IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON SECURE COMMUNICATION FOR THE NEXT GENERATION 5G AND IOT NETWORKS

Received February 4, 2020, accepted February 17, 2020, date of publication February 27, 2020, date of current version March 11, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2976874

DIVDS: Docker Image Vulnerability Diagnostic System

SOONHONG KWON AND JONG-HYOUK LEE, (Senior Member, IEEE)
Protocol Engineering Laboratory, Department of Software, Sangmyung University, Cheonan 31066, South Korea

Corresponding author: Jong-Hyouk Lee (jonghyouk@smu.ac.kr)

This work was supported by the research grant from Sangmyung University.

ABSTRACT Since the development of Docker in 2013, container utilization projects have emerged in various fields. Docker has the advantage of being able to quickly share application build environments among developers through container technology, but it does not provide security guarantees for known security vulnerabilities inside Docker images. Since the Docker images are shared without a means of security vulnerability diagnostic, polluted Docker images can be distributed so that the Docker-based application build environments can be easily collapsed. In this paper, we introduce a Docker Image Vulnerability Diagnostic System (DIVDS) for a reliable Docker environment. The proposed DIVDS diagnoses Docker images when uploading or downloading the Docker images from a Docker image repository.

INDEX TERMS Container, Docker image security, Docker image vulnerability evaluation.

I. INTRODUCTION

Since the deployment of the Docker platform, various research and container utilization projects on container-based virtualization technologies have emerged. Nowadays Docker containers are widely used to provision multiple applications over shared physical hosts in a lighter form than a traditional Virtual Machine (VM) platform [1]. Docker creates a quantitative form of a Docker image, which can be easily shared and distributed through a Docker image repository (e.g., Docker Hub, Gitlab, Quay.io, etc.). However, while the rapid sharing and deployment of Docker images has the advantage of allowing developers to share a variety of real-time application deployment environments [2], [3], as there is no separate Docker image vulnerability diagnosis procedure, the Docker platform is easily exposed to various security attacks. Docker images are divided into official and community Docker images, and both Docker images would have known security vulnerabilities. Older package containing vulnerabilities in Docker images can be exposed to various types of attacks (e.g., denial of service, gain privilege, etc.) [4], [5]. For instance, there have been cases of distributing cryptocurrency mining programs using Docker images, and this attack showed that various attacks can be performed using Docker images regardless of the target operating system (OS). Therefore, security for Docker images has become one of the main concerns in establishing a reliable Docker environment.

This Docker image vulnerability problem comes from the fact that users do not perform a separate security verification work on downloaded Docker images or Docker images to be uploaded in a Docker image repository. Therefore, we propose a Docker Image Vulnerability Diagnostic System (DIVDS) that detects known vulnerabilities and evaluates Docker images based on vulnerability scores.

The main contributions of this paper are the following.

- The proposed DIVDS enables users to identify the vulnerability level of each Docker image through a developed vulnerability evaluation process, which is based on a combined relationship of vulnerable software packages and vulnerability information in a Docker image.
- The proposed DIVDS prevents users from downloading or uploading vulnerable Docker images to a Docker image repository. It thus makes possible to maintain a reliable Docker based application build environment.

The remainder of this paper is organized as follows. Section II introduces Docker and Docker images. Section 3 presents an overview of the proposed DIVDS. Section 4 shows how the DIVDS detects known vulnerabilities and how the vulnerabilities are evaluated for Docker images. Section 5 evaluates the proposed DIVDS and discusses possible limitations of the proposed system. Section 6 concludes this paper.

The associate editor coordinating the review of this manuscript and approving it for publication was Antonio Skarmeta Gómez.

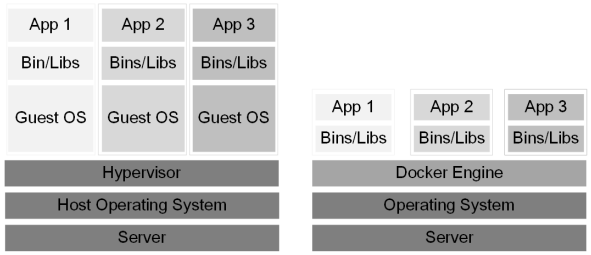

FIGURE 1. Virtual machine and Docker architecture.

