

# University of Wales Trinity Saint David

School of Applied Computing – MSc Computer Networks and Cyber Security



## A Research on Security in Docker

Presented by: Aneena Elizabeth Saju (2105165)

Supervisor: Michael Dacey

February 2023

Project Dissertation submitted as part of the requirements for  
the award of Degree of Master of Science

# CONTENTS

|                                                                                       |    |
|---------------------------------------------------------------------------------------|----|
| 1. INTRODUCTION .....                                                                 | 5  |
| 2. LITERATURE REVIEW .....                                                            | 7  |
| 2.1 DOCKER AND ARCHITECTURAL SECURITY .....                                           | 9  |
| 2.1.1 Linux Security Overview .....                                                   | 9  |
| 2.1.2 Features of Linux that Support Docker .....                                     | 10 |
| 2.1.3 Security in Docker .....                                                        | 10 |
| 2.2 EXISTING DOCKER CONTAINER SECURITY MECHANISMS.....                                | 11 |
| 2.2.1 Resource Isolation.....                                                         | 11 |
| 2.2.2 Resource Control.....                                                           | 14 |
| 2.2.3 Limitation of Kernel Capability .....                                           | 14 |
| 2.3 ANALYSIS OF THREATS FOR DOCKER CONTAINER.....                                     | 14 |
| 2.3.1 Components Security .....                                                       | 14 |
| 2.3.2 Container Escape Attack.....                                                    | 15 |
| 2.3.3 Sensitive Information Leakage .....                                             | 15 |
| 2.3.4 Network Mode Security .....                                                     | 15 |
| 2.4 DOCKER IMAGE SECURITY.....                                                        | 15 |
| 2.5 ATTACKER MODELS.....                                                              | 16 |
| 2.6 DIFFERENT EXPLOITS IN A DOCKER CONTAINER.....                                     | 16 |
| 2.6.1 Container Image Malware.....                                                    | 16 |
| 2.6.2 Privilege Escalation Exploits .....                                             | 17 |
| 2.6.3 Access Control Exploits.....                                                    | 17 |
| 2.6.4 Network Security Exploits.....                                                  | 18 |
| 2.6.5 Orchestrator Container Exploits.....                                            | 18 |
| 2.7 SOME PAPERS AND ARTICLES THAT SHOWED DIFFERENT WAYS TO ACCESS<br>CONTAINERS ..... | 18 |
| 3. RESEARCH METHODS .....                                                             | 21 |
| 4. RESEARCH DEVELOPMENTS .....                                                        | 24 |

|                                                                |    |
|----------------------------------------------------------------|----|
| 4.1 SYSTEM CONFIGURATION .....                                 | 24 |
| 4.2 ANALYSIS OF WSL2 AND DOCKER .....                          | 25 |
| 5. IMPLEMENTATION .....                                        | 28 |
| 5.1 EXPLORATION OF DOCKER CONTAINER APPLICATIONS .....         | 28 |
| 5.1.1 Method: SSHing into the Docker container .....           | 28 |
| 5.1.2 Method 1: OpenSSH-based Docker shell. ....               | 28 |
| 5.1.3 Method 2: Using the docker exec command in a shell ..... | 31 |
| 5.1.4 Method 3: Pivoting through a server via SSH .....        | 32 |
| 5.1.5 Method 4: Pivoting using Metasploit.....                 | 34 |
| 6. RESULT AND DISCUSSION .....                                 | 37 |
| 6.1 Method 1: OpenSSH-based Docker shell. ....                 | 37 |
| 6.2 Method 2: Using the docker exec command in a shell. ....   | 38 |
| 6.3 Method 3: Pivoting through a server using SSH.....         | 39 |
| 6.4 Method 4: Pivoting using Metasploit.....                   | 41 |
| 7. CONCLUSION.....                                             | 44 |
| 8. FUTURE SCOPE.....                                           | 45 |
| 9. REFLECTION .....                                            | 46 |
| REFERENCES .....                                               | 47 |

## TABLE OF FIGURES AND TABLES

|                                                                                         |        |
|-----------------------------------------------------------------------------------------|--------|
| Figure 1: Docker makes use of the Linux kernel's virtualization capabilities. [9] ..... | 7      |
| Figure 2: A Docker container development. [9].....                                      | 8      |
| Figure 3: Container Vs Virtual Machines [9] .....                                       | 8      |
| Figure 4: The Docker installation.....                                                  | 26     |
| Figure 5: WSL2 command line.....                                                        | 27     |
| Figure 6: Dockerfile for Docker shell based on OpenSSH. ....                            | 29     |
| Figure 7: SSH key generation. ....                                                      | 29     |
| Figure 8: Building of the Dockerfile image.....                                         | 30     |
| Figure 9: Docker image was run.....                                                     | 30     |
| Figure 10: Dockerfile for an image to run the docker exec command.....                  | 31     |
| Figure 11: Building the Dockerfile.....                                                 | 31     |
| Figure 12: Dockerfile for attacker container .....                                      | 32     |
| Figure 13: Docker-compose file for pivoting through a server. ....                      | 33     |
| Figure 14: Docker-compose for file2.yml. ....                                           | 34     |
| Figure 15: Docker-compose file for pivoting using metasploit .....                      | 35     |
| Figure 16: Dockerfile used for attacker service.....                                    | 36     |
| Figure 17: Docker-compose for the file. ....                                            | 36     |
| Figure 18: SSHing into the Docker container .....                                       | 37     |
| Figure 19: Output of the netstat -tuln command .....                                    | 38     |
| Figure 20: Docker image run and the docker exec command executed. ....                  | 39     |
| Figure 21: top command output.....                                                      | 39     |
| Figure 22: Access gained using docker-exec.....                                         | 40     |
| Figure 23: The command used to create SSH socks. ....                                   | 40     |
| Figure 24: ProxyChains nmap scan result. ....                                           | 41     |
| Figure 25: Access gained to attacker container using docker-exec command.....           | 42     |
| Figure 26: nmap scan in the target container.....                                       | 42     |
| Figure 27: Exploit of Metasploit auxiliary module .....                                 | 43     |
| <br>Table 1: Comparison of VMs and Containers [12] .....                                | <br>11 |

## 1. INTRODUCTION

Since the beginning of cloud technologies, people have had concerns about their security. Given that cloud computing is still in its infancy, many do not completely trust its security features. By promoting their solutions using accepted terminology, security firms create marketing responses for customers. [1] Even though many difficulties exist, modern cloud security solutions minimise many security and privacy risks.

A crucial element of cloud computing is virtualization technology, which enables hardware or software abstraction. [2] Docker is a programme that uses containers to make it easier to create, deploy, and execute applications. Using containers, a developer can bundle a programme with all of its components, including libraries and other dependencies, and send it out as a single package. By doing this, the developer can be guaranteed that the programme will function on any other system that has Docker installed, regardless of any unique settings that may be present on that machine and which may be different from those present on the machine used for creating and testing the code. Modern virtualization methods like Docker containers can operate more effectively than old-fashioned virtual machine methods. The Docker images can be readily built by the developer to their specifications. By using these images, a Docker user can easily install local Docker containers with moderate demand on hardware resources like CPU and RAM. [3] These images are instances, and each instance has the same underlying dependencies, which are represented as containers. Containers are lighter than alternative methods like virtual machines (VMs) since they only contain the instructions and code required for the application to run. Containers are perfect for various stages of the software development cycle because they can run several instances on a single real or virtual computer. [4]

Docker is a prominent container Platform as a Service (PaaS), where users can deploy and run their applications in Docker containers on top of a cloud-based infrastructure. This paradigm abstracts the infrastructure and underlying host operating system from the user, leaving them just with the task of packaging and distributing their application to the PaaS provider. Since Docker enables developers to quickly design, create, test, and deploy applications while optimising resource utilisation, many firms, like Google and Amazon, are implementing Docker into their software development life cycle. Because of the popularity of Docker, the development community has established repositories for Docker images, such as Docker Hub, to promote file sharing and boost reusability. [5] Additionally, there are standalone PaaS systems that are built on top of Docker that enable customers to deploy and operate containerized applications, such as Google's Cloud Run and Red Hat's OpenShift.

However, virtual machine technology currently performs better in terms of security while Docker container technology still has issues with security performance. To launch an attack when Docker users download and build container instances above the malicious images, for instance, Docker image producers may purposefully attach dangerous files like backdoors, computer viruses, etc. [6] Two

security firms, Fortinet and Kromtech, discovered 17 compromised Docker images containing cryptomining malware in June 2018 and had been downloaded 5 million times indicating that this code was downloaded into normally safe web applications by hackers who were able to inject commands into insecure containers. [7] On the other side, a trustworthy image developer can leave their private information in the created image files, which could result in the disclosure of crucial information. Additionally, an active Docker container instance could use up too many resources on the computer, launching a DoS attack. It is vital to eliminate the dangers that Docker container technology encounters to prevent its continued growth in popularity. [6]

The aim of this research is to examine the security of Docker containerisation, for that first, we examine the prevailing research on Docker security and discuss some of the fundamental ideas behind Docker container technology. The main objectives addressed in this paper primarily is to examine the security controls given by Docker, the problems with security, and the resources available for exploiting security flaws, secondarily to deploy images, scripts, and tools that show how to exploit containers to assess the security of containers, and finally to analyse critically how to exploit containers, how to stop exploits, and to check how secure Docker is. Techniques for detecting various threats are explained, and significant experiment results are provided.

In this dissertation, research about security in docker was tested. Instead of securing docker, attacks were performed on docker images and containers in different ways to identify various weaknesses. Identifying weaknesses and remedying them helps improve the security posture and reduce the risk of data breaches and other security incidents. By performing attacks on docker containers and images, a better understanding of the security risks associated can be gained. Performing security assessments and simulated attacks into docker containers and images provide remarkable benefits in terms of hands-on experience, improved security, and increased awareness of the best security practices of docker environments.

For this, many previous works, trusted links, and videos were referred to. Some of the previous research papers discussed the various attacks that could affect the Docker environments and how to ensure security, and the various measures that should be undertaken so as keep Docker protected. Some of the other research papers discussed various exploits that could occur in the Docker containers and images. One of the links described how to do penetration testing inside a docker container. Some of the videos that were referred to spoke about the various images in the Docker hub that are not secure and easy to attack.

## 2. LITERATURE REVIEW

In terms of containerizing applications, Docker has swiftly emerged as one of the top initiatives. Started as an open-source project, the Docker Platform enables the packing, shipping, and running of software in portable containers. Docker containers have the distinct advantages of being hardware- and platform-agnostic. These containers are not dependent on any specific language, framework, or packaging scheme. Any technological environment can execute Docker containers. These containers are no longer dependent on a specific stack or provider thanks to this functionality. [8] At platform-as-a-service (PaaS) provider dotCloud, Solomon Hykes started Docker as an internal open-source project. Open-source Docker's first version was made available in March 2013. As seen in Figure 1, Docker was created to make use of several APIs to access Linux kernel virtualization features. LXC was at first the platform's default execution environment for Docker. On the other hand, Docker discontinued LXC and introduced its own libcontainer library with version 0.9 on March 13, 2014. The Docker libcontainer library was created using the Go programming language. [9]

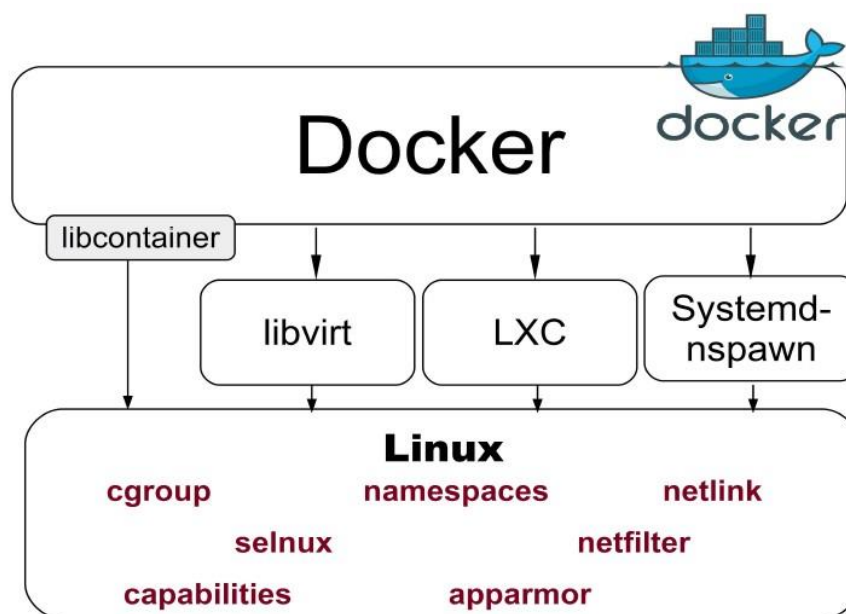


Figure 1: Docker makes use of the Linux kernel's virtualization capabilities. [9]

There are several processes involved in creating containers using Docker. We look for images in the nearby Docker library to use for constructing containers. For creating a local Docker container, Docker provides images and command lists. It is possible to download the necessary image from the main Docker repository, though, if it is not already there. The next stage is to construct a container and to create the appropriate adjustments. The entire procedure is depicted in Figure 2. [9]

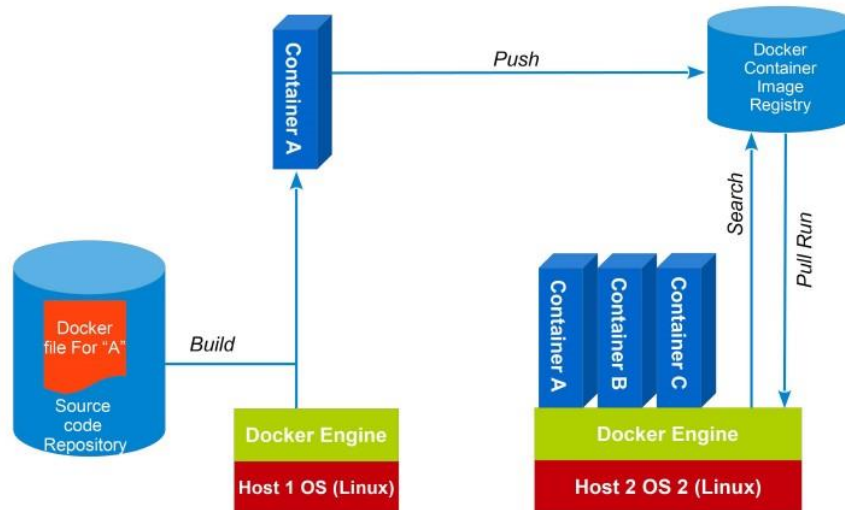


Figure 2: A Docker container development. [9]

