Security Audit of Docker Container Images in Cloud Architecture

Waheeda Syed Shameem Ahamed
Information Systems Security
Concordia University of Edmonton
Edmonton, AB, Canada
wsyedsha@student.concordia.ab.ca

Pavol Zavarsky
Information Systems Security
Concordia University of Edmonton
Edmonton, AB, Canada
pavol.zavarsky@concordia.ab.ca

Bobby Swar
Information Systems Security
Concordia University of Edmonton
Edmonton, AB, Canada
bobby.swar@concordia.ab.ca

2021 2nd International Conference on Secure Cyber Computing and Communications (ICSCCC) | 978-1-6654-4415-6/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICSCCC51823.2021.9478100

***Abstract*—Containers technology radically changed the ways for packaging applications and deploying them as services in cloud environments. According to the recent report on security predictions of 2020 by Trend Micro, the vulnerabilities in container components deployed with cloud architecture have been one of the top security concerns for development and operations teams in enterprises. Docker is one of the leading container technologies that automate the deployment of applications into containers. Docker Hub is a public repository by Docker for storing and sharing the Docker images. These Docker images are pulled from the Docker Hub repository and the security of images being used from the repositories in any cloud environment could be at risk. Vulnerabilities in Docker images could have a detrimental effect on enterprise applications. In this paper, the focus is on securing the Docker images using vulnerability centric approach (VCA) to detect the vulnerabilities. A set of use cases compliant with the NIST SP 800-190 Application Container Security Guide is developed for audit compliance of Docker container images with the OWASP Container Security Verification Standards (CSVS). In this paper, firs vulnerabilities of Docker container images are identified and assessed using the VCA. Then, a set of use cases to identify presence of the vulnerabilities is developed to facilitate the security audit of the container images. Finally, it is illustrated how the proposed use cases can be mapped with the requirements of the OWASP Container Security Verification Standards. The use cases can serve as a security auditing tool during the development, deployment, and maintenance of cloud microservices applications.**

Keywords—Docker, microservices, use cases, application container security, container security verification, security audit

I. INTRODUCTION

Containers are transforming enterprise applications by allowing deployment of applications rapidly, with reduced development overhead, lower cost, efficient use of resources and increased business agility. Microservices are developed in container technology as the applications are broken down into smaller components and independent of services. Docker is a containerization platform, which is a Platform as a Service (PaaS) product that uses operating system virtualization to deliver software applications as packages. Docker containers are easy to deploy in cloud infrastructure and are integrated with cloud providers like AWS, Microsoft Azure, Google, and Digital Ocean. However, direct access of containers to the host kernel is considered as one of the major security weaknesses of containers' security and privacy management. According to Forester's survey, 53% of the enterprises consider security as their biggest concern when developing their applications with container technology [1].

Fig. 1 shows a Docker architecture workflow for creating Docker images using the Docker registry and deployment of images into containers with cloud providers. Docker operates as a client-server architecture and has three major components: (i) Docker client, (ii) Docker host, and (iii) Docker registry. As shown in Fig 1, the Docker client is the primary component for interacting with Docker daemon. The client uses the commands "Docker build" to build the image, "Docker pull" to pull the image from the Docker host and "Docker run" to execute and create an image. The Docker host, which is the server, gets the request from the Docker client. The Docker registry is the public repository that stores the images. As shown in Fig 1, the Docker registry pulls the image from the Docker trusted registry of Docker that could be a private registry created in any of the public cloud providers. The public cloud providers like Amazon Elastic Container Registry, Microsoft Azure Container Registry, and Google Cloud Container Registry, store the images by fetching the images from Docker hub that is a trusted Docker registry on the cloud.