II. RELATED WORK

A. DOCKER

Docker is an open source container platform for developing, deploying, and running applications released by Docker. Inc in March 2013. Containers are generally compared to existing Virtual Machine (VM). In the case of existing VM platforms, an entire guest OS is virtualized on a host OS. This method has an advantage of being able to virtualize and use various OSs on the host OS and relatively simple to use, but there is a limitation that it is difficult to use in the operating environment due to the disadvantage that the environment is relatively heavy and slow. Container technology has emerged to address these limitations. Containers, along with the Linux kernel, use kernel features such as Control groups (Cgroups) and namespaces to separate processes so that they can run independently. Namespaces, underlying technology for the Docker container platform, are a feature of the Linux kernel that provides isolated and independent spaces for file system, process, network, IPC, hostname, and user. Cgroups is a feature of the Linux kernel that provides control over the resources (e.g., memory, CPU, I/O, network, device, etc.) of a Linux system and provides ownership of the group for use by specific users on the resources [2], [5].

Docker was first based on Linux kernel isolation technology called LinuX Container (LXC), which had started with chroot on Linux, but after version 11 of Docker, it did not designate support for LXC as a default execution environment. Docker has replaced LXC with a self-implementation called libcontainer. Finally, Docker was improved by a runC that complies with the standard Open Container Initiative (OCI) specifications of namespaces and Cgroups. Figure 1 shows the structural differences between VM and Docker [5].

B. DOCKER IMAGES

Docker constructs a container environment based on its Docker images [2]. Docker packages all the files which need for application, library, middleware, OS, network configuration, etc. into a quantitative form called a Docker image. The Docker image is uploaded to a remote repository called Docker image repository used to share the Docker images among users. In the case of the repository, it can be configured as public or private repositories. If we upload a Docker image through the *docker pull image name: tag* command without giving parameters related to a separate remote repository,

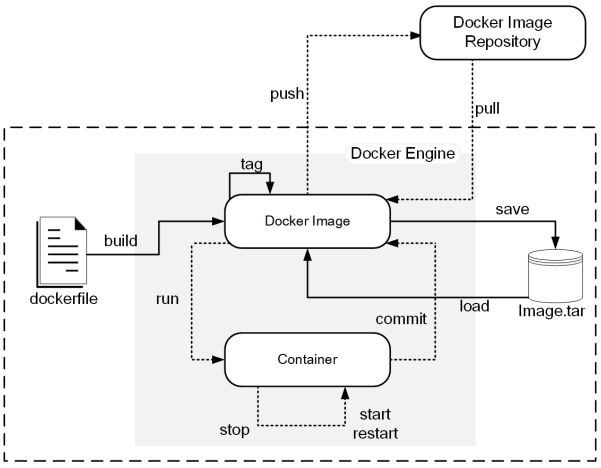

FIGURE 2. Docker Image deployment process.

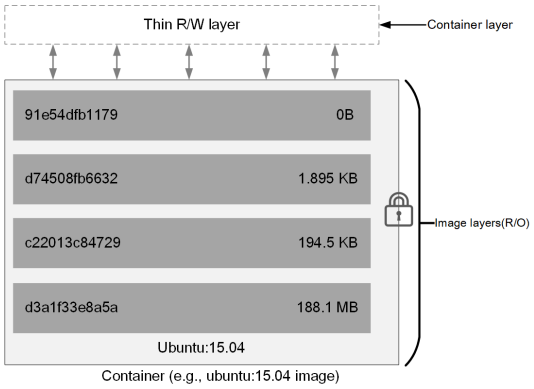

FIGURE 3. Ubuntu container layer.

the Docker image is uploaded to Docker Hub by default, but it can be also uploaded to a private repository with the private repository information in the command. Figure 2 shows how a Docker image is created by the *docker build* command and uploaded or downloaded to the remote repository.

The Docker platform allows its Docker image to be created as a container by the *docker run* command. When the *docker run* command is performed through the Docker client, the Docker first copies the Docker image to the file system area managed by the Docker, adding one layer to the top of the base image layer to create the container. Afterwards, the Docker platform shows the user an integrated view using the union file system in which the Docker image layer structure consisting of multiple layers appears as a single file system. In addition, all tasks that the user reads and writes through commands such as *apt-get install* or *apt-get upgrade* inside the container are recorded in the container layer created at the top of the base image layer. This is because the Docker engine controls the file system and places the base image layer in a read only mode and the container layer in a read/write mode. Figure 3 shows that a container image layer is created at the top of the base image layer as the user performs its work in the container.

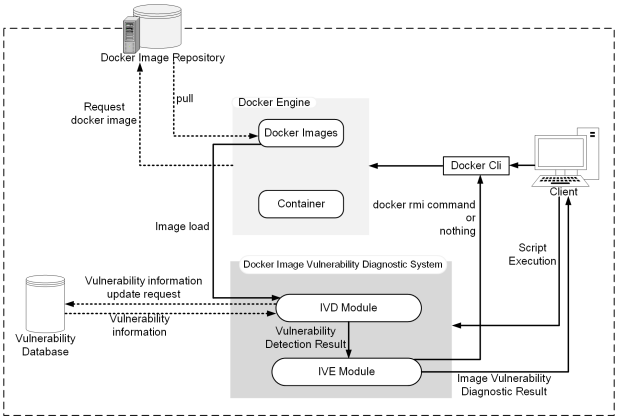

FIGURE 4. Architecture of the DIVDS.