When running applications in a virtual environment, especially in a shared-host cloud system, security can be one of the biggest concerns. It is argued that virtual machines generated by hypervisor-based virtualization solutions are more secure than containers because they add an additional layer between the host and the applications. The host kernel cannot be reached by an application running inside a virtual machine; only the VM kernel can. Hence, before attacking the host kernel, the application must get beyond the VM kernel and the hypervisor to escalate out of a VM, as shown in Figure 3. Nevertheless, containers have direct access to the host kernel, so an attacker can enter the host system with considerably less effort when using them. This makes container security an issue. [10]

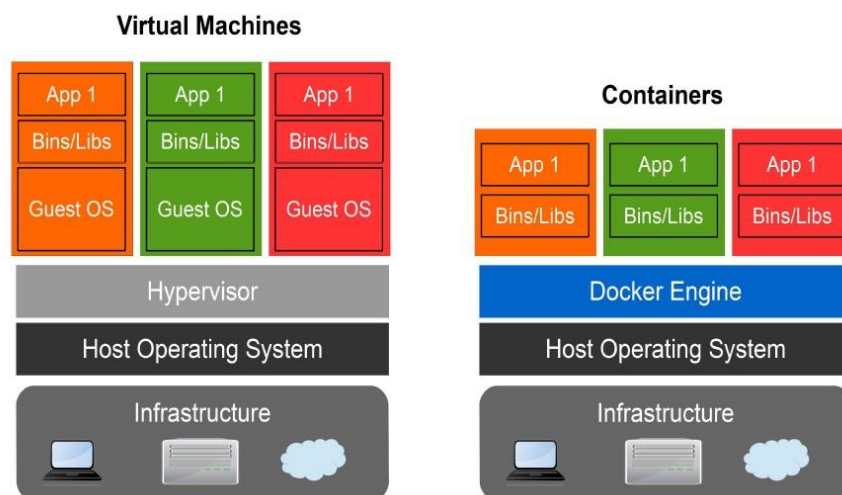


Figure 3: Container Vs Virtual Machines [9]

It is necessary to evaluate both the "inside" and "outside" sides of the system to understand Docker's security flaws. Both inside the system and from the outside, security threats are possible. Attacks from outsiders aim to breach the container, corrupt the data, and take advantage of resources. While hostile

users who are present inside can conduct insider assaults by gaining access to the Docker commands. Both types of attackers have the same primary goal, which is to harm and take use of the intellectual resources contained within the container. [9]

Insider attacks in the context of Docker are nefarious actions carried out by a person who is a member of the organisation and has access to the Docker infrastructure. These attacks can take a variety of shapes, including inserting malicious code into a container image, using a compromised account to obtain unauthorised access to resources, or stealing confidential information from a container. Strong security controls must be put in place to guard against insider attacks, including access limitations, network segmentation, and monitoring to find and stop the illegal behaviour. To make sure that the security rules and procedures are effective in securing the Docker environment, it is also crucial to routinely review and update them. In the context of Docker, "outsider attacks" are hostile actions carried out by individuals outside of a company who wants to acquire unauthorised access to the Docker infrastructure. These assaults can take a variety of shapes, including trying to access the system through network-based attacks, exploiting holes in the host system or the Docker daemon, or using tampered container images. Strong security controls, including firewall rules, access controls, and network segmentation, should be put in place to prevent unauthorised access to the Docker environment to defend against outside threats. To make sure they are secure against known vulnerabilities, the host system and the Docker daemon should also receive regular updates. To find and stop the use of compromised photos, it is also advised to employ container image scanning technologies.

## 2.1 DOCKER AND ARCHITECTURAL SECURITY

### 2.1.1 Linux Security Overview

A Docker image is fundamentally a GNU/Linux kernel that has been carefully packaged with all the libraries and tools required to execute the application in concern (for example, a MySQL database instance). Before delving into the mechanics of Docker, it is crucial to assess the condition of Linux security and comprehend the technologies that protect the host operating system from harmful attacks. Under the name of DAC, or Discretionary Access Control, the Linux kernel inherits the fundamental, if antiquated, security mechanisms from Unix itself. In accordance with this ownership structure, this privilege-based system enables the existence of many users and groups, giving them access to the same data, system locations, and applications. However, a system administrator (or superuser) can overrule these rights, which presents a security concern. [11]

Using a cryptographic API (Application Programming Interface) through a kernel module to further harden privileges across the system installation is one of the key enhancements to this fundamental structure. Another improvement is the use of LSM, or Linux Security Modules, which expands upon DAC and adds a middle layer through which apps can register and receive call-backs. Implementing isolation rules and regulations makes sure that the user privileges and application rights are kept separate

throughout the runtime. SELinux and AppArmor are two important and often-used LSM implementations. While reducing security concerns and retaining a strong competence in virtualization environments, one of which is Docker, both have grown over time to meet the always-changing landscape of software applications. [11]

Both systems' primary flaws are their dependency on user-defined rules and regulations. As a result, the tool's usability is now directly important for system configuration. In an empirical investigation on this topic, Schreuders, McGill, and Payne [12] introduced a graphical toolkit to assist administrators in the exploration and application of existing SELinux regulations utilising visualisation. Despite its limitations, the study demonstrated how visual aids improved SELinux's overall usability and made it easier for administrators to spot policy infractions.

The same team of researchers used a thorough comparison of SELinux, AppArmor, and a third LSM implementation known as FBAC-LSM in a related study that was carried out the following year. According to this comparison, FBAC-LSM was preferred among participants above the other two due to its user-friendly interface, which consequently allowed for better policy implementation and inspection. This security system is no longer in use, though. [12] [11]

### 2.1.2 Features of Linux that Support Docker

Namespaces and control groups are two distinct aspects that directly allow the creation of containers, including Docker and LXC. Namespaces are employed to offer the required mechanisms for orchestrating complicated infrastructure and separate operations at various levels. This can be applied to other applications as well, not just containers and virtualization. A certain degree of isolation, or "sandboxing," is ensured by giving a process access to a specific namespace (or several). This technique allows for the isolation of numerous containers, each of which is basically a process operating on the host OS. Namespaces guarantee isolation, while control groups—often abbreviated to cgroups—manage resources and offer the tools required for programmatic resource allocation to a process. The administrator can give each cgroup a set amount of resources through the usage of a virtual file system (VFS) in order to (a) make sure they don't use up all of the host resources; (b) prevent them from starving the system resources for other programmes. There are hardware-oriented policies too, like a device control group that can limit read/write access to physical devices. The resources are typically CPU, memory, or network related. [11]

### 2.1.3 Security in Docker

A thorough study by Dua, Raja, and Kakadia assessed different container technologies from the perspective of their applicability in a PaaS (Product as a Service) context and compared them to current virtualization technologies, such Xen and KVM. [13] Table 1 offers a high-level comparison. Containers have a number of benefits over conventional virtualization methods, especially in terms of

performance and in general manageability. From the security perspective, all kinds of containers can gain from the host OS's Mandatory Access Control via programmes like SELinux and AppArmor. [14]

| Parameters    | Containers                                                                                  | Virtual Machines                                                                                                          |
|---------------|---------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| Guest OS      | The OS and Kernel are shared by all users. Physical memory is loaded with the kernel image. | Every VM has its own kernel loaded into its own memory space and operates on virtual hardware.                            |
| Communication | Signals, pipes, and sockets are common IPC components.                                      | Takes place via Ethernet devices                                                                                          |
| Security      | Access control laws must be enforced                                                        | Depending on how Hypervisor is implemented                                                                                |
| Performance   | Compared to containers, the underlying Host OS provide a near-native performance            | Since the machine instructions are translated from the Guest OS to the Host OS, virtual machines have a little overhead.  |
| Isolation     | Subdirectories can be shared and mounted openly.                                            | It is not possible for guests and guest hosts to share libraries and files.                                               |
| Startup time  | Considering that the host OS is already running, containers can be started quickly.         | VMs take a while to start up.                                                                                             |
| Storage       | Since the base OS is shared, containers use less storage.                                   | Since the OS kernel and its related programmes must be installed, saved, and run, virtual machines require extra storage. |

Table 1: Comparison of VMs and Containers [12]

## 2.2 EXISTING DOCKER CONTAINER SECURITY MECHANISMS

The virtual objects are not isolated using a separate operating system or by virtualizing hardware in a Docker container. Instead, it makes use of Linux's Namespace and Cgroup mechanisms to ensure the secure isolation of running applications and to implement resource management. Additionally, it makes use of the kernel feature to increase security. [15]

### 2.2.1 Resource Isolation

Secure isolation in Docker is achieved by using Linux's Namespace mechanism, its objective is to stop compromised containers from interfering with other containers with resources and hence guarantees resource access transparency. Process isolation is achieved by Docker by enclosing active processes in

containers within namespaces and confining their visibility and permissions towards other processes operating in other containers as well as the underlying host. [14]

Every container instance can possess its entire network, file system, and IPC (Inner-Process Communication) using Namespace. This represents the fact that virtualization technology has expanded to include a new branch called container technology. [14]

#### A) Process Isolation

This approach is supported by PID namespaces, which separate a container's process ID number space from the hosts. [10] When the Linux clone function is called with CLONE\_NEWPID specified as an argument, a new PID namespace is created. An isolated PID environment is offered by the new namespace. This namespace's processes and any subprocesses they produce are still considered to be part of it. [14]

A process can only observe the other processes inside its own namespace or in its "children" namespaces since PID namespaces are hierarchical. As a result, after a new namespace is formed and assigned to a container, the host can view and influence the processes operating inside the new PID namespace of the container, but those processes are unable to see or interact with other processes that are active on the host or in other containers. It is more difficult to attack other processes if the attacker is unable to monitor them. [10]

The PID namespaces also permit each container to have a separate init-like process (PID 1), which when ended, terminates all the processes in the namespace. When something questionable is found, the administrator can use this technique to completely shut down a container. [10]

#### B) IPC Isolation

Processes that are run in containers must be constrained so that they are not able to interfere with those running on the host computer or in other containers, and that they are only permitted to communicate with one another via a specific set of IPC resources. [10]

Once the parameter CLONE\_NEWIPC is given when calling clone, a new IPC namespace made up of a collection of IPC object identifiers will be formed. Communication between processes from various IPC namespaces is prevented by the IPC objects created in one namespace being accessible only within that namespace and hidden in other namespaces. To provide communication isolation between processes in various containers as well as isolation between processes in containers as well as the host computer, Docker containers employ the IPC Namespace technique to separate the communications of the processes belonging to separate namespaces. [14]

#### C) Network Isolation

To stop network-based attacks like Man-in-the-Middle (MitM) and ARP spoofing, network isolation is crucial. Containers must be set up so they cannot eavesdrop on or interfere with the network traffic of other containers or the host. [10]

Once `CLONE_NEWNET` is set when running `clone`, a new network namespace would be generated. Processes may have access to a full, independent network protocol stack through a network namespace. To prevent container instances from accessing the external network, Docker uses the Network Namespace technique to construct a private network for each container. [14]

#### D) UTS Isolation

Once `CLONE_NEWUTS` is defined, a new UTS namespace is formed when using `clone`. By duplicating the namespace's identifier, which contains the aforementioned processes, the new UTS namespace's identifier is initialised. Every Docker container instance has a unique hostname and domain name because of the UTS Namespace protocol. [14]

#### E) User Isolation

A new user namespace will be allocated when the `clone` function is called with `CLONE_NEWUSER`. Users or user groups of a container may only be able to access their privileges inside of the container thanks to the user namespace. Every container instance in a Docker container will have its own root privilege, but this won't affect the processes operating on the host computer and in other containers because Docker containers employ a user namespace to achieve this. [14]

#### F) Filesystem Isolation

The host and container filesystems must be shielded against unauthorised access and change to obtain filesystem isolation. [10]

When calling `clone`, a new Mount namespace will be generated once `CLONE_NEWNS` is specified. For processes inside it, the Mount namespace offers a file hierarchical view. To guarantee safe file isolation, Docker uses the Mount namespace to enable separate file hierarchy views for container instances. [14]

Some kernel filesystems, such as those under `/sys`, `/proc/sys`, `/proc/sysrq-trigger`, `/proc/irq`, and `/proc/bus`, are not namespaced, and a Docker container must mount them to use them. Because of this, a container has the problem of inheriting the host's view of these filesystems and being able to access them directly. [10]

Docker uses two filesystem protection mechanisms to limit the threats that a compromised container could pose to the host through these filesystems: (a) denying containers write access

to these filesystems; (b) forbidding any process of a container from remounting any filesystem inside the container. [10]

### 2.2.2 Resource Control

One of the most frequent attacks on a multi-tenant system is a denial-of-service (DoS) attack, in which a process or a group of processes tries to use up all the system's resources while interfering with the functioning of the other processes. It needs to be feasible to restrict the resources allotted to each container to stop this kind of attack. [10]

The Control Group in Linux, commonly known as Cgroup, is used by Docker containers to address this problem and make sure that processes inside container instances have only access to the limited system resources, such as CPU, memory, block I/O, bandwidth, etc. As a result, it makes sure that each container gets its fair number of resources and stops any container from using them altogether. Additionally, they enable Docker to set restrictions and limitations on the resources allotted to each container. [10] [14]

The Linux kernel will trigger OOM (Out-Of-Memory), which will kill the relevant operating processes or containers, whenever the system resource runs out. Therefore, DoS attacks can be efficiently controlled thanks to the Cgroup mechanism. [14]

### 2.2.3 Limitation of Kernel Capability

When executing permission is being checked for in a classic UNIX system, users are split into two categories: root users and non-root users. Root users have a great deal of privilege and can ignore the kernel's permission check. However, non-root users must rigorously abide by the execution authorization in order to complete their responsibilities. The capability mechanism for more granular access control is introduced in Linux 2.2 and works with both processes and files [16]. Non-root users will now be able to finish tasks that were previously only available to root users. [14]

Because the host system and Docker containers share a kernel, the host computer is able to do various tasks. As a result, it is not essential to provide containers with complete root permission and eliminating some kernel capabilities will not have an impact on the availability and functionality of a Docker container. Instead, the level of system security is raised. [14]

## 2.3 ANALYSIS OF THREATS FOR DOCKER CONTAINER

### 2.3.1 Components Security

The modules for the Docker client, Docker daemon, and Docker registration are all included in a Docker container. Each module operates separately from the others. They may construct, operate, and delete a container instance together and upload and download Docker images via the Docker Hub. [17] [14]

### 2.3.2 Container Escape Attack

The kernel-sharing approach used by Docker container technology lowers the performance cost of virtualization and relies on the kernel to control the behaviour of container instances. Some exposures present in the host computer's system may enable processes operating in containers to gain access to the host computer's system, resulting in privilege escalation [18]. This so-called escape exploit renders container isolation ineffective, negatively impacting computer security as a result. [14]

### 2.3.3 Sensitive Information Leakage

Leakage of private data poses a serious issue for Docker container technology. If API authentication data or database passwords have been compromised, users may experience significant consequences. But the entire service could be terminated at any time if security flaws are discovered. More focus on the auditing of sensitive information is very necessary. To secure the information security of Docker users, adequate authentication techniques for accessing sensitive information are required while Docker containers are running on the host machine. [14]

### 2.3.4 Network Mode Security

There are four different network modes available in Docker: none, host, container, and bridge. A Docker container operates in none mode when no network parameters have been specified and it has its own independent network namespace. In host mode, containers and the host computer share the same network namespace and use the host system's IP address and ports. Containers can be allocated to the specified network namespace of other containers in container mode. Bridge mode allows Docker containers to communicate by using the host computer's ports. [14]

## 2.4 DOCKER IMAGE SECURITY

Regulating and producing Docker Images is a crucial part of how Docker images are put together. In essence, the Docker Engine generates these image files based on a set of instructions. An image's creation and all the circumstances in which it is used collectively produce intriguing metrics that may also include harmful or out-of-date images. These vulnerabilities can also be discovered using widely used frameworks like CVE (Common Vulnerability Assessment) and tools like Docker Notary. The necessary code, configuration mapping, and packages are all contained in image files [19]. The result is that an image can be malicious or even contain susceptible packages that could have security concerns based on any of these components. Currently, there are numerous methods and tools for finding these things, but the static and dynamic analysis is one area of study. [11]

The following actions can be taken to guarantee the safety of your Docker images:

Utilize authorized photos from reliable sources: The software provider offers official images, which are typically thought to be more secure than images created by other parties.

