of such lack of citations. We think that the implementation and further evaluation of the monitoring techniques in practice would contribute in mitigating this threat.

6.3. External validity

External validity concerns the extent to which the results of a study can be generalized. The clarity, size, duration, and complexity of the task, including the questions, used for the evaluation of the framework might differ from real-world conditions and limit the generalizability of the evaluation results. To address this threat, we aim and call for replication. Furthermore, the participants involved in the evaluation of the framework are selected based on contacts and collaborators, i.e., through convenience sample. While the number of involved participants might be considered as small, we do not consider this as a major threat since the aim of the evaluation is to get a preliminary qualitative feedback and not to generalize over a population of actors.

6.4. Reliability

Reliability concerns the extent to which the operations of a study can be repeated by other researchers, achieving the same results. We thoroughly detail the approach that we adopted to perform the literature review. Moreover, we detail the evaluation process of the framework by reporting the introduction to the context and framework given to the participants, characteristics of the participants, evaluation task, and post-task questions. These details should enable reproduction of our study under comparable contexts.

7. Conclusion

Container-based virtualization is gaining popularity in different domains. Different techniques for monitoring containers security are proposed. However, there are no guidelines supporting the selection of suitable container monitoring techniques for the tasks at hand.

We review the literature to identify relevant techniques for monitoring container-based virtualization environments with the goal to provide a comprehensive overview of these techniques. We further categorize the identified techniques to help developers understand their purpose, technical characteristics, applicability, and effectiveness.

As a result, we present CONSERVE, a multi-dimensional decision support framework that can be applied in different domains. An evaluation of the framework by a mix of eighteen researchers and practitioners shows a high level of interest, and points out to potential benefits. Mainly, the majority of the participants in the evaluation agree that the framework is easy to use (83%), clear (78%), and useful (94%). Moreover, all the participants agree that the framework supports an efficient selection of container monitoring techniques by saving time and effort in matching the techniques with the tasks at hand.

7.1. Implications to research and practice

The knowledge that the framework provides, including the characteristics, functionalities, pros, and cons of the container monitoring techniques, could be used by researchers to enhance

the performance of the techniques and/or design more effective solutions. This can be done by e.g., investigating approaches that would significantly mitigate the current cons of the techniques or by combining a number of techniques to achieve multiple monitoring-layers and, thus, improve the security of the container-based virtualization environment.

A combination of threat-based risk analysis and the framework could be used by practitioners to make informed decisions about the required monitoring techniques with respect to the identified threats to the container-based virtualization environment and the associated level of risk.

Given that the framework considers techniques used in different domains and provides clear insight into the trade-offs and performance evaluations of each technique, the effectiveness of the framework can be potentially more significant in domains such as automotive in which cybersecurity and container-based technology are relatively new, and performance requirements are critical. The framework enables practitioners in such domains to identify monitoring techniques used in other domains, and select the most suitable approach based on the available resources and the expected performance requirements.

7.2. Future work

To enhance the current version of the framework, we will address the comments that we received via the evaluation. Moreover, to complement the conducted evaluation and get further insights, we plan to evaluate the framework using more scenarios and involving more participants.

Furthermore, we plan to provide an interactive presentation of CONSERVE to assist the exploration and selection process of container monitoring techniques. We also plan to continuously maintain and update the framework to reflect eventual evolution of the considered monitoring techniques as well as include newly developed techniques.

CRediT authorship contribution statement

Rodi Jolak: Conceptualization, Methodology, Resources, Investigation, Data curation, Writing - original draft, Writing - review & editing, Supervision. **Thomas Rosenstatter:** Conceptualization, Resources, Writing - original draft, Writing - review & editing. **Mazen Mohamad:** Conceptualization, Resources, Writing - original draft, Writing - review & editing. **Kim Strandberg:** Conceptualization, Resources, Writing - original draft, Writing - review & editing. **Behrooz Sangchoolie:** Conceptualization, Resources, Writing original draft, Writing - review & editing. **Nasser Nowdehi:** Conceptualization, Resources, Writing - original draft, Writing - review & editing. **Riccardo Scandariato:** Conceptualization, Methodology, Investigation, Resources, Writing - original draft, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We would like to thank the participants who took a part in the evaluation of CONSERVE. This research was partially supported by the Swedish VINNOVA FFI project CyReV: Cyber Resilience for Vehicles with diary numbers: 2018-05013 (1st phase) and 2019-03071 (2nd phase).