TABLE 1. Metadata being extracted from a Docker image.

Data Field	Description
Image ID	256-bit ID for each unique Docker image
Image Name	Identifier of each Docker image following a specific naming policy
Last Update Time	Accurate data and last update time for Docker images
Layer ID	Unique ID for each layer and the relationship between layers
Commands	Docker image build related commands

TABLE 2. Data structure for the IVD vulnerability database.

Data Field	Description
Timestamp	Accurate analysis time by Clair
Vulnerability ID	Unique CVE identifier to identify the vulnerability
Severity Ranking	Severity rating of each vulnerability
Description of CVE	Description of each identified vulnerability
Associated Packages	Name and exact version of the package associated with each vulnerability
Layer ID	Specific image layer in which the vulnerability exists

III. OVERVIEW OF THE DIVDS

In this section, we present an overview of the proposed DIVDS. The architecture of the DIVDS is shown in Figure 4. First, the user downloads a Docker image from the Docker Hub using the *docker pull* command (e.g., *docker pull imageName:tag*) to diagnose the Docker image if the Docker image contains known vulnerabilities. The five fields of information described in Table 1 are extracted from the downloaded image for security vulnerability analysis [6]. Last Update Time, which shows the exact data and the latest update time of the image among the data fields specified in Table 1, can be checked by using the *docker inspect* command, which can check the system architecture and OS of the Docker image.

The IVD module and IVE module are then used to detect known vulnerabilities in Docker images and evaluate them by calculating the image vulnerability score.

The IVD module operates on the basis of Clair, an open source tool designed to detect known vulnerabilities within the Docker image [7]. The IVD module collects several types of vulnerability information from a set of resources (e.g., Ubuntu CVE Tracker, Debian Security Bug Tracker, RedHat

FIGURE 5. Docker image vulnerability detection result.

FIGURE 6. CVE information on the whitelist.

Security Data, etc.), shown in Table 2. It also performs static analysis on a Docker image to extract version information and OS metadata from each layer for all software packages installed in the Docker image [8]. After completing all the preparation for the Docker image vulnerability detection, the IVD module compares the metadata of the Docker image with the software package vulnerability information received from a set of resources to detect unsafe known vulnerabilities. The software package vulnerability detection results of the IVD module represent the layer ID of the analyzed Docker image, the name of the analyzed Docker image, the total number of vulnerabilities present in the Docker image, the number of unapproved vulnerabilities, the CVE severity, the vulnerable package name and version, and the CVE information. For the CVE severity, the National Vulnerability Database (NVD) is defined using the Common Vulnerability Scoring System (CVSS). The CVSS assigns severity scores based on formulas that include exploitable and impact metrics, while the NVD provides severity levels of "Low", "Medium" and "High" based on the CVSS scores [9]. Figure 5 shows an example of the vulnerability detection result with the 'java:latest' Docker image by the IVD module.

Since the IVD module has been developed based on Clair, users can approve vulnerabilities in Docker images using a whitelist that allows the Docker images containing the whitelisted vulnerabilities. Figure 6 shows an example of the whitelisted vulnerability information 'CVE-2019-14855:gnupg' in the 'java:latest' image.

The IVE module calculates the vulnerability score of the Docker image and diagnoses the Docker image by

