A review of security risks and countermeasures in containers

Samuel Martínez-Magdaleno*,
Victor Morales-Rocha and Ramón Parra

National Laboratory of Information Technology,
Autonomous University of Ciudad Juarez,
Av. Del Charro 450 Nte, Cd. Juárez, Chihuahua, México
Email: al182948@alumnos.uacj.mx
Email: victor.morales@uacj.mx
Email: rparra@uacj.mx
*Corresponding author

Abstract: Containers are environments that allow software developers to package applications, along with their libraries, dependencies, and all the resources necessary for their operation. Due to the advantages of containers, compared to virtual machines, their use has increased in recent years. However, the nature of containers to share both, the resources, and the kernel of the host system, produces a variety of security problems. This paper describes how application containers work, to latter present a review of the security risks to this technology, as well as the countermeasures to mitigate them. A classification has been made of the risks as well as the security mechanisms used in this environment. Finally, according to different works that were analysed, a relation of the risks and the corresponding mechanisms to counteract them is presented.

Keywords: containers; container security; container risk; application containers.

Reference to this paper should be made as follows: Martínez-Magdaleno, S., Morales-Rocha, V. and Parra, R. (xxxx) 'A review of security risks and countermeasures in containers', *Int. J. Security and Networks*, Vol. X, No. Y, pp.xxx–xxx.

Biographical notes: Samuel Martínez-Magdaleno is currently pursuing his MS in Applied Computing at the Autonomous University of Ciudad Juarez. He received his BS in Computer Systems Engineering from the Technological Institute of Ciudad Juarez in 2017. His research interests include cloud computing and container technologies.

Victor Morales-Rocha is currently a Professor at the Autonomous University of Ciudad Juarez. He received his PhD from the Polytechnic University of Catalonia in 2009. His main area of interest is information security.

Ramón Parra is currently a full time Professor at the Autonomous University of Ciudad Juarez. He earned his PhD in Electrical Engineering at New Mexico State University in 1986. His main area of interest is cloud computing.

1 Introduction

Virtualisation technologies are growing in popularity and are the pillars on which the cloud computing paradigm works. Such is the case of containers, which are a lightweight virtualisation technology that seeks to solve the problems caused by the use of virtual machines, thus generating a greater number of adherents to the use of container technologies (Pahl et al., 2019; Martin et al., 2018). This is largely because containers are the standard method for microservices-based architecture deployments, allowing software to be developed in short periods, as well as to allow this software to be continually delivered and deployed, and thus generate cost savings (Sultan et al., 2019; Combe et al., 2016).

Therefore, it is important to describe the operation of containers, the risks when using them, and the existing security mechanisms, as well as the classification of the available literature for each of these areas. To achieve this, the present work is organised as follows: Section 2 discusses the concept of containers. Section 3 presents a classification of the intrinsic risks to the use of containers and their ways of implementation. Section 4 shows the existing security mechanisms for each category described in section 3. Section 5 shows the results of the selected studies. Finally, Section 6 presents the conclusions of this work.

2 Containers

Containers are a lightweight virtualisation technology or virtualisation at the operating system level. They are a response to the problems presented by solutions based on virtual machine technologies, such as lack of portability, slow provisioning of resources, and density of virtual machines. In the case of virtual machine technology, they use a hypervisor that is responsible for providing resource isolation for each virtual machine at the hardware levels, each of these machines includes a complete copy of the operating system with applications and data in addition to the host operating system (Pahl et al., 2019). In contrast, in container technology, they share the kernel and resources of the host operating system, allowing them to occupy fewer resources and start faster than a virtual machine since they are specific to a particular operating system family (Al-Dhuraibi et al., 2018). They are currently classified into operating system containers and application containers. For this investigation, only application containers are considered (Souppaya et al., 2017).