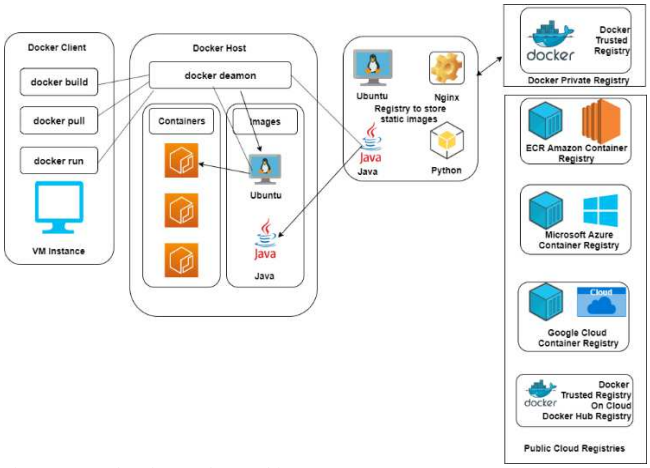

Fig 1. Example of a Docker architecture

The core components of Docker are images, registries, containers and Docker hosts, of which the containers and images are high-risk components. Docker on execution searches for a simple Dockerfile that defines the steps needed to generate a Docker image, which is then used to instantiate the containers [2]. Dockerfile is the configuration file that is needed to create an image and each instruction in the Dockerfile creates a layer in the image. As shown in Fig 2, the blocks of image layers are the result of the instructions executed in the Dockerfile. The topmost layer of the Docker image is the read/write layer and the below layers are only readable layers. The base images are the operating system images like Ubuntu and Alpine that form the underlying layer.

The major concern in deployment of containers is stability and security of images. Vulnerabilities can be in the package version of the base image, in the configuration file, authentication and authorization. The Docker image is an initial step for developing an application in the container

technology. The main objective of this paper is to propose a checklist to facilitate security audit of requirements of OWASP Container Security Verification Standards [3] aligned with the NIST SP 800-190 Application Container Security Guide [4].

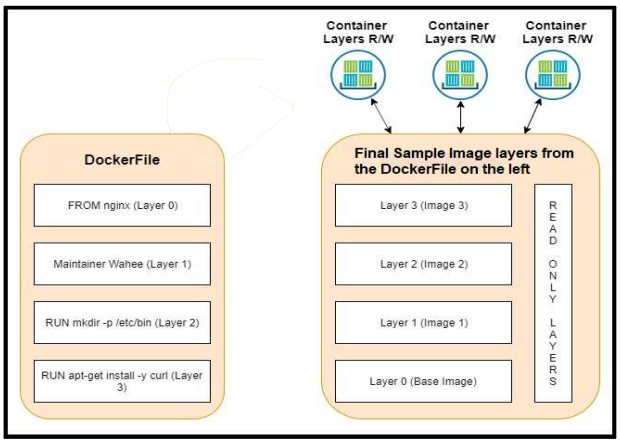

Fig 2. Layer representation of a container image

This paper is organized as follows, Section II provides the related works in the security of Docker images, Section III describes the development of use cases for security audit of container images. Finally, Section IV concludes the paper.

II. Related Works

The U.S. National Institute of Security and Technology (NIST) provides the application container security guidelines in the NIST SP 800-190 [4]. According to the security guidelines, the primary data to protect in the container technology are the images and the containers as these comprise the files and dependencies required to run the applications [4].

The research in Docker image security has focused on static code analysis of images, base image security, authentication, and authorization. Durante et al. performed a detailed analysis of Docker vulnerabilities and patches of the vulnerabilities and applied a static code analysis to understand how the vulnerabilities spread over time [5]. The findings in [5] indicate that image vulnerabilities can be detected by analyzing the behavior of the software, through regression testing or robustness testing. In relation to [5], for performing the robustness and regression testing, the checklist for security audit proposed in the following sections is for verification of security controls and configuration of an image during the development, deployment and testing phase of an application.

Zerouali et al. [6] analyzed relation between outdated containers and vulnerable packages installed in the container deployment. The images were extracted from Docker Hub and analyzed. The analysis of the state of the packages in Docker containers revealed that outdated container packages in images are one of the reasons for the presence of bugs and vulnerabilities in the Docker image. It was also observed that the Debian repository from the Docker was widely used [6].