Maintain regular image updates: To guarantee that the images include the most recent security updates and fixes, it is crucial to update them frequently.

Use as few base images as possible: The basic image's vulnerabilities are less likely to exist the smaller it is.

Check for weaknesses in images: A variety of programmes are available that can check pictures for known vulnerabilities and make fixes suggestions.

Use digitally signed images: Docker Content Trust enables to use of signed images to help assure the validity and integrity of the images.

## 2.5 ATTACKER MODELS

An attacker model is a model of how an attacker could try to take advantage of security holes in a system. It is employed to recognise potential security issues and provide information on the creation of countermeasures to lessen those risks. There are two perspectives when discussed about containers. The first viewpoint (inside a container) is an attacker model in which an attacker has acquired access to a container. The attacker has access to everything inside the container and can run commands inside it. This kind of attack is known as a container escape since the attacker primarily concentrates on escaping the isolation that the container delivers. The second viewpoint (not inside a container) is that of an attacker who has unrestricted access to a host. Although the attacker has some access to the host, they do not have full control. This type of attack is known as a Docker daemon attack because the attacker will use Docker (particularly the Docker daemon) on the host to get access. [20]

## 2.6 DIFFERENT EXPLOITS IN A DOCKER CONTAINER

The act of employing flaws or vulnerabilities in a Docker container or its application to gain unwanted access or control of the container or the host system it runs on is known as "exploiting" a container. This can be accomplished in a number of ways, including by taking advantage of software flaws, configuration errors, or stolen credentials.

### 2.6.1 Container Image Malware

Malicious code or payloads that are embedded within container images are referred to as container image malware. The threat of malware in container images is so constant that, according to a InfoQ analysis, more than half of the publicly accessible container images on Docker Hub, the most well-known public container registry, include exploitable vulnerabilities. When used and executed, these pictures may compromise the host system or steal private data. Before deploying container images in a production environment, it is crucial to fully validate and scan them for malware. Malware can run inside the environment alongside the "legal" contents of the container image if containers containing malware are deployed. [21] Additionally, executing containers with restricted access and utilising secure images from reliable sources will lessen the danger of malware infected container images.

Malware attacks in containers are comparatively simple to stop. The primary answer is to use a container image scanner to scan container images before deploying them, and never blindly trust a container image no matter where it came from. It's crucial to understand that malware can often only be found using container image scanners if it has been tracked using publicly accessible vulnerability databases. As a result, even if the photos pass the scans, unidentified malware can be present. However, the vast majority of malware that might be concealed within photographs will be found via image scanning. By prohibiting hostile parties from introducing weak images into the registry, restricting access to the container registry can also lower the risk of container infection. [21]

### 2.6.2 Privilege Escalation Exploits

When a process running inside a container successfully accesses resources that it shouldn't be able to control (such as programmes running in other containers or on the host operating system), this is known as a privilege escalation exploit. These flaws provide an attacker access to more sensitive information, privileged accounts, or system resources than they otherwise would. Exploiting a web application's vulnerability to access the underlying operating system is an example of a privilege escalation attack, as is leveraging a poorly configured service to access confidential information. The software that executes containers, the container runtime, frequently has bugs that lead to privilege escalations. [21]

Threats of container privilege escalation can be reduced in two major ways. The first is to confirm that all of the container tooling is current. In this manner, known attacks won't affect the runtime, orchestrator, and other components of the stack. Locking down container access privileges using all of the tools available is the second component of the approach. For this, host-level frameworks like SELinux or AppArmor can be applied. [21]

### 2.6.3 Access Control Exploits

If containers are given more rights than they should, problems could occur. Attackers that succeed in taking over the container in that situation could be able to utilise it to access other resources on the host. The container comes with unneeded rights by default, so they don't need to take advantage of a software flaw to accomplish this. When appropriate security procedures are not followed or installed properly, this can occur. It is significant to remember that access control vulnerabilities in container systems affect the host system, orchestration software, and underpinning infrastructure in addition to the container technology itself. [21]

Strong access controls must be implemented in order to effectively mitigate this kind of danger. When defining access control rights for each container, be sure to follow the concept of least privilege and refrain from making mistakes like permitting containers to run in "privileged" mode (which gives the container root access to host-level resources). Most of the time, these rights can be controlled utilising the access control architecture of the orchestrator. [21]

#### 2.6.4 Network Security Exploits

By allowing containers to communicate over the network, improperly implemented networking rules might also serve as a channel for attack or worsen exploits that occur from other sources. It's crucial to set up container networking correctly, utilise only secure, recent container images, keep the host system and containerization software updated, and set up host networking correctly in order to safeguard a container environment from network security attacks. Additionally, it's critical to keep a regular eye out for any strange activity in the container environment and to have a reaction strategy ready in case of a security problem. [21]

Within the first two tiers of your stack, you can enforce this regulation. First, limit network communications between containers using the orchestrator. For instance, network policies can be created in Kubernetes to provide guidelines for container network traffic. Next, check the container pictures to see whether any unsecured network settings are there. For instance, based on the image, image scanners can assist in locating vulnerable network ports that are configured to be open inside containers. [21]

#### 2.6.5 Orchestrator Container Exploits

The deployment, scaling, and management of containerized applications are tasks that may be managed and automated using a container orchestrator. The container orchestrators Kubernetes, Docker Swarm, and Apache Mesos are a few examples. Container orchestrators can have flaws that attackers can use, just like any other software. [21]

There are numerous methods that can be used to protect the orchestrator. The first is to implement least privilege using the Role-Based Access Control (RBAC) mechanisms offered by the orchestrator. Second, deploying SELinux and AppArmor on nodes can protect them from orchestrator vulnerabilities. Lastly, if the orchestrator supports them, auditing features can be utilised to identify dubious requests. The risk of an exploit against one workload "spilling over" to damage another can be reduced by separating unrelated workloads using abstractions like Kubernetes namespaces. By putting these procedures into place, the orchestrator can be properly shielded from threats and vulnerabilities, assuring the system's seamless operation. [21]

### 2.7 SOME PAPERS AND ARTICLES THAT SHOWED DIFFERENT WAYS TO ACCESS CONTAINERS

McGrew, in his paper [22] explores the security issues connected to using Docker to operate multi-container applications. McGrew's research focuses on the potential compromise of a Docker environment by an attacker who takes advantage of flaws in one or more of the containers. Before discussing different attack vectors that could be exploited to compromise a multi-container application, McGrew first gives an overview of Docker and its architecture. With a multi-container application, he

continues, if one container is compromised, the attacker might be able to access the other containers as well as the host operating system. McGrew also highlights the numerous security features Docker offers, such as namespace isolation, resource restrictions, and access controls, to help reduce these threats. He does point out that these characteristics can be challenging to set up correctly and could not offer sufficient defence against all kinds of attacks. Last but not least, McGrew offers several practical suggestions for protecting multi-container Docker applications, including the use of image vulnerability scanners, restricting container rights, and putting network segmentation into practise. Ultimately, McGrew's study emphasises the significance of being aware of the security concerns posed by Docker and offers insightful advice for programmers and system administrators who are in charge of safeguarding multi-container applications.

In his paper [23], Shaun McCullough explains how Docker can be used to build multi-container environments that make it possible to test and share lateral movement scenarios in a secure and regulated way. McCullough starts with giving a general explanation of lateral movement, which is the technique an attacker uses to access numerous computers within a network by taking advantage of flaws or vulnerabilities. He argues that because they include several systems and have the potential to be harmful if not carried out in a controlled environment, lateral movement scenarios can be challenging to test and communicate. Next, McCullough goes on to explain how Docker can be used to build multi-container setups that mimic networks and make it possible to test and share lateral movement scenarios. He explains that Docker offers a portable and light approach to build and manage numerous containers, each of which can stand in for a distinct network system or service. McCullough gives thorough instructions on how to use Docker to set up a multi-container environment that is suitable for lateral movement testing. Also, he talks about some of the difficulties and restrictions associated with utilising Docker for this purpose, including the requirement for correct network configuration and the potential for resource limitations. Overall, McCullough's work shows how Docker might be a useful tool for researchers and security experts that need to test and communicate lateral movement situations in a secure setting. These experts can better understand how attackers might move laterally within a network and devise more effective techniques for detecting and stopping these types of attacks by using Docker to construct multi-container environments.

In Lenny Zeltser article [24], he describes how to use Docker to execute the well-known penetration testing programme Metasploit Framework without the need for any installation or configuration. Zeltser starts off by giving a general introduction of Docker and how it can be used to build and execute containers that include apps and all of their dependencies. The advantages of using Docker to execute Metasploit, including enhanced portability and reproducibility as well as streamlined setup and teardown, are then discussed. Zeltser gives step-by-step instructions on how to install and run a Metasploit container using Docker, including how to download and set up the required Docker image, how to set up and connect to the container, and how to use Metasploit inside the container to carry out

various kinds of penetration testing activities. Also, Zeltser talks on some of the potential drawbacks and difficulties of utilising Docker to run Metasploit, including the necessity for adequate resource management for containers and the potential for incompatibilities between various versions of Metasploit and its dependencies.

Swordfish-"Pentest-In-Docker" Security's GitHub repository [25] offers a selection of Docker containers that are pre-configured with different security tools and scripts to aid security professionals and researchers in doing penetration testing tasks. The repository contains a Docker Compose file that details the many containers and how they relate to one another in addition to comprehensive setup and usage instructions. A variety of security tools are contained in the containers, including Wireshark, Nmap, Nikto, Metasploit, and others. The project's objective is to deliver a suite of penetration testing tools that are simple to deploy, adaptable, and run in a Docker environment for increased portability and simpler dependency management. The community is encouraged to add new tools or improve the ones that are already there by contributing to the repository. In conclusion, the "Pentest-In-Docker" repository is a helpful tool for security experts and researchers who want to use Docker to carry out penetration testing tasks in a flexible and portable way.

Security experts and penetration testers can use the "Pivot Cheatsheet for Pentesters" document from NullSweep [26] as a reference on how to pivot and move laterally through networks using compromised hosts. The importance of lateral movement in penetration testing and network pivoting are introduced at the outset of the study. Following that, it offers a rundown of popular network pivoting methods and implements, such as local and remote port forwarding, SOCKS proxies, and VPNs. The article then gives detailed instructions on how to carry out network pivoting using each of these methods, along with examples and screenshots to show how the procedure is carried out. The guidelines address a variety of situations, such as pivoting via vulnerable hosts and firewalls and gaining access to internal systems that are not readily available on the Internet. The document provides both technical instructions and recommendations on how to accomplish network pivoting while remaining undetected and stealthy. This involves limiting the impact of the pivoting on the target network, avoiding suspicious activity, and employing encrypted communication routes.

### 3. RESEARCH METHODS

This paper focuses on finding out how secure Docker is, for this purpose freely accessible and publicly available software was used, and attacks were performed on Linux-based workstation.

Some of the many ways to exploit a container were considered.

The first was using OpenSSH to gain shell access in Docker. For this purpose, any of the base images can be used, here the Ubuntu base image was considered. Using Ubuntu is better because many users may find it simpler to understand as this image is well-known and is a widely used operating system. Even installing and managing necessary dependencies for applications is easy because of Ubuntu's extensive repository of packages and libraries. OpenSSHing into a Docker container is done to get access to a shell or command line interface with the container directly. This can be also seen as white-hat hacking, as OpenSSH provides a way for security professionals to securely access and assess the security of remote systems. It can be used to test the system's security measures and identify vulnerabilities. OpenSSH gives a secure and encrypted connection between two machines, ensuring that sensitive data is protected during transmission. By logging into a Docker container using OpenSSH, the container can be accessed from another machine or device. OpenSSH can manage and configure a Docker container by logging in and making changes to files, installing packages, and setting environment variables. Key-based authentication, a safe and practical method of accessing remote machines, is supported by OpenSSH. In order to access remote computers, whether they are real-world servers or Docker containers, the same key-based authentication can be used when logging into a Docker container. In the Dockerfile, port 22 was exposed, this means making the SSH service accessible to the outside world by mapping the container's port 22 to a port on the host machine. Exposing the SSH service on port 22 will increase the attack surface of the system and can make it more vulnerable to attacks.

The second one was using the docker exec command. It is similar to using SSH and OpenSSH, it allows one to log into a remote machine and run commands on it. However, the docker exec is specifically used for Docker containers, while SSH and OpenSSH are more general-purpose tools. When docker exec is used to log into a running container, one gets access to the container's environment and file system, which allows one to run commands and make changes to the container, which can be useful for tasks like debugging, maintenance, or testing. Commands can be issued and communicated with the container once logged in, just like one would with a remote server. From within a container, services could be set up, install packages, and even manage files. As a result, logging into a running Docker container using docker exec is comparable to accessing a remote server in the real world because it gives the ability to communicate with the container and control its services and configuration.

The next way is pivoting through a server using SSH, pivoting in a penetration test means using the primary point of entry to obtain access to additional systems within a network. This technique is used by ethical hackers to mimic the actions of an attacker who could break into a system and attempt to take over the entire network. When SSH to pivot through a Docker container is used, first one must log into the container using `docker exec` or `ssh` and then use the `ssh` command within that container to connect to another system. The Docker container would act as a bridge or gateway between the local and remote systems as if one was directly connected to it. In order to perform activities like troubleshooting, testing, or security assessments, this technique might be helpful because it enables one to connect to remote systems that would otherwise be inaccessible. It necessitates opening the network of the Docker container and permitting inbound connections, both of which could pose security threats. By pivoting through a Docker container, one could gain access and test remote systems in a controlled and isolated environment. In the real world, if a system administrator needs access to a remote server that is located in a restricted network, the system administrator would have to set up Docker container on the server that has access to both their local machine and remote server and log into the Docker container using `docker exec` or `ssh`, and then use the `ssh` command within the container to connect to the remote server. Here, the Docker container would be acting as the bridge between the system administrator's local machine and remote server.