2.1 Application containers

Application containers, also known as application virtualisation, are intended to run one or more applications for each virtual instance. Currently, they are used to package an application or service with its dependencies, configuration files, and libraries, which are required for its operation (Pahl et al., 2019). In contrast to traditional application architectures, which are layered, and each layer has a server or virtual machine, container architectures usually have an application that is divided into components, each of these components has a role in specific and run in a separate container (Sultan et al., 2019).

2.2 Components and phases involved in the process of developing a containerised application

Containers share the kernel of the host operating system with the different containerised applications that are running. This allows them to have greater ease and agility of use, compared to virtual machines (Pahl et al., 2019). The components of the system and the kernel of the host machine take care of keeping each of the instances of the applications separately (Al-Dhuraibi et al., 2018). However, the shared use of the resources and kernel of the host system produces several security problems that we will discuss later in this work.

2.2.1 Image creation

During the first phase of the development of the containerised application, the application components are created and put into one or more images. The images are packages that include all the files necessary for the execution of a container (libraries, binaries, and configuration files, among others) (Pahl et al., 2019). The images use the concept of immutability, that is, they function as entities in which no modifications are made. Once an updated image is available, a new implementation of the container is implemented (Sultan et al., 2019).

2.2.2 Image testing and accreditation

The second phase consists of the process of testing and accreditation of images. This phase is generally managed by the developers who were responsible for packaging the application in the first phase. Immediately after performing the image assembly process, test automation tools are used, these tools perform tests to validate the operation of the final application. Finally, the security team oversees carrying out the accreditation of the images (Pittenger, 2016).

2.2.3 Image storage and retrieval

Once the image has been created and the accreditation and testing process has been completed, a place is required to store and retrieve the images. It is here where the third phase of the cycle provides us with services that perform this task (Souppaya et al., 2017). Registries are services that provide developers with a simple way to store images. These services can be deployed and used on-site or obtained from a provider such as Amazon or Docker (Martin et al., 2018).

2.2.4 Management of container administrators

In the last phase, the images are extracted from the registers, which can be carried out by a system administrator or by a trigger that is in an automation process (Martin et al., 2018). After the image was extracted, the orchestrator handles the deployment and management of container instances. This last phase is responsible for delivering a usable application, ready to respond to user requests (Sultan et al., 2019).

2.3 Implementation of container administrators

There are four main ways to implement containers. The first option is a local implementation. The second is a local implementation using a type 1 hypervisor that runs directly on the hardware (bare metal). The third option is to use a Type 2 hypervisor which runs on a host operating system. Finally, the last option is to use a cloud provider where we mount the container (Al-Dhuraibi et al., 2018).

3 Intrinsic risks to the use of containers

The main risks in using containers are circumscribed to the image, the orchestrator, the container, the host operating system, and the implementation. These risks occur in most container deployments, as they are intrinsic to the main components of this technology.

3.1 Image risks

Images are static packages that contain all the files, libraries, and binaries necessary for the execution of a certain application (Pahl et al., 2019). These packages are generally built using updated components, which are free of vulnerabilities for a few days or weeks (Pittenger, 2016). Therefore, it is required to update the components of the images periodically, which in turn, maintains the integrity of the image. Otherwise, the images are left with outdated and vulnerable components (Souppaya et al., 2017).

In addition to the risks of outdated and vulnerable components, malicious files may be intentionally or inadvertently included. This malicious program embedded in images can be obtained by using images for which its provenance is uncertain (Pittenger, 2016). Malicious files have the same capabilities as any other image component, so a malicious user can use it to attack other containers or hosts within their environment (Sultan et al., 2019).

There are also risks of images with configuration flaws. For example, an image where passwords are stored in plaintext for connections to databases or SSH. This allows an attacker to analyse the image content and violate other services (Souppaya et al., 2017).

3.2 Orchestrator risks

Orchestrators were designed assuming that users that interact with them will be system administrators and therefore would have a complete control of the environment (Sultan et al., 2019). Since orchestrators provide their own authentication service, it can lead to inappropriate practices in account management. For example, using passwords without proper security policies, not setting an adequate access level, or leaving unused accounts in the orchestrator. This is caused by the fact that this service is separated from other existing authentication services, in other words, they are managed less rigorously (Souppaya et al., 2017).