Shu et al. [7] performed a study on the state of security vulnerabilities for both official and community images from the Docker Hub repository considered as the base images for writing a Dockerfile. Docker Image Vulnerability Analysis (DIVA) framework is proposed in [7] to automatically discover, download and analyze the Docker images for their security vulnerabilities. The experimentation analyzed a set of images and observed that (i) both official and community images contained an average more than 180 vulnerabilities, (ii) many images had not been updated for hundreds of days, and (iii) vulnerabilities were commonly propagated from parent or child images. Based on the observation in [7], there is a need for a systematic methods while analyzing the content of the Docker containers. The use case checklist presented in the following section of this paper is intended to provide a systematic method for security audit of Docker images during the development of microservices application.

Tunde et al. [8] evaluated a set of static and dynamic attack detection schemes using 28 real-world vulnerabilities in Docker cntainers. They compared the vulnerabilities detected through these detection schemes and provided a conclusion that dynamic anomaly detection schemes improve the detection rate of vulnerabilities compared to the static vulnerability schemes. For static vulnerability detection in Docker containers, an open-source tool Claire for detection and analysis of static vulnerabilities was used. For dynamic vulnerability detection, the evaluation was performed using an unsupervised anomaly detection.

The OWASP Container Security Verification Standards [3] were published in July 2019 to enhance security of the projects deployed with containers. The OWASP Container Security Verification Standards define security requirements or tests that can be used by architects, developers, testers, security professionals and even consumers to define a secure container with respect to its infrastructure [3]. While there has been a lot of research on container security related to the framework, performance, and vulnerability analysis, the use cases developed in the following section are intended to assist application developers in aligning their applications with the OWASP Container Security Verification Standards requirements to avoid vulnerabilities in the Docker image.

III. Use Cases and Checklist for Security Audit of Docker Container Images

To facilitate security audit of Docker images in container images, a checklist of use cases is developed to verify security of the images.. The developed use cases are intended to assist DevSecOps teams in testing and verification the applications for their compliance with the OWASP Container Security Verification Standards. Though there are integrated tools for finding the vulnerabilities in containers, the checklist can be used by the DevSecOps teams in the organization while developing, testing their applications and verifying compliance with the OWASP standard requirements and the NIST container security guidelines to reduce likelihood of vulnerabilities in the microservices applications. This section is divided into three sub-sections: (i) scanning the commonly used Docker images in a cloud environment and identification vulnerabilities in the images; (ii) developing the use cases based on the NIST application container security guidelines; and (iii) verification of the use cases with the OWASP Container Security Verification Standards.

A. Vulnerability scanning of the Docker images

In this section, five categories of images: (i) operating system images, (ii) database, (iii) language, (iv) web component, and (v) application platform images are scanned for their vulnerabilities. A Vulnerability Centric Approach (VCA) is applied for assessing security of the Docker images

by scanning the images and reviewing the configurations for possible vulnerabilities. The Google Container Registry (GCR), a private container image registry that runs on Google Cloud, and one of the cloud providers of Docker were used for scanning the images. In the experiments, a built-in API was used for scanning the image vulnerabilities, orchestration and performance measurement. In our experiments, in the Google Container Registry, a Vulnerability Scanning API was enabled to automatically scan the images once they were pushed to the container registry. To scan the images in this Registry, the following steps were performed:

1) A set of images is pulled from the Docker Hub Registry, a public repository for Docker images, into the Google Container Registry under a project created.
2) Once the images are pulled from the Docker Hub Registry repository, it is tagged with the project created in the Google Container Registry
3) The image tagged in Step 2 is pushed to the specific project in the Google Container Registry.
4) The images are automatically scanned for vulnerabilities once they are pushed to the container registry (since the Vulnerability Scanning API is enabled).
5) The vulnerabilities in the images are evaluated based on the Common Vulnerability Scoring System (CVSS) and for each image critical and high impact vulnerabilities are analyzed.