References

Abed, A.S., Azab, M., Clancy, C., Kashkoush, M.S., 2020. Resilient intrusion detection system for cloud containers. Int. J. Commun. Netw. Distrib. Syst. 24 (1), 1–22.

Abed, A.S., Clancy, C., Levy, D.S., 2015. Intrusion detection system for applications using linux containers. In: International Workshop on Security and Trust Management. Springer, pp. 123–135.

Ackley, D.H., Hinton, G.E., Sejnowski, T.J., 1985. A learning algorithm for Boltzmann machines. Cogn. Sci. 9 (1), 147–169.

Agache, A., Brooker, M., Iordache, A., Liguori, A., Neugebauer, R., Piwonka, P., Popa, D.-M., 2020. Firecracker: Lightweight virtualization for serverless applications. In: 17th USENIX Symposium on Networked Systems Design and Implementation. NSDI 20, USENIX Association, Santa Clara, CA, pp. 419–434, URL https://www.usenix.org/conference/nsdi20/presentation/agache.

Agarwal, R., Srikant, R., et al., 1994. Fast algorithms for mining association rules. In: Proc. of the 20th VLDB Conference, pp. 487–499.

Apache, 2021. Apache hadoop. Apache http://hadoop.apache.org/. (Accessed January 2021).

AppArmor, 2021. A linux application security system. AppArmor https://gitlab.com/apparmor/apparmor/-/wikis/home. (Accessed January 2021).

Arnautov, S., Trach, B., Gregor, F., Knauth, T., Martin, A., Priebe, C., Lind, J., Muthukumaran, D., O'Keeffe, D., Stillwell, M.L., Goltzsche, D., Eyers, D., Kapitza, R., Pietzuch, P., Fetzer, C., 2016. SCONE: Secure linux containers with intel SGX. In: Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation. In: OSDI'16, USENIX Association, USA, pp. 689–703.

Bélair, M., Laniepce, S., Menaud, J.-M., 2019. Leveraging kernel security mechanisms to improve container security: A survey. In: Proceedings of the 14th International Conference on Availability, Reliability and Security. In: ARES '19, Association for Computing Machinery, New York, NY, USA, http://dx.doi.org/10.1145/3339252.3340502.

Berger, C., Nguyen, B., Benderius, O., 2017. Containerized development and microservices for self-driving vehicles: experiences & best practices. In: 2017 IEEE International Conference on Software Architecture Workshops. ICSAW, IEEE, pp. 7–12.

Bernstein, D., 2014. Containers and cloud: From LXC to docker to kubernetes. IEEE Cloud Comput. 1 (3), 81–84. http://dx.doi.org/10.1109/MCC.2014.51.

Boyatzis, R.E., 1998. Transforming Qualitative Information: Thematic Analysis and Code Development. sage.

cAdvisor, 2020. A software for analyzing and exposing resource usage and performance data from running containers. GitHub Repository https://github.com/google/cadvisor. (Accessed December 2020).

Casalicchio, E., Iannucci, S., 2020. The state-of-the-art in container technologies: Application, orchestration and security. Concurr. Comput.: Pract. Exper. 32 (17), http://dx.doi.org/10.1002/cpe.5668, e5668, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5668, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.5668, e5668 cpe.5668.

Chandramouli, R., Chandramouli, R., 2017. Security Assurance Requirements for Linux Application Container Deployments. US Department of Commerce, National Institute of Standards and Technology.

Chen, J., Feng, Z., Wen, J.-Y., Liu, B., Sha, L., 2019. A container-based DoS attack-resilient control framework for real-time UAV systems. In: 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, pp. 1222–1227.

Cohen, J., 1960. A coefficient of agreement for nominal scales. Educ. Psychol. Meas. 20 (1), 37–46. http://dx.doi.org/10.1177/001316446002000104, arXiv:https://doi.org/10.1177/001316446002000104.

Combe, T., Martin, A., Di Pietro, R., 2016. To docker or not to docker: A security perspective. IEEE Cloud Comput. 3 (5), 54–62.

containers, L., 2021. Infrastructure for container projects. Kubernetes https://linuxcontainers.org/. (Accessed June 2021).

Creswell, J.W., Creswell, J.D., 2017. Research Design: Qualitative, Quantitative, and Mixed Methods Approaches. Sage publications.