3.3 Container risks

Because individual containers can access other containers and the host operating system over the network, it can lead to unlimited network access, exposing other resources in the environment (Sultan et al., 2019). This risk is due in part to the fact that operational tools and processes are not designed to analyse virtualised network traffic used in a containerised environment and poorly separated virtual networks. If a container is compromised, it can scan the network for vulnerabilities that can be exploited (Souppaya et al., 2017).

3.4 Host operating system risks

Containers provide software-level isolation, yet the use of shared kernel results in a larger attack surface compared to hypervisors (Martin et al., 2018). However, there are specific container operating systems that reduce the attack surface. For example, they remove libraries and package managers that generally allow database applications or web servers to run (Sultan et al., 2019).

Although optimised operating systems are used in the container environment, these systems still provide basic libraries that are essential for the operation of the system. For example, primitive kernel libraries that are used to call or manage other processes (Souppaya et al., 2017). These libraries, like any other software component, are not vulnerability-free and as they are in the system architecture and the container manager, they have an impact on the host system and the containers it runs (Pittenger, 2016).

3.5 Implementation risks

Implementation risks are subject to the selected implementation mechanism. For example, if a local implementation is chosen, the containerised environment will have the inherent risks described previously. In this first implementation mechanism the user is responsible for implementing the appropriate security mechanisms to prevent attacks on each of the components of the containerised application or service. In the case of opting for a local implementation using type 1 or type 2 hypervisors, the security of the containers can be improved as mentioned in Mavridis and Karatza (2018), however, this generates inherent risks to the use of hypervisors and virtualisation (Asvija et al., 2019).

In addition to local implementations, an implementation with a cloud provider available on the market could be chosen. In this case, the provider is responsible for using the appropriate security mechanisms to guarantee full operation for the user. However, there are also security risks inherent in the PaaS service model (Kumar and Goyal, 2019).

Table 1 A classification of the intrinsic risks to the use of containers

Category	*Risks*
Image risks	Outdated components
	Configuration flaws
	Malware
	Plaintext passwords
Orchestrator risks	Unlimited administrator access
	Unauthorised access
	Wrong separation of network traffic
	Trust in the orchestrator node
Container risks	Unlimited network access
	Insecure configurations in the container runtime environment
	Clandestine containers
OS risks	Shared kernel
	Insecure OS components
	Wrong user accesses
	Manipulation of the OS file system
Implementation risks	Password reset attacks
	Hypervisor vulnerabilities
	Intrinsic risks of PaaS

Table 1 groups the risks mentioned in each of the categories previously described.

4 Security mechanisms

There are two main groups of security mechanisms for the protection of containers. The first one is to use software, which is usually found in the features and modules of Linux operating systems. The second is based on hardware, which is usually on the host machine where the container is running. The following sections discuss both, software, and hardware-based solutions.

4.1 Software-based solutions

Software-based solutions implement security mechanisms to protect containers in two main ways. The fist is found by default in the kernel of the operating system and is known as Linux kernel features (LKF). They are made up of four main components: namespaces, CGroups, capabilities, and SecComp (Sultan et al., 2019). The second way is through additional modules known as Linux security modules (LSM), they are additional modules that users can load or configure for additional protection to the containerised environment, with SELinux and AppArmor being the most widely used in the literature (Souppaya et al., 2017). The following sections describe each of these features and modules.

Namespaces

Namespaces are responsible for isolating and virtualising the resources a container can interact with (Martin et al., 2018). This includes file systems, network interfaces, hostname, user information, and processes (Sultan et al., 2019). In other words, it guarantees that a container can only see the resources it is allowed to interact with, and it cannot see the processes that are being executed in other containers (Souppaya et al., 2017). Therefore, one container is prevented from gaining privileged access to sockets or interfaces of another container.