Another way is pivoting using the Metasploit framework. Meterpreter is a payload which a Metasploit component. It is an open-source tool that is used for security testing and vulnerability assessment. It is a kind of reverse shell that gives the compromised system an interactive shell that enables remote access and control by an attacker. By using Metasploit to pivot within a Docker container, it isolates the pivoting activity from the rest of the network. Using Metasploit to pivot inside a Docker container is often done as part of a security testing or vulnerability assessment engagement in the real world. This procedure is carried out to assess the network or system's security and find any flaws or vulnerabilities to attack or that an attacker could exploit. The procedure typically entails installing Metasploit in a Docker container and then utilising the tool to find and exploit weaknesses in the target system. After gaining entry, the attacker can utilise the Meterpreter payload to navigate the target system and learn more about its security measures.

This research focuses on container attacks on Linux-based machines, and the use of free, open-source and publicly available software. It also goes into detail about `docker-compose` and how to coordinate multiple containers. Once the image is built and the container for that image is run, the `docker exec` command or the `ssh` command could be used to gain access into the container. `ssh` command can be used only if the necessary packages are installed in the container. Inside the container, commands or programs that are available in the base image or any additional images that were added to the container. This could be running services, compiling, and running code, and executing scripts. When the container is started, the command or entry point can be specified that one wants to run inside the container. For

instance, by entering the command to start the server, one can launch a container which runs a web server.

When the container is up and running, one can connect to relevant ports on the container to view the web server. By using the shell or other tools, one can access and modify files and folders inside the container and even interact with the container's file system. However, unless a mechanism is set up to persist the modifications, all the modifications made to the container's file system would be lost when the container is stopped. By default, Docker containers have restricted access to system resources and are isolated from the host system and other containers. The container can, however, be set up to have access to more resources or to operate in privileged mode if necessary.

To test the consistency of the results, after starting the container that runs a web server on a specific port, port scanning tools like Nmap, telnet or netstat were used to check if the ports are open. The port scanning step is repeated several times to see if the results were consistent. The security of Docker containers is tested by trying different attack scenarios. The test environment is set up with a Docker host and containers running different services. Using appropriate tools, the attacks are run that are to be tested, here for remote code execution. The behaviour of the Docker containers is monitored, using docker logs or other monitoring tools. The attacks are repeated multiple times to test the consistency of the results, different attack tools can also be used. The results are analysed to check if any vulnerabilities were exploited.

## 4. RESEARCH DEVELOPMENTS

Docker can be run on a variety of environments, some of them being Linux, Windows, macOS, Cloud-based environments, and virtual machines, but the primary environment for running Docker is Linux. Linux offers a lightweight, adaptable environment for running Docker containers and commands, and Linux was the platform on which Docker was first developed. However, running Docker on other operating systems allows using Docker with some performance overhead by allowing it to run Docker commands inside a virtual machine that utilises a Linux kernel. Additionally, running Docker commands in cloud-based environments is simple to operate and administer Docker containers on the cloud due to the variety of hosting and management capabilities provided by these cloud providers.

Docker commands can be run on macOS using Docker Desktop for Mac and the containers on macOS run in a Linux virtual machine. Similarly, Docker Desktop in Windows uses Linux virtual machine. Cloud systems like Amazon Web Services (AWS), Microsoft Azure or Google Cloud Platform (GCP), can also be used to run Docker containers. Virtual machines, which can be used to build isolated environments for testing or development, can be used to run Docker commands. Virtual machines like VMware or VirtualBox can be used to execute Docker instructions.

Ultimately, the choice of environment for running Docker depends on one's specific requirements. Since a Windows system was used here, Docker Desktop for Windows and Windows Subsystem for Linux version 2 (WSL2) were installed and checked, as that would be an efficient and straightforward option.

### 4.1 SYSTEM CONFIGURATION

The system used for the installation was a 64-bit version. Docker Desktop version 4.16.3 was installed in the system, here, HP Pavilion Laptop 14. Docker Desktop makes creating and sharing microservices and containerised applications simpler. It is a one-click installation programme for Linux, Mac and Windows environments. It is also an open-source platform that offers a simple graphical user interface (GUI) to control the images, containers and applications created from the computers. It handles file system issues, port mappings, and other default settings, and is frequently updated with security and bug patches.

A Windows Subsystem for Linux version 2 (WSL2) as a backend was necessary to run the Docker commands. To install WSL2, a Windows version of 10 or later was necessary. Microsoft created the Windows Subsystem for Linux (WSL) 2, a full Linux kernel that enables Linux distributions to function without having to worry about maintaining virtual machines. Docker Desktop speeds up CPU and memory-intensive processes like creating containers by only using the necessary amount of CPU and memory resources. Similarly, WSL2 considerably reduces the time needed to launch a Docker daemon

after a cold start. Attacks can be performed in many ways, here I have used a simple way of attacking through the WSL2 command line.

## 4.2 ANALYSIS OF WSL2 AND DOCKER

As mentioned above, Docker version 4.16.3 and WSL2 were installed on the system. In order to check the installed program, an example is run using the WSL2 command line. Figure 4 shows the *docker info*, this command displays information about the Docker installation, like version number, the number of containers and images, and the number of system resources being used, etc. Figure 5 shows an inbuilt example from the Docker Hub that was run using the WSL2 command line. This example is the hello-world image. It was pulled from the Docker Hub using the *docker pull* command, the image was then run using the *docker run* command to get the output of the image. The official hello-world image's output is a simple single static binary which prints some texts as standard output. It can be trivially run by any arbitrary user. The Docker daemon has to be running in WSL2 integrations in the Docker settings, otherwise, the containers may not run if the daemon is not running.

WSL2 was installed from the command prompt by running the command *wsl --install*, this will install all the WSL2 components. WSL2 does not come with built-in applications. Instead, it provides a Linux environment that allows installing our own Linux applications and tools. When WSL2 is installed, it was prompted to install a Linux distribution, such as Ubuntu, Debian, or Kali Linux. Here, Ubuntu was installed using *wsl --install -d Ubuntu* command. In order to update the WSL2 kernel with the installed Ubuntu distribution, *wsl --update* command was run. Once, the distribution was installed and updated, the terminal was ready to be used for running commands and even for installing additional packages, applications, and tools.

```
aneena06@LAPTOP-PFDF8TOB: /mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles$ docker info
Client:
Context:    default
Debug Mode: false
Plugins:
  buildx: Docker Buildx (Docker Inc., v0.10.0)
  compose: Docker Compose (Docker Inc., v2.15.1)
  dev: Docker Dev Environments (Docker Inc., v0.0.5)
  extension: Manages Docker extensions (Docker Inc., v0.2.17)
  sbom: View the packaged-based Software Bill Of Materials (SBOM) for an image (Anchore Inc., 0.6.0)
  scan: Docker Scan (Docker Inc., v0.23.0)

Server:
Containers: 15
  Running: 0
  Paused: 0
  Stopped: 15
Images: 11
Server Version: 20.10.22
Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Native Overlay Diff: true
  userxattr: false
Logging Driver: json-file
Cgroup Driver: cgroupfs
Cgroup Version: 1
Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog

Swarm: inactive
Runtimes: io.containerd.runtime.v1.linux runc io.containerd.runc.v2
Default Runtime: runc
Init Binary: docker-init
containerd version: 9ba4b250366a5ddde94bb7c9d1def331423aa323
runc version: v1.1.4-0-g5fd4c4d
init version: de40ad0
Security Options:
  seccomp
   Profile: default
Kernel Version: 5.15.79.1-microsoft-standard-WSL2
Operating System: Docker Desktop
OSType: linux
Architecture: x86_64
CPUs: 8
Total Memory: 7.645GiB
Name: docker-desktop
ID: X6NS:BJ2N:PJBM:SNBC:G3BH:I67I:ZESN:X6MP:WLFW:QPHP:XULM:NRFY
Docker Root Dir: /var/lib/docker
Debug Mode: false
HTTP Proxy: http.docker.internal:3128
HTTPS Proxy: http.docker.internal:3128
No Proxy: hubproxy.docker.internal
Registry: https://index.docker.io/v1/
Labels:
Experimental: false
Insecure Registries:
  hubproxy.docker.internal:5000
  127.0.0.0/8
Live Restore Enabled: false
```

Figure 4: The Docker installation

```
aneena06@LAPTOP-PFDF8TO x + v
WARNING: No blkio throttle.write_iops_device support
aneena06@LAPTOP-PFDF8TO: /mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles$ docker pull hello-world
Using default tag: latest
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:aa0cc8055b82dc2509bed2e19b275c8f463506616377219d9642221ab53cf9fe
Status: Downloaded newer image for hello-world:latest
docker.io/library/hello-world:latest
aneena06@LAPTOP-PFDF8TO: /mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

Figure 5: WSL2 command line

The tools necessary for exploiting are not installed separately, but instead installed in the Dockerfile and Docker compose YAML file. However, there are some tools and packages that are often included in some Docker images.

The packages and dependencies are installed using the *apt-get* or *yum* depending on the operating system used in the container. Here, since the Ubuntu distribution is used, *apt-get* is used to install, update and remove packages and dependencies from the APT repository. *yum* is the package manager used in Red Hat-based distributions including CentOS, Red Hat Enterprise Linux (RHEL). The Docker images that are written in scripting languages required the interpreter and its libraries. For instance, python and pip were needed to install the required Python packages for the images that were written in Python. Networking tools such as curl, wget or netstat were included for Docker containers that required network access. Security tools like Nmap, chkrootkit, or clamav were included to ensure the security of the container. The necessary tools and packages are included in the Dockerfile and Docker compose YAML files to keep the image as small and efficient as possible. This helps in improving the speed and security of containerised applications.

## 5. IMPLEMENTATION

### 5.1 EXPLORATION OF DOCKER CONTAINER APPLICATIONS

#### 5.1.1 Method: SSHing into the Docker container

SSHing into a Docker container enables one to gain access to a shell prompt within the container and carry out administration operations including software installation, configuration file management, and problem-solving. Since it gives users more control and insight into the underlying system, it can be used as an alternative to running a command directly within a Docker container. When a certain tool is missing from the container image or when modifications have to be made to the container that can only be made using the command line, SSHing into a Docker container can be helpful. Using OpenSSH and the `docker exec` command are the two ways to gain shell access to a Docker container.

#### 5.1.2 Method 1: OpenSSH-based Docker shell.

A base image was selected from the Docker Hub, here, the Ubuntu image was selected. In order to enable SSH, the docker image had to be pre-configured with an OpenSSH server. For this, a Dockerfile was created, in the Dockerfile, the networking tool `net-tools` were installed along with the OpenSSH package was installed and any dependencies that were necessary to run the SSH server. Figure 6 shows a snippet of the Dockerfile that was created. The Dockerfile starts by pulling the `ubuntu:latest` base image. The OpenSSH server is installed, a new user is added named `sshuser`, and a group named `sshgroup`. The Dockerfile then copies the SSH public key file to the `authorized_keys` file in the container, changes the ownership of the key file and starts the SSH services. It then runs to change the `sshuser` password to `sshpas`, exposes port 22 and runs the SSH daemon.

Next, the SSH key pair was set up for secure authentication. Either a new key pair could be generated, or the existing key pair could be used. The public key to the Docker container had to be copied and the SSH server had to be configured to use it. Figure 7 shows the SSH key generation that was done. Here, when it was prompted to give the file in which the key has to be stored, `idkey` was given as the file name.

The SSH port (usually port 22) was exposed in the Dockerfile. The Docker image was built, and the container was run, specifying the exposed SSH port to map it to the host port. The build image was then run. Figure 9 shows the image that was created named `sshaneena` and the container that was run by mapping port 22 to host port 2022. To verify that the container was running the `docker ps` command is given. After this, the Docker container was connected to the SSH client, here the `ssh` command-line tool.

```

aneena06@LAPTOP-PFDF8TOB:/mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles$ cat Dockerfile
FROM ubuntu:latest

RUN apt-get update && apt-get install -y \
net-tools

RUN apt update && apt install -y openssh-server

RUN groupadd sshgroup && useradd -ms /bin/bash -g sshgroup sshuser
RUN mkdir -p /home/sshuser/.ssh
COPY idkey.pub /home/sshuser/.ssh/authorized_keys
RUN chown sshuser:sshgroup /home/sshuser/.ssh/authorized_keys && chmod 600 /home/sshuser/.ssh/authorized_keys
RUN service ssh start
RUN echo 'sshuser:sshpass1' | chpasswd
EXPOSE 22
CMD ["/usr/sbin/sshd","-D"]
aneena06@LAPTOP-PFDF8TOB:/mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles$

```

Figure 6: Dockerfile for Docker shell based on OpenSSH.

```

aneena06@LAPTOP-PFDF8TOB x + v - □ ×
aneena06@LAPTOP-PFDF8TOB:/mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/aneena06/.ssh/id_rsa): idkey
idkey already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in idkey
Your public key has been saved in idkey.pub
The key fingerprint is:
SHA256:Cn/3TlWmrheWG6Yud+u/R+MY0EJEj7ZrW0aoUSA3A7k aneena06@LAPTOP-PFDF8TOB
The key's randomart image is:
+---[RSA 3072]-----+
|      oo=oo      |
|      .o +.o     |
|      . +.. o    |
|      E  oo. +   |
|      . S. oo.o.  |
|      o . o o+*.  |
|      o ...o.*==. |
|      . .oo*o+..  |
|      *+=o++     |
+---[SHA256]-----+

```

Figure 7: SSH key generation.

```
aneena06@LAPTOP-PFDF8TOB: /mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles$ vi Dockerfile
aneena06@LAPTOP-PFDF8TOB: /mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles$ docker build -t sshaneena .
[+] Building 4.6s (15/15) FINISHED
=> [internal] load build definition from Dockerfile 0.1s
=> => transferring dockerfile: 526B 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [internal] load metadata for docker.io/library/ubuntu:latest 2.8s
=> [auth] library/ubuntu:pull token for registry-1.docker.io 0.0s
=> [1/9] FROM docker.io/library/ubuntu:latest@sha256:9a0bdd4188b896a372804be2384015e90e3f84906b750c1a53539b585f 0.0s
=> => resolve docker.io/library/ubuntu:latest@sha256:9a0bdd4188b896a372804be2384015e90e3f84906b750c1a53539b585f 0.0s
=> [internal] load build context 0.0s
=> => transferring context: 616B 0.0s
=> CACHED [2/9] RUN apt-get update && apt-get install -y net-tools 0.0s
=> CACHED [3/9] RUN apt update && apt install -y openssh-server 0.0s
=> CACHED [4/9] RUN groupadd sshgroup && useradd -ms /bin/bash -g sshgroup sshuser 0.0s
=> CACHED [5/9] RUN mkdir -p /home/sshuser/.ssh 0.0s
=> [6/9] COPY idkey.pub /home/sshuser/.ssh/authorized_keys 0.0s
=> [7/9] RUN chown sshuser:sshgroup /home/sshuser/.ssh/authorized_keys && chmod 600 /home/sshuser/.ssh/authorize 0.4s
=> [8/9] RUN service ssh start 0.5s
=> [9/9] RUN echo 'sshuser:sshpassl' | chpasswd 0.6s
=> exporting to image 0.1s
=> => exporting layers 0.1s
=> => writing image sha256:99639185e470e780d69a717dc51a0cd8026a693a7750208dd7e66c9d759ed190 0.0s
=> => naming to docker.io/library/sshaneena 0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
aneena06@LAPTOP-PFDF8TOB: /mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles$ docker run -d -p 2023:22 sshaneena
```

Figure 8: Building of the Dockerfile image.