CRIU, 2021. A checkpoint/restore functionality for linux. CRIU https://criu.org/Main_Page. (Accessed January 2021).

De Benedictis, M., Lioy, A., 2019. Integrity verification of docker containers for a lightweight cloud environment. Future Gener. Comput. Syst. 97, 236–246.

Docker, 2020. A lightweight, standalone, executable package of software that includes everything needed to run an application. Docker https://www.docker.com. (Accessed December 2020).

Du, Q., Xie, T., He, Y., 2018. Anomaly detection and diagnosis for container-based microservices with performance monitoring. In: International Conference on Algorithms and Architectures for Parallel Processing. Springer, pp. 560–572.

Elsevier, 2021. Scopus – expertly curated abstract & citation database. Elsevier https://www.elsevier.com/solutions/scopus. (Accessed January 2021).

Falco, 2020. A behavioral activity monitoring with container support. GitHub Repository https://github.com/draios/oss-falco. (Accessed December 2020).

Forrest, S., Hofmeyr, S.A., Somayaji, A., Longstaff, T.A., 1996. A sense of self for unix processes. In: Proceedings 1996 IEEE Symposium on Security and Privacy. IEEE, pp. 120–128.

Fourati, M.H., Marzouk, S., Drira, K., Jmaiel, M., 2019. DOCKERANALYZER: Towards fine grained resource elasticity for microservices-based applications deployed with docker. In: 2019 20th International Conference on Parallel and Distributed Computing, Applications and Technologies. PDCAT, IEEE, pp. 220–225.

Gantikow, H., Reich, C., Knahl, M., Clarke, N.L., 2019. Rule-based security monitoring of containerized workloads. In: CLOSER. pp. 543–550.

Gantikow, H., Zöhner, T., Reich, C., 2020. Container anomaly detection using neural networks analyzing system calls. In: 2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing. PDP, IEEE, pp. 408–412.

Garcia-Teodoro, P., Diaz-Verdejo, J., Maciá-Fernández, G., Vázquez, E., 2009. Anomaly-based network intrusion detection: Techniques, systems and challenges. Comput. Secur. 28 (1–2), 18–28.

Haldar, V., Chandra, D., Franz, M., 2004. Semantic remote attestation: A virtual machine directed approach to trusted computing. In: USENIX Virtual Machine Research and Technology Symposium, Vol. 2004.

Heapster, 2020. A software for container cluster monitoring and performance analysis. GitHub Repository https://github.com/kubernetes-retired/heapster. (Accessed December 2020).

Howard, M., Lipner, S., 2006. The Security Development Lifecycle, Vol. 8. Microsoft Press Redmond.

Huang, D., Wu, H., 2018. Chapter 2 - virtualization. In: Huang, D., Wu, H. (Eds.), Mobile Cloud Computing. Morgan Kaufmann, pp. 31–64. http://dx.doi.org/10.1016/B978-0-12-809641-3.00003-X, URL http://www.sciencedirect.com/science/article/pii/B978012809641300003X.

InfluxDB, 2021. A platform for building and operating time series applications. InfluxDB https://www.influxdata.com. (Accessed January 2021).

Kamthania, S., 2019. A novel deep learning RBM based algorithm for securing containers. In: 2019 IEEE International WIE Conference on Electrical and Computer Engineering. WIECON-ECE, IEEE, pp. 1–7.

Keele, S., et al., 2007. Guidelines for Performing Systematic Literature Reviews in Software Engineering. Technical Report, Citeseer.

Khalimov, A., Benahmed, S., Hussain, R., Kazmi, S.A., Oracevic, A., Hussain, F., Ahmad, F., Kerrache, C.A., 2019. Container-based sandboxes for malware analysis: A compromise worth considering. In: Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing, pp. 219–227.

Kubernetes, 2020. An open-source system for automating deployment, scaling, and management of containerized applications. Kubernetes https://kubernetes.io. (Accessed December 2020).

Landis, J.R., Koch, G.G., 1977. The measurement of observer agreement for categorical data. Biometrics 33 (1), 159–174, URL http://www.jstor.org/stable/2529310.

Laskov, P., Düssel, P., Schäfer, C., Rieck, K., 2005. Learning intrusion detection: supervised or unsupervised? In: International Conference on Image Analysis and Processing. Springer, pp. 50–57.