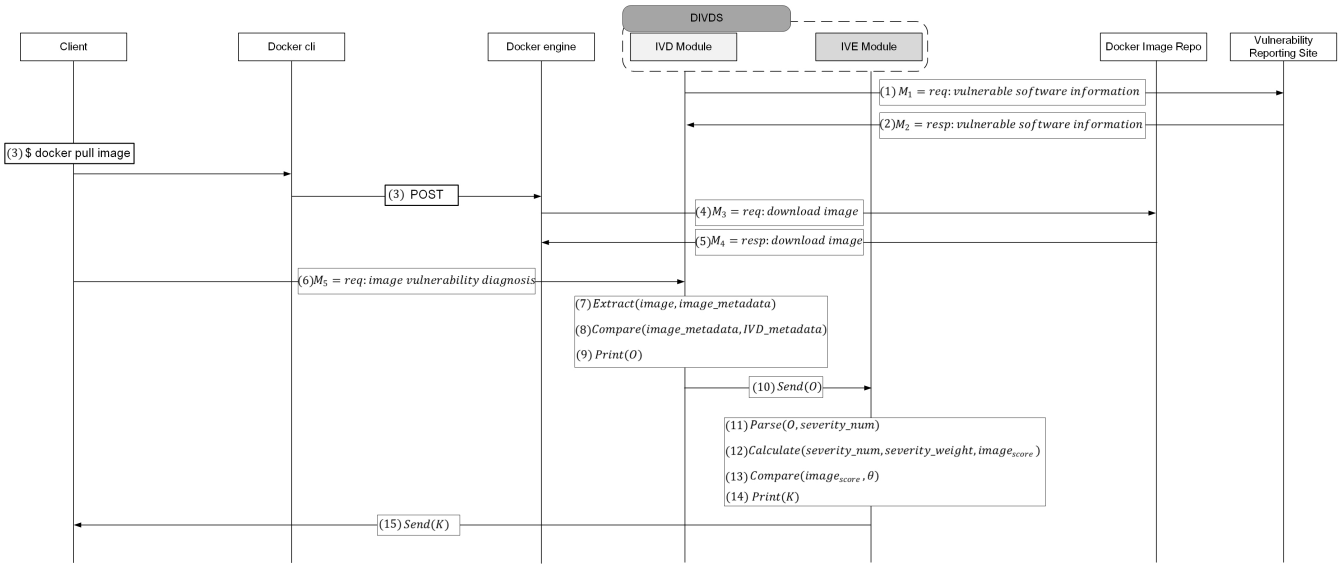

FIGURE 7. Docker image download process of the DIVDS.

TABLE 3. Notation of DIVDS operation procedure.

Notation	Description
$image$	Docker image downloaded from a Docker image repository
$Extract(x, y)$	Extract y from x
$Compare(x, y)$	Compare x with y
$Print(x)$	Print x data
$Send(x)$	Send x data
$Parse(O, severity\_num)$	Extract the number of the CVE severity from O
$Calculate(x, y, z)$	Calculate z using x and y

comparing the calculated Docker image vulnerability score with the vulnerability threshold score. First, the IVE module receives a Docker image vulnerability detection result from the IVD module and parses the CVE severity from the Docker image vulnerability detection result to count the number of severity-specific counts. After counting the number of counts per severity, the vulnerability score of the Docker image is calculated according to the defined score per severity. Although the score by severity of the vulnerability is based on the CVSS score, it is possible to define the score by severity according to the user's settings. If the Docker image vulnerability score is calculated based on the score by defined vulnerability severity and is above the vulnerability threshold score, the image is diagnosed as a vulnerable image and cannot be uploaded or downloaded. The vulnerability threshold score can also be defined by the user. As an example, Figure 7 shows the DIVDS operating procedure when downloading a Docker image from a Docker image repository. Table 3 shows the notation used in Figure 7.

The operation process of the proposed DIVDS is as follows.

1) The IVD module of the DIVDS continuously requests vulnerable software package metadata to the vulnerability reporting site
2) Upon receiving requests from the IVD module, the corresponding vulnerability reporting site provides vulnerable software package metadata to the IVD module
3) The client requests the Docker image from Docker image repository through '*docker pull image*' command
4) Docker engine, which receives the '*docker pull*' command through the Docker cli, requests for the Docker image download from Docker image repository
5) Docker image repository provides the requested Docker image by the client to the Docker engine
6) Client who downloads the Docker image from Docker image repository loads Docker image into the DIVDS to diagnose vulnerability in the Docker image
7) For the Docker image loaded from Step (6), the IVD module of the DIVDS extracts the software package metadata, OS metadata, and the metadata defined in Table 1 (*image_metadata*) installed in the Docker image
8) The IVD module detects vulnerable software package installed in the Docker image by comparing the vulnerable software package metadata (*IVD_metadata*) with the metadata of the Docker image extracted in Step (7)
9) The IVD module outputs the vulnerable software package information existing in the Docker image (O) through the comparison process performed in Step (8)
10) The IVD module loads the Docker image vulnerability information output (O) in Step (9) into the IVE module for Docker image vulnerability evaluation
11) The IVE module extracts and counts the number of CVE severities (*severity_num*), one of the values required for the Docker image vulnerability evaluation, from the Docker image vulnerability result (O) received from the IVD module in Step (10)
12) The IVE module calculates the Docker image vulnerability score ($image_{score}$) using the number of CVE

severities (*severity_num*) extracted in Step (11) and the severity weight defined in Table 4 (*severity_weight*)

13) The IVE module compares the Docker image vulnerability score ($image_{score}$) calculated in Step (12) with the Docker image vulnerability threshold score (θ) defined in the IVE module
14) The IVE module outputs the Docker image vulnerability evaluation result (K), which is the result of comparison between the Docker image vulnerability score ($image_{score}$) and Docker image vulnerability threshold score (θ) performed in Step (13)
15) The IVE module provides the Docker image vulnerability evaluation result (K) from Step (14) to the client

Figure 7 shows the DIVDS's operation when downloading a Docker image. If the Docker image vulnerability score $image_{score}$ exceeds the Docker image vulnerability threshold score θ by comparing $image_{score}$ with θ in Step (12), the Docker image is deleted by the Docker engine. Otherwise, the Docker image is stored in the local image storage.