CGroups

This Linux feature establishes how many processing resources (CPU), memory, input/output (I/O), and bandwidth is assigned to each process. It is responsible for limiting how many resources a container can use, unlike namespaces that delimit which resources the containers can see (Martin et al., 2018). This ensures that a container does not completely consume the resources of a system and leaves other containers or processes without resources (Sultan et al., 2019). For example, if we have a host operating system with 16 GB of total memory and 12 containers managed by the container manager, 1 GB can be allocated to each container, preventing it from interfering with the operations of other containers (Souppaya et al., 2017).

Capabilities

Linux capabilities break down permissions by layers or easy-to-use levels, which prevents unnecessary permissions from being assigned. For example, let us assume that we have a web server inside a container, which requires binding to a specific port, generally TCP port 80. This action requires a superuser permission assignment. In this example, the CAP_NET_BIND_SERVICE capability can be assigned, allowing the container to bind to the specific port without assigning superuser permissions to the entire container. By having this capability enabled, attacks will be limited exclusively to the operating gateway (Sultan et al., 2019).

SecComp

Secure computing profiles (SecComp) are a feature of the Linux kernel to filter out the system calls, reducing potential threats since most attacks take advantage of vulnerabilities that occur in system calls (Sultan et al., 2019). Docker includes default profiles that eliminate unsafe and unnecessary system calls, and custom profiles can be created and executed to further limit the capabilities of the container (Souppaya et al., 2017).

Linux security modules

According to Sultan et al. (2019), LSMs have their origin in 2001, when the US National Security Agency (NSA) proposed the inclusion of the security enhancement module (SELinux), starting with kernel version 2.5. Subsequently, the development of modules that support multiple security schemes began. However, they are not widely accepted within the security community, as there is no agreement on what their correct implementation should be. For example, a user may decide not to implement any module, which causes the default Linux capabilities module to be loaded. Despite this, performing a proper configuration or using automation tools that perform the configuration ensures a better level of security.

Mandatory access control

Mandatory access control (MAC) determines access to resources through access policies. These policies are controlled by a system administrator and are used by technologies, such as SELinux and AppArmor, allowing MAC to provide isolation (Win et al., 2014). For example, one of these technologies can be used to limit container access to file directories, processes, and network sockets (Souppaya et al., 2017). Consequently, the ability of a container to affect other containers or host machines is limited.

Decentralised Docker trust

Decentralised Docker trust (DDT) is a solution that appears as an alternative to address the risks of DoS attacks. It greatly reduces DoS attacks, in addition to providing a

service to verify the signatures of Docker images. This solution runs within the same operating system where the Docker container is running and works through REST requests that receive metadata which is stored in a blockchain. After instructions are saved in a blockchain, the DDT interpreter updates its internal database of keys and signatures, allowing the user to verify the signatures of the images (Xu et al., 2017).

Docker image vulnerability diagnostic system

Image vulnerabilities are mainly caused by users failing to verify images that are downloaded or uploaded to the image repository. Docker image vulnerability diagnostic system (DIVDS) is an answer to this problem as it detects known vulnerabilities and evaluates images based on a vulnerability score. The solution proposed by Kwon and Lee (2020) uses two modules that are in charge of collecting vulnerability information from different sources and once this information is available, they are compared with the image metadata to determine the vulnerabilities that could exist in the image.

PINE

PINE is a proposal that seeks to solve the security challenge of how to contain the computing resources used by the services in the containers. In general, when you have a containerised environment with services that share the storage of the system, you can run into the problem of one service occupying or delaying the execution of other services. PINE avoids this problem, as storage resources are allocated to services adaptively depending on their performance behaviour (Li et al., 2019).

X-Containers

Traditional containerisation technology is based on the principle of unique concern, which tells us that containers should only be concerned with serving a single task. This same principle was taken for the development of the X-Containers solution, where each of the containers runs an operating system library (LibOS). Unlike traditional container technologies, LibOS in conjunction with exokernels allow the management of hardware resources in applications without going directly through the kernel, this is achieved with an abstraction of operating system libraries (Shen et al., 2019).