From data analyzed by scanning a set of images in each category of the above defined five categories, it was observed that vulnerabilities of the images are due to package version issues, privileges given to the default user and configuration issues in Dockerfile. Considering the vulnerabilities and exposure details, it is important to harden the security of images to mitigate likelihood of security incidents such as Denial of Service and Man in the Middle attacks.

B. *Use Cases for Security Audit of Docker Images*

A set of use cases are proposed in this section for security audit of Docker images. Based on the vulnerabilities and their CVE details, the use cases can be divided into four phases of the security audit of Docker images.

Phase 1: Docker base image inspection (Use Cases A1 – A5)

Docker image inspection is for dependent images required for an application. The Docker inspect command and the Docker Host Security tool, which is an audit tool, facilitate a detailed inspection of an image in the local repository. As the base image pull command is the first command in any Dockerfile and it is mostly fetched from the Docker Hub Registry, inspecting the top base images with their updated versions is essential. Trust is one of key factors in security and when relying on any open-source data it is important to verify whether the data is trusted. Docker Hub registry is a central repository and depends on the publisher of the official images whether to sign the image or not. The base images need to be checked for their signed integrity before they are included in the application. To check the trust of the base images, the following use cases A1 to A5 are proposed.

Use Case A1 - *Check whether the image is an official image:* This can be done by the Docker command "Docker search <image_name> --filter is-official=true". Official images in the Docker repository are the certified and trusted images.

Use Case A2 - *Check the trust of the base image:* The images in the Docker repository can be signed or not, so it is recommended to verify integrity of the images which are fetched from the public repository. Docker Content Trust provides the ability to use digital signatures when the data is sent and received from the repository [9]. This can be achieved by enabling the DOCKER_CONTENT_TRUST flag to 1 in the environment variables. When this flag is enabled and if an image is not a certified or a signed image, an error that the image is untrusted occurs and the image will not be extracted from the DockerHub.

Use Case A3 - *Check the content of the image*: Docker images are comprised of a set of layers and it is needed to verify the layers in the image to evaluate the space occupied by the images, their contents, and other details. The content of the image can be checked through the Docker inspect command which gives the details of network configuration, the hash value and layer details of the image in a JSON format. The Dive tool could be used to check the unused spaces and the efficiency of a Docker image [14]. This helps in uninstalling the packages which are not being used in the image as this could improve the performance of building the image.

Use Case A4 - *Uninstalling the unnecessary packages:* The size of a Docker image includes the dependent packages used to build an image. Including unnecessary dependencies in the Dockerfile increases the size of the image. It would slow down the deployment process and also increases the possibility of attacks. One way to uninstalling unnecessary packages is to include a Dockerignore file which consists of a list of patterns. The files and directories matching the patterns are excluded during the build process of a Docker image [15].

TABLE I. USECASE ID WITH DESCRIPTION AND MAPPING TO DEVSECOPS

Phase	Use Case ID	Description	DevSecOps
Docker Base Image Inspection	A1	Check whether the image is an official image	Development, Security, Operations
	A2	Check the trust of the base image	Operations, Security
	A3	Check the content of the image	Security
	A4	Uninstalling unnecessary packages	Operations
	A5	Disabling the build cache	Operations
Dockerfile Configurations	B1	Base image version	Development
	B2	Running apt-get install and apt-get update	Development
	B3	Using COPY instead of ADD	Development
	B4	HealthCheck for container images	Development, Operations
	B5	Secrets not stored in Dockerfile	Development
Image Authentication	C1	Image signing with DCT	Development, Security
	C2	Registry authentication	Development, Security

Phase	Use Case ID	Description	DevSecOps
Image Authorization	D1	Creating a user for each container	Development and Operations
	D2	Permissions for user ID is removed	Development, Security, Operations

Use Case A5 - *Disabling the build cache*: When an image is built through Dockerfile, it steps through each instruction of the Dockerfile, executing them in chronological order. When instruction is executed, it checks the cache for any existing image layer of that instruction, and if present it uses that image. But this could cause issues when there is a change in the instruction of the Dockerfile and the layer is used form cache. Therefore, it is preferred to create each and every layer of the image whenever the Dockerfile is executed to build.