IV. MAIN MODULES OF THE DIVDS

In this section, we present the detailed operations of the IVD and IVE modules.

A. IVD MODULE

The main function of the IVD module is to detect known vulnerabilities in the Docker image based on metadata received from a set of resources (e.g., Ubuntu CVE Tracker, Debian Security Bug Tracker, RedHat Security Data, etc.) where software package vulnerability information is stored.

The Docker image vulnerability detection function is represented as Algorithm 1. All software package information installed within the Docker image is provided by the IVD module(D) as previously mentioned in Section 2. Set $D = \{D[1], D[2], D[3], \ldots, D[\alpha]\}$, where α represents the total number of software packages installed in the Docker image. Define $D.index$ as the index of the software package. $D.layerID$ represents the layer ID of the Docker image, and $D.name$ describes the name of the software package installed in the Docker image. $D.version$ represents the version of software package. The IVD vulnerability database (V) is obtained by a collector that stores V from a set of resources. Then $V = \{V[1], V[2], \ldots, V[\beta]\}$ is defined, where β represents the total number of entries of V. Each entry of V consists of the index of the entry, $V.index$; Layer ID of the Docker image, $V.layerID$; Name of the vulnerable software package, $V.name$; Version of the vulnerable software package, $V.version$; the CVE name of the vulnerable software package, $V.cvename$; the CVE severity of vulnerable software package, $V.severity$; the CVE description of vulnerable software package, $V.description$. In Algorithm 1, D and V are used as inputs. The output is represented by entry O of vulnerable software packages installed in the Docker image. Then O is defined as $O = \{O[1], O[2], \ldots, O[L]\}$, where L is the last index for the list.

In Algorithm 1, each software package is checked with a layer ID, an attribute of D. The layer ID of the Docker image, represented by $D[x].layerID$, to compare with the attribute of V ($V[y].layerID$). If an entry in V ($V[y].layerID$) equals $D[x].layerID$, then it is compared to the attribute of V ($V[y].name$) with a vulnerable software package installed in the Docker image represented by $D[x].name$. Then, if there is an entry in V ($V[y].name$) that is the same as $D[x].name$, then the version of the vulnerable software package, represented by $D[x].version$, is compared to the attribute of V ($V[y].version$). This process is repeated α times, so all software packages installed in the Docker image are checked. At the end of Algorithm 1, O is returned as output, and the user can get a list of vulnerable software packages installed in the Docker image.

Algorithm 1 Docker Image Vulnerability Detection

```
1:  begin
2:  input D, V
3:  index ← 1
4:  for x ← 1 to α do
5:      for y ← 1 to β do
6:          if D[x].layerID = V[y].layerID then
7:              if D[x].name = V[y].name then
8:                  if D[x].version = V[y].version then
9:                      O[index]←(D[x].index, V[y].index)
10:                     index ← index + 1
11:                 end if
12:             end if
13:         end if
14:     end for
15: end for
16: output O
17: end
```

B. IVE MODULE

The main function of the IVE module is to evaluate Docker image vulnerabilities by counting the number of the CVE severity in the Docker image vulnerability detection result obtained through the IVD module and calculating the Docker image vulnerability scores according to the severity scores defined in Table 4 for each vulnerability. The vulnerability of the Docker image is evaluated by comparing the calculated Docker image vulnerability score with the user-defined Docker image vulnerability threshold score. To evaluate the vulnerability of Docker images, we define weights by the CVE severity as shown in Table 4. The scores by the CVE severity were selected based on the CVSS scores, and not limited to them. Those scores by the CVE severity can be defined according to a user definition.

Since CVEs printed in the Docker image vulnerability detection results obtained through the IVD module may not be vulnerabilities. In such cases, the whitelist function allows the user to disable detection for the CVE, which may not be a vulnerability. For this reason, if a user sets up CVE

TABLE 4. Weight of severity.

Severity	Weight
$w_{unknown}$	0.0
$w_{negligible}$	0.5
w_{low}	2
w_{medium}	5.45
w_{high}	7.95
$w_{critical}$	9.5

information for software packages that are not vulnerable to whitelist, the IVD module prints its CVE severity as "Approved" when detecting Docker image vulnerability. In addition, the IVE module does not take into account "Approved" vulnerabilities when evaluating Docker image vulnerability. In other words, the IVE module conducts the vulnerability evaluation of the vulnerable software packages that are installed within a Docker image that the user does not set in the whitelist.