ConPan

Generally, for finding outdated or vulnerable content in Docker packages, professionals and researchers create scripts or do the work manually. ConPan is a tool that performs this work in an automated way. The interaction with the tool is made through CLI or API calls. Its objective is to help researchers and professionals seeking to analyse the content of Docker containers and obtain data on the software packages that are installed (Zerouali et al., 2019).

4.2 Hardware-based solutions

Hardware-based solutions implement two main security mechanisms. These mechanisms are classified into trusted platform module (TPM) and trusted execution technology (TXT) (Sultan et al., 2019). TPM is a chip that is installed on the host machine and performs certain cryptographic operations to protect a device against unauthorised modification of firmware and software. Regarding TXT technology, it works similarly to TPM, but it is exclusive to Intel processors.

vTPM

Due to the widespread use of cloud computing and virtualisation, researchers have had to look for alternatives to TPM modules. Unlike traditional environments, where TPMs are installed on the host machines, they cannot be installed on virtual machines. This fact originated the development of virtual modules TPM or also known as vTPM. The modules can work directly on the kernel of the operating system or in a dedicated container (Hosseinzadeh et al., 2016).

Intel SGX

Intel Software Guard Extensions (SGX) is a set of instructions that allows Intel hardware to provide confidentiality and integrity to application data and code when the underlying software (hypervisor or kernel) has been compromised. This technology was developed in 2015 and is mainly responsible for container protection at the level of the cloud provider or host machine (Kumar and Goyal, 2019).

Table 2 A classification of security mechanisms for containers

Type	*Module*	*Mechanism*
Software	Linux kernel features (LKF)	Namespaces
		CGroups
		Capabilities
		Seccomp
	Linux security modules (LSM)	SELinux
		AppArmor
	Other	Decentralised Docker trust (DDT)
		Docker image vulnerability diagnostic system (DIVDS)
		PINE
		X-Containers
		ConPan
Hardware	vTPM	vTPM in the OS kernel
		vTPM in a dedicated container
	Trusted execution technology	Intel SGX

Table 2 groups the main security mechanisms for containers already described.

5 Review of risks and countermeasures

This section shows the results of the literature review regarding security mechanisms in containers, associating the risks, and countermeasures already described in previous sections. In this review, studies that provided some validation, evaluation, solution, opinion, or experience in the context of containers, light virtualisation, virtualisation, or IaaS and PaaS as middleware to support the execution of containers, were selected.

The results of this review are shown in Table 3. This table is organised as follows. The first column shows the reference and the year. Column two marks which security risks are taken into consideration and are arranged as follows:

1 image

2 orchestrator

3 container

4 operating system

5 implementation.

Column three describes which security mechanisms are used. Finally, column four indicates the type of contribution.

Most of the selected studies contribute via a review or solution. Regarding review studies, it was found that many authors mention the security features that exist by default in the Linux kernel, such as namespaces, CGroups, SecComp, and capabilities. This is largely because Linux is a popular operating system on servers for its low cost and freedom of configuration. However, it was noticed that there are not many studies that cover these characteristics from a more general point of view or considering other operating systems such as Windows.

Regarding the studies that provide a solution, it was found that most authors use features or modules of the Linux kernel. Also, the fact that some authors propose tests or use cases in controlled environments, can make the effectiveness in a business environment questionable.

Finally, we can see that certain studies describe a more specific solution. For example, in Kwon and Lee (2020), a DIVDS is proposed to detect vulnerabilities and evaluates images using a score.