Phase 2: Dockerfile Configurations (Use Cases B1 - B5)

When developing an application, it is a responsibility of the developer to consider security aspects of the application. Below are the configurations to consider while writing a Dockerfile as recommended in [10].

Use Case B1 - *Base image version:* The base image version is one of the reasons for the vulnerabilities in Docker container images. The usual ways to include the base image in the Docker file are through the latest tag or through a version tag. When using the latest tag, the immutability of the image is violated as the updates results in changes of the size and other features of the image. When using a specific version tag, for example, 1.0, it is just a name given by the publisher for that image and it may change or be deleted in the future. The best way to pull the image while mentioning in the Dockerfile is through hash value, the digest of an image. The image ID of an image is a SHA256 hash value which is unique to each image. When the image tag changes or the latest tag gets updated, a new SHA256 is created but the SHA256 remains the same for the image which is considered as the immutable identifier for the image.

Use Case B2 - *Running apt-get install and apt-get update*: In Linux terminology, the Advanced Package Tool (apt) is a command-line tool to interact with the package system. In Docker, the packages are installed and updated using this tool. When writing the Dockerfile, the apt-get command is used with the RUN command in the Dockerfile and it can be with multiple arguments. The apt-get install and apt-get update methods are mostly used in the Dockerfile when the packages need to be installed are updated with the RUN command [10]. There are two ways to install and update the packages through Dockerfile.

(i) apt-get update and apt-get install in a separate line: When the image is built, all the packages are updated with the latest version, but immutability of the image is lost [11].

(ii) *apt-get update and apt-get install in the same lines*: When the image is built, it is ensured that there is no intervention between the execution process as the cache is executed after all the packages are installed listed in the RUN command [11].

Use Case B3 - *Using COPY instead of ADD:* The commands which create the layers in the image are RUN, ADD and COPY command. Other commands create intermediate layers but not writeable layers.

ADD: This command copies files from the source to the destination directory, and if the destination directory is not available the command creates it implicitly. The command is also used to fetch data from remote URL's which may result in the Man-in-the-Middle vulnerability when there is a need to download data directly into a secure location and the attacker can modify the content of the file being downloaded [12]. The advantage of using ADD is when we need to extract an archive files or zipped files that are used. But this could result in vulnerabilities in the archive files. Most of the vulnerabilities in the images are through their inbuilt packages and libraries.

COPY: When using COPY, and when file is copied from a remote URL, a secure TLS connection is used and the source is validated [12]. When an external file is copied, it needs to be authenticated, which is secured and the preferred method for file transfer.

Use Case B4 - *HealthCheck for Container Images:* Healthcheck for container images is to determine the state of the services whether they are running or not. In this healthcheck, we can specify the interval time on how frequently the state of the container has to be checked. When the healthcheck is enabled, the container can have one of the three states: (i) Starting - when the container is starting, (ii) Healthy - when the healthcheck command in the Docker file executes successfully and the container is up in running state, and (iii) Unhealthy - If the container takes a long time to start, then timeout interval in the healthcheck command can be defined to give the timeout error after a specified time.

Use Case B5 - *Secrets not stored in Dockerfile*: A Docker image is shared in different infrastructures and designed to run in any environment. If any of the packages need credentials, they have to be passed in the runtime and not built into the image. When a Dockerfile contains any hardcoded secrets or configuration values, they are not encrypted. A secret can be a certificate, SSH key or password. Docker secrets can be created and invoked when container is initialized. The secrets are created and sent to the Docker engine through TLS connection.

Phase 3: Image Authentication (Use Cases C1 - C2)

The integrity of an image assures that the image is not changed when it is pulled from the repository and not from any third-party resources. Below are the scenarios through which integrity of the image can be verified.