The Docker image vulnerability score calculation by the IVE module is performed according to Equation (1) below, where sv represents the CVE severity of the vulnerable software package installed in the Docker image. And, r_{sv} is the total number of vulnerable software package CVEs from the result of the Docker image vulnerability detection (O) obtained through the IVD module. w_{sv} means the weight for each CVE severity defined in Table 4.

$$image_{score} = \sum_{i=1}^{r_{sv}} w_{sv} \tag{1}$$

Equation (1) calculates the Docker image vulnerability score($image_{score}$) and compares it with the Docker image vulnerability threshold score θ. By comparing $image_{score}$ and θ according to Equation (2), if $image_{score}$ is higher than θ, upload or download is impossible for the Docker image. However, according to Equation (3), when $image_{score}$ is less than θ, upload or download is possible for the Docker image.

$$image_{score} = \sum_{i=1}^{r_{sv}} w_{sv} > \theta \tag{2}$$

$$image_{score} = \sum_{i=1}^{r_{sv}} w_{sv} < \theta \tag{3}$$

Figure 8 shows the result of performing a Docker image vulnerability evaluation on the 'java:latest' Docker image through the IVE module. As shown in Figure 8, the vulnerable Docker image is detected.

Next, to understand how to perform a Docker image vulnerability evaluation by the IVE module, a description is provided with Algorithm 2, where the input O is obtained from Algorithm 1, and the output K is a result of the IVE module.

Each entry in O consists of index of the software package $dindex$ installed in the Docker image and index of the IVD database, where $vindex$ is defined as $vindex = \{vindex[1], vindex[2], \ldots, vindex[\mu]\}$. μ means the entry

```
latest: Pulling from library/java
5040bd298390: Pull complete
fce5728aad85: Pull complete
76610ec20bf5: Pull complete
60170fec2151: Pull complete
e98f73de8f0d: Pull complete
11f7af24ed9c: Pull complete
49e2d6393f32: Pull complete
bb9cdec9c7f3: Pull complete
Digest: sha256:c1ff613e8ba25833d2e1940da0940c3824f03f802c449f3d1815a66b7f8c0e9d
Status: Downloaded newer image for java:latest
Untagged: java:latest
Untagged: java@sha256:c1ff613e8ba25833d2e1940da0940c3824f03f802c449f3d1815a66b7f8c0e9d
Deleted: sha256:d23bdf5b1b1b1afce5f1d0fd33e7ed8afbc084b594b9ccf742a5b27080d8a4a8
Deleted: sha256:0132aeca1bc9ac49d397635d34675915693a8727b103639ddee3cc5438e0f60a
Deleted: sha256:c011315277e16e6c88687a6c683e388e2879f9a195113129a2ca12f782d9fcf9
Deleted: sha256:3181aa7c07970b525de9d3bd15c4c3710a2ab49fd5927df41e5586d9b89b1480
Deleted: sha256:b0053647bc72f97b7a9709a505a20a7a74a556c6aa025979e36532ff3df7cb8d
Deleted: sha256:0877f4904e80b44741cc07706b19c6d415724b20128f4b26ee59faec9a859416
Deleted: sha256:dbf7b16cf5d32dfec3058391a92361a09745421deb2491545964f8ba99b37fc2
Deleted: sha256:4cbc0ad7007fe8c2dfcf2cdc82fdb04f35070f0e2a04d5fa35093977a3cc1693
Deleted: sha256:a2ae92ffcd29f7ededa0320f4a4fd709a723beae9a4e681696874932db7aee2c
Delete your docker image
297.4
```

FIGURE 8. DIVDS execution screen.

of the IVD database. $O[i]$ also contains information about the CVE severity $cve\_severity$ associated with vulnerable software packages, defined as $cve_severity = \{cve_severity[1], cve_severity[2], \ldots, cve_severity[\tau]\}$where τ represents the number of the CVE severity list. Therefore, the CVE severity list is defined as $O[i].C\_severity[j]$ [10].

