INDIAN INSTITUTE OF TECHNOLOGY, JODHPUR

VIRTUALIZATION AND CLOUD COMPUTING - CSL 7510

COURSE PROJECT REPORT

ANALYSIS OF DOCKER FROM A SECURITY PERSPECTIVE

Prepared by:

Abhijit S Iyer (M21CS001)

Poonam Kashyap (M21CS012)

An analysis of Docker from a security perspective, based on the research papers ***To Docker or Not to Docker: A Security Perspective*** and ***Security Analysis and Threat Detection Techniques on Docker Container***

Background

Over the last few years, the use of virtualization technologies has increased dramatically and as a result, the demand for efficient and secure virtualization solutions shoots up. Container-based virtualization and Hypervisor based virtualization are two main types of virtualization that have emerged in the market in which Container-based solutions provide a more lightweight and efficient virtual environment. Among all container solutions, *Docker*, a complete packaging and software delivery tool, is leading the market. Docker is written in Go language and was first released in March 2013. Docker launches a container from an existing image, performs modification and installations inside the container, and then stops the container and saves its state as a new image(docker commit). Docker containers create a controlled environment on the host machine to run code safely. Docker software runs as a daemon on the host machine and launches containers, controls isolation and is also responsible for managing the container's image including pulling and pushing images on a Docker Hub(online public repository).

Docker security relies on three factors:

- Isolation of processes at the user-space level managed by the Docker daemon
- Enforcement of this isolation by the kernel
- Security of network operations.

The standard isolation configuration is relatively strict (i.e. default configuration). The only flaw is that all containers share the same network bridge, allowing Address Resolution Protocol (ARP) poisoning attacks between containers on the same host. Linux kernel security modules apply security limitations on containers for host hardening. Default hardening protects the host from containers but does not protect containers from other containers.

As docker uses network resources for image distribution and remote control of the docker daemon through Unix socket. The socket used is a Unix socket owned by root:docker and can be changed to a TCP socket. When a TCP socket is used, any connection gives root privileges to the host.

Docker container technology is better than traditional virtualization technology-virtual machines but it has poor security due to the unmatured auditing procedures of Docker image releasing. We illustrate the existing security mechanism in Docker and analyze the threats for docker containers.

Problem Statement

In order to protect the security of host computers or local Docker containers from the attacks of malicious Docker containers and by various adversaries either directly or indirectly, it is, therefore, necessary to identify the potential threats present in Docker images and to find the risks if Docker container instances are on the host computer. Docker containers do not isolate the virtual objects via virtualization hardware or using an independent operating system. Instead, it uses the namespace mechanism in Linux to promise secure isolation from running environments and uses the cgroup mechanism in Linux to implement computer resource management. In addition, it uses the kernel capability to strengthen security.

Docker Container contains independent modules of Docker client, Docker Daemon and Docker Register, all of which are independent and work together to complete the tasks. Any kind of exposures existing in these modules will directly affect the security of Docker containers and host computers. Container Escape attack is also a major threat in which exposures existing in the host machine helps the process running in the container to escape to the host system in order to achieve higher privileges. Sensitive information leakage may also occur causing the whole service to cease anytime.

Existing security implementations on docker containers focus mainly on the relationship between the host machine and the container. However, today, Docker containers are now part of a complex ecosystem, which includes containers and various repositories and orchestrators, that is highly automated. Container solutions embed automated deployment chains that are meant to speed up the code deployment processes. These chains are often composed of third-party elements running on different platforms provided by various providers, raising concerns about code integrity. This can cause multiple vulnerabilities that an outsider could exploit to enter the system. Container ecosystem security has yet to be thoroughly investigated, despite being fundamental to container adoption. The primary reasons why we focus on the Docker ecosystem are:

- Docker as a container system is already being used/ will be used by almost 92% of DevOps systems.
- Security is the first barrier to the adoption of this container environment.
- Docker provides the opportunity to run experiments and explore the practicality of attacks done by adversaries.