Use Case C1 - *Image signing with Docker Content Trust (DCT)*: Docker Content Trust is a feature of Docker used to verify the client-side or runtime verification of integrity of an image and verification of the publisher of specific image tags through digital signatures. An image can also be signed and pushed to the Docker registry with the Docker trust command and it is built on top of the Notary feature in Docker. For signing an image, the Docker registry and Notary server are required. Notary in Docker is a collection of a trusted content that includes the sign collections of the consumers and publishers of the images. The Notary is responsible for operations necessary to create, manage, and distribute metadata of the images to ensure integrity and freshness of the content of the image.

Use Case C2 - *Registry Authentication:* Registry stores the container images and should be authenticated before being

accessed by a user. DockerHub provides the option of creating private registries for the organizations where they can store their internal images. This registry is integrated with the cloud environment like Google Container Registry. Authentication is an important step to consider in the process as the images are stored in the registry and if an illegitimate user gets access to the registry, there is a risk of exploiting the image. If the image is configured as private, the registry access should be given to the users who are working on the applications. The effective way to authenticate the registry is to provide the account credentials through encrypted Dockercfg config file. This file secures credentials when images are pushed or pulled from private registries.

Phase 4: Image Authorization (Use Cases D1 - D2)

Use Case D1 - *Creating a user for each container:* Images which are created as container images should not be created with a root user as this might cause access restriction bypass by an external users. By default, when base image is pulled from the Docker Hub by the FROM command of the Dockerfile, the root privileges are by default. Containers provide isolation, but there is a direct interaction with the host kernel from the underlying operating system. To reduce potential risks, containers should always run under non-privileged user. Users can be created based on groups working on the application or an individual user for each container.

Use Case D2 - *Permissions for user ID are removed:* The user ID is referred to as UID and is managed by the Linux kernel. The kernel system calls to determine whether the requested user has privileges to access the container. This is achieved by creating a user under a group and providing privileges for a specific user to run the container. Unwanted permissions for a user should be removed to prevent possible escalation attacks resulting in an unauthorized access to the container resources or in a Denial of Service.

C. Verification of compliance with the OWSAP Container Security Verification Standards

The use cases of the previous paragraphs can be used to verify compliance of the container images with the OWASP Container Security Verification Standards. The OWASP Container Security Verification Standards provides a framework of security requirements and controls for design, development and testing of container-based solutions [3]. In OWASP Container Security Verification Standards there are three levels of security requirements:

TABLE II. SECURITY AUDIT OF COMPLIANCE OF DOCKER APPLICATION CONTAINER IMAGES WITH REQUIREMENTS OF THE OWASP CONTAINER SECURITY VERIFICATION STANDARDS

Verification of OWASP CSVS	Levels of OWASP CSVS			Use Case ID													
	L1	L2	L3	A1	A2	A3	A4	A5	B1	B2	B3	B4	B5	C1	C2	D1	D2
Verify that within each container image, a new user is created, which is then used to perform all operations within the containers.			✓													✓	
Verify that all base images are explicitly specified, using their hash instead of name and tag.			✓						✓								
Verify that the signature of each image is verified before productive usage.			✓		✓									✓			
Verify that only required software packages are installed in the images.	✓	✓	✓				✓										
Verify the Dockerfile uses the COPY directive instead of the ADD directive unless the source is fully trusted.	✓	✓	✓								✓						
Verify that Docker's health checking functionality is used for all containers and their status is monitored.		✓	✓									✓					
Verify that garbage collection is enabled on image registries and running on a regular basis.																	
Verify that after a container has been actively accessed (e.g., for troubleshooting), is deleted and replaced by a new instance (container) of the image		✓	✓														
Verify Docker content trust is enabled and enforced.					✓									✓			
Verify that Dockerfile or Docker-compose file does not contain any sensitive information like API keys and passwords.													✓				
Verify that an odd number of image registries (e.g., DTR) with a minimum of three registries is used.			✓														
Verify that containers are always created based on the most recent corresponding image and not local caches.	✓	✓	✓					✓		✓							
Verify that all images are using tags whereas only production/master is allowed to use the default latest tag.		✓	✓						✓								
Verify that the CI/CD tools and systems are connected to the Docker infrastructure to enable changes in nodes, images, or the network to be tested and rolled out fully automated.		✓	✓														

Level 1 (L1): This is the minimum requirement on security needed for all containers. A container-based infrastructure achieves the CSVS Level 1 if it adequately defends against well-known security threats that are easy to discover and easy to abuse [3].