The purpose of Algorithm 2 is to calculate the Docker image vulnerability score, compare it with the Docker image vulnerability threshold score, and then evaluate the Docker image vulnerability. First, $O[i].cve\_severity$, number by the CVE severity, and weight by the CVE severity are needed to calculate the Docker image vulnerability score. In case of the CVE severity weight, it is converted into cve_weight value as defined in Table 4, and cve_weight can be denoted through $E[i].severity$ of IVE module(E). The CVE severity name must be parsed from O to calculate the total score of the CVE severity ($image_{score}$). To get the number of the CVE severity, parsing is possible in O through each O entry $O[o\_num].C\_severity$. For each $C\_severity$, $C[cve\_severity].status$ (vulnerable software package the CVE severity status) is identified as either "Approved" or "Unapproved". As the CVE information is set in the whitelist, 0 is added if status is "Approved". Otherwise 1 is added to $numCvesv$. Using $total\_score$ and $numCvesv$, we can calculate the $image_{score}$ as shown in Equation (1). Thereafter, when the $image_{score}$ is higher than the Docker image vulnerability threshold score (θ), the corresponding Docker image is deleted. Otherwise the Docker image is stored. The output of Algorithm 2 is K, which can be seen in Figure 8.

V. EVALUATION

We have confirmed that the Docker does not provide security guarantees for vulnerable software packages installed within the Docker image. In this paper, when uploading or downloading Docker images, the proposed DIVDS is applied to the Docker environment. To demonstrate the applicability of the DIVDS, we evaluated whether the DIVDS works correctly to evaluate the Docker image vulnerability. To demonstrate the applicability of the DIVDS, we have established a Docker environment in the x86_64 CentOS 7 environment and installed the IVD module and the IVE module that make

Algorithm 2 Docker Image Vulnerability Evaluation

```
begin
input O
image_score ← 0
for i ← 1 to r_sv do
    cve_weight ← weight of E[i].severity
    total_score ← 0
    io ← 0
    for all elements of O[i].index do
        io ← io + 1
        o_num ← O[i].index[io]
        ic ← 0
        numCvesv ← 0
        for all elements of O[o_num].C_severity do
            ic ← ic + 1
            cve_severity ← O[o_num].C_severity[ic]
            if C[cve_severity].status ="Approved" then
                numCvesv ← numCvesv + 0
            else
                numCvesv ← numCvesv + 1
            end if
        end for
        total_score ← total_score + numCvesv
    end for
    image_score ← total_score × cve_weight
end for
if image_score > θ then
    Delete image
else
    Save image
end if
K ← image_score
output K
end
```

TABLE 5. DIVDS result for the official Docker images.

Imagename:tag	Image vulnerability score	Image vulnerability result
ubuntu:latest	76.2	Save image
java:latest	297.4	Delete image
centos:latest	7.95	Save image
wordpress:latest	183.45	Save image

up DIVDS. The IVE module uses the CVE severity-specific scores defined in Table 4 for the calculation of image vulnerability scores. In addition, for the Docker image vulnerability threshold score θ, 200 points were set as default for the evaluation of Docker image vulnerabilities through the IVE module.

Table 5 below shows the results of vulnerability evaluates performed on the official Docker images 'ubuntu:latest', 'java:latest', 'centos:latest', and 'wordpress:latest' with 10M+ downloads from the Docker Hub [11].

In the existing Docker environment, when the Docker image upload or download is performed, the Docker image vulnerability evaluation is not carried out, so it is exposed to various types of security attacks (e.g., denial of service, gain privilege, execution code, etc.).

To compensate for this, the DIVDS currently provides vulnerability evaluation for known security vulnerabilities that exist inside Docker images that are not provided in existing Docker environments. However, for the IVD module, there is a disadvantage that the DIVDS are based on Clair that performs static analysis, making it difficult to detect anomalies [12], [13] that may occur during a container execution. To secure the disadvantages of the current DIVDS, the application of dynamic analysis techniques to detect anomalies [4], [14]–[17] that may occur during container execution is necessary.

VI. CONCLUSION

In this paper, we proposed the Docker Image Vulnerability Diagnostic System (DIVDS) to establish a reliable Docker environment that can quickly share and distribute various execution environments. The proposed DIVDS mainly consists of the IVD module and IVE module. The IVD module detects vulnerable software packages installed in a Docker image. The IVE module calculates a Docker image vulnerability score based on the detection results obtained from the IVD module and compares with a Docker image vulnerability threshold score to decide if the image is allowed or not for use. By checking the operation of DIVDS through algorithms and execution screens, we showed that the system can establish a reliable Docker environment. However, there is a limitation that it is difficult to detect abnormal behavior that may occur at container runtime based on the static analysis of the current DIVDS. In the future, we will apply deep learning technique [16], [17] to the DIVDS and develop DIVDS based on static and dynamic analysis to detect not only vulnerability evaluation of images but also abnormal behaviors that can occur at container runtime.