```
aneena06@LAPTOP-PFDF8TOB: /mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles$ docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
sshaneena            latest          37c5cce20598    54 minutes ago 231MB
dockerfiles-worker   latest          32629c630ac3    6 days ago     231MB
dockerfiles-vote     latest          9d10158dc9cc    6 days ago     231MB
dockerfiles-result   latest          7e5fda1f3d24    6 days ago     231MB
mysql                latest          cdf3aa69f5f0    9 days ago     517MB
sshu                 latest          6047d5cd0bc4    2 weeks ago    231MB
redis                alpine          53278c28ec70    2 weeks ago    29.9MB
aneena               latest          b56c13ff44da    2 weeks ago    231MB
joomla               latest          5f6ec0b23869    3 weeks ago    633MB
alpine               latest          042a816809aa    4 weeks ago    7.05MB
mysql                5.6            dd3b2a5dcb48    13 months ago 303MB
hello-world          latest          feb5d9fea6a5    16 months ago 13.3kB
postgres             9.4            ed5a45034282    2 years ago    251MB
aneena06@LAPTOP-PFDF8TOB: /mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles$ docker run -d -p 2023:22 sshaneena
805c9a28dd9e1388de0be85e5743fede67ed8b130e7b309ddd9f517d7e4903f7
aneena06@LAPTOP-PFDF8TOB: /mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED    STATUS    PORTS    NAMES
805c9a28dd9e   sshaneena "/usr/sbin/sshd -D"      3 seconds ago Up 2 seconds 0.0.0.0:2023->22/tcp focused_spence
aneena06@LAPTOP-PFDF8TOB: /mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles$
```

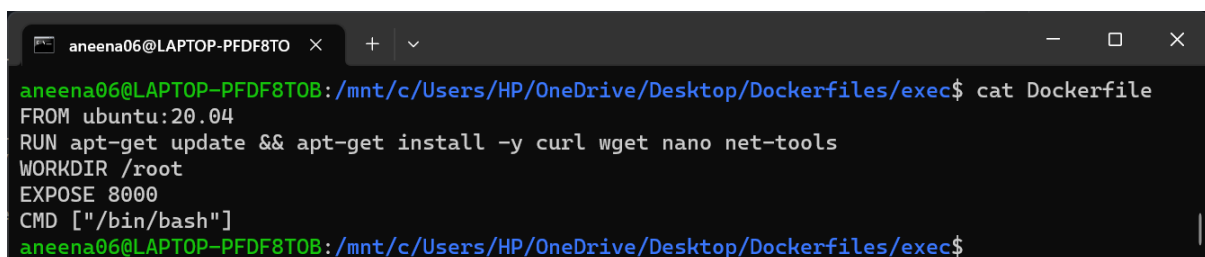
Figure 9: Docker image was run.

SSHing inside a Docker container is typically not advised because it violates containerization principles. A single process or a small number of processes can run in an isolated, self-contained environment known as a container. When SSHed into a container, a shell is shown and the ability to execute several processes is given, which might result in unpredictable behaviour and security problems. Furthermore, since changes made inside the container are not preserved when the container is restarted, it could also make it more difficult to manage the state and configuration of the container.

When SSHing into a container, already two services have been reached because OpenSSH is running in the container to use SSH. Additionally, it was much simpler to use Docker's built-in approach of using the *docker exec* command to conduct SSH commands. Without an SSH server, this command could be used to access the shell or execute remote commands.

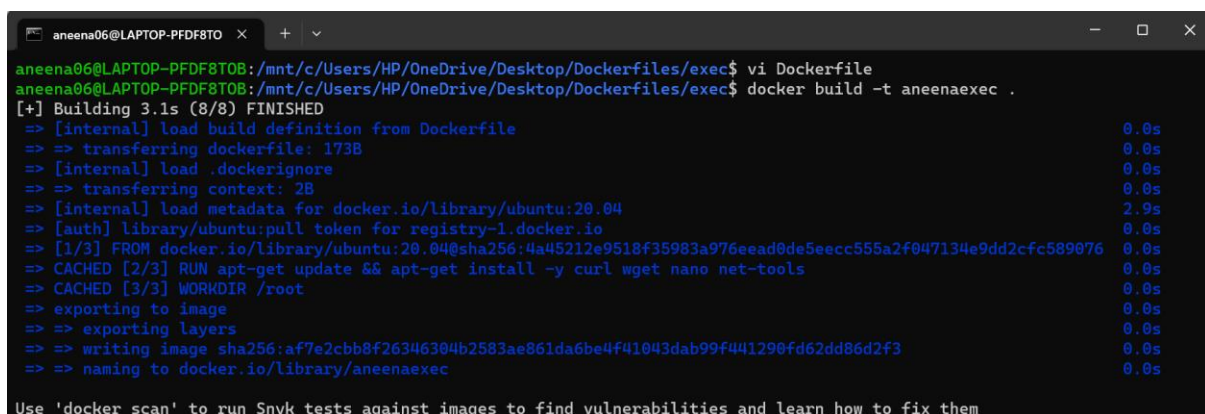
### 5.1.3 Method 2: Using the docker exec command in a shell

Shell access can be obtained in an active container by using the built-in remote shell API of Docker using the built-in *docker exec* command. When this command was executed, the container's prompt was reached. Any Linux command could be run from there. For instance, the *top* command was used. This command displayed details on the performance and currently active system processes. Ending it returned to the bash shell prompt for Docker. Docker exec was a significantly more convenient and lightweight way to remote into containers when compared to SSH. Figure 10 shows the Dockerfile that was created using the base image of the latest Ubuntu 20.04 image, and installs curl, wget, nano and net-tools. The working directory was set to /root and the container was run in the foreground using the /bin/bash command. The container was then started, and the *docker ps* command was given to get the container ID or the container name. Next, the *docker exec* command was given. The command given will open an interactive terminal session inside the container, allowing one to run commands and interact with the container as if one was directly logged in to the host.



```
aneena06@LAPTOP-PFDF8TOB: /mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles/exec$ cat Dockerfile
FROM ubuntu:20.04
RUN apt-get update && apt-get install -y curl wget nano net-tools
WORKDIR /root
EXPOSE 8000
CMD ["/bin/bash"]
aneena06@LAPTOP-PFDF8TOB: /mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles/exec$
```

Figure 10: Dockerfile for an image to run the docker exec command.



```
aneena06@LAPTOP-PFDF8TOB: /mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles/exec$ vi Dockerfile
aneena06@LAPTOP-PFDF8TOB: /mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles/exec$ docker build -t aneenaexec .
[+] Building 3.1s (8/8) FINISHED
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 173B 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [internal] load metadata for docker.io/library/ubuntu:20.04 2.9s
=> [auth] library/ubuntu:pull token for registry-1.docker.io 0.0s
=> [1/3] FROM docker.io/library/ubuntu:20.04@sha256:4a45212e9518f35983a976eead0de5eccc555a2f047134e9dd2cfc589076 0.0s
=> CACHED [2/3] RUN apt-get update && apt-get install -y curl wget nano net-tools 0.0s
=> CACHED [3/3] WORKDIR /root 0.0s
=> exporting to image 0.0s
=> => exporting layers 0.0s
=> => writing image sha256:af7e2cbb8f26346304b2583ae861da6be4f41043dab99f441290fd62dd86d2f3 0.0s
=> => naming to docker.io/library/aneenaexec 0.0s
Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
```

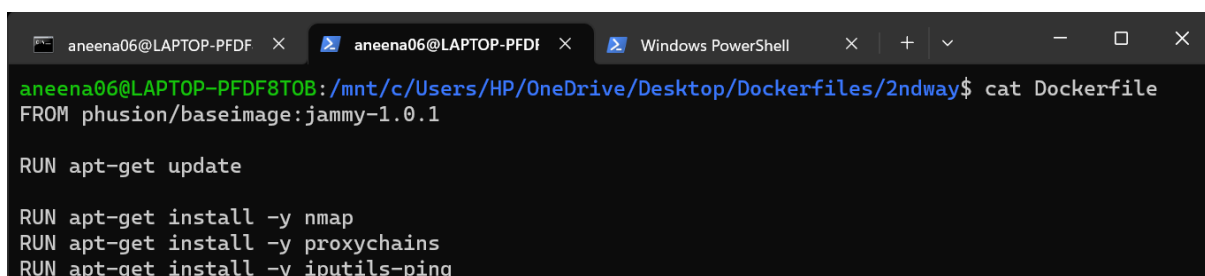
Figure 11: Building the Dockerfile.

SSH has been the standard method for logging into a Linux or Unix server remotely and is widely used in DevOps workflow. The SSH approach also functions well with Docker containers.

#### 5.1.4 Method 3: Pivoting through a server via SSH

SSH pivot method is used in network security to acquire access to networks that are not immediately accessible from the current location. In order to send one's traffic to the intended network, this technique uses an intermediary server, also referred to as a jump or pivot server. At first, the container was run and connected to the container using the `docker exec -it <container-name>` command. Once inside the container, SSH tunnelling was set up using the command `ssh -L <localport>:<target port> <username>@<hostname of intermediate server>`. Figure 13 shows the docker-compose YAML file that was created for this purpose, the first line identifies the version of the compose file in use, here version 2 was used because some of the IPAM configurations that are available in version 2 are not yet available in version 3. Then, the attacker and pivot container were created. The pivot container uses the keto/ssh image from the Docker hub which is a Dockerfile for ssh. Marks says, Docker supports custom networks in bridge mode, which can be used to create subnets with predetermined IP addresses to divide the data [27]. The appropriate gateway and subnet details were used to create customised subnetworks under the networks section. attack-net and target-net were the two subnetworks created. The target-net would be located at 172.16.201.0/24, whereas the attack-net would be at 172.16.200.0/24. Each host would be allocated to one or more networks once the networks were formed. Both networks have pivot as their member. Allocating the hostname would set the hostname of the container and using extra\_hosts would add those hosts to the /etc/host file. Two ports were open on target1, port 80 for HTTP and 3306 for mysql. Figure 12 shows the Dockerfile for the attacker container that was built locally instead of pulling an image from the Docker hub. This Dockerfile uses the Phusion/baseimage:jammy-1.0.1 as the base image and installs three additional applications nmap, proxychains and ping.

Using the command `docker-compose -f file2.yml up` the system which has four containers was started. Figure 14 shows the same.

A screenshot of a terminal window with three tabs: 'aneena06@LAPTOP-PFDF', 'aneena06@LAPTOP-PFDF', and 'Windows PowerShell'. The active tab is the first one. The terminal shows the command 'cat Dockerfile' being executed, displaying the following content:

```
aneena06@LAPTOP-PFDF8T0B:/mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles/2ndway$ cat Dockerfile
FROM phusion/baseimage:jammy-1.0.1

RUN apt-get update

RUN apt-get install -y nmap
RUN apt-get install -y proxychains
RUN apt-get install -y iputils-ping
```

Figure 12: Dockerfile for attacker container

```

aneena06@LAPTOP-PFDF8T0B: /mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles/2ndway$ cat file2.yml
version: '2.1'

services:
  attacker:
    build: .
    container_name: attacker
    ports:
      - "443:443"
      - "22:2222"
    hostname: attacker
    extra_hosts:
      - "pivot:172.16.200.11"
      - "target1:172.16.201.12"
      - "target2:172.16.201.13"

    networks:
      DMZ-net:
        ipv4_address: 172.16.200.10

  pivot:
    image: keto/ssh
    container_name: pivot
    hostname: pivot
    environment:
      SSH_PASSWORD: "mypassword"
    extra_hosts:
      - "attacker:172.16.200.10"
      - "target1:172.16.201.12"
      - "target2:172.16.201.13"
    networks:
      DMZ-net:
        ipv4_address: 172.16.200.11
      target-net:
        ipv4_address: 172.16.201.11

  target1:
    image: eystsen/vulnerablewordpress

    container_name: target1
    hostname: target1
    extra_hosts:
      - "pivot:172.16.201.11"
      - "target2:172.16.201.13"
    ports:
      - "80:80"
      - "3306:3306"
    networks:
      target-net:
        ipv4_address: 172.16.201.12

  target2:
    build: .
    container_name: target2
    hostname: target2
    extra_hosts:
      - "pivot:172.16.201.11"
      - "target1:172.16.201.12"
    networks:
      target-net:
        ipv4_address: 172.16.201.13

networks:
  DMZ-net:
    driver: bridge
    driver_opts:
      com.docker.network.enable_ipv6: "false"
    ipam:
      driver: default
      config:
        - subnet: 172.16.200.0/24
          gateway: 172.16.200.3
  target-net:
    driver: bridge
    driver_opts:
      com.docker.network.enable_ipv6: "false"
    ipam:
      driver: default
      config:
        - subnet: 172.16.201.0/24
          gateway: 172.16.201.3

```

Figure 13: Docker-compose file for pivoting through a server.