Lei, L., Sun, J., Sun, K., Shenefiel, C., Ma, R., Wang, Y., Li, Q., 2017. SPEAKER: Split-phase execution of application containers. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, pp. 230–251.

Li, W., Xia, Y., Lu, L., Chen, H., Zang, B., 2019. TEEV: Virtualizing trusted execution environments on mobile platforms. In: Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. In: VEE 2019, Association for Computing Machinery, New York, NY, USA, pp. 2–16. http://dx.doi.org/10.1145/3313808.3313810.

Lunt, T.F., Jagannathan, R., Lee, R., Whitehurst, A., Listgarten, S., 1989. Knowledge based intrusion detection. In: Proceedings of the Annual AI Systems in Government Conference, Washington, DC.

Lwakatare, L.E., Karvonen, T., Sauvola, T., Kuvaja, P., Olsson, H.H., Bosch, J., Oivo, M., 2016. Towards DevOps in the embedded systems domain: Why is it so hard? In: 2016 49th Hawaii International Conference on System Sciences. Hicss, IEEE, pp. 5437–5446.

Martin, A., Raponi, S., Combe, T., Di Pietro, R., 2018. Docker ecosystem–vulnerability analysis. Comput. Commun. 122, 30–43.

Mattetti, M., Shulman-Peleg, A., Allouche, Y., Corradi, A., Dolev, S., Foschini, L., 2015. Securing the infrastructure and the workloads of linux containers. In: 2015 IEEE Conference on Communications and Network Security. CNS, IEEE, pp. 559–567.

Merkel, D., 2014. Docker: lightweight linux containers for consistent development and deployment. Linux J. 2014 (239), 2.

Morabito, R., 2017. Virtualization on internet of things edge devices with container technologies: A performance evaluation. IEEE Access 5, 8835–8850.

Morabito, R., Petrolo, R., Loscri, V., Mitton, N., Ruggeri, G., Molinaro, A., 2017. Lightweight virtualization as enabling technology for future smart cars. In: 2017 IFIP/IEEE Symposium on Integrated Network and Service Management. IM, IEEE, pp. 1238–1245.

OCI, 2021. Open container initiative. OCI https://opencontainers.org/. (Accessed January 2021).

OpenAttestation, 2021. An open source project providing a SDK for managing host integrity verification. OpenAttestation https://wiki.openstack.org/wiki/OpenAttestation. (Accessed January 2021).

Pahl, C., 2015. Containerization and the paas cloud. IEEE Cloud Comput. 2 (3), 24–31.

Pham, C., Estrada, Z., Cao, P., Kalbarczyk, Z., Iyer, R.K., 2014. Reliability and security monitoring of virtual machines using hardware architectural invariants. In: 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. pp. 13–24. http://dx.doi.org/10.1109/DSN.2014.19.

Pinto, S., Santos, N., 2019. Demystifying ARM TrustZone: A comprehensive survey. ACM Comput. Surv. 51 (6), http://dx.doi.org/10.1145/3291047.

Randazzo, A., Tinnirello, I., 2019. Kata containers: An emerging architecture for enabling MEC services in fast and secure way. In: Sixth International Conference on Internet of Things: Systems, Management and Security. IOTSMS, pp. 209–214. http://dx.doi.org/10.1109/IOTSMS48152.2019.8939164.

Ray, J.J., 1982. The construct validity of balanced likert scales. J. Soc. Psychol. 118 (1), 141–142.

Rodríguez, P., Haghighatkhah, A., Lwakatare, L.E., Teppola, S., Suomalainen, T., Eskeli, J., Karvonen, T., Kuvaja, P., Verner, J.M., Oivo, M., 2017. Continuous deployment of software intensive products and services: A systematic mapping study. J. Syst. Softw. 123, 263–291.

Sailer, R., Zhang, X., Jaeger, T., Van Doorn, L., 2004. Design and implementation of a TCG-based integrity measurement architecture. In: USENIX Security Symposium, Vol. 13, 2004, pp. 223–238.

Sayed, M.M., Azab, M., 2019. The time machine: Smart operation-resilience in presence of attacks and failures. In: 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference. IEMCON, IEEE, pp. 0127–0132.

SELinux, 2020. A flexible mandatory access control (MAC) for linux. GitHub Repository https://github.com/SELinuxProject. (Accessed December 2020).

Sha, L., 2001. Using simplicity to control complexity. IEEE Softw. 18 (4), 20–28.