Table 3 Summary of selected studies

References	*Type of risk*					*Mechanisms*	*Type of contribution*
	1	*2*	*3*	*4*	*5*		
Asvija et al. (2019)				X	X	vTPM, Intel SGX	Review
Azab and Domanska (2016)	X		X			LKF	Experience
Bacis et al. (2015)	X		X			LMS	Proposal
Bélair et al. (2019)	X	X	X		X	LKM, LSM, TPM	Review
Catuogno et al. (2018)	X	X				LKF, LSM	Methodology
Chelladhurai et al. (2016)	X	X				LKF, LSM	Methodology
Chen et al. (2019)	X	X	X	X		LKF, LSM	Framework
Combe et al. (2016)	X	X	X	X	X	LKF, LSM	Review
De Benedictis and Lioy (2019)	X	X	X	X	X	LKF, LSM, vTPM	Solution
Fernandez and Brito (2019)	X	X	X		X	Intel SGX	Solution
Garg and Garg (2019)	X		X			LKF, LSM	Experience
Gomes et al. (2018)	X		X			LKF	Tool
Gupta et al. (2019)			X	X	X	LKF, LSM, TPM	Review
Hertz (2016)	X		X			LKF, LSM	Review
Hosseinzadeh et al. (2016)			X	X	X	vTPM	Solution
Huang et al. (2019)	X	X	X			LKF	Analysis
Ibrahim and Hemayed (2019)				X	X	vTPM	Systematic review
Jian and Chen (2017)	X		X			LKF	Proposal
Kehrer et al. (2019)	X	X	X		X	LKF	Solution
Kong et al. (2019)			X		X	LKF	Solution
Kumar and Goyal (2019)			X	X	X	Intel SGX, Hardware VM, vTPM	Review
Kwon and Lee (2020)	X					DIVDS	Solution
Li et al. (2019)			X			Pine	Solution
Lin et al. (2018)	X		X			LKF, LSM	Systematic evaluation
Loukidis-Andreou et al. (2018)	X		X			LKF, LSM	Solution

Table 3 Summary of selected studies (continued)

References	*Type of risk*					*Mechanisms*	*Type of contribution*
	1	*2*	*3*	*4*	*5*		
Manu et al. (2016a)	X		X		X	LKF, LSM, VM	Study
Manu et al. (2016b)	X	X	X		X	Heuristics	Study
Martin et al. (2018)	X	X	X	X	X	LKF, LSM	Review
Mavridis and Karatza (2018)	X		X	X	X	LKF, LSM	Study
Pittenger (2016)	X		X	X		LKF	Opinion
Raj et al. (2016)	X		X			LKF, LSM	Study
Shen et al. (2019)		X		X	X	X-Containers	Solution
Souppaya et al. (2017)	X	X	X	X	X	LKF, LSM, TPM	Review
Sultan et al. (2019)	X	X	X	X	X	LKF, LSM, vTPM	Review
Tian et al. (2019)			X	X		LKF, Intel SGX	Evaluation
Tosatto et al. (2015)	X	X	X	X	X	LKF, LSM	Review
Win et al. (2014)					X	LSM, MAC	Proposal
Win et al. (2017)	X					LKF, LSM	Solution
Xu et al. (2017)	X					DDT	Solution
Yang et al. (2019)			X			LKF	Experiment
Zerouali et al. (2019)	X		X			ConPan	Tool

6 Conclusions

Container engines provide a good level of security by using predetermined security mechanisms that limit the use of resources and the scope of containers. However, as a result of specific configuration requirements of a production environment, this default configuration is usually modified. Most of the time, this can result in a less secure environment.

It has been observed that the main challenges of this technology arise when a container environment requires a specific configuration. For example, a host machine with containers that store web services, database containers, or containers that have a complete application.

Having a well-defined classification of security risks of container technology helps both developers and researchers to create more secure software execution environments by designing and implementing the appropriate mechanisms for each situation. Likewise, the proposed classification of existing security mechanisms presents researchers and developers with a general idea about various mechanisms that exist by default in the kernel of the operating system, the purpose of which is to reduce the risks previously described in this work.

Future research work may focus on proposing solutions that automate the configuration process on different modules and avoid manual configuration by the user, which could guarantee more secure container environments. Another possible direction for future research may be the combination of containers with other virtualisation technologies such as hypervisors type 1 or type 2 and exokernels, which would help to eliminate several of the risks inherent to the use of containers.

