Concordia Institute for Information System Engineering
(CIISE)

Concordia University

INSE 6130: Operating Systems Security

Project Progress Report

Implementing Recent attacks and a security application on container

Submitted to:

Professor Suryadipta Majumdar

Date: March 16, 2022

	Student Name	**Student Id**
1	Mohit Balu	40221594
2	Bikramjeet Singh	40192900
3	Harpreet Kaur	40190384
4	Nithya Sri Bommakanti	40220572
5	Milanpreet Kaur	40204741
6	Gouresh Chauhan	40194834
7	Srividya Poshala	40192542
8	Rabiatou Oubbo Modi	40155873

Table of Contents

Introduction:

Over the last few years, the use of virtualization technologies has been increased dramatically. This makes the demand for efficient and secure virtualization solutions more obvious. There are two main types of virtualization solutions: container-based and hypervisor that have emerged in the market. Docker was introduced in 2013, to solve the time-consuming and costly process of application development and service delivery. Docker is a container-based virtualization platform that provides a lightweight and efficient virtual environment and separates the applications into their containers, where they share the resources, however, interacts with the operating system independently.

As virtualization is becoming mainstream, the security concerns pertaining to it are also coming to the surface. In this project, we are focusing on the Docker environment, its possible attack vectors, and defense mechanisms against its vulnerabilities. This progress report mentions the planning and the progress of the following:

(i) development of the attack scenarios,
(ii) The exploitation of the vulnerabilities,
(iii) defense mechanisms against the vulnerabilities,
(iv) and documentation of the whole process.

The basic vulnerability that has piqued our interest and we are most likely working in our attack scenario is exploiting the exposed docker socket and container's vulnerabilities. Misconfigured docker registry could leak confidential data and lead to a compromised situation that interrupts the business operations. We have implemented two attack scenarios using Ubuntu as our base image for docker. For the defense mechanism, we have used Python SDK for Docker to defend the vulnerabilities.

We are working on two different attack scenarios, both of which host a docker environment having few images and a few containers running. We are using Ubuntu as our base operating system as a target and Kali Linux as our attacking system, both of which reside in the same virtual network.

Planning and Preparation:

Our planning and preparation for the project implementation included the following course of actions, a few of which were decided before we started the implementation of the project and a few of them were figured out during the course of the project:

1. Learn Docker basics and read user guides from Docker official documentation (docs.docker.com)
2. Complete Docker courses on Udemy to get a hands-on idea of the docker environment and commonly used commands.
3. Learn about the latest common vulnerabilities (CVE) in the containerized environment and common security misconfigurations.
4. Complete "The docker rodeo" lab on tryhackme.com to get a hands-on idea of exploiting vulnerabilities in the docker environment.
5. Read published papers and conference materials about container security to get in-depth knowledge of the specific vulnerabilities.
6. Document all the illustrations, learnings, and findings on our private project repository on GitHub.
7. Participate in the weekly recurring meeting to discuss the ideas and actions plans.

Setting Up the Environment:

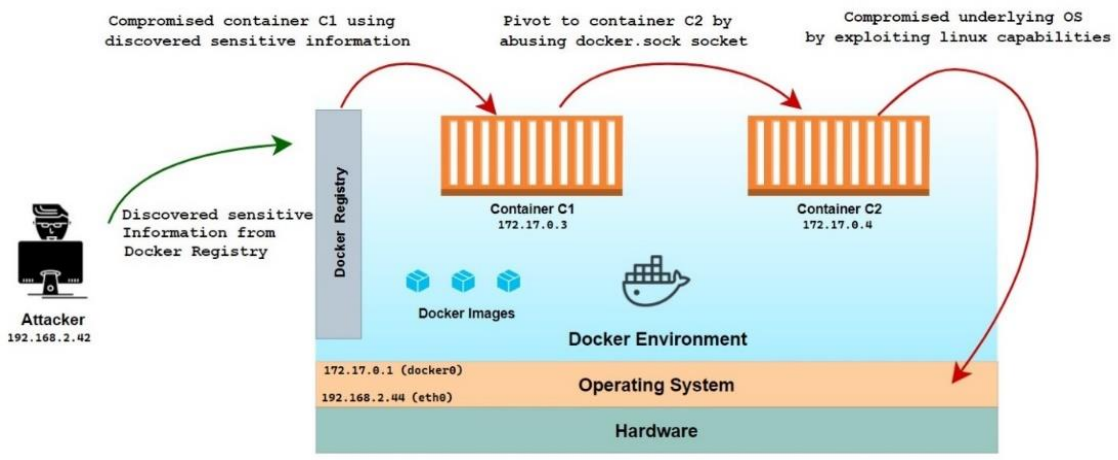
Before starting the implementation of attack and defense mechanisms, we decided to set up the environment on our local machine inside virtual environments and collaborate on GitHub for sharing the implementation steps and key-points through a private repository. GitHub is a collaboration tool used by software developers to work on a single project without the need to share a common network. It helps to manage a large collection of documents and regularly updates them.