REFERENCES

[1] R. Shu, X. Gu, and W. Enck, "A study of security vulnerabilities on docker hub," in *Proc. 7th ACM Conf. Data Appl. Secur. Privacy (CODASPY)*, 2017, pp. 269–280.
[2] D. Merkel, "Docker: Lightweight Linux containers for consistent development and deployment," *Linux J.* vol. 2014, no. 239, p. 2, 2014.
[3] C. Boettiger, "An introduction to docker for reproducible research," *ACM SIGOPS Oper. Syst. Rev.*, vol. 49, no. 1, pp. 71–79, Jan. 2015.
[4] O. Tunde-Onadele, J. He, T. Dai, and X. Gu, "A study on container vulnerability exploit detection," in *Proc. IEEE Int. Conf. Cloud Eng. (ICE)*, Jun. 2019, pp. 121–127.
[5] R. Yasrab, "Mitigating docker security issues," 2018, *arXiv:1804.05039*. [Online]. Available: http://arxiv.org/abs/1804.05039
[6] A. Zerouali, T. Mens, G. Robles, and J. M. Gonzalez-Barahona, "On the relation between outdated docker containers, severity vulnerabilities, and bugs," in *Proc. IEEE 26th Int. Conf. Softw. Anal., Evol. Reeng. (SANER)*, Feb. 2019, pp. 491–501.
[7] *Clair*. Accessed: Jan. 14, 2020. [Online]. Available: https://github.com/quay/clair/tree/master/Documentation
[8] O. Henriksson and M. Falk, "Static vulnerability analysis of docker images," M.S. thesis, Blekinge Inst. Technol., Karlskrona, Sweden, 2017.
[9] *CVSS Score*. Accessed: Jan. 14, 2020. [Online]. Available: https://www.first.org/cvss/
[10] J.-H. Lee, S.-G. Sohn, B.-H. Chang, and T.-M. Chung, "PKG-VUL: Security vulnerability evaluation and patch framework for package-based systems," *ETRI J.*, vol. 31, no. 5, pp. 554–564, Oct. 2009.

[11] *Docker Hub*. Accessed: Feb. 1, 2020. [Online]. Available: https://hub.docker.com
[12] A. Zimba, Z. Wang, and H. Chen, "Multi-stage crypto ransomware attacks: A new emerging cyber threat to critical infrastructure and industrial control systems," *ICT Express*, vol. 4, no. 1, pp. 14–18, Mar. 2018.
[13] A. Leandros Maglaras, K.-H. Kim, H. Janicke, M. A. Ferrag, and J. Tiago Cruz, "Cyber security of critical infrastructures," *ICT Express*, vol. 4, no. 1, pp. 42–45, Mar. 2018.
[14] P. P. W. Pathirathna, V. A. I. Ayesha, W. A. T. Imihira, W. M. J. C. Wasala, E. A. T. D. Edirisinghe, and N. A. G. Arachchilage, "Security testing as a service with docker containerization," in *Proc. 17th Int. Conf. Adv. ICT Emerg. Regions (ICTer)*, Sep. 2017, pp. 1–7.
[15] S. Srinivasan, A. Kumar, M. Mahajan, D. Sitaram, and S. Gupta, "Probabilistic real-time intrusion detection system for docker containers," in *Proc. Int. Symp. Secur. Comput. Commun.* Singapore: Springer, 2018, pp. 336–347.
[16] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2017, pp. 1285–1298.
[17] A. Shenfield, D. Day, and A. Ayesh, "Intelligent intrusion detection systems using artificial neural networks," *ICT Express*, vol. 4, no. 2, pp. 95–99, Jun. 2018.

SOONHONG KWON is currently pursuing the bachelor's degree in computer engineering from Sangmyung University, Cheonan, South Korea. He is currently a Research Assistant with the Protocol Engineering Laboratory, Sangmyung University. His research interests include protocol engineering, network security, and system security.

JONG-HYOUK LEE (Senior Member, IEEE) received the Ph.D. degree in computer engineering from Sungkyunkwan University, Suwon, South Korea. He was with INRIA, France, for IPV6 vehicular communication and security research. He was an Assistant Professor with TELECOM Bretagne, France. In 2013, he moved to Sangmyung University, Cheonan, South Korea. He is currently an Author of the Internet Standards: IETF RFC 8127, IETF RFC 8191, and IETF RFC 8691. His research interests include protocol engineering and performance analysis. He was a recipient of the Best Paper Award from the IEEE WiMob 2012, the 2015 Best Land Transportation Paper Award from the IEEE Vehicular Technology Society, the Haedong Young Scholar Award, in 2017, and the IEEE Systems Journal Best Paper Award from the IEEE Systems Council, in 2018. He was a Tutorial Speaker at the IEEE WCNC 2013, the IEEE VTC 2014 Spring, and the IEEE ICC 2016. He was introduced as the Young Researcher of the month by the National Research Foundation of Korea Webzine, in 2014.