```
aneena06@LAPTOP-PFDF8TO: /mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles/2ndway$ vi file2.yml
aneena06@LAPTOP-PFDF8TO: /mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles/2ndway$ docker-compose -f file2.yml up
[+] Building 0.3s (11/11) FINISHED
=> [2ndway-attacker internal] load build definition from Dockerfile                                0.1s
=> transferring dockerfile: 303B                                                                0.0s
=> [2ndway-attacker internal] load .dockerignore                                                 0.0s
=> transferring context: 2B                                                                      0.0s
=> [2ndway-target2 internal] load build definition from Dockerfile                                0.0s
=> transferring dockerfile: 303B                                                                0.0s
=> [2ndway-target2 internal] load metadata for docker.io/phusion/baseimage:jammy-1.0.1            0.0s
=> [2ndway-target2 1/5] FROM docker.io/phusion/baseimage:jammy-1.0.1                          0.0s
=> CACHED [2ndway-target2 2/5] RUN apt-get update                                                0.0s
=> CACHED [2ndway-target2 3/5] RUN apt-get install -y nmap                                       0.0s
=> CACHED [2ndway-target2 4/5] RUN apt-get install -y proxychains                               0.0s
=> CACHED [2ndway-target2 5/5] RUN apt-get install -y iputils-ping                             0.0s
=> [2ndway-target2] exporting to image                                                            0.0s
=> exporting layers                                                                               0.0s
=> writing image sha256:825aaa1985545e8068298e2b76e80f710be90e2bbcb43733466bcd3814c50c08      0.0s
=> naming to docker.io/library/2ndway-attacker                                                  0.0s
=> writing image sha256:9ec3935faff49c675882f68a5be8de2a7fbc546fb2827ee4e64c698911371e7a      0.0s
=> naming to docker.io/library/2ndway-target2                                                  0.0s
=> [2ndway-target2 internal] load .dockerignore                                                 0.0s
=> transferring context: 2B                                                                      0.0s
[+] Running 6/6
# Network 2ndway_DMZ-net Created                                                                0.1s
# Network 2ndway_target-net Created                                                            0.0s
# Container pivot Created                                                                      0.2s
# Container target2 Created                                                                    0.1s
# Container attacker Created                                                                    0.1s
# Container target1 Created                                                                    1.0s
Attaching to attacker, pivot, target1, target2
target2 | *** Running /etc/my_init.d/00_regen_ssh_host_keys.sh...
```

Figure 14: Docker-compose for file2.yml.

### 5.1.5 Method 4: Pivoting using Metasploit.

Using Metasploit to pivot in a container is a method of routing network traffic through a middle container from a compromised system to a different system or network, also known as port forwarding or tunnelling. With this technique, an attacker can acquire access to networks that are not directly reachable from the attacker's machine by routing their traffic through the compromised system. This was achieved by getting inside the container using the `docker exec -it <container_name>` command. Once a shell access to the container is open, the metasploit's port forwarding or routing functionality is used to pivot through the target container to reach other targets on the network. Figure 15 shows the docker-compose YAML file that was created for this method, it defines a multi-container application composed of three services, attacker, vulnerable\_app and db. The attacker service was build using the Dockerfile.attacker (shown in figure 16), and the container is named attacker, an IP address of 172.16.238.10 is assigned on the attacker\_net network, which is the bridge network with the subnet of 172.16.238.0/24. The attacker net network, which has the subnet 172.16.238.0/24, is established as a bridge network to link the containers. It depends on the vulnerable\_app service and has an environment variable RHOSTS set to vulnerable\_app. The eystem/vulnerablewordpress:latest image from Docker hub was used to generate the vulnerable\_app service, and the container was named vulnerable\_app. Its IP address on the attacker\_net network is 172.16.238.20, and it exposes ports 80 and 443 to the host. The environment variable WORDPRESS\_DB\_PASSWORD, which is dependent on the database service, is set to mypassword. The mysql:5.7 image from Docker hub is used to construct the database service, which has the container name db. On the attacker\_net network, the IP address 172.16.238.30 was given, and the environment variables MYSQL\_ROOT\_PASSWORD and MYSQL\_DATABASE

are set to mypassword and wordpress, respectively. Figure 2 shows the Dockerfile that was used for the attacker service, this Docker image was build based on the Kali Linux operating system and has nmap and Metasploit framework packages pre-installed.

Using the command *docker-compose up* the system which has three containers was started. Figure 17 shows the same.

```
aneena06@LAPTOP-PFDF8T0B:/mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles/aneena$ cat docker-compose.yml
version: '3'

services:
  attacker:
    build:
      context: .
      dockerfile: Dockerfile.attacker
    image: kali
    container_name: attacker
    tty: true
    stdin_open: true
    networks:
      attacker_net:
        ipv4_address: 172.16.238.10
    depends_on:
      - vulnerable_app
    environment:
      RHOSTS: vulnerable_app
    command: /bin/bash

  vulnerable_app:
    image: eystsen/vulnerablewordpress
    container_name: vulnerable_app
    ports:
      - 80:80
      - 443:443
    networks:
      attacker_net:
        ipv4_address: 172.16.238.20
    environment:
      WORDPRESS_DB_PASSWORD: mypassword
    depends_on:
      - db

  db:
    image: mysql:5.7
    container_name: db
    environment:
      MYSQL_ROOT_PASSWORD: mypassword
      MYSQL_DATABASE: wordpress
    networks:
      attacker_net:
        ipv4_address: 172.16.238.30

networks:
  attacker_net:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 172.16.238.0/24

aneena06@LAPTOP-PFDF8T0B:/mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles/aneena$
```

Figure 15: Docker-compose file for pivoting using metasploit

```

aneena06@LAPTOP-PFDF8T0B:/mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles/aneena$ ls
Dockerfile Dockerfile.attacker aneena.yml docker-compose.yml
aneena06@LAPTOP-PFDF8T0B:/mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles/aneena$ cat Dockerfile.attacker
FROM kalilinux/kali-rolling

RUN apt-get update && \
    apt-get install -y nmap \
    metasploit-framework

aneena06@LAPTOP-PFDF8T0B:/mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles/aneena$

```

Figure 16: Dockerfile used for attacker service.

```

root@0c4770f6504f: / x root@0c4770f6504f: / x + v
aneena06@LAPTOP-PFDF8T0B:/mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles/aneena$ docker-compose build
[+] Building 200.9s (7/7) FINISHED
=> [internal] load build definition from Dockerfile.attacker 0.0s
=> => transferring dockerfile: 155B 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [internal] load metadata for docker.io/kalilinux/kali-rolling:latest 2.9s
[auth] kalilinux/kali-rolling:pull token for registry-1.docker.io 0.0s
=> CACHED [1/2] FROM docker.io/kalilinux/kali-rolling@sha256:03c3b10b0064082e38eaf03c2022eee87d673c8ae4a26efe2db9052600facf18 0.0s
=> [2/2] RUN apt-get update && apt-get install -y nmap metasploit-framework 174.8s
=> exporting to image 23.1s
=> => exporting layers 23.1s
=> => writing image sha256:5efbefff998c924491aab4b4b9854f1e8d79173bcd59c7d8f468f8d0bd397163 0.0s
=> => naming to docker.io/library/kali 0.0s
aneena06@LAPTOP-PFDF8T0B:/mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles/aneena$ docker-compose up
[+] Running 4/4
# Network aneena_attacker_net Created 0.0s
# Container db Created 0.2s
# Container vulnerable_app Created 0.4s
# Container attacker Created 0.1s
Attaching to attacker, db, vulnerable_app
db | 2023-02-18 20:37:36+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 5.7.41-1.el7 started.
db | 2023-02-18 20:37:36+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
db | 2023-02-18 20:37:36+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 5.7.41-1.el7 started.
db | 2023-02-18 20:37:36+00:00 [Note] [Entrypoint]: Initializing database files
db | 2023-02-18T20:37:36.651591Z 0 [Warning] TIMESTAMP with implicit DEFAULT value is deprecated. Please use --explicit_
defaults_for_timestamp server option (see documentation for more details).
db | 2023-02-18T20:37:36.965976Z 0 [Warning] InnoDB: New log files created, LSN=45790
db | 2023-02-18T20:37:37.049788Z 0 [Warning] InnoDB: Creating foreign key constraint system tables.
db | 2023-02-18T20:37:37.063068Z 0 [Warning] No existing UUID has been found, so we assume that this is the first time t
hat this server has been started. Generating a new UUID: 14825d81-afcc-11ed-b58c-0242ac10ee1e.
db | 2023-02-18T20:37:37.068524Z 0 [Warning] Gtid table is not ready to be used. Table 'mysql.gtid_executed' cannot be o

```

Figure 17: Docker-compose for the file.

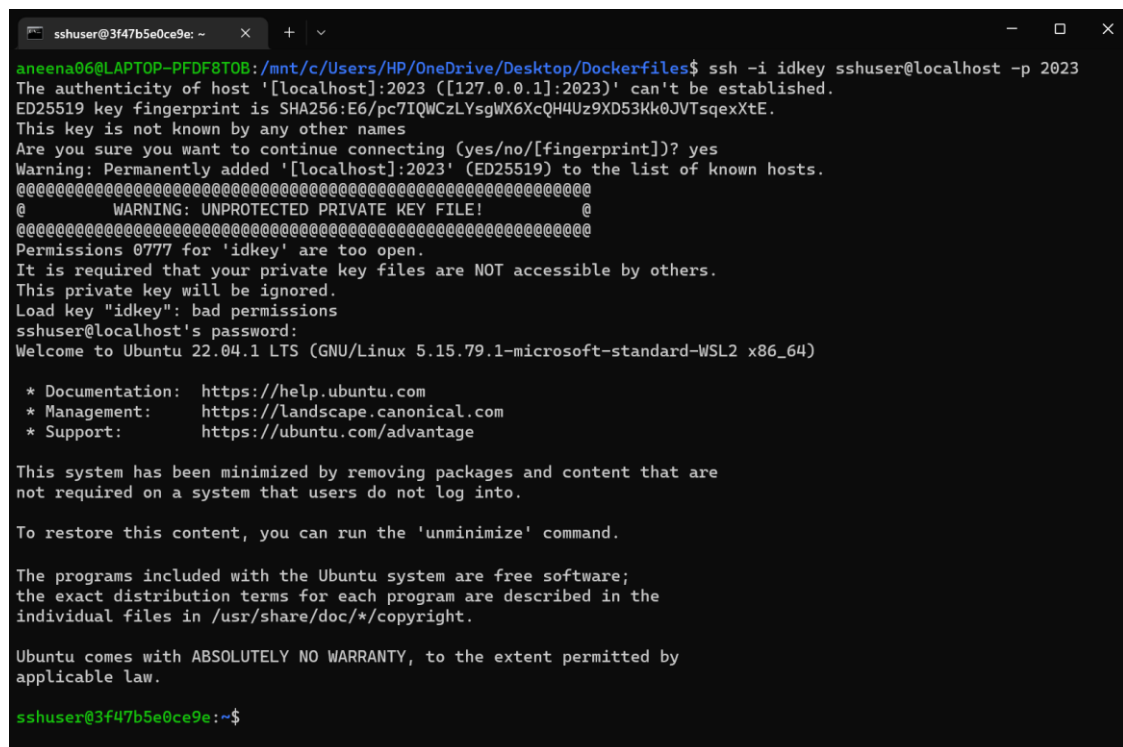
## 6. RESULT AND DISCUSSION

### 6.1 Method 1: OpenSSH-based Docker shell.

When the Dockerfile was run with built-in OpenSSH, a connection to the container could be made with the SSH client, specifying the private key and the IP address or the hostname of the container. Figure 18 shows that the command was used to connect to docker's default IP address that is, the localhost, it could also be run by explicitly specifying the container's IP address. Once the connection was made to the container it could be used for interacting with the container just like one would with any remote machine running an SSH service.

The result of an OpenSSH-based shell depends on the configuration of the container, as well as the commands executed within the shell. Here, a command prompt was presented where commands can be executed and even interact with the container's file system and network.

Since the container is running a Linux distribution, a wide range of standard command-line tools can be accessed, like ls for listing files, cat for reading file contents, and grep for searching text. Additional packages and tools can also be installed. Here, as shown in figure 19, as the networking tools were installed, *netstat -tuln* was given to see the network status and protocol statistics, -t shows all tcp connections, -u shows all udp connections, -l displays only listening ports, -n displays network addresses as IP addresses instead of hostnames.



```
aneena06@LAPTOP-PFDF8TOB: /mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles$ ssh -i idkey sshuser@localhost -p 2023
The authenticity of host '[localhost]:2023 ([127.0.0.1]:2023)' can't be established.
ED25519 key fingerprint is SHA256:E6/pc7IQWCzLYsgWX6XcQH4Uz9XD53Kk0JVTsqexXtE.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[localhost]:2023' (ED25519) to the list of known hosts.
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@                WARNING: UNPROTECTED PRIVATE KEY FILE!                @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Permissions 0777 for 'idkey' are too open.
It is required that your private key files are NOT accessible by others.
This private key will be ignored.
Load key "idkey": bad permissions
sshuser@localhost's password:
Welcome to Ubuntu 22.04.1 LTS (GNU/Linux 5.15.79.1-microsoft-standard-WSL2 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

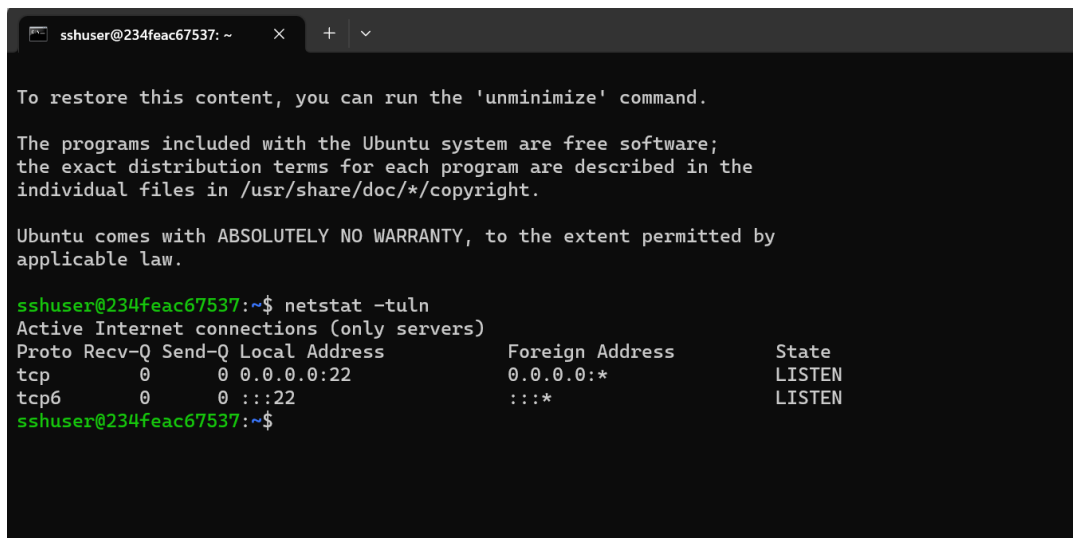
To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

sshuser@3f47b5e0ce9e:~$
```

Figure 18: SSHing into the Docker container

A terminal window with a dark background and light green text. The window title is 'sshuser@234feac67537: ~'. The output shows Ubuntu notices and the result of the 'netstat -tuln' command. The command output is a table of active internet connections.

```
sshuser@234feac67537: ~  
To restore this content, you can run the 'unminimize' command.  
  