Soltesz, S., Pötzl, H., Fiuczynski, M.E., Bavier, A., Peterson, L., 2007. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. In: Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007, pp. 275–287.

Souppaya, M., Morello, J., Scarfone, K., 2017. NIST Special Publication 800-19 – Application Container Security Guide. Technical Report, National Institute of Standards and Technology (NIST), http://dx.doi.org/10.6028/NIST.SP.800-190.

Srinivasan, S., Kumar, A., Mahajan, M., Sitaram, D., Gupta, S., 2018. Probabilistic real-time intrusion detection system for docker containers. In: International Symposium on Security in Computing and Communication. Springer, pp. 336–347.

Sultan, S., Ahmad, I., Dimitriou, T., 2019. Container security: Issues, challenges, and the road ahead. IEEE Access 7, 52976–52996.

Sysdig, 2020. A universal system visibility tool with native support for containers. GitHub Repository https://github.com/draios/sysdig. (Accessed December 2020).

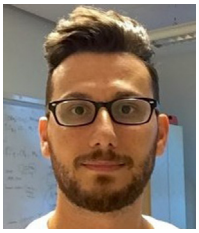
SystemTap, 2020. A free software (GPL) infrastructure to simplify the gathering of information about the running linux system. SystemTap https://sourceware.org/systemtap. (Accessed December 2020).

Wieringa, R.J., 2014. Design Science Methodology for Information Systems and Software Engineering. Springer.

Wohlin, C., 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, pp. 1–10.

Yun, H., Yao, G., Pellizzoni, R., Caccamo, M., Sha, L., 2013. Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In: 2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium. RTAS, IEEE, pp. 55–64.

Zou, Z., Xie, Y., Huang, K., Xu, G., Feng, D., Long, D., 2019. A docker container anomaly monitoring system based on optimized isolation forest. IEEE Trans. Cloud Comput..



Rodi Jolak is a post-doctoral researcher in software engineering at the joint Department of Computer Science and Engineering between Chalmers and University of Gothenburg in Sweden. His research activities focus on software engineering, software architectures, software design, human–computer interfaces, and security. Rodi received a Ph.D. degree in software engineering from the University of Gothenburg in 2020. He also practiced his role as a software engineer in industry for more than two years. See http://www.rodijolak.com for more.

Thomas Rosenstatter is a researcher in the Department of Digital Systems at RISE Research Institutes of Sweden and received a Ph.D. degree in Computer Science and Engineering at the Chalmers University of Technology in Sweden. His research focuses on cybersecurity in transport and the design and development of secure and resilient cyber–physical systems.

Mazen Mohamad received his master's degree in software engineering in 2016 from Chalmers University of technology in Sweden, and is currently working towards a Ph.D. in Computer Science and Engineering at the Software Engineering division of the Computer Science and Engineering department of Chalmers University of Technology and University of Gothenburg. His research interests include security assurance of cyber–physical systems.

Kim Strandberg is a senior security engineer at the Department of Research and Development at Volvo Cars and an industrial Ph.D. student in the Department of Computer Science and Engineering at Chalmers University of Technology, Sweden. He has two BSc. and two MSc. within the area of computer science and engineering. He has been working as an engineer within the IT area for 16 years and with automotive cyber security for about six years. His main research field is automotive cyber security, with an emphasis on secure and resilient automotive system design and development.

Behrooz Sangchoolie is a researcher in the Department of Electrification and Reliability at RISE Research Institutes of Sweden. He is the technical coordinator of National and European research projects in the area of dependable and secure computing and has served on many program committees for conferences and workshops in the area. His current research interests include the use of fault and attack injection experiments for dependability and security assessment of computer systems as well as to conduct interplay analyses between non-functional requirements such as safety and security.

Nasser Nowdehi is an automotive cybersecurity technical specialist in the department of Research and Development at Volvo Cars, Sweden. He has a Ph.D. degree in automotive cybersecurity from Chalmers University of Technology and a MSc. degree in computer systems and networks (specialized in cybersecurity) from the same university. His main research interests include intrusion detection systems, V2X security, and cyber resilient systems.

Riccardo Scandariato (Ph.D. 2004) is a full professor and the head of the Institute of Software Security at the Hamburg University of Technology (TUHH), in Germany. His work focuses on the design of secure and privacy-friendly applications, particularly in the realms of micro-services, IoT ecosystems, and cyber–physical systems.