We installed Ubuntu operating system on the hypervisor (as our host OS) for hosting the docker environment, to contain the vulnerable components inside the virtual environment only.

We are using python programming language to build the scripts for defending against the vulnerabilities present in the environment. Also, we are using Docker SDK for Python - A Python library for the Docker Engine API which lets you do anything the docker command does, but from within Python apps such as run containers, manage containers, manage Swarms, etc.

Implementation:

Attack Scenario 1 Setup:


Compromised container C1 using discovered sensitive information
Pivot to container C2 by abusing docker.sock socket
Compromised underlying OS by exploiting linux capabilities
Discovered sensitive Information from Docker Registry
Docker Registry
Container C1
172.17.0.3
Container C2
172.17.0.4
Docker Images
Docker Environment
Attacker
192.168.2.42
172.17.0.1 (docker0)
192.168.2.44 (eth0)
Operating System
Hardware

Docker Registry:

A Docker registry is a distribution system for Docker images. There will be different images, and each may contain multiple tags and versions. By default, the registry runs on port 5000 without authentication and TLS.

Docker Socket:

docker.sock is a socket file used for communication from the docker engine API or the CLI to run commands on the system through the docker daemon. The two most used sockets are, Unix sockets – which are used when processes want to communicate locally; so, they are faster, and on the other hand, the TCP sockets are used to communicate via the internet or network. The exposed docker socket is exploited by the attacker by using docker-engine API to download any custom image hosted by him or her on the victim machine and use it for his or her benefit by mounting file system on the docker container image.

Attack 1 Execution:

Compromising Container C1

A Docker Environment is running in an organization, and it has several images and a few containers running. It also has a docker registry service running on port 5000 for developers to access the status of images and containers.

An attacker (or an adversary) is in the same network as the system on which the docker engine is installed (say Ubuntu). S(he) does not have access to the target Ubuntu system (it's very secure) however S(he) could see that there is a docker environment in the network as the docker registry can be accessed on port 5000. The docker registry itself does not pose any serious security risk as it does not allow the user to create, delete or publish images (unless it is writable). The objective of the attacker is to find and exploit the vulnerabilities in the docker environment (if any) and eventually gain access to the underlying Ubuntu operating system.

Compromising Container C2

Container C1 is already compromised which has docker.sock file mounted at /var/run/docker.sock. This can be leveraged to communicate with the docker engine and get access to other running containers.

To communicate with Docker Engine using docker. sock, 'docker' command-line tool should be installed however we are assuming that installation for the 'docker' command line is blocked due to security reasons. So, we will be using Unix socket to communicate with Docker Engine.

Compromising Underlying Operating System [Goal]

Sometimes containers need to be run with extra privileges to perform some operations on the host operating system itself. These containers are run with —a privileged flag which adds some special capabilities to the running container, however, it is not considered a best practice for the security of the system. Any user (or attacker) having access to the container can easily elevate his/her access to the underlying operating system.

As remediation of the above issue, containers are run with the specific capability (whichever is needed) instead of running it as a privileged container (which possesses all the capabilities). These capabilities can be listed with the capsh --print command.

In this attack scenario, Container C2 is also running with the capability SYS_ADMIN. SYS_ADMIN capability allows the container to perform system administration operations such as quotactl, mount, umount, swapon, swapoff, sethostname, and setdomainname. Although not running a container as a privileged one, decreases the security risk however these specific capabilities can also be exploited by issuing some commands and access to the underlying operating system can be gained.

Attack 1 Defense:

To Defend the vulnerabilities that are present in attack 1, we have implemented python SDK for docker scripts. First, we have designed the security measures according to the vulnerabilities that are present in the attack scenario. In this, we have decided to implement five scripts. From which two scripts we have implemented and currently we are working on three defense mechanism scripts.

1. **Designing security Measures according to vulnerabilities present in attack scenario 1: [*WORKING*]:**

    - Implement a python script to detect (and generate an alert) if any of the running docker containers is mounting docker. sock socket file.
    - Implement a python script to detect (and generate an alert) if any of the docker containers are running with unnecessary capabilities which can be abused by an adversary.
    - Implement a python script to generate an alert when someone logins via SSH to the docker container from an unidentified IP address.