The programs included with the Ubuntu system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by  
applicable law.  
  
sshuser@234feac67537:~$ netstat -tuln  
Active Internet connections (only servers)  
Proto Recv-Q Send-Q Local Address           Foreign Address         State  
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN  
tcp6       0      0 :::22                  :::*                     LISTEN  
sshuser@234feac67537:~$
```

Figure 19: Output of the netstat -tuln command

## 6.2 Method 2: Using the docker exec command in a shell.

When a container is run using the docker exec, an interactive shell within the container is open. This allows one to execute commands and perform various tests to check the security and functionality of the container. Various options to control the behaviour of the command can be provided, such as -i and -t options are used to provide an interactive shell within the container. Once access to the container is made, commands can be executed as if one were logged into the container. This can be useful for testing the security of the container, running diagnostic tests, or troubleshooting issues that may arise. Figure 20 shows the docker exec command that was run using an interactive shell, with the container id.

The output of the commands run using the docker exec will be displayed on the terminal. This output could be used to verify the container's functioning ability, that is if it is running as expected and even to identify any potential security vulnerabilities or other issues that could arise. Figure 21 shows the output of the top command that provides a real-time dynamic view of the processes that are running on the system, the system resources utilisation like CPU, memory and I/O, this command is used to identify resource-hungry processes, troubleshoot system performance problems, and even to monitor the overall health of the system.

```

root@ea00d46ba8cf: ~
exit
aneena06@LAPTOP-PFDF8T0B:/mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles/exec$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS          NAMES
ea00d46ba8cf  aneenaexec  "/bin/bash"             4 minutes ago  Up 2 minutes  unruﬄled_williamson
aneena06@LAPTOP-PFDF8T0B:/mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles/exec$ docker exec -it ea00d46ba8cf bash
root@ea00d46ba8cf:~# exit
exit
aneena06@LAPTOP-PFDF8T0B:/mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles/exec$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
aneenaexec    latest   739c583463df   11 minutes ago  131MB
sshaneena     latest   37c5cce20598   2 hours ago    231MB
dockerfiles-vote latest   9d10158dc9cc   6 days ago     231MB
dockerfiles-result latest   7e5fda1f3d24   6 days ago     231MB
dockerfiles-worker latest   32629c630ac3   6 days ago     231MB
mysql         latest   cdf3aa69f5f0   9 days ago     517MB
mysql         latest   6047d5cd0bc4   2 weeks ago    231MB
redis         alpine   53278c28ec70   2 weeks ago    29.9MB
aneena        latest   b56c13ff44da   2 weeks ago    231MB
joomla        latest   5f6ec0b23869   3 weeks ago    633MB
alpine        latest   042a816809aa   4 weeks ago    7.05MB
mysql         5.6     dd3b2a5dcb48   13 months ago  303MB
hello-world   latest   feb5d9fea6a5   16 months ago  13.3kB
postgres      9.4     ed5a45034282   2 years ago    251MB
aneena06@LAPTOP-PFDF8T0B:/mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles/exec$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS          NAMES
ea00d46ba8cf  aneenaexec  "/bin/bash"             5 minutes ago  Up 3 minutes  unruﬄled_williamson
aneena06@LAPTOP-PFDF8T0B:/mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles/exec$ docker exec -it ea00d46ba8cf bash
root@ea00d46ba8cf:~#

```

Figure 20: Docker image run and the docker exec command executed.

```

root@a94a2507f008: ~
aneena06@LAPTOP-PFDF8T0B:/mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles/exec$ docker run -it -p 4500:8000 aneenaexec
root@a94a2507f008:~# top
top - 21:16:48 up 5:45, 0 users, load average: 0.08, 0.07, 0.08
Tasks: 2 total, 1 running, 1 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 0.1 sy, 0.0 ni, 99.8 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 7828.0 total, 198.2 free, 2299.2 used, 5330.6 buff/cache
MiB Swap: 2048.0 total, 2009.6 free, 38.4 used, 5231.6 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR S  %CPU  %MEM    TIME+  COMMAND
    1 root        20   0   4112   3372  2928 S   0.0   0.0   0:00.03 bash
    9 root        20   0   5968   3224  2768 R   0.0   0.0   0:00.00 top

root@a94a2507f008:~# netstat -tuln
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
root@a94a2507f008:~#

```

Figure 21: top command output

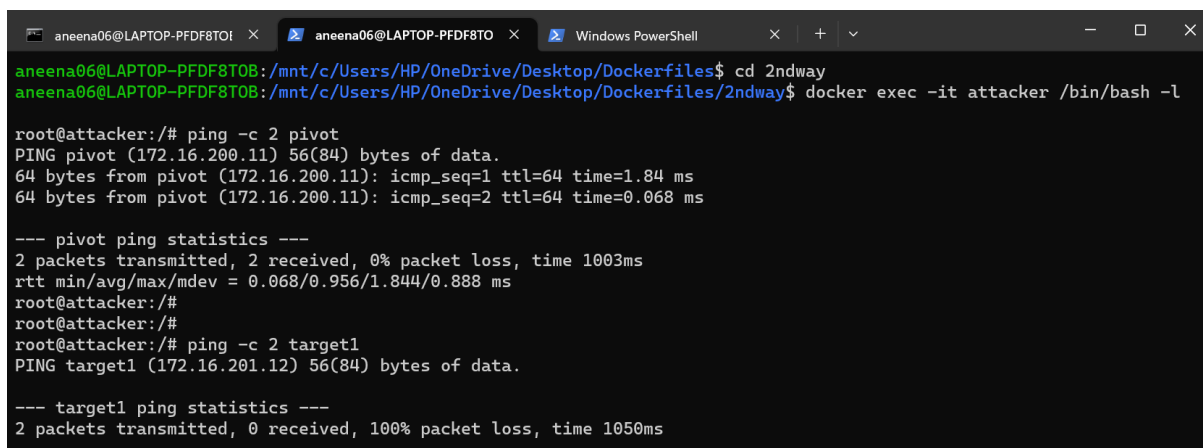
### 6.3 Method 3: Pivoting through a server using SSH.

Pivoting through a server through SSH in a container is a technique used to gain access to additional network resources from the container. In order to pivot, an SSH connection is established from the container to the server, and then that connection is used to access additional resources on the server's network. This is useful when the container is isolated from the network, but there is a server on the same network that can be used as a jumping-off point to access additional resources.

As shown in figure 22, the docker exec command was used to connect to the attacker container. After the connection is established, the connection is checked using the ping command, ping to pivot and target1 container is checked. The pivot container responded but the target1 container did not.

On the pivot container, a SOCKS proxy was built via SSH. A SOCKS proxy, a general-purpose proxy server is a network protocol that enables a client computer to access any remote server through a proxy server. All the traffic that is present between the client and the server is routed through the SOCKS proxy, which connects to another server. In the attacker machine, the `/etc/proxychains.conf` file showed the port is to be used for ProxyChains, the final line showed a default entry that prefers port 9050 to 127.0.0.1. The command used to create SSH Socks was given as shown in figure 23, `ssh -D local-address:port -f -N username@gatewayaddress`. Then, as shown in figure nmap scanning was done, but the target1 container was not visible from the attacker container. The proxychains was added to the command so that the nmap instructions were executed through the proxy that runs on the pivot container, the command given was `proxychains nmap --unprivileged 172.16.201.12`.

ProxyChains will successfully scan the target1 by proxying all network traffic through the SSH tunnel on the pivot container. Target1 was operating a WordPress website that is insecure, and the HTTP and MySQL Server ports are open, the nmap scan showed them as open ports as shown in figure 24.



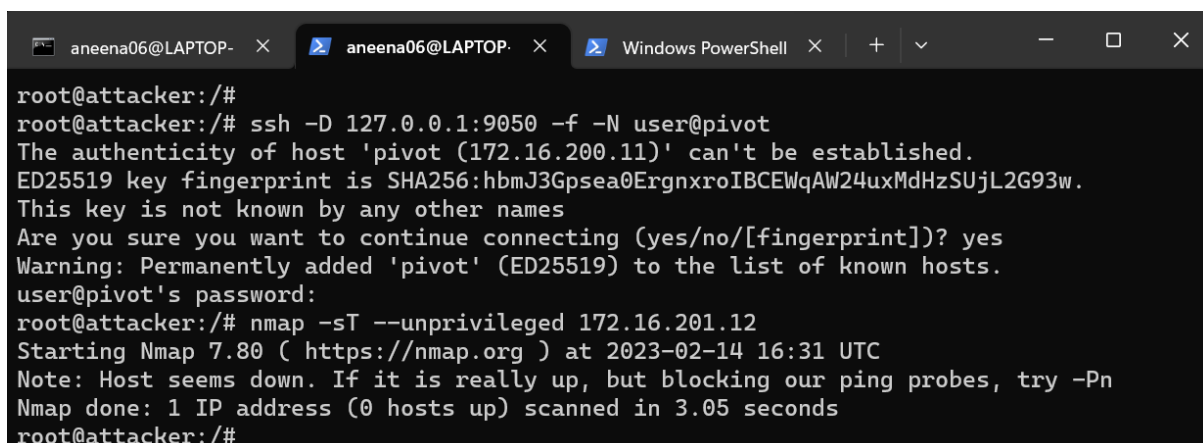
```
aneena06@LAPTOP-PFDF8TOI x aneena06@LAPTOP-PFDF8TO x Windows PowerShell x + v - □ x
aneena06@LAPTOP-PFDF8TOB:/mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles$ cd 2ndway
aneena06@LAPTOP-PFDF8TOB:/mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles/2ndway$ docker exec -it attacker /bin/bash -l

root@attacker:/# ping -c 2 pivot
PING pivot (172.16.200.11) 56(84) bytes of data.
64 bytes from pivot (172.16.200.11): icmp_seq=1 ttl=64 time=1.84 ms
64 bytes from pivot (172.16.200.11): icmp_seq=2 ttl=64 time=0.068 ms

--- pivot ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1003ms
rtt min/avg/max/mdev = 0.068/0.956/1.844/0.888 ms
root@attacker:/#
root@attacker:/# ping -c 2 target1
PING target1 (172.16.201.12) 56(84) bytes of data.

--- target1 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1050ms
```

Figure 22: Access gained using docker-exec.



```
aneena06@LAPTOP- x aneena06@LAPTOP- x Windows PowerShell x + v - □ x
root@attacker:/#
root@attacker:/# ssh -D 127.0.0.1:9050 -f -N user@pivot
The authenticity of host 'pivot (172.16.200.11)' can't be established.
ED25519 key fingerprint is SHA256:hbmJ3Gpsea0ErgnxroIBCEWqAW24uxMdHzSUjL2G93w.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'pivot' (ED25519) to the list of known hosts.
user@pivot's password:
root@attacker:/# nmap -sT --unprivileged 172.16.201.12
Starting Nmap 7.80 ( https://nmap.org ) at 2023-02-14 16:31 UTC
Note: Host seems down. If it is really up, but blocking our ping probes, try -Pn
Nmap done: 1 IP address (0 hosts up) scanned in 3.05 seconds
root@attacker:/#
```

Figure 23: The command used to create SSH socks.

```
aneena06@LAPTOP- x aneena06@LAPTOP- x Windows PowerShell x + v - □ x
|S-chain|-<>-127.0.0.1:9050-<>-172.16.201.12:2068-<--timeout
|S-chain|-<>-127.0.0.1:9050-<>-172.16.201.12:70-<--timeout
|S-chain|-<>-127.0.0.1:9050-<>-172.16.201.12:5960-<--timeout
|S-chain|-<>-127.0.0.1:9050-<>-172.16.201.12:32779-<--timeout
|S-chain|-<>-127.0.0.1:9050-<>-172.16.201.12:9220-<--timeout
|S-chain|-<>-127.0.0.1:9050-<>-172.16.201.12:11967-<--timeout
|S-chain|-<>-127.0.0.1:9050-<>-172.16.201.12:5222-<--timeout
|S-chain|-<>-127.0.0.1:9050-<>-172.16.201.12:3301-<--timeout
|S-chain|-<>-127.0.0.1:9050-<>-172.16.201.12:49165-<--timeout
|S-chain|-<>-127.0.0.1:9050-<>-172.16.201.12:1063-<--timeout
|S-chain|-<>-127.0.0.1:9050-<>-172.16.201.12:3580-<--timeout
|S-chain|-<>-127.0.0.1:9050-<>-172.16.201.12:16993-<--timeout
|S-chain|-<>-127.0.0.1:9050-<>-172.16.201.12:9091-<--timeout
|S-chain|-<>-127.0.0.1:9050-<>-172.16.201.12:52822-<--timeout
|S-chain|-<>-127.0.0.1:9050-<>-172.16.201.12:3007-<--timeout
|S-chain|-<>-127.0.0.1:9050-<>-172.16.201.12:17-<--timeout
|S-chain|-<>-127.0.0.1:9050-<>-172.16.201.12:57797-<--timeout
|S-chain|-<>-127.0.0.1:9050-<>-172.16.201.12:5269-<--timeout
|S-chain|-<>-127.0.0.1:9050-<>-172.16.201.12:1126-<--timeout
|S-chain|-<>-127.0.0.1:9050-<>-172.16.201.12:4900-<--timeout
|S-chain|-<>-127.0.0.1:9050-<>-172.16.201.12:515-<--timeout
|S-chain|-<>-127.0.0.1:9050-<>-172.16.201.12:1028-<--timeout
|S-chain|-<>-127.0.0.1:9050-<>-172.16.201.12:4004-<--timeout
Nmap scan report for target1 (172.16.201.12)
Host is up (0.00062s latency).
Not shown: 998 closed ports
PORT      STATE SERVICE
80/tcp    open  http
3306/tcp  open  mysql

Nmap done: 1 IP address (1 host up) scanned in 0.66 seconds
```

Figure 24: ProxyChains nmap scan result.

## 6.4 Method 4: Pivoting using Metasploit.

When one pivot's using Metasploit, a connection is established for a new path for the traffic to reach its intended target. For this first one system is compromised that is located in a different network from the target, and then that system is used as pivots to reach the target.

Here, as shown in Figure 25, connection to the attacker container is gained using the docker exec command, this gave an interactive session. Then the Metasploit framework is open by giving the msfconsole command, at first a running vulnerable service is identified on the target container, this was identified using the *nmap -sV <target-continaer-ip>*. Then Metasploit is used to exploit the vulnerability and gain a foothold on the target container. Figure 26 shows the nmap command result which shows that there were two open ports, port 80 for https and port 3306 for mysql. One such vulnerability that was exploited was the Metasploit auxiliary module (as shown in figure 27) that is used for scanning for open ports on a target host. The module sends SYN packets to a range of ports and determines which ports are open, closed or filtered. This module can be used to check that a specific service is listening to a specific port as well as for identifying potential attack vectors on the target

network. This is sometimes used as a warm-up for more specialised attacks specially to figure out which port are accessible on a specific system so that attacks can be targeted to those ports.