Methodology

Although Docker, when used in its default recommended configuration does not have any security flaws, it provides users with many options to start and run docker images with various options that expose the host system, the host network and other docker containers to be put in vulnerable positions from which there may be no recovery. Some of these launch options include

- -net=host parameter on docker run, forces the Docker daemon to not place the container in a separate NET namespace, allowing the container to take advantage of the host computer's network stack, which is a huge security breach.
- -uts=host allows the container to access the hosts' name and domain in the network since the Docker daemon places the container into the host's UTS namespace.

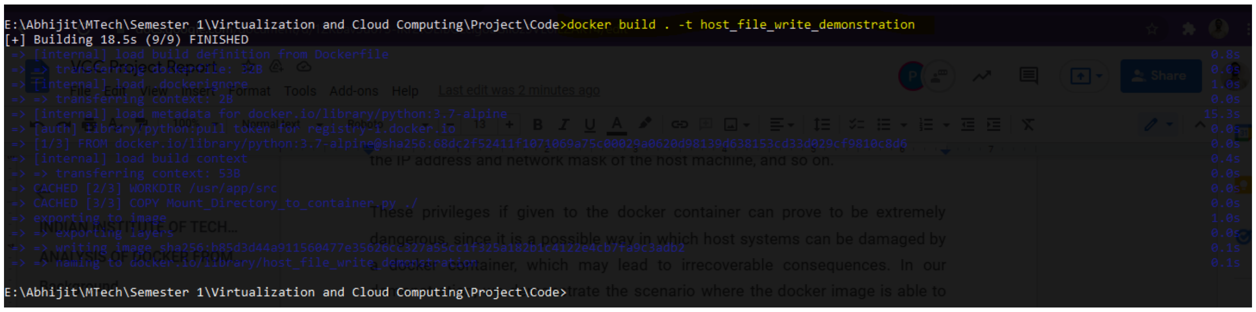
- -cap-add <CAPABILITY_NAME> allows the docker daemon to give specific privileges to the docker container, some of which may be potentially harmful to the host.

In our implementation, we take a look at giving the capability/ status of accessing the host file system to the Docker container with which will be able to access full authority over the specific directory of the host system. Some of the capabilities include mounting and unmounting of the host file system to the docker container, executing privileged OS kernel instructions, changing the IP address and network mask of the host machine, and so on.

These privileges if given to the docker container can prove to be extremely dangerous since it is a possible way in which host systems can be damaged by a docker container, which may lead to irrecoverable consequences. In our demonstration, we demonstrate the scenario where the docker image is able to access the host file system and make changes to the host system.

To start with, we wrote and built a docker image that runs a python application that tries to gain access to a directory present in the host's file system, by reading content from a file and writing to a file in the same directory.

The command that was used to build the docker image is as shown below:



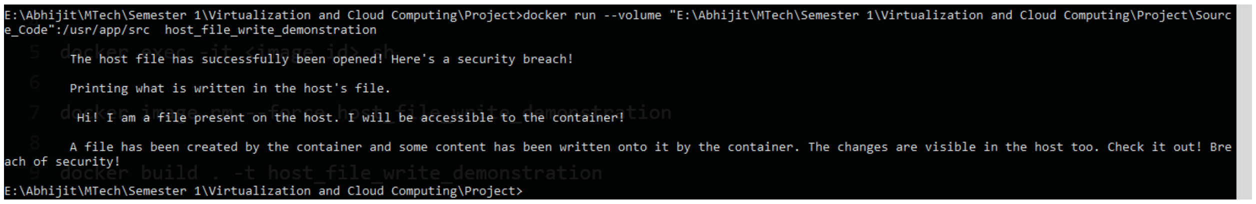
```
E:\Abhijit\MTech\Semester 1\Virtualization and Cloud Computing\Project\Code>docker build . -t host_file_write_demonstration
[+] Building 18.5s (9/9) FINISHED
 => [internal] load build definition from Dockerfile                                                   0.8s
 => => transferring dockerfile: 32B                                                                    0.0s
 => [internal] load .dockerignore                                                                      1.0s
 => => transferring context: 2B                                                                        0.0s
 => [internal] load metadata for docker.io/library/python:3.7-alpine                                  15.3s
 => [auth] library/python:pull token for registry-1.docker.io                                          0.0s
 => [1/3] FROM docker.io/library/python:3.7-alpine@sha256:68dc2f52411f1071069a75c00029a0620d98139d638153cd33d929cf9810c8d6  0.0s
 => [internal] load build context                                                                      0.4s
 => => transferring context: 53B                                                                       0.0s
 => CACHED [2/3] WORKDIR /usr/app/src                                                                  0.0s
 => CACHED [3/3] COPY Mount_Directory_to_container.py ./                                               0.0s
 => exporting to image                                                                                 1.0s
 => => exporting layers                                                                                0.0s
 => => writing image sha256:b85d3d44a911560477e35626cc327a55cc1f325a182b1c4122e4cb7fa9c3adb2           0.1s
 => => naming to docker.io/library/host_file_write_demonstration                                       0.1s

E:\Abhijit\MTech\Semester 1\Virtualization and Cloud Computing\Project\Code>
```