2. **Implemented Defense Mechanism for first attack scenario: [*COMPLETED*]**

    - We have implemented docker SDK for python, to defend the first part of the attack. In this script, it detects if any information (such as PASS, KEY, APIKEY, TOKEN) is being disclosed in the manifest file of the images on the Docker registry.
    - We have implemented the HTTP authentication for the docker registry to protect the registry service from being accessed by unauthorized users.

3. **Monitoring tools we are working on to detect the vulnerabilities in the docker: [*WORKING*]**

    - We are also working on monitoring tools: Docker bench and Clair. We are analyzing both tools and will finalize which will detect the vulnerabilities more efficiently.

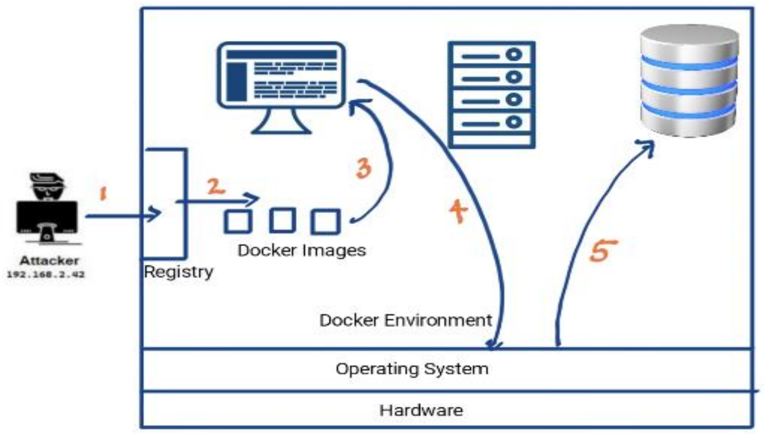
4. **Future Deliverables: [Planning to implement]**

    - We are planning to design a security application that consists of all the defense scripts.

Further, we will be working on implementing a second attack scenario and expect to work out the defense mechanism for it.

Attack Scenario 2 Setup:

We have designed the scenario for our attack 2. The preliminary idea is that the attacker tries to exploit the web application for the credit reporting organization which stores all information related to the credit card of the users.
Below is the diagram for the idea of the exploit.



The idea for the exploit is divided into 5 parts as explained below:

1. The attacker discovers the unprotected registry is writable and writes his own malicious image.
2. An adversary then pushes that image to the registry with runC exploit.
3. Legitimate user starts the container executing the runC exploit along with it.
4. Gets access to the host Operating system (Docker-engine).
5. The attacker tries to access the container hosting database to get sensitive info

Attack 2 Defense:

We will design the security measures according to vulnerabilities present in attack scenario 2 as we have done in scenario 1 and we will implement the defense scenario related to this particular attack.

Team Member's Contributions:

Each Member has put equal contributions to the project. Everyone has read the paper and attended the meetings regularly to attain a better understanding of the project. The background research has been done by everyone and the understanding was discussed.

Member's Individual Contributions:

Attack Scenario 1 Vulnerabilities:

1. Unprotected Docker registry service
2. Sensitive information disclosure in image manifests
3. Abusing exposed docker socket inside the container (docker.sock)
4. Abusing SYS_ADMIN Linux capability inside the container.

	Sub Tasks	Contribution
1.	Reading about different known vulnerabilities and misconfigurations in the Docker environment.	All
2.	Designing Attack Scenario 1 by chaining different vulnerabilities as per real world context.	Mohit Balu, Nithya Sri Bommakanti, Harpreet Kaur
3.	Prerequisites for Attack Scenario 1 (Installing specific versions and implementing misconfigurations).	Gouresh Chauhan, Rabiatou Oubbo Modi
4.	Performing attack on the environment including initial compromise, pivoting to other containers, docker escape, host OS compromise.	Mohit Balu, Milanpreet Kaur, Bikramjeet Singh
5.	Documentation of development of Attack Scenario and attack steps on common Github repo.	Harpreet kaur, Srividya Poshala

Attack Scenario 2 Vulnerabilities:

1. Unprotected Docker registry service
2. CVE-2019-5736 (runC docker escape exploit)

Sub-Tasks:

