Docker container hardening method based on trusted computing

Yuang Shen and Xuejun Yu
Chaoyang District, Beijing, China, Beijing University of Technology, 100124
chaojwudilan@126.com; yuxuejun@bjut.edu.cn

Abstract. In view of the incomplete isolation of docker, the image file is easy to be tampered with, and the problem of insecure container operation. Based on the analysis of the existing isolation mechanism and security enhancement technology of docker container, this article uses trusted computing technologies such as cryptographic algorithms, integrity measurement, real-time monitoring, etc., a hardening method for docker containers is proposed. The feasibility of the reinforcement method was verified by experiments. The results show that this method can realize that docker is in a trusted and secure environment during the entire process of downloading the image from the container to the container, and ensuring that the container and the image file are not tampered with. When the container is enabled, the system resources are in a monitorable state, which greatly improves the credibility and security of the docker container.

1. Introduction
Docker is a lightweight virtualization technology. Compared with traditional virtualization methods, docker is different in that traditional virtual machine technology is to virtualize a set of hardware from the host machine, run a complete operating system on it, and run the required application processes on the system. The application process in the docker container runs directly on the host's kernel. It does not have its own kernel inside, nor does it perform hardware virtualization. Therefore, the docker container is more portable than the traditional virtual machine virtualization method.

But neither the docker container nor the underlying Linux kernel technology it relies on has matured, far from being as tested as the virtual machine technology. At least for now, containers have not provided the same level of security guarantees as virtual machines [1].

Aiming at some existing problems of docker, scholars and some engineers have also proposed some methods to improve the isolation and reinforcement of docker. Wang Juan, School of Computer, Wuhan University, etc. Designed and implemented a credible enhanced docker container based on trusted computing related technologies for container and image tampering, malicious processes and unauthorized communication of docker [2]. Matthew Bajor uses TLS-enabled docker daemon. To use the docker daemon, provide a client certificate generated from the same CA as the certificate used by docker / Swarm. Steven Vaughan-Nicols published an article in 2014 saying that open source container company docker announced that it finally added digital signature verification to docker images [3]. Logscape shows the use of docker commands and netcat to import data into Logscape and start monitoring docker resource utilization [4]. Daniel Walsh proposed using SELinux to control the access allowed to docker container processes [5].

What this article does is to address the problems of image tampering, container runtime attacks, and relatively poor isolation during docker container startup [6]. Propose a container reinforcement method
using trusted computing technology to construct a trusted chain from image download to container startup [7]. From the acquisition of the image to the start of the container, the entire process uses trusted computing technology, docker container-related supporting technologies and tools, performs trusted measurement and trusted container hardening, and resources, file systems, processes, Container communication, etc. for real-time monitoring and resource limitation. Ensure that the entire docker-initiated process reduces the risk of attack and minimizes losses in the event of an attack.

2. Docker container reinforcement method

2.1. The traditional method of docker isolation mechanism
The isolation implementation principle of docker itself is summarized as follows:
The use of Namespaces to achieve the isolation of the system environment [8], Namespaces allows a process and its child processes to obtain from the shared host kernel resources (network stack, process list, mount point, etc.) an isolation area that is only visible to them, allowing the same All processes in Namespace are aware of each other's changes and know nothing about external processes, as if running in an exclusive operating system.
Use CGroups to limit the resource usage of this environment. For example, on a 16-core 32GB machine, only containers use 2 core 4GB. Using CGroups can also set weights for resources, calculate usage, and control the start and stop of tasks (processes or threads);

2.2. Docker hardening method based on trusted computing
This paper proposes a docker container hardening method based on trusted computing. Reinforcement of containers by establishing a trusted chain by downloading from container to container launch. According to the objects involved in the docker container startup process, the container startup process is divided into two phases, namely the image download phase and the container startup phase.
Trusted security of the image is a prerequisite for the normal operation of the docker container [9]. For mirroring, this article reinforces three phases, the download phase of docker from the docker hub ensures that the hub has not been tampered with, the number of image vulnerabilities in the downloaded docker image did not exceed the threshold, and the docker image has not been tampered with locally. Through the above method, the trust chain construction of the docker image part is realized.
After ensuring that the image is trusted, start the container based on the image, strengthen the container, and build a trusted chain for container operation. First build a container monitoring platform, real-time monitoring of started container clusters to ensure that docker's resource usage is within a reasonable range. On the other hand, the container is reinforced. docker is reinforced by limiting container capabilities, monitoring docker container networking, and limiting internal processes of the container. Strengthen isolation, enhance the security of docker, prevent external attacks or minimize the impact after encountering attacks and tampering.

3. Design of docker container hardening method based on trusted computing

3.1. Design ideas
According to the security analysis of the docker container described above, in view of possible security vulnerabilities and untrusted problems in the image and the container, trusted computing technology is used to reinforce the entire startup process of the docker container.