```
root@0c4770f6504f: /  X root@0c4770f6504f: /  + v
aneena06@LAPTOP-PFDF8T0B:/mnt/c/Users/HP/OneDrive/Desktop/Dockerfiles/aneena$ docker exec -it attacker /bin/bash
[ (root@0c4770f6504f)-[/]
# msfconsole

Unable to handle kernel NULL pointer dereference at virtual address 0xd34db33f
EFLAGS: 00010046
eax: 00000001 ebx: f77c8c00 ecx: 00000000 edx: f77f0001
esi: 803bf014 edi: 8023c755 ebp: 80237f84 esp: 80237f60
ds: 0018  es: 0018  ss: 0018
Process Swapper (Pid: 0, process nr: 0, stackpage=80377000)

Stack: 90909090909090909090909090909090
       90909090909090909090909090909090
       90909090.90909090.90909090
       90909090.90909090.90909090
       90909090.90909090.09090900
       90909090.90909090.09090900
       .....
       cccccccccccccccccccccccccccc
       cccccccccccccccccccccccccccc
       cccccccc.....
       cccccccccccccccccccccccccccc
       cccccccccccccccccccccccccccc
       .....cccccccccc
       cccccccccccccccccccccccccccc
       cccccccccccccccccccccccccccc
       .....
       ffffffffffffffffffffffffffff
       ffffffff.....
       ffffffffffffffffffffffffffff
```

Figure 25: Access gained to attacker container using docker-exec command.

```
root@0c4770f6504f: /  X root@0c4770f6504f: /  + v
=[ metasploit v6.3.2-dev ]
+ -- ==[ 2290 exploits - 1201 auxiliary - 409 post ]
+ -- ==[ 968 payloads - 45 encoders - 11 nops ]
+ -- ==[ 9 evasion ]

Metasploit tip: View all productivity tips with the
tips command
Metasploit Documentation: https://docs.metasploit.com/

msf6 > nmap -sV 172.16.238.20
[*] exec: nmap -sV 172.16.238.20

Starting Nmap 7.93 ( https://nmap.org ) at 2023-02-18 20:40 UTC
Nmap scan report for vulnerable_app.aneena_attacker_net (172.16.238.20)
Host is up (0.0000070s latency).
Not shown: 998 closed tcp ports (reset)
PORT      STATE SERVICE VERSION
80/tcp    open  http   Apache httpd 2.4.7 ((Ubuntu))
3306/tcp  open  mysql  MySQL 5.5.62-0ubuntu0.14.04.1
MAC Address: 02:42:AC:10:EE:14 (Unknown)

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 6.84 seconds
msf6 > search
Usage: search [<options>] [<keywords>:<value>]
```

Figure 26: nmap scan in the target container.

```

msf6 > use auxiliary/scanner/portscan/tcp
msf6 auxiliary(scanner/portscan/tcp) > set RHOSTS 172.16.238.20
RHOSTS => 172.16.238.20
msf6 auxiliary(scanner/portscan/tcp) > set ports 80, 443
ports => 80, 443
msf6 auxiliary(scanner/portscan/tcp) > show options

Module options (auxiliary/scanner/portscan/tcp):



| Name        | Current Setting | Required | Description                                                                                                                                                                                         |
|-------------|-----------------|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CONCURRENCY | 10              | yes      | The number of concurrent ports to check per host                                                                                                                                                    |
| DELAY       | 0               | yes      | The delay between connections, per thread, in milliseconds                                                                                                                                          |
| JITTER      | 0               | yes      | The delay jitter factor (maximum value by which to +/- DELAY) in milliseconds.                                                                                                                      |
| PORTS       | 80, 443         | yes      | Ports to scan (e.g. 22-25,80,110-900)                                                                                                                                                               |
| RHOSTS      | 172.16.238.20   | yes      | The target host(s), see <a href="https://docs.metasploit.com/docs/using-metasploit/basics/using-metasploit.html">https://docs.metasploit.com/docs/using-metasploit/basics/using-metasploit.html</a> |
| THREADS     | 1               | yes      | The number of concurrent threads (max one per host)                                                                                                                                                 |
| TIMEOUT     | 1000            | yes      | The socket connect timeout in milliseconds                                                                                                                                                          |



View the full module info with the info, or info -d command.

msf6 auxiliary(scanner/portscan/tcp) > exploit

[*] 172.16.238.20: - 172.16.238.20:80 - TCP OPEN
[*] 172.16.238.20: - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/portscan/tcp) > set ports 80, 3306
ports => 80, 3306
msf6 auxiliary(scanner/portscan/tcp) > exploit

[*] 172.16.238.20: - 172.16.238.20:80 - TCP OPEN
[*] 172.16.238.20: - 172.16.238.20:3306 - TCP OPEN
[*] 172.16.238.20: - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/portscan/tcp) >

```

Figure 27: Exploit of Metasploit auxiliary module

The results of performing these activities provided insights into the security posture of Docker containers and the overall network. By successfully SSHing into a Docker container and using docker-exec to get access into a container indicated that the container was not secured properly and there were known vulnerabilities that could be exploited. By being able to pivot using SSH from a compromised Docker container to other systems indicated that there was a lack of proper network segmentation and that access to sensitive systems and data were not limited. By being able to pivot from a compromised system to other systems using Metasploit showed that there was a lack of proper vulnerability management and testing.

By running these, the security of Docker containers could be tested and could even be used for identifying any vulnerabilities, this could help to proactively identify and fix security issues before being exploited by malicious attackers. These tests could even help to better understand the risks associated with running containerized applications, this could eventually help to implement appropriate security measures to mitigate those risks. These tests could also help to stay up-to-date with the latest security trends and practices, and to ensure that the Docker containers are as secure as possible. Overall, the importance of performing activities like these is to proactively identify and address the security risks in Docker environment, in order to prevent security breaches and protect the critical assets.

## 7. CONCLUSION

By constructing several containers, each of which represents a distinct component of the networked system, Docker may be used to simulate complicated networked systems. A Docker network can be used to connect these containers to one another and enable communication between them. In addition to networking, Docker offers a mechanism to emulate various architectures and operating systems, which is helpful for testing and development. Multiple container applications offer additional internal networks for attackers to engage with, after the application's exterior attack surface has been compromised. In this dissertation, four different ways to access a container were tested. From this it was studied that SSHing into a Docker container and using `docker-exec` to gain access into a Docker container were both legitimate ways to access a container and its environment. And pivoting using SSH and Metasploit were both techniques used to expand an attacker's access within a network. By using SSH pivoting, an attacker would be able to switch between different network segments using a remote SSH host as a proxy. Metasploit pivoting would establish a similar task by creating a pivot connection between two separate subnets.

By being able to access containers using these techniques, it was understood that attackers could use various techniques to gain access and move around within a network, and security professionals would need to ensure that they have robust defenses in place to mitigate the risks associated with these attacks. It also suggests that there were security vulnerabilities in the Docker environment that had to be addressed, like the container image being used could be outdated or there could be any vulnerable software that could be exploited by an attacker. It could also indicate that the container was not secured properly, such as having weak SSH credentials or not disabling root login.

In general, this type of testing could also help in finding security gaps in a Docker system and could even provide a chance to strengthen security. To avoid potential attacks or security and data breaches, it is crucial to deal with these problems. Despite the fact that Docker offers an isolated environment for running applications, it is crucial to configure the environment correctly to protect the host system and other containers from security threats. Unpatched vulnerabilities, improperly configured container networking, and insecure settings are a few potential security risks with Docker that could make it possible for an attacker to take advantage of the Docker environment and move on to other portions of the network. Best practices that could assist organizations in identifying and addressing potential security vulnerabilities in their systems and networks, include frequent security audits, vulnerability scanning, penetration testing, and even security awareness training.

## 8. FUTURE SCOPE

As container access to containers was possible using these exploits, there could be several things that could be achieved in the future, one of them being penetration testing. The Docker infrastructure could be explored in depth and several other attacks could be carried out to discover further vulnerabilities, those which could potentially lead to deeper access and control over the Docker environment. The Docker environment could also be used as a platform to launch further attacks on other systems, within or outside the Docker infrastructure. These could be used for testing activities like lateral movement, data exfiltration or even launching additional malware or exploits. The Docker infrastructure could have to tested as a target for penetration testing exercises or even for red teaming, in order to simulate realistic attacks and test the security of the environment. This could involve using similar strategies and methods used to obtain the initial access, but with the intention of finding and fixing environmental weaknesses. By detecting and reporting vulnerabilities, creating, and testing new security controls and features, and using the knowledge acquired from these attacks, Docker and other containerization technologies could be made more secure.

## 9. REFLECTION

By doing these activities, I gained a deeper understanding of the security risks and vulnerabilities associated with Docker containers. SSHing into a container involves remotely being able to access a running Docker container. An access is possible if a container is not secured properly, either due to weak passwords or other vulnerabilities. This showed the importance of securing the SSH server of a container. The `docker-exec` is a command that is used to run a command within a container. By using this command, I was able to gain access to a container and its file system, which could be useful for performing reconnaissance or even for launching attacks. This made me realize that it is important to limit the use of `docker-exec` and any similar commands to only trusted users, as that is the base and basic command in Docker that could be used for many attacks that could be done within a container. Moving laterally inside a network or a system to access other services or systems is referred to as pivoting. Here, I shifted from a compromised Docker container to other networked systems using SSH. This underlined that there is a need for appropriate network segmentation and access controls to reduce the harm that could result from a hacked container. One of the popular tools for penetration testing and vulnerability analysis includes Metasploit. I understood that I could be able to switch from a compromised Docker container to other systems on the network by using Metasploit framework. This demonstrated the significance of routinely testing networks and Docker containers for flaws and vulnerabilities in order to spot them and fix them before attackers can take use of them.

I took up from this that it's crucial to secure Docker containers correctly and put best practices into practice, such as using strong passwords, turning off unused services and ports, and keeping software up to current with the most recent security patches. Implementing suitable access controls, such as limiting SSH access to just trusted users or hosts, is also crucial. I also realized how important it is to consistently check and audit Docker containers for security problems, such as vulnerabilities and shady behavior. Implementing intrusion detection and prevention systems, reviewing logs and analytics, and utilizing security scanning tools could do this. I learned about the various security threats connected to Docker containers through doing this, and the significance of taking preventative actions to safeguard them. Organizations can lower the risk of security breaches and safeguard their crucial assets by adhering to best practices and routinely checking for vulnerabilities.

## REFERENCES

- [1] R. Yasrab, "Mitigating Docker Security Issues," 2021.
- [2] L. R.-M. J. C. M. L. Luis M Vaquero, "A break in the clouds: towards a cloud definition," *ACM SIGCOMM*, vol. 39, no. 1, pp. 50-55, 2009.
- [3] D. V. S. H. N. & M. Morris, "Use of Docker for deployment and testing of astronomy software," *Astronomy and Computing*, vol. 20, pp. 105-119, 2017.
- [4] S. M. T. N. J. C. Kelly Brady, "Docker Container Security in Cloud Computing," *IEEE*, 2020.
- [5] S. Winkel, "Security Assurance of Docker Containers: Part 1," *ISSA*, pp. 35-42, 2017.
- [6] H. C. S. W. C. H. Delu Huang, "Security Analysis and Threats Detection Techniques on Docker Container," in *IEEE 5th International Conference on Computer and Communications (ICCC)*, 2019.
- [7] J. Biggs, "TechCrunch," 15 June 2018. [Online]. Available: <https://techcrunch.com/2018/06/15/tainted-crypto-mining-containers-pulled-from-docker-hub/>. [Accessed 2 January 2023].
- [8] J. Turnbull, *The Docker Book*, 2014.
- [9] R. Yasrab, "Mitigating Docker Security Security Issues," 12 August 2021.
- [10] T. Bui, "Analysis of Docker Security," Finland, 2015.
- [11] E. K. R. D. T. R. M. P. Samuel P. Mullinix, "On Security Measures for Containerized Applications Imaged With Docker," vol. 11, p. August, 2020.
- [12] T. M. C. P. Z. Cliffe Schreuders, "Empowering End Users to Confine Their Own Applications: The Results of a Usability Study Comparing SELinux, AppArmor, and FBAC-LSM," *ACM Transactions on Information and System Security*, vol. 14, no. 2, p. September, 2011.
- [13] A. R. R. D. K. Rajdeep Dua, "Virtualization vs Containerization to Support PaaS," *IC2E '14: Proceedings of the 2014 IEEE International Conference on Cloud Engineering*, pp. 610-614, 2014.

- [14] H. C. S. W. C. H. Delu Huang, "Security Analysis and Threats Detection Techniques on Docker Container," *IEEE 5th International Conference on Computer and Communications*, pp. 1214-1220, 2019.
- [15] F. J, " Docker: A Software as a Service, Operating System-Level Virtualization Framework," *Code4Lib Journal*, 2014.
- [16] A. G. M. Serge E. Hallyn, "Linux Capabilities: making them work," in *Proceedings of OLS, Volume One*, 2008.
- [17] G. Ouyang, "Build Local Private Repository of Docker," vol. 24, no. 10, pp. 156-157, 2018.
- [18] J. Z, "Design and implementation of Docker container security monitoring system," 2017.
- [19] M. S. C. J. Brady K, "Docker Container Security in Cloud Computing," in *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, 2020.
- [20] J. Vrancken, "A Methodology for Penetration Testing Docker Systems," 2020.
- [21] C. Tozzi, "slim.ai," 12 January 2022. [Online]. Available: <https://www.slim.ai/blog/5-common-container-exploits/>. [Accessed 26 January 2023].
- [22] W. McGrew, "An Attacker Looks at Docker: Approaching Multi-Container Applications".
- [23] S. McCullough, "Using Docker to Create Multi-Container Environments for Research and Sharing Lateral Movement Scenarios," 2017.
- [24] L. Zeltser, "Zeltser," 12 May 2016. [Online]. Available: <https://zeltser.com/metasploit-framework-docker-container/#>. [Accessed 10 February 2023].
- [25] S. Security, "GitHub," 15 Decemabr 2020. [Online]. Available: <https://github.com/Swordfish-Security/Pentest-In-Docker>. [Accessed 11 January 2023].
- [26] Nullsweep, "Nullsweep," 18 September 2019. [Online]. Available: <https://nullsweep.com/pivot-cheatsheet-for-pentesters/>. [Accessed 15 February 2023].
- [27] M. Marks, "GitHub," 6 December 2017. [Online]. Available: <https://github.com/docker/labs/blob/master/networking/A2-bridge-networking.md>. [Accessed 5 February 2023].

- [28] S. Solutions, "SecuriumSolutions," 12 May 2022. [Online]. Available: <https://securiumsolutions.com/blog/introduction-to-docker-and-how-docker-can-be-used-as-pentesting/>. [Accessed 13 December 2022].
- [29] S. Shah, "TelePort," 3 February 2022. [Online]. Available: <https://goteleport.com/blog/how-to-set-up-ssh-keys/>. [Accessed 15 January 2023].
- [30] J. Wallen, "TechRepublic," 18 May 2022. [Online]. Available: <https://www.techrepublic.com/article/deploy-docker-container-ssh-access/>. [Accessed 22 January 2023].
- [31] A. Kanitkar, "GoLinuxCloud," [Online]. Available: <https://www.golinuxcloud.com/docker-container-ssh/>. [Accessed 24 January 2023].
- [32] K. Ghimire, "GKiran," [Online]. Available: <https://gkiran.com.np/blog/pivoting>. [Accessed 30 January 2023].