References

Al-Dhuraibi, Y., Paraiso, F., Djarallah, N. and Merle, P. (2018) 'Elasticity in cloud computing: state of the art and research challenges', *IEEE Transactions on Services Computing*, Vol. 11, No. 2, pp.430–477.

Asvija, B., Eswari, R. and Bijoy, M. (2019) 'Security in hardware-assisted virtualization for cloud computing – state of the art issues and challenges', *Computer Networks*, Vol. 151, pp.68–92.

Azab, A. and Domanska, D. (2016) 'Software provisioning inside a secure environment as Docker containers using STROLL file-system', *IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pp.674–683.

Bacis, E., Mutti, S., Capelli, S. and Paraboschi, S. (2015) 'Docker policy modules: mandatory access control for Docker containers', *IEEE Conference on Communications and Network Security (CNS)*, pp.749–750.

Bélair, M., Laniepce, S. and Menaud, J. (2019) 'Leveraging kernel security mechanisms to improve container security: a survey', *ARES '19: Proceedings of the 14th International Conference on Availability, Reliability, and Security*, pp.1–6.

Catuogno, L., Galdi, C. and Pasquino, N. (2018) 'An effective methodology for measuring software resource usage', *IEEE Transactions on Instrumentation and Measurement*, Vol. 67, No. 10, pp.2487–2494.

Chelladhurai, J., Chelliah, P. and Kumar, S. (2016) 'Securing Docker containers from denial of service (DoS) attacks', *IEEE International Conference on Services Computing*, pp.856–859.

Chen, J. et al. (2019) 'A container-based DoS attack-resilient control framework for real-time UAV systems', *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp.1222–1227.

Combe, T., Martin, A. and Di Pietro, R. (2016) 'To Docker or not to Docker: a security perspective', *IEEE Cloud Computing*, Vol. 3, No. 5, pp.54–62.

De Benedictis, M. and Lioy, A. (2019) 'Integrity verification of Docker containers for a lightweight cloud environment', *Future Generation Computer Systems*, Vol. 97, pp.236–246.

Fernandez, G. and Brito, A. (2019) 'Secure container orchestration in the cloud: policies and implementation', *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pp.138–145.

Garg, S. and Garg, S. (2019) 'Automated cloud infrastructure, continuous integration, and continuous delivery using Docker with robust container security', *IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, pp.467–470.

Gomes, J. et al. (2018) 'Enabling rootless Linux containers in multi-user environments: the udocker tool', *Computer Physics Communications*, Vol. 232, pp.84–97.

Gupta, R., Gupta, P. and Singh, J. (2019) *Security and Cryptography*, Elsevier, Cambridge.

Hertz, J. (2016) *Abusing Privileged and Unprivileged Linux Containers*, pp.1–53, NCC Group Publication, United Arab Emirates.

Hosseinzadeh, S., Laurén, S. and Leppänen, V. (2016) 'Security in container-based virtualization through vTPM', *IEEE/ACM 9th International Conference on Utility and Cloud Computing*, pp.214–219.

Huang, D., Cui, H., Wen, S. and Huang, C. (2019) 'Security analysis and threats detection techniques on Docker container', *IEEE 5th International Conference on Computer and Communications*, pp.1214–1220.

Ibrahim, F. and Hemayed, E. (2019) 'Trusted cloud computing architectures for infrastructure as a service: survey and systematic literature review', *Computers & Security*, Vol. 82, pp.196–226.

Jian, Z. and Chen, L. (2017) 'A defense method against Docker escape attack', *ICCSP '17: Proceedings of the 2017 International Conference on Cryptography, Security, and Privacy*, pp.142–146.

Kehrer, S., Riebandt, F. and Blochinger, W. (2019) 'Container-based module isolation for cloud services', *IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pp.177–186.

Kong, T. et al. (2019) 'A secure container deployment strategy by genetic algorithm to defend against co-resident attacks in cloud computing', *IEEE 21st International Conference on High Performance Computing and Communications*, pp.1825–1832.