	Sub Tasks	Contribution
1.	Designing Attack Scenario 2 by chaining a known vulnerability (CVE-2019-5736) and unprotected registry service.	Mohit Balu, Bikramjeet Singh, Harpreet kaur
2.	Figuring out pre-requisites and compatibility of different docker components' versions (Docker Engine, Docker Client, containerd, runc) required for the implementation of runC vulnerability.	Gouresh Chauhan, Rabiatou Oubbo Modi
3.	Implementation of Attack Scenario 2.	Bikramjeet Singh, Mohit Balu, Nithya Sri Bommakanti
4.	Documentation of Attack Scenario 2.	Srividya Poshala. Milanpreet Kaur

Defense Tasks:

	Defense Tasks	Contribution
1.	Designing security Measures according to vulnerabilities present in Attack Scenerio1 & 2	Milanpreet Kaur, Mohit Balu, Bikramjeet Singh
2.	Implemented defense mechanism using Python SDK for docker.	Milanpreet Kaur, Srividya Poshala.
3.	Working on monitoring tools to detect the vulnerabilities	Gouresh Chauhan, Rabiatou Oubbo Modi
4.	Planning to design a framework for the security application.	Nithya Sri Bommakanti, Harpreet kaur
5.	Documentation of security measures	Harpreet kaur, Milanpreet Kaur, Srividya Poshala

Challenges Faced:

1. We have faced some challenges while working on this project. One challenge was that team members were not familiar with docker. We spent some time familiarizing ourselves with it and learning it.

2. Many articles on the internet guide about a running container with docker. sock and Unix sockets but commands for functioning containers using docker.sock and Unix sockets were a challenge to figure out. Anyhow, docker-engine API documentation came to our rescue for the challenge we were facing.

3. Another challenge was to find a working exploit with a reverse shell, to gain access to the operating system. All of the available writeups were limited to running the 'ps aux' command to prove the control over OS. We tried to modify the exploit to gain the reverse shell on OS, and we almost managed to run it, unfortunately, ended up with a 'broken pipe' as soon as the first command was executed on the reverse shell. After several attempts to fix it, the final solution was to go ahead with a bind shell written in python to make it work.

4. Faced compatibility issues of Virtual machines Ubuntu with the Mac M1 and Windows 11.

5. We struggled to figure out such vulnerability in docker components, which would let us do initial compromise of the environment and also further allow us to pivot to other containers or to perform docker escape.

Compromising initially by uploading malicious images would have given us access to the container itself however escaping from that container would have been very difficult, unless the container is intentionally run with some potential misconfiguration by a legitimate user, which is the least likely scenario in the real world.

Similarly, to implement the docker escape scenario, we would have required access to an already running container with some misconfiguration, which was possible by compromising some explicitly installed applications on the docker container. In the end, we figured out that an attacker outside the docker environment can do an initial compromise by pushing malicious image (with runC exploit) into an unprotected docker registry and at the same time perform docker escape through runC exploit and get access to the host OS.

What's Next?

1. **I**mplementing the remaining defense mechanisms for Attack 1.
2. Need to do the documentation of Attack Scenario 2.
3. Need to implement remaining defense mechanisms for Attack 1.
4. Working on an implementation of a detection tool for the vulnerabilities.
5. Need to design security measures & implement those defense mechanisms for Attack 2.

References:

[1] “Docker Desktop overview,” *Docker Documentation*, Mar. 14, 2022. https://docs.docker.com/desktop/ (accessed Mar. 15, 2022).
[2] “Docker Engine API v1.41 Reference.” https://docs.docker.com/engine/api/v1.41/ (accessed Mar. 15, 2022).
[3] “Examples using the Docker Engine SDKs and Docker API,” *Docker Documentation*, Mar. 14, 2022. https://docs.docker.com/engine/api/sdk/examples/ (accessed Mar. 15, 2022).
[4] “How To Run Docker In Docker Container [3 Methods Explained],” Jun. 25, 2021. https://devopscube.com/run-docker-in-docker/ (accessed Mar. 15, 2022).
[5] Atucom, “Smallest Python Bind Shell.” https://blog.atucom.net/2017/06/smallest-python-bind-shell.html (accessed Mar. 15, 2022).
[6] “TryHackMe | Cyber Security Training,” *TryHackMe*. https://tryhackme.com (accessed Mar. 15, 2022).
[7] Anton, “Understanding Docker container escapes,” *Trail of Bits Blog*, Jul. 20, 2019. https://blog.trailofbits.com/2019/07/19/understanding-docker-container-escapes/ (accessed Mar. 15, 2022).
[8]”TryHackMe | The Docker Rodeo” *TryHackMe. The Docker Rodeo.* https://tryhackme.com