3.2. Overall design framework
In the entire reinforcement scheme, a trust chain is designed to ensure the trust chain's transmission during the entire process of downloading and starting the docker container, thereby ensuring the trustworthiness and security of the docker container.
The entire container reinforcement can be divided into two parts according to the object, which are image and container ,as shown in Figure 1.
3.2.1. Image measurement module. Before downloading the image, first verify the publisher of the image. When others perform a pull operation on this image, docker uses the publisher’s public key to verify whether the image is consistent with the published image and whether it has been tampered with to ensure the image file trusted before downloading.

After the image is downloaded locally, you need to check whether the downloaded image itself is vulnerable through image verification to avoid affecting the running status of the host and other containers when the container starts.

Before the container starts, after the image is downloaded locally, the module obtains the hash value of each part of the image through the algorithm and saves it. When the container starts, it calculates it again and compares it with the previous value. If it succeeds, it starts the container. Ensure that the image before the container starts is trusted, as shown in Figure 2.

3.2.2. Container reinforcement monitoring method. When the container is started, it is also necessary to monitor the entire container cluster to prevent an unexpected state of a certain container, which will affect the security and credibility of other containers and even the host machine.

This module plans to use container management tools to achieve real-time monitoring of resource usage and performance indicators, CPU and memory usage, and network utilization. Control the communication between containers by setting IPtables rules [10]. By designing a process whitelist, internal processes are restricted. Limiting memory usage can prevent DoS attacks and application memory leaks. By limiting the CPU, if an attacker can obtain one or a group of containers and completely

![Figure 1. Trust chain delivery process](image)

![Figure 2. Image measurement module flowchart](image)
occupy the CPU on the host, the attacker can prevent other containers on the host from allocating appropriate CPU resources. By limiting the file system, it prevents attackers from writing to the file system and defends against some attacks on the system. By limiting the kernel's capabilities, it prevents attackers from exploiting it and gains all root privileges on the system. By applying resource limits, limit the number of child processes that a process can fork to spawn, or the number of file descriptors allowed to open. Others include the use of the Linux security module, which provides another important security check on the access rights of processes and users to ensure the trustworthiness of the docker container running process.

After the cluster is started, for each container, ensure that its internal resources and capabilities are in a state that can be monitored when the container is running. Reinforce the container to strengthen the safety of the container. In the process of container operation, real-time monitoring of container resources must be implemented to ensure that the container runs in a trusted environment. For the file system, through some methods such as disk quotas and access permissions, strengthen the isolation of the file system. Limit the resources used by each container to avoid a container consuming too much CPU resources [11]. Monitor the external communication of the container, and set the specified external container to complete the interconnection with this container; complete the monitoring of the process inside the container, and only the specified process can run. Reinforcement of the container is achieved through the above methods, ensuring the credibility of the container during operation, as shown in Figure 3.

![Flow chart of container reinforcement measurement method](image)

**Figure 3.** Flow chart of container reinforcement measurement method

### 4. Docker container hardening scheme based on trusted computing

#### 4.1. Image measurement method

**4.1.1. Docker image signature module.** Through the docker image signature mechanism, the credibility of docker's content publishing mechanism can be guaranteed. The specific process is that when the user downloads the image from the docker hub, he can also receive a certificate containing the publisher's public key, so that the user can verify whether the image really comes from the publisher. When the content trust mechanism is enabled, the docker engine will only use signed images and refuse to run any images with mismatched signatures or digests, as shown in Figure 4.
4.1.2. **Docker image scanning module.** The docker image is pulled from the docker hub. Through the mirror signature mechanism, it can be guaranteed that in the docker hub, the uploaded image and the downloaded image are the same image. The image file itself has not been tampered with, but it cannot guarantee that the uploaded image will be certain is a trusted image. It may be that when the image is built, the image file is injected with malicious commands or vulnerable components, so the image file needs to be detected for vulnerabilities. In response to mirror attacks, this article uses mirror scanners to respond. In the middle of downloading docker images to the docker container startup process, we add mirror scans to ensure the transmission of the trust chain. The image scanning module in this article uses the Trivy image scanner [12]. The specific image scanning process is: First, Trivy will periodically obtain vulnerability data elements from the configured source address and store them in the vulnerability database to match the possible vulnerabilities in the image file. The client extracts the features of the local image and stores them in the database. These image features are used to match the CVE vulnerability database, and each feature is requested to associate features and vulnerabilities, avoiding multiple repeated scanning of the image. When there is a new vulnerability metadata update, a system update prompt is performed. The image scan results show that images without vulnerabilities can be started, and other images are untrusted images. The image scanning module can ensure the credibility of the image file and complete the trust chain transfer after the image file is downloaded, as shown in Figure 5.

![Diagram](image.png)

**Figure 4.** Image signature flowchart
4.1.3. **Docker image file measurement module.** The docker images are downloaded to the local area using the docker pull command. Each image is stored locally in the specified directory in the form of a folder. Before the container is started, the host may be exposed to external attacks, causing the image file to be tampered with, which may cause the container to be untrusted. This module uses a cryptographic algorithm to measure the trustworthiness of the image file to ensure that each time the image before the container is started is a trusted image. The specific process of the module is: first, after the first download of the docker image file, use the sha256 algorithm to hash the downloaded image file and store it in the specified file. In order to prevent the host machine from being attacked, the storage metrics are stored. Files are uploaded to trusted cloud storage to ensure that they cannot be tampered with. Each time after starting the docker container through this image file, the image file is hashed again using the sha256 algorithm and compared with the first hash value in cloud storage. If all the hash values are the same, it means that the image file has not been tampered with. If the hash values are not equal, it means that the file has been tampered with and the image is not trusted. It is forbidden to start the container through this image [13]. This module is the last step in the trust chain before the container starts. It ensures that the image file in the host machine has not been tampered with after ensuring that the downloaded image is a trusted and secure image. Before the container starts, the image is trusted, as shown in Figure 6.

![Diagram](image)

**Figure 5.** Docker image scanning design method

**Figure 6.** Docker image measurement method flowchart
4.2. Container reinforcement monitoring method

4.2.1. Container reinforcement. After the container is started, the isolation of the container is strengthened by limiting the capacity of the container and enabling the security module, thereby achieving the effect of container reinforcement.

Limiting container memory: By default, each docker container can use all the memory of the host machine. This module uses a memory limit method to prevent a container from consuming all resources of the host machine and causing denial of service, thereby other containers in the cluster cannot execute normally.

Restrict file system: Prevent the container's root file system from being written to the container by setting the container to be read-only when the container is running. Using this method can clarify the data writing strategy of the container at runtime, reducing the risk of container tampering. The specific method is to use docker run to use the --read-only parameter and set the container's file system to be completely read-only: or you can add: ro after the data volume parameter to achieve a similar effect.

4.2.2. Container monitoring. Container monitoring is divided into monitoring during cluster operation and container internal monitoring.

Cluster monitoring platform: When a container is running, it needs to use a monitoring module to monitor the resource usage of each container in real time. This paper uses a visual container monitoring platform to monitor the resource usage of container clusters in real time. The entire platform architecture includes cAdvisor + InfluxDB + Grafana, of which cAdvisor is a monitoring tool developed by Google that can monitor container clusters in real time from the webpage, but its disadvantage is that it cannot save data, so it is only used as a data to obtain the docker cluster resource usage source. Then use the influxdb database to store the container resource usage obtained by cAdvisor. It can be operated in both web pages and commands to create databases and users. Finally, Grafana is used as a visual display of the final monitoring resource, which can use InfluxDB as a data source, as shown in Figure 7 and Figure 8.

![Figure 7. Monitoring platform design process](image)

Monitoring container networking: There are multiple ways for docker containers to communicate with the outside world. The principle is to use namespace network isolation and virtual network devices.
Setting iptables rules can restrict unauthorized communication of containers and strengthen container security. By setting the iptables rule in the docker container, the specified IP is added to the blacklist, and unauthorized communication between docker and the specified IP is prohibited. For each individual container of the cluster, monitoring is also required. By writing a process whitelist, the processes inside each container can be guaranteed to be safe and reliable processes. First set the file to write the executable process. Through the background program, get the current process inside the container once in a while. By comparing with the whitelist process, it can be run if it exists. Reliability, as shown in Figure 9.

![Figure 9. Container process monitoring flowchart](image)

5. Experimental test

5.1. Experimental environment
The experimental environment of this experiment was carried out in the Ubuntu version of the Linux environment, the memory is 4G, the storage is 256G, and the docker version is 17.03.2.

5.2. Experimental process
The image download mechanism is used to ensure that the image download is reliable. First, open the image signature mechanism and download the untrusted image, which shows that the image download fails, as shown in Figure 10. After downloading the trusted image, the image download is successful, as shown in Figure 11.

![Figure 10. Image signature is not enabled](image)
After the image download is successful, the downloaded image is scanned by the image scanner, and the vulnerabilities in the image can be scanned. Using trivy to scan the downloaded redis image, you can display the image vulnerabilities in the image, as shown in Figure 12.

Figure 12. Image scan results

For the image attack after download, pass the password verification to ensure that the image before the container starts is credible. First write a script to get the image file information from the local image file storage path. The image file is stored in the lib directory. After obtaining the path information of the file, the metric value calculation is performed on the image file through the sha-256 algorithm and stored in the specified file. Before the container is started, the image file is re-measured. Comparing the two measurements, Figure 13 is the result of the image file not being tampered with, and Figure 14 simulates some file modifications, which shows that the image file has been tampered with.

Figure 13. Container has not been tampered with
Enable container memory limit function in Figure 15, check the current container resource usage of the current cluster, the penultimate column of memory usage, except for the first container with memory limit turned on, the memory goes online at 500M, all other containers are the same as the host 3.8G, it is easy Under attack, a container consumes too much memory, and a container with memory limit turned on will not affect other containers too much.

Make docker read-only. The construction of docker images can only be achieved through dockerfile. It is forbidden to change the container while the container is running. After the container is set to read-only, a string is written to the file inside the container, which shows that the file inside the container cannot be modified, as shown in Figure 16. Set the container to read-only, execute the command to write information to the container file, and display the write failure.

Build a container monitoring platform to perform real-time monitoring of the resource usage of the currently running containers. First download the three images of cadvisor, influxdb, and grafana respectively. Start the container cadvisor, which is used to obtain the resource usage of the currently running container. It can see the resource usage in the webpage in real time, but cannot be stored, as shown in Figure 17. Connect the data collected by cadvisor to the database influxdb, and create database tables and users in influxdb, as shown in Figure 18. Finally deploy the database to granafa, and you can see the resource container usage in the panel through configuration, as shown in Figure 19, showing the CPU and memory usage of the influxdb container.
The monitoring test of the internal process of the container first writes the name of the process that is prohibited from starting the process in the blacklist file, the test list is the executable program of test, and then starts the process monitoring module. This module is a daemon process and reads the current
time once at a fixed time. The executed process reads the process blacklist for matching. If the blacklist process is matched, the blacklist process is killed, as shown in Figure 20.

![Figure 20. Kill blacklist process](image)

Container network monitoring, set iptables rules to specify that ip can communicate with the container, reducing unauthorized communication. First ping the 172.17.0.2 ip in the container, showing that the communication is successful, as shown in Figure 21, then write iptables rules to prohibit 172.17.0.2 from communicating with the container, and execute the ping command again, the communication fails.

![Figure 21. Successful container networking](image)

Figure 22. Limiting container networking

5.3. Experimental conclusions

As the most popular lightweight virtualization technology today, docker container faces great security risks due to its imperfect isolation. This paper addresses the problems of malicious images in docker containers, images being tampered with, unavailable resources monitoring after container startup, and malicious processes inside the containers. This paper proposes a hardening method based on trusted computing. Through monitoring of image scanning and image measurement, a monitoring platform is established. Methods such as process monitoring and communication monitoring strengthen the security of docker's entire life cycle from download to startup, and construct a reinforced docker container. By testing docker 17.03.2, it was successfully verified that the method proposed in this paper can strengthen the isolation of docker, reduce external attacks or minimize the adverse effects under external attacks,
and at the same time can monitor the running status of the container in real time to ensure its safe and controllable operating state.

6. References

[1] SLOMINSKIA, MUTHUSAMY, KHALAF R. Building a Multi-tenant Cloud Service from Legacy Code with Docker Containers [DB/OL]. [2016-08-13]. http://iexplore.ie.org/document/7092950/

[2] Zhang Yao, Wang Linlin. Research on Docker Security. J Network Security Technology and Application, 2017

[3] Huawei Docker Practice Group. Advanced and Practical Docker. Beijing: Machinery Industry Press, 2016: 313-354

[4] Wang Juan, Hu Wei. Docker-based trusted container. J Wuhan University, 2017

[5] Ma Yue; Huang Gang Research on application software virtualization based on Docker School of Computer, Nanjing University of Posts and Telecommunications, 2015

[6] Logscape.com, Monitoring Docker part II, 2014

[7] Rafi Chen, Antoine Joux, Collisions of SHA-0 and Reduced SHA-1. ACM, 2005

[8] SUN H L. The Analysis of Docker Source Code Part I: The Architecture of Docker

[9] Yang Wenlin, Tan Xi Docker Vulnerability Analysis and Security Enhancement J Electronic Technology and Network Engineering 2018

[10] BOETTIGER C. An introduction to Docker for reproducible research J. ACM SIGOPS Operating Systems Review, 2015, 49(1):71-79

[11] Qiu Chen, Design and Implementation of Performance Monitoring and Logging Service for Docker Containers [D] Zhejiang University 2016

[12] TAN L, XU Z W. Development of the transitive trusted chain based on TPM J. The Computer Science, 2008, 35(10):15-18 (Ch)

[13] Liu Siyao, Li Qiang, Li Bin Research on container isolation based on Docker technology [J] Design research and application 2015