Kumar, R. and Goyal, R. (2019) 'On cloud security requirements, threats, vulnerabilities, and countermeasures: a survey', *Computer Science Review*, Vol. 33, pp.1–48.

Kwon, S. and Lee, J. (2020) 'DIVDS: Docker image vulnerability diagnostic system', *IEEE Access*, Vol. 8, pp.42666–42673.

Li, Y. et al. (2019) 'PINE: optimizing performance isolation in container environments', *IEEE Access*, Vol. 7, pp.30410–30422.

Lin, X. et al. (2018) 'A measurement study on Linux container security: attacks and countermeasures', *Association for Computing Machinery*, pp.418–429.

Loukidis-Andreou, F., Giannakopoulos, I., Doka, K. and Koziris, N. (2018) 'Docker-sec: a fully automated container security enhancement mechanism', *IEEE 38th International Conference on Distributed Computing Systems*, pp.1561–1564.

Manu, A. et al. (2016a) 'A study, analysis and deep dive on cloud PAAS security in terms of Docker container security', *International Conference on Circuit, Power and Computing Technologies [ICCPCT]*, pp.1–13.

Manu, A. et al. (2016b) 'Docker container security via heuristics-based multilateral security-conceptual and pragmatic study', *International Conference on Circuit, Power and Computing Technologies [ICCPCT]*, pp.1–14.

Martin, A., Raponib, S., Combea, T. and Di Pietro, R. (2018) 'Docker ecosystem – vulnerability analysis', *Computer Communications*, Vol. 122, pp.30–43.

Mavridis, I. and Karatza, H. (2018) 'Combining containers and virtual machines to enhance isolation and extend functionality on cloud computing', *Future Generation Computer Systems*, Vol. 94, pp.674–696.

Pahl, C., Brogi, A., Soldani, J. and Jamshidi, P. (2019) 'Cloud container technologies: a state-of-the-art review', *IEEE Transactions on Cloud Computing*, Vol. 7, No. 3, pp.677–692.

Pittenger, M. (2016) 'Addressing the security challenges of using containers', *Network Security*, Vol. 2016, No. 12, pp.5–8.

Raj, A., Pai, S., Kumar, A. and Gopal, A. (2016) 'Enhancing security of Docker using Linux hardening techniques', *2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT)*, pp.94–99.

Shen, Z. et al. (2019) 'X-containers: breaking down barriers to improve performance and isolation of cloud-native containers', *ASPLOS '19: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp.121–135.

Souppaya, M., Morello, J. and Scarfone, K. (2017) 'Application container security guide', *NIST*, pp.800–190.

Sultan, S., Ahmad, I. and Dimitriou, T. (2019) 'Container security: issues, challenges, and the road ahead', *IEEE Access*, Vol. 7, pp.52976–52996.

Tian, D. et al. (2019) 'A practical Intel SGX setting for Linux containers in the cloud', *Ninth ACM Conference on Data and Application Security and Privacy*, pp.255–266.

Tosatto, A., Ruiu, P. and Attanasio, A. (2015) 'Container-based orchestration in the cloud: state of the art and challenges', *Ninth International Conference on Complex, Intelligent, and Software Intensive Systems*, pp.70–75.

Win, T., Tianfield, H. and Mair, Q. (2014) 'Virtualization security combining mandatory access control and virtual machine introspection', *IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pp.1004–1009.

Win, T., Tso, F., Mair, Q. and Tianfield, H. (2017) 'PROTECT: container process isolation using system call interception', *14th International Symposium on Pervasive Systems, Algorithms and Networks*, pp.191–196.

Xu, Q. et al. (2017) 'Blockchain-based decentralized content trust for Docker images', *Multimed Tools*, Vol. 77, pp.18223–18248.

Yang, T. et al. (2019) 'Docker's security analysis of using control group to enhance container resistance to pressure', *10th International Conference on Information Technology in Medicine and Education (ITME)*, pp.655–660.

Zerouali, A. et al. (2019) 'ConPan: a tool to analyze packages in software containers', *IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pp.592–596.