Level 2 (L2): Requirements for the containers which are developed for sensitive business logic that requires additional protection. A container-based infrastructure achieves this level when it adequately defends against most of the risks associated with the containers [3].

Level 3 (L3): This is the highest level of requirements in the container security verification standards. The L3 requirements are for the container-based solutions requiring significant levels of security verifications, such as in critical infrastructure [3]. The failure to achieve this level in the organization could result in a significant impact on the organization's operations.

The use cases presented in this paper can be used for security audit of container images to verify compliance with the OWASP Container Security Verification Standards. Each verification rule can be mapped with a use case as shown in Table II. The use cases based security audit can be used during the development, testing and audit of cloud microservices applications.

IV. Conclusion

In this paper, critical vulnerabilities of Docker images are identified and use cases are developed to facilitate security audit of Docker application container images performed by the development, operations and security teams throughout the application development lifecycle. The use cases presented in the paper follow the U.S. NIST guidelines on application containers security and assist in verification of compliance with the OWASP Container Security Verification Standards. For the future work, the set of use cases can be extended for vulnerabilities in the orchestration of containers that is important for cloud microservices application container interactions.

References

[1] J. Shetty, "A State-of-Art Review of Docker Container Security Issues and Solutions". American International Journal of Research in Science, Technology, Engineering & Mathematics, 2017.

[2] T. Yarygina, A. Bagge, "Overcoming Security Challenges in Microservice Architectures". 11-20. 10.1109/SOSE.2018.00011, 2018.

[3] OWASP, "OWASP Container Security Verification Standards", July 2019.

[4] M. Souppaya, J.Morello, K. Scarfone, "Application Container Security Guide", NIST Special Publication 800-190, September 2017.

[5] A. Duarte, N. Antunes"An Empirical Study of Docker Vulnerabilities and of Static Code Analysis Applicability". 27-36. 10.1109/LADC.2018.00013, 2018.

[6] A. Zerouali, T. Mens, G. Robles, Jesus M.Gouzalez-Barohana, "On the Relation Between Outdated Docker Containers, Severity Vulnerabilities, and Bugs", 2018.

[7] R. Shu, X. Gu, and W. Enck, "A Study of Security Vulnerabilities on Docker Hub", *7th ACM Conference on Data and Application Security and Privacy*. ACM, pp. 269–280, 2017.

[8] O. Tunde-Onadele, J. He, T. Dai and X. Gu, "A Study on Container Vulnerability Exploit Detection," *2019 IEEE International Conference on Cloud Engineering (IC2E)*, Prague, Czech Republic, 2019, pp. 121-127. DOI: 10.1109/IC2E.2019.00026.

[9] "Content Trust in Docker", Available: https://docs.Docker.com/ engine/security/trust/content_trust/

[10] Jorge Silva, "9 Common Mistakes in Dockerfile", runnable, Available at : https://runnable.com/blog/9-common-Dockerfile-mistakes.

[11] Docker Documentation, "Best Practices for writing Dockerfile". Available:https://docs.Docker.com/develop/develop-images/Dockerfile_best-practices/

[12] Liran Tal, "Why you should use COPY instead of ADD when building Docker image".Available at: https://dev.to/lirantal/why-you-should-use-copy-instead-of-add-when-building-Docker-images-5fkj

[13] "The New Norm: TrendMicro Security Predictions for 2020", TrendMicro,Available::https://www.trendmicro.com/vinfo/us/security /research-and-analysis/predictions/2020.

[14] "How to analyze and explore the contents of Docker images", October 2019, Available at:https://www.ostechnix.com/how-to-analyze-and-explore-the-contents-of-Docker-images/.

[15] "DockerFile Reference", Available : https://docs.Docker.com/engine/ reference/builder/#Dockerignore-file