`docker build . -t host_file_write_demonstration`

Once the image has been generated, we ran the image using the following command:

```
docker run --volume "E:\Abhijit\MTech\Semester 1\Virtualization and
Cloud Computing\Project\Code":/usr/app/src
host_file_write_demonstration
```

A snapshot of the same can be seen below:



```
E:\Abhijit\MTech\Semester 1\Virtualization and Cloud Computing\Project>docker run --volume "E:\Abhijit\MTech\Semester 1\Virtualization and Cloud Computing\Project\Sourc
e_Code":/usr/app/src  host_file_write_demonstration

         The host file has successfully been opened! Here's a security breach!

         Printing what is written in the host's file.

          Hi! I am a file present on the host. I will be accessible to the container!

         A file has been created by the container and some content has been written onto it by the container. The changes are visible in the host too. Check it out! Bre
ach of security!

E:\Abhijit\MTech\Semester 1\Virtualization and Cloud Computing\Project>
```

The command has the following parameters specified to it:

- **--volume <Directory_name>**
  This parameter mounts the specified directory onto the container, allowing the container to gain full access to the host's file system.

A file in the host system, named 'File_from_Host.txt' was created in the host machine with certain text written into it. Upon the creation of this container, the container with its privileges was able to read this file and the same has been shown to be displayed in the console image above.

Similarly, a file was created by the container and was placed in the host's file directory with some content written to this. This shows the two-way access that a container has over the host system's file directory. It also demonstrates a vulnerable security flaw in the docker system.

Results

Demonstration Video

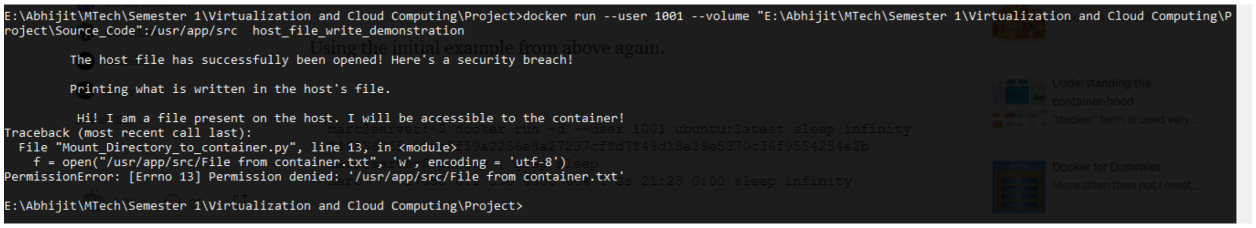
From the above-mentioned methodology procedure, it was observed that the **Docker container was able to gain full access to the file system of the host computer, which puts many critical files and folders of the host system at a high risk** of being compromised by a container which may export the data to other web applications, clients or other containers. Therefore, this brings to light a major security issue in using Docker containers, wherein critical files of the host system have become exposed to the container. However, it is also important to note that these privileges are not given to the containers by default. It is something that can be exercised by container image builders as a mechanism to share files between a host system and a container. But when they do so, they would need to keep in mind such security vulnerabilities, since the same image if run on client machines may cause damage to client machine files and other critical data, leading to unwanted outcomes between developers and their clients.

The reason behind this is that **Docker Engine** can be run with root privileges, and hence if a process breaks out of the container, it will have the same privileges as a system administrator. Therefore, we need to be extremely careful about the privileges that we give a container during its execution. This risk can be mitigated by avoiding the use of ready to download containers.

A container is not a trust boundary. The only difference between a normal process that runs in the system and a docker container process is the metadata that declares it as a container process.

We can control the container from getting any sort of root privileges by provisioning it with a **User ID (UID)** or a **Group ID (GID)**, which can be supplied to the container by making it a part of the dockerfile.

A demonstration of the same can be seen below:



```
E:\Abhijit\MTech\Semester 1\Virtualization and Cloud Computing\Project>docker run --user 1001 --volume "E:\Abhijit\MTech\Semester 1\Virtualization and Cloud Computing\P
roject\Source_Code":/usr/app/src  host_file_write_demonstration

        The host file has successfully been opened! Here's a security breach!

        Printing what is written in the host's file.

         Hi! I am a file present on the host. I will be accessible to the container!
Traceback (most recent call last):
  File "Mount_Directory_to_container.py", line 13, in <module>
    f = open("/usr/app/src/File from container.txt", 'w', encoding = 'utf-8')
PermissionError: [Errno 13] Permission denied: '/usr/app/src/File from container.txt'

E:\Abhijit\MTech\Semester 1\Virtualization and Cloud Computing\Project>
```

Here, we used the docker run command by specifying the container with a user ID and showed that the container was unable to access the host file system and was unable to create and modify files.

Therefore, it is critical to note that although Docker by default does not expose security vulnerabilities, it does allow users to take advantage of the available features in docker daemon to be utilized and put important components of a system such as a file system, network configurations, kernel parameters at high risk.

It was also observed that applications being hosted by the container and the libraries that are present in the container also put other containers and host systems at high risk of direct and indirect adversaries. Apart from this, it has also been detected that thousands of Docker images present in DockerHub have critical vulnerabilities, which may give enough cues for attackers to attack critical systems and expose critical data.

Another security flaw in using Docker containers is during automated build systems. When changes are pushed to a Github repository, an automated build system would be triggered through Jenkins which would build an image and immediately push it to DockerHub within a span of 5 to 10 minutes. This is again a highly risky methodology because the image would not have been checked for vulnerabilities and is directly being pushed onto production.

Future Directions

Although there are some scenarios where Docker containers can be used relatively safely, there are enough ways in which Docker containers can be exploited and surrounding systems can be made vulnerable. However, there are third-party tools that Docker has integrated into its daemon to check for vulnerabilities in Docker images.

Snyk is a developer security platform for securing code, dependencies, containers, and infrastructure as code. Snyk's vulnerability scan not only lists the issues but also guides developers towards actions required to reduce the risk of vulnerability. Snyk is an open-source security scanning tool that is going to be integrated with the latest versions of Docker Desktop for providing security to Docker's official images and It is integrated directly into Docker Hub.

Orchestration is a tool that will be used to automate repetitive tasks. For designing a private cloud, Orchestrators are required to tie with the ticketing system of clients so that it can automate user requests. An Orchestrator also resolves many security issues by limiting misuse of Docker through higher levels of abstraction and removing host dependency and thus enabling better isolation. Further experiments in this direction will lead to a more enhanced security system.

Another area where the default docker specification can possibly improve is by putting containers into namespaces. This would ensure that a container process would access only those resources which belong to that particular namespace, thereby avoiding misuse of host system resources.

References

- https://ieeexplore.ieee.org/document/7742298
- https://ieeexplore.ieee.org/document/9064441
- https://www.freecodecamp.org/news/7-cases-when-not-to-use-docker/
- https://medium.com/@mccode/processes-in-containers-should-not-run-as-root-2feae3f0df3b
- https://medium.com/@mccode/understanding-how-uid-and-gid-work-in-docker-containers-c37a01d01cf

Demonstration Video Link:

https://youtu.be/AlArv2PgkKI

Project Presentation Link:

https://docs.google.com/presentation/d/141nIxf5aisncQYM5WHstUY0mxWUsn8kZ1oQaT4kRUqk/edit?usp=sharing

Source Code Repository Link: