Vulnerability Analysis of Container-based Virtualization

Thesis submitted in partial fulfillment of requirements
For the degree of
Master of Computer Science and Engineering
of
Computer Science and Engineering Department
of
Jadavpur University

by

Sanchita Mondal
Regn. No. - 154127 of 2020-2021
Exam Roll No. - M4CSE22003

under the supervision of

Mridul Sankar Barik
Assistant Professor

Department of Computer Science and Engineering
Jadavpur University
Kolkata, West Bengal, India
2022

Certificate from the Supervisor

This is to certify that the work embodied in this thesis entitled **"Vulnerability Analysis of Container-based Virtualization"** has been satisfactorily completed by **Sanchita Mondal** (Registration Number 154127 of 2020-21; Class Roll No. 002010502003; Examination Roll No. M4CSE22003). It is a bona-fide piece of work carried out under my supervision and guidance at Jadavpur University, Kolkata for partial fulfilment of the requirements for the awarding of the **Master of Engineering in Computer Science and Engineering** degree of the Department of Computer Science and Engineering, Faculty of Engineering and Technology, Jadavpur University, during the academic year 2021-22.

Mridul Sankar Barik,
Assistant Professor,
Department of Computer Science and Engineering,
Jadavpur University.
(Supervisor)

Forwarded By:

Prof. Anupam Sinha,
Head,
Department of Computer Science and Engineering,
Jadavpur University.

Prof. Chandan Majumdar,
DEAN,
Faculty of Engineering & Technology,
Jadavpur University.

Department of Computer Science and Engineering
Faculty of Engineering And Technology
Jadavpur University, Kolkata - 700 032

Certificate of Approval

This is to certify that the thesis entitled **"Vulnerability Analysis of Container-based Virtualization"** is a bona-fide record of work carried out by **Sanchita Mondal** (Registration Number 154127 of 2020-21; Class Roll No. 002010502003; Examination Roll No. M4CSE22003) in partial fulfilment of the requirements for the award of the degree of **Master of Engineering in Computer Science and Engineering** in the **Department of Computer Science and Engineering, Jadavpur University**, during the period of September 2021 to June 2022. It is understood that by this approval, the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein but approve the thesis only for the purpose of which it has been submitted.

Examiners:

(Signature of The Examiner)

(Signature of The Supervisor)

Department of Computer Science and Engineering
Faculty of Engineering And Technology
Jadavpur University, Kolkata - 700 032

Declaration of Originality and Compliance of Academic Ethics

I hereby declare that the thesis entitled **"Security Vulnerability Analysis of Container-based Virtualization"** contains literature survey and original research work by the undersigned candidate, as a part of his degree of **Master of Engineering in Computer Science and Technology** in the **Department of Computer Science and Engineering, Jadavpur University**. All information have been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by these rules and conduct, I have fully cited and referenced all materials and results that are not original to this work.

Name: Sanchita Mondal

Examination Roll No.: M4CSE22003

Registration No.: 154127 of 2020-21

Thesis Title: "Vulnerability Analysis of Container-based Virtualization"

Signature of the Candidate:

ACKNOWLEDGEMENT

I am pleased to express my gratitude and regards towards my Project Guide **Shri Mridul Sankar Barik**, Assistant Professor, Department of Computer Science and Engineering, Jadavpur University, without whose valuable guidance, inspiration and attention towards me, pursuing my project would have been impossible.

Last but not the least, I express my regards towards my friends and family for bearing with me and for being a source of constant motivation during the entire term of the work.

Sanchita Mondal
MCSE Final Year
Exam Roll No. - M4CSE22003
Regn. No. - 154127 of 2020-21
Department of Computer Science and Engineering,
Jadavpur University.

Contents

List of Figures

List of Tables

Abstract

Even though system virtualization is not really a new paradigm, the way it is implemented in present system architectures provides a powerful platform for system development, the benefits of which have only recently been realised as a result of the widespread implementation of commodity hardware and software systems. Virtualization, in concept, incorporates the employment of an encapsulating software layer that surrounds or supports an operating system and provides the same inputs, outputs, and behaviour as a physical device. However, new dangers and security challenges have arisen as a result of the design, implementation, and deployment of virtualization technologies thereby giving rise to the requirement for a reliable and secure virtualization solution.Over time we have come across various virtualisation technologies like virtual machines, containers, unikernels.

Container technologies have been used for a long time, but Docker is the most popular and widely used among them. Along with so many benefits, it also has a few drawbacks, the most critical of which is security.We attempted to develop a prerequisite and postcondition classification framework for attacker privileges in this project, which can be utilised to generate attack graphs particular to Docker.Advanced multi-step, multi-host attacks are common in today's computer networks. So, attack graphs are a more efficient way since they are built from a network's recognised vulnerabilities and indicate security concerns via attack paths that aren't visible in the results of primitive methodologies.

Chapter 1

Introduction

Large and small businesses have traditionally deployed applications on "bare metal" servers with operating systems installed. Due to cost savings and flexibility, numerous virtual machines (VMs) on the same physical server have recently become popular. Containers too have grown popular for application deployment in recent years due to their smaller footprints than VMs, their ability to start and stop more quickly, and their ability to encapsulate application binaries and their dependencies/libraries in standalone units for seamless mobility. A typical container ecosystem includes a code repository (e.g., GitHub) where container images are created from source code and libraries, then submitted to an image registry (e.g., Docker Hub) for deployment as application containers.There are several container based platforms of which more popular ones are Docker, Kubernetes, Amazon ECS.However, the widespread use of containers has resulted in several security issues, including attackers hacking credentials, source codes, and sensitive data from image repositories and code repositories, launching DoS attacks on application containers, and gaining root access to misuse the underlying host resources, to name a few.

To address the security problems, various research studies on container security have been conducted, with some focused on vulnerability analysis and others on mitigation strategies. However, the majority of related works focus on a single vulnerability, threat, use-case, or container subsystem.Hence in this work we have tried to define a prerequisite and postcondition classification scheme for attacker privi- leges that can be used for attack graph generation specific to Docker, which is a container-based open source platform for developing, deploying, managing and running applications.Using naive methods such as counting the number of flaws or studying the vulnerabilities individually, generate incomprehensible and limited security assessment results. While attack graphs prove to be a more efficient approach as graph developed from a network's identified vulnerabilities show security threats via attack paths that aren't visible in the findings of primitive methodologies.

1.1 Research Objectives

1. To study different virtualisation methods, its security, vulnerabilities and use cases.
2. To analyze vulnerabilities (from NVD) associated with different virtualization techniques and determine pre and post conditions.

1.2 Contribution of the Thesis

As of the date of this thesis there are over 1 lakh vulnerabilities published in NVD. Only vulnerabilities reported between the years 2021 to 2014 with the keyword 'Docker' are chosen for this research. We identified 115 vulnerabilities, but 15 of them were omitted since the impact score had not yet been submitted or the vulnerabilities were undergoing reanalysis. Finally, we studied at 100 vulnerabilities and built a set of rules based on the experimental data that would produce preconditions and postconditions for Docker vulnerabilities.

1.3 Outline of the Thesis

The thesis is organized as follows: Chapter 2 reviews the related work with a focus on attack graph generation approaches.Chapter 3 describes the virtualisation technologies briefly. Chapter 4 focuses on Docker, it's architecture including its ecosystem. Chapter 5 explains an enhanced prerequisite and postcondition rule based model. Chapter 6 concludes the work and discusses the future work.

Chapter 2

Related Works

Virtualization technologies have brought immense benefits in terms of resource utilization and ease of management. Even though virtualization intends to provide isolation, as long as the VMM and guest VMs reside in the same host. Solutions designed without taking into account their potential use in virtualized environments are at higher risk of introducing security vulnerabilities. As more support for security technologies is implemented at the hardware level (encryption, key management, isolation of resources, etc.) virtualization technologies will benefit from this, and as a result, the final users.

Federico Sierra-Arriaga et al. [2] the following topics: first, the security issues and challenges that come with migrating from stand-alone solutions to virtualized environments—special attention is paid to the Virtual Machine Monitor, as it is a key component in a virtualized solution; second, the impact (sometimes negative) that these new technologies have on existing host security strategies; and third, how virtualization technologies can be used to provide better security for hosts.The survey's goal is to provide the most up-to-date information about virtualization security challenges by presenting real-world instances of security flaws that have been revealed in the past, identifying the solutions developed to address those issues, and describing how and where they are used.

Container advancements have aided businesses and organisations in improving their processes and enabling new business models. However, the inherent security vulnerabilities posed in the container ecosystem have hampered its full implementation. The security landscape in containers was first analysed in this article. [8]used the STRIDE methodology in particular to identify vulnerabilities, threats, and the implications of threats throughout the entire container ecosystem. Many of the vulnerabilities they discovered are related to the containers' shared access to the kernel of the host operating system. While isolation measures (e.g., namespaces) and resource management mechanisms (e.g., cgroups) were in place, they were vulnerable to misconfigurations and indiscriminate usage of system functions and capabilities.The multiple external entities engaged in generating the code, building the image, configuring the installation, setting up network connectivities, and finally deploying the programme in production containers greatly increased the attack surfaces from an ecosystem perspective.They evaluated the merits and shortcomings of existing mitigation measures in relation to the highlighted security concerns in containers, in particular. Most existing mitigation measures, in their opinion, have inherent limitations and are insufficient to meet the security concerns posed by container systems.Several studies exist that are primarily concerned with the investigation and analysis of threats and vulnerabilities in the container ecosystem. One

study [9] compiled 223 container-related exploits from a public database and organised them into a two-dimensional attack taxonomy. The web app, server, library, and kernel had hierarchical layers, and the repercussions of attacks, such as sensitive information leaking, remote control, denial of service, and kernel privilege escalation, had another dimension. The authors discovered that 56.8% of these exploits are successful against the default container configurations. They studied 11 exploits that can bypass the isolation provided by the container to achieve privilege escalation. The authors then propose a defense mechanism to defeat those identified privilege escalation exploits. These studies emphasize the current vulnerable state of the container environment.Each kernel security mechanism restricts kernel permissions from different angles while the relationships among them are intricate and complicate. Improper configuration of these security mechanisms might lower their protective capability.However, the focus of their research was on privilege escalation attacks and how to fight against them using kernel security features.

Another study[10] looked into the Docker platform's security by looking at the vulnerabilities mentioned in the Common Vulnerabilities and Exposures (CVE) database. The authors of this paper employed static code analysis (SCA) techniques on vulnerable and patched versions of the Docker codebase to examine the differences between the two and the efficiency of SCA technologies in discovering vulnerabilities. This study analysed Docker's source code largely using static code analysis techniques, but did not relate it to real-world use scenarios or offer practical mitigation strategies.

The need for rapid deployment and lower costs has prompted a transition from virtual machine deployment to containerisation, which has increased application flexibility and performance. Containers, on the other hand, require a fully functional operating system to run, increasing an application's attack surface. As the next step toward network functions softwarization, [11] introduces unikernel network functions. The usage of unikernels allows network functions to be implemented as lightweight images that can run on a hypervisor or on the hardware layer directly. It is demonstrated that unikernel network functions achieve equivalent performance to container-based network functions while offering much improved security.Unikernels have a small memory footprint, are easy to package, and have faster startup times. Furthermore, because the self-contained environment only allows low-level functionality, Unikernels reduce the attack surface.In [5] an outline of Unikernels' strengths and limitations is presented. It is demonstrated that some Unikernels are vulnerable to known attacks such as buffer overflows and that recommended practises for mitigating common vulnerabilities have yet to be implemented.

Furthermore, for REST service written in Java, Go, and Python, article[12] compares the performance of unikernels versus containers. The results show that unikernels perform at least as well as or better than the corresponding containers in terms of memory consumption and execution/response times. However, the former consumes much more memory than the latter.

Today's computer networks are subjected to increasingly complex and numerous attacks. In order to assess such risks, vulnerability scanners are commonly used to determine the number, kind, and location of vulnerabilities on our networks. Such tools, on the other hand, analyse the vulnerabilities separately and do not demonstrate how they interact to disclose combinations of them that could constitute a substantial threat to our networks. To defend such networks, we must first identify each channel into the network and then prevent any malicious access via those paths. To analyse a network's overall vulnerability, vulnerabilities must be classified according to their multi-step and multi-host nature.There have been several attack graph generation algorithms proposed, each with varying levels of applicability. Prerequisite/postcondition models are among the most intuitive and

straightforward. Two well-known examples of this approach are TVA (Topological Analysis of Network Attack Vulnerability)[13] and NETSPA (Network Security Planning Architecture)[14]. TVA uses a knowledge library of exploit conditions in terms of preconditions and postconditions that relate to exploitation steps to produce attack graphs.

On top of TVA system, Aksu et al.[7] presented a study. They created a set of preconditioning and postconditioning guidelines and assessed their accuracy.They defined an enhanced categorization of attacker privileges used for generating attacker privileges from the vulnerabilities in NVD by rule-based techniques and using machine learning in order to automate the generation of attack graphs.To develop microservice based architectural module topology, Ibrahim et al.[15] used container-based deployment configuration files. They used Docker Compose in particular to extract the topology of the module orchestration. They also created "attack graphs," which represent actions that attackers could take to achieve their malicious purpose.

In the paper[16] MulVAL's methodology is extended to depict cyberattacks on machine learning systems. The suggested modification expands the capabilities of attack graph-based risk assessment by introducing hitherto unmodeled paradigms, giving security practitioners a tactical tool for assessing the impact and estimating the risk of a cyberattack on ML systems.Enterprises and organisations have been using threat models and attack graphs for more than 20 years to map the actions of prospective adversaries, analyse the effects of vulnerabilities, and visualise attack scenarios rather than analysing complex data within them and attempting to extract useful information for security control prioritisation and proper risk mitigation.To extract information about the influence of vulnerabilities and attack phases on enterprise networks, the methodology provided in [17] uses graph modelling algorithms such as mathematical series analysis, clustering, and optimization. They created attack scenarios and determined the impact of each step from all potential system attacks. To that purpose, they've added two new capabilities to earlier automatic generation methodologies: Prioritizing discovered vulnerabilities and analysing the impact of system states on the overall network in order to recommend which system states, vulnerabilities, and configurations pose the greatest overall risk to the ecosystem.For network risk estimations, probabilistic security metrics generated using attack graphs are frequently utilised. In [18], preexisting metrics are extended to describe realistic scenarios including networks with IDSs that have time latency and attackers and detection systems that can learn. The vulnerability is shown as a function of the time until detection. The metric represents the multi-dimensional nature of vulnerability, encompassing the attacker's and IDS's adaptive capabilities, IDS response time, attacker expertise, and attack graph structure.

Chapter 3

Technology Background

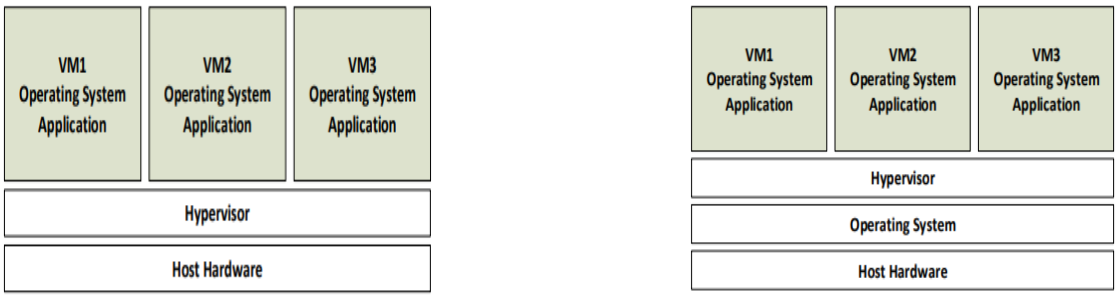
Before virtualization came into use, for every application to run a separate physical server is required and this server is required to run even when it is not utilised to its maximum capacity. This increases the operational cost as well as power consumption. Also the increase in the number of servers leads to infrastructure challenge thereby increasing infrastructure costs. The advent of virtualization has been able to overcome these challenges to a great extent. We begin our analysis of various popular virtualization solutions in this chapter, explaining how each works. This chapter provides a technical overview of the various types of virtualization, security vulnerabilities of each as well as their countermeasures.

3.1 Virtual Machines

A Virtual Machine (VM) is a compute resource that uses software instead of a physical computer to run programs and deploy applications. The virtual machine runs as a process in an application window, similar to any other application, on the operating system of the physical machine. More than one virtual machine can run on a single host machine. A hypervisor is a software that hosts different VMs with different operating systems for running on a single physical machine. Since VMs are capable of running on multiple operating system environments it saves time, physical space as well as cost. Also hypervisor allocates resources to VMs as well as monitors them by coordinating with the underlying primary operating system.

Hypervisors are of two types: Type I and Type II, as shown in Figure 3.1.Type I hypervisors, also known as bare metal directly runs on server hardware. Since hypervisor has direct interaction with hardware resources it minimizes overhead and also provides better hardware resource utilization. Type I is more secure,faster and more efficient than Type II but is hard to set up.

The type II hypervisors, also known as hosted runs on top of the supported OS. Here VMM runs as part of an extended host and the OS allocates and schedules the resources therefore causes overhead.



(a) Type I hypervisor

(b) Type II hypervisor

Figure 3.1: Classification of Hypervisor [1]

3.1.1 Security Vulnerabilities

Issues with the Hypervisor

1. Scalability: Various advantages of the virtual environment can lead to rapid scaling of virtual machines. Consequently comes along the problem of management tasks. The administrative tasks like upgradation, configuration are often not automated and thereby significantly multiply the impact of catastrophic events, e.g. worm attacks.

2. Mobility: Usually hypervisors, for each VM, store virtual disk which can be copied and run from various physical machines. This poses a security threat as it would be easy for attackers to copy the VM and access data on their own machine. And since this is a copy it would not show any record of intrusion.

3. Hypervisor Intrusion: The Hypervisor does the work of abstraction and resource allocation between hosts and guests. The main goal of the intruder is gaining privilege of the hypervisor and compromising the hypervisor by running arbitrary code on the hypervisor.

4. Hypervisor Modification: The original hypervisor may be modified externally where the system calls of the hypervisor may be changed to run malicious codes. Using an embedded hypervisor may be a solution to this problem or guest can verify integrity of the hypervisor. Even creating a trusted connection with trusted platform module can be another solution [19].

Issues between VMs or between VMs and VMM

1. Guest to VMM: The most serious attack done by the attacker on the VMM who has the control of a guest VM. With the increasing use of virtualisation this type of attack becomes more common. Thus it is very important to ensure protection against this attack.

2. Host to VMM: In order to gain control of the VMM attacker may attack on the host. The attack is not necessarily done by a privileged user.

3. Guest to Guest: This type of attack from a guest VM on another guest VM may break the isolation provided by the hypervisor thereby gaining access to control other guest VM or influence its behaviour [2].

Issues with Host OS

1. Guest to Host: Host machine is the physical hardware upon which the virtualization takes place. A guest VM may escape the hypervisor and execute its code in the host OS or as privileged guest.

2. Denial of Service: A denial-of-service (DoS) or distributed denial-of-service (DDoS) attack is an attempt to prevent a computer resource from being used by its intended users. It can be done by flooding the service with bad requests, causing it to stop responding to valid ones. The resources of virtual computers on the same physical machine are shared. If an attacker uses one virtual machine to get all of the host system's resources, leaving all other VMs to starve to death, due to a lack of resources, resulting in a denial of service attack in the virtual environment.

3. Information Leakage: The unreported backdoor in the Host system's I/O system may allow the Guest VM to access unprivileged information. Backdoors are normally opened for testing purposes, but in the unlikely case of an event, they may be left open, allowing an attacker to access sensitive information travelling between the Host and other guest VMs, as well as execute malicious code on the host or the guest.

Issues with Management Interface

The management interface is used to control, maintain, and monitor the guest VMs. The location of the management interface is determined by the type of virtualization utilised in the system.

Guest VMs have the potential to compromise the management interface, resulting in information leakage and the execution of arbitrary code. Cloud systems with web-based administration interfaces for controlling and monitoring virtual machines create attack surfaces for the web.

In a virtual environment, an ineffective virtual machine management policy will result in VM sprawl. VM sprawling occurs when the number of virtual machines (VMs) continues to rise but the majority of them are idle or never wake up, causing the host machine's resources to be significantly wasted.

Issues with the Network

Within the cloud, network virtualization works effectively, but it introduces various attack surfaces to the cloud infrastructure. Because virtual networks are software-based, they have a number of attack surfaces that can be found on any software system. Because software-based networks lack the level of security as physical networks, it is relatively easy to compromise and reroute network traffic, resulting in significant loss.

Since networks can be attacked during VM migration, information leakage can occur, raising concerns about the insanity of the VM image that is being migrated.Several virtual networks can be compromised, posing a serious threat to the cloud architecture as a whole.

3.1.2 Countermeasures of the Security Vulnerabilities

From the hypervisor and host OS to guest OSs, applications, and storage, the security of a virtualization solution is largely dependent on the individual security of each component[20]. To identify or prevent attacks, sound security practises must be maintained, such as maintaining software up-to-date with security updates, employing secure configuration baselines, antivirus software, or other relevant mechanisms.

1. Hardening the VMM (hypervisor): In most scenarios, as stated in the preceding sections, attackers take advantage of hypervisor vulnerabilities. Hence, the Hypervisor's security is crucial to the virtualization platform's security. Presently, there are three main aspects of Hypervisor security hardening: the first is to build a lightweight Hypervisor; the second is to protect the integrity of the Hypervisor using trusted computing technology; and the third is to improve the Hypervisor's defence capability by configuring a virtual firewall and allocating host resources appropriately. Latest security patches should be deployed as soon as they become available. For exposed terminals, proper firewall and antivirus configuration is required to avoid black box attacks[21].

2. Secure Identity and Access Management: In a virtual environment, access control refers to the practise of limiting access to a resource to authorised VMs. Physical resources will be used effectively, and communication between VMs and between VM and VMM will be more reliable, due to a well-designed access control strategy. Identity and Access Management allows the authorized people to have access to the appropriate resources at the right time and for the legitimate purposes. Instead of a single administrator having access to the entire system, role-based access to the resources should be implemented [22].

3. vIDS/vIPS: The Virtual Intrusion Detection System (vIDS) / Virtual Intrusion Prevention System (vIPS) safeguards the virtual environment by gathering and analysing data from the network and the host to see if there are any signs of attack. Benefits of deploying vIDS/vIPS in a virtual environment include: monitoring and analysis of virtual network behaviour, configuration of the system and assessment of its weaknesses, exceptional statistics and analysis, ensure that security policies are consistent, security event notification, protection of vulnerable applications from virtual network attacks, spyware detection [23].

4. Securing the network: To eliminate information leakage, arbitrary code execution, network traffic rerouting, and VM sanity, a secure network is required. Secure communication channels, such as SSL and IPsec, must be used to secure networks. A Virtual Firewall (VF) is a packet filtering and monitoring firewall that is deployed and run entirely within a virtual environment. Our network and computers may be protected by a virtual firewall from outside dangers such as hackers and harmful attacks.

3.1.3 Case Study: VMWare Security Analysis

VMware ESX Server can be used by physical hardware machines for hosting virtual machines using virtualisation. VMWare tools optimizes overall performance, usability and manageability of virtual machines. For security purposes firewall is the high level security. Though it comes with many tools necessary for enterprise infrastructure it suffers from a few security issues. They are communication issues, VM escape, VM monitoring, denial of service, VM migration and compliance issues.

1. Isolation and shared resources: VMs can communicate with each other through the shared network. This makes ESX server prone to attacks from malicious virtual machines.

2. Life Cycle of a VM in VMWare: All information of a virtual machine is stored in an image file.This file can be easily copied, distributed, deleted. And hence malicious users can use VM image file to mitigate other machines so as to launch attacks on other virtual machines.

3. Security Bottlenecks in VMWare:

3.2 Containers

Though VMs provide several benefits it comes with a huge overhead on the system while imitating the hardware into the virtual environment. Recently containers has emerged as a favourable solution to the VMs that are costly to migrate or scale. Unlike VM, that uses hypervisor based virtualisation where each VM may run separate OS, Containerization virtualizes OS level resources. Containers encapsulate standard OS processes along with their dependencies to create containers. This has been shown in Fig. 3.2. A container is an independent, self-sufficient package for running an application or service. Docker is the most popular container platform in the cloud community [24].

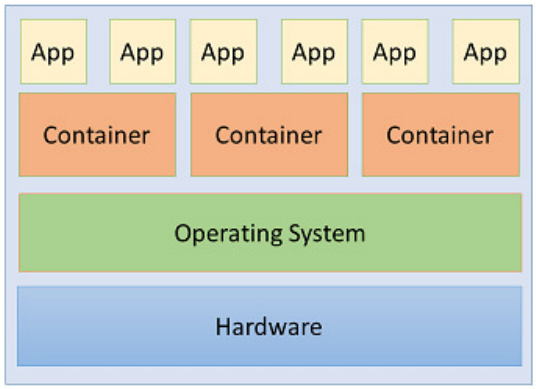


Figure 3.2: Container based Virtualisation [2]

3.2.1 Container Components

Image

The components of an app are built and placed into an image in the first phase of the container lifecycle (or perhaps into multiple images). A container image is a package that includes all the files needed to run the container. The httpd binary, as well as associated libraries and configuration files, would be included in an image to run Apache, for example. All additional OS functionality is provided by the OS kernel within the underlying host OS, hence an image should only contain the executables and libraries required by the app.

Registry

Registries are services that allow developers to conveniently store images. These images are tagged and catalogued for identification and version control, and can be downloaded created by others. Self-hosted registries and registries as a service may also be options.Registries provide APIs that allow common image-related operations to be automated.

Orchestrator

Orchestrators allow DevOps personas or automation to pull images from registries, deploy those images into containers, and manage the containers while they are running. An orchestrator's abstraction allows a DevOps persona to specify how many containers should execute a particular image and how much memory, processing, and storage space should be allotted to each. The orchestrator is aware of the current state of each host, as well as the resources available to each host, and decides which containers will execute on which hosts. Container resource consumption, job execution, and machine health are all monitored by orchestration tools across hosts. An orchestrator may automatically restart containers on new hosts, depending on its configuration.

3.2.2 Container Technology Architecture

Container Technology Architecture consists of the following five tiers as shown in Fig:3.3

1. Developer systems: create images and transmit them to be tested and accredited.
2. Testing and accreditation systems: validate and verify picture content, sign images, and transmit them to the register.
3. Registries: images are stored in registries and distributed to the orchestrator upon request.
4. Orchestrators: images are converted into containers, and containers are deployed to hosts.
5. Hosts: run and stop containers according to the orchestrator's instructions.

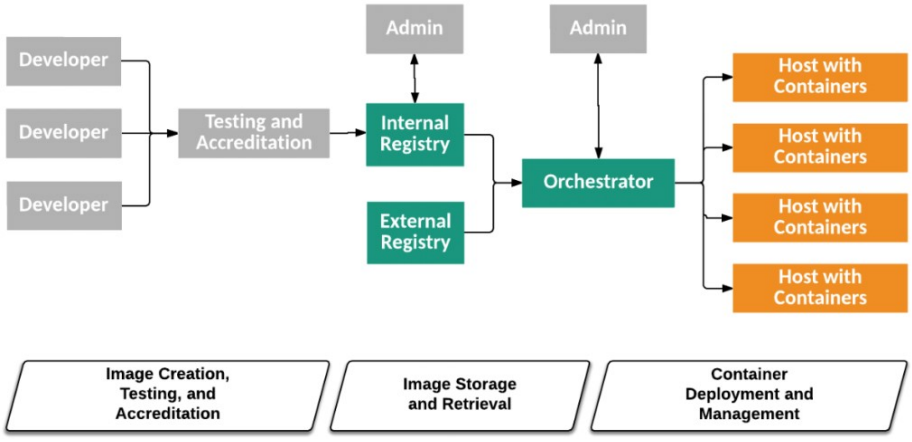


Figure 3.3: Container Architecture[3]

3.2.3 Vulnerabilities of Components of a Container

Image Vulnerabilities :-

Throughout the process of image building, image storage, image deployment there are several attack surface.This has been depicted in Fig. 3.4

Image Creation Vulnerability The image creation step needs a Dockerfile (details of this have been discussed in chapter 4) containing all the instructions for building an image which is then converted to container image. This step may encounter a number of potential security risks. At each step of building the image any intruder can easily take following malicious actions:

1. Accessing build secrets.
2. Adding malware software into the image intentionally.
3. Attacking the host of the image.
4. Enumerate network topology accessible from the build structure.

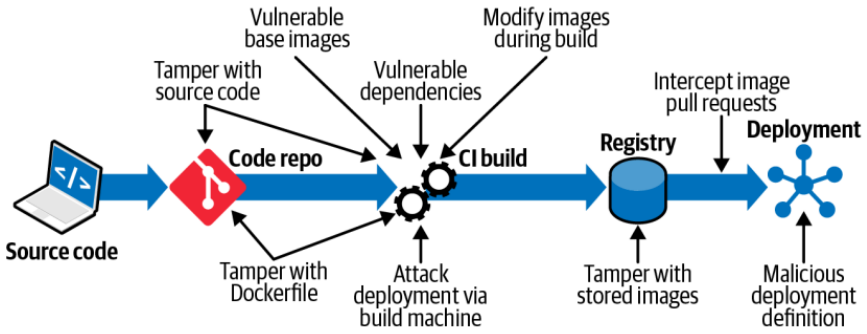


Figure 3.4: Container Image Attack Vectors[4]

If the components of the image are not up to date there may be known vulnerabilities. Images may have configuration defects.[25]

Image Storage Vulnerability After building of the image, it needs to be stored in a repository. Images may contain sensitive information which may be subject to confidentiality risks if the connection to registries are done over insecure connections. Also insufficient authentication and authorization requirements may lead to loss of significant technical details. An attacker can also modify or replace the images in registries. Intruder can also intercept in network traffic or steal developer's credentials.

Image Deployment Vulnerability The main security concern of the deployment stage of container images is ensuring that the right image has been deployed. Also lack of admission control may cause failure in certain additional checks to run container images.

Orchestrator Risks :-

When a single orchestrator runs many different apps then the access provided to each user must be restricted to specific bound otherwise malicious user may affect the operation of other containers managed by the orchestrator.

Including own authentication directory service leads to weaker account management practice. Traffic between individual nodes is routed over a virtual network and there is a potential risk of poorly separated inter-container traffic. Weak configuration of orchestrator may lead to exposure of risks of the orchestrator and all the container components managed by the orchestrator.

Container Risks :-

Usually containers have access to communicate with other containers as well as the host OS over the virtualised network. If a container is compromised then it may expose other resources in the network to risk.

Improper runtime configuration of the containers potentially lowers the security of the system. If a container runs in the privileged mode it has access to all other containers in the system as well as host OS thereby allowing it to affect operations of other containers. Containers may still be compromised owing to weaknesses in the apps they execute, even if businesses take the safeguards recommended in this guidance. This isn't a problem with containers per such; rather, it's the result of common software faults within a container environment.

Host OS Risks :-

Every host OS has an attack surface. It is a collection of all possible ways for attackers to gain access to and exploit the vulnerabilities of the host OS. The more the attack surface, the more likely an attacker will locate and exploit a vulnerability, thereby compromising the host OS and the containers that operate on top of it. Even with container-specific OSs.

The usage of a shared kernel usually results in an increased inter-object attack surface than encountered with hypervisors, despite the fact that containers provide strong software-level resource isolation. Insecure container settings increase the risk of file manipulation on host volumes. If a container is authorised to alter the files in those directories. These modifications may have an impact on the host's stability and security, as well as the stability and security of any containers operating on it.

3.2.4 Countermeasures for Major Vulnerabilities

Image Countermeasures

To give more actionable and trustworthy results, organisations should employ tools that include the pipeline-based build approach and the immutable nature of containers and images into their design. Images should be configured to run as non-privileged users. Validation of image configuration settings including vendor recommendations and third-party best practises, maybe the tools and processes that can be used.

Container images should be scanned for embedded malware on a regular basis. Malware signature sets and behavioural detection algorithms should be used in the monitoring processes. Organizations should maintain a list of trusted images and registries, and only allow images from this list to run in their environment, this may be effective in reducing the possibility of untrustworthy

or malicious components being deployed. Multilayered approach may be taken that includes discrete identification of each image,validation of image signatures before image execution, constant monitoring and maintenance of these repositories to ensure images within them are maintained and updated.

Registry Countermeasures

Organizations should set up their development tools, orchestrators, and container runtimes to connect to registries only over encrypted channels. the main goal is to ensure that all data sent to and extracted from a registry is encrypted in transit and happens between trusted endpoints.

There are two main ways to reduce the risk of using stale photos. To begin, organisations can prune registries of images that are unsafe or susceptible and should no longer be used.

Based on time triggers and labels connected with photographs, this procedure can be automated. Second, operational methods should prioritise using immutable identifiers to retrieve pictures that define discrete versions of images to be used. Authentication should be enforced for all access to registries that include proprietary or sensitive images. To ensure that only images from trusted entities can be added to a registry, any write access to it should require authentication.

Orchestrator Countermeasures

Orchestrators should use a least privilege access model in which users are only provided the ability to perform specified operations on the specific hosts, containers, and images that their job roles require, especially because of their broad scope of control. Since user accounts have the capacity to affect all resources in the environment, access to cluster-wide administrative accounts should be strictly regulated. Strong authentication mechanisms, such as requiring multifactor authentication rather than just a password, should be used by businesses.

Orchestrators should be set up to segregate network traffic into discrete virtual networks based on the level of sensitivity. Orchestration platforms should be designed to offer features that make all of the apps they operate more secure.

Container Countermeasures

A vulnerable runtime puts all of the containers it supports, as well as the host itself, in risk. Organizations should employ tools to search for Common Vulnerabilities and Exposures (CVEs) vulnerabilities in the runtimes deployed, upgrade any vulnerable instances, and verify that orchestrators only allow deployments to properly maintained runtimes.

Container egress network traffic should be controlled by organisations. These restrictions should be in place at network borders at the very least, preventing containers from sending traffic across networks with varying levels of sensitivity. Separate environments for development, test, production, and other scenarios should be established, each with its own set of controls to offer role-based access control for container deployment and management activities.

Host OS Countermeasures

Organizations should not mix containerized and non-containerized workloads on the same host instance, in addition to grouping container workloads onto hosts by sensitivity level. Containers

should never be able to mount sensitive directories on a host's file system, particularly those storing operating system configuration settings.

Organizations should employ tools to keep track of which directories containers are mounting and to prevent the deployment of containers that break these policies. Organizations should use management methods and tools to ensure that the components used for base OS management and functionality are up to date. Organizations should check for and apply updates to all software components used within the OS using tools offered by the OS vendor or other reputable organisations on a regular basis. Not only should the OS be kept up to date with security updates, but it should also be kept up to date with the latest component updates advised by the vendor.

3.3 Unikernels

Recently Unikernel concept has gained a lot of attention from the research community. Unikernels are specialised light weight bootable disk images designed to run a single process. It aims at reducing memory footprint and image size of applications by integrating application code and its dependencies into a single bootable binary images. Though it cannot do multi-processing, multi-threading is possible. Also it is capable of providing security as there is a distinct kernel and there are typically less attack options.

Fig.3.5 depicts the ecosystem of Unikernels, the differences between the development and production environments, as well as the Unikernel distribution. In development, an OS is required to set up the Unikernel and its features, but in production, the Unikernel should be deployed without a host OS for improved performance and security. The Unikernel image can be hosted and modified through an online repository. Furthermore, the environment/functionalities differ based on the Unikernel distribution. A package manager (e.g. Conan) for developing and downloading Unikernel compatible software, a toolstack/domain manager (e.g. Cosmos) for managing unprivileged domains, and an API (e.g. Rest) for remotely managing Unikernels via issuing commands are among the features.

3.3.1 Types of Unikernels

Clean Slate

A clean slate Unikernels make no attempt to replicate a traditional operating system in any way. They are written in a single programming language and provide external communication (networking) interfaces in that same language. MirageOS[1] (OCaml) and IncludeOS[2] (C++) are two examples. Language-specific virtual machines, such as the Java Virtual Machine (JVM), can be used as actual virtual machines with the help of these unikernels.

Legacy

Legacy Unikernels, on the other hand, use a subset of POSIX to allow unmodified software to operate, with some requiring only modest configuration modifications. They don't enable timesharing; instead, the virtualization layer is responsible for this.

[1] https://mirage.io/
[2] https://github.com/includeos/IncludeOS

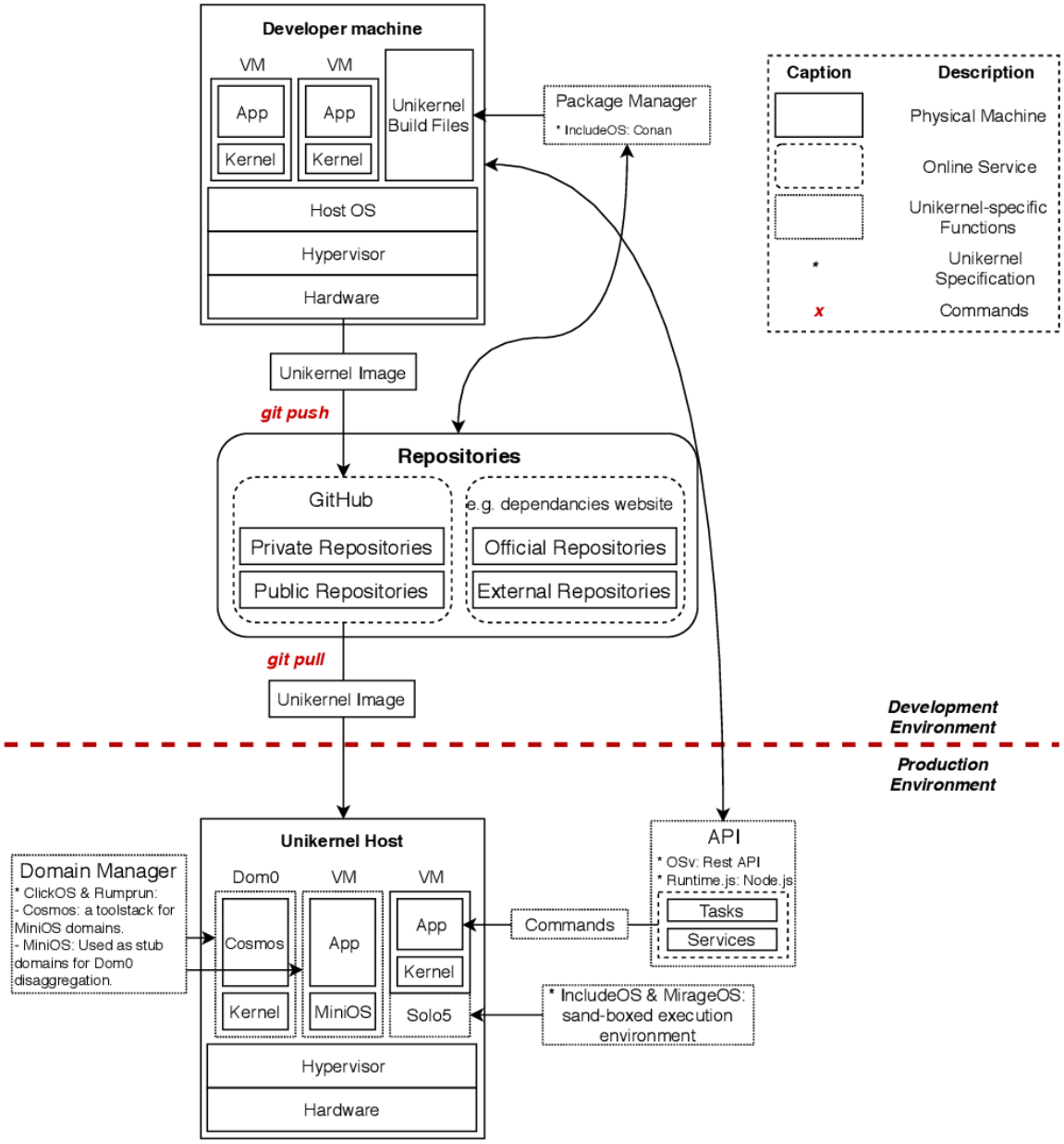


Figure 3.5: Unikernel Ecosystem[5]

3.3.2 Advantages of Unikernel over Containers

Performance

The performance of containers is said to be their main advantage. Containers run directly on the kernel layer, eliminating the overhead of a hypervisor or a virtual hardware layer. As a result, there is a low resource usage and excellent performance. Unikernels, on the other hand, produce comparable outcomes. This is due to the application's decoupling from not just other services but also unwanted libraries, which helps to compensate the overhead provided by the hypervisor and virtual hardware layer [11].

Security

The level of security that containers provide is one of their most significant drawbacks. Containers are insecure in comparison to virtual machines (VMs). Containers, unlike virtual machines, make use of kernel capabilities such as cgroups and namespaces. This indicates that the kernel provides resource isolation natively. If one container is compromised, it affects all of the others. A compromised container, for example, may begin to use all available network resources. Unikernels, on the other hand, run as virtual machines (VMs), benefiting from the complete resource isolation offered by a hypervisor and the virtual hardware layer [11].

Isolation

Isolation is another drawback of using a shared kernel. All operating containers are affected by changes to the kernel, upgrades, and even loading and unloading modules. Unikernels, on the other hand, can run as VMs or directly on the hardware layer, with hypervisor-based or full isolation. As a result, they are more adaptable to such changes. It's only important to make sure the changes don't compromise the hypervisor or the hardware below it [11].

3.3.3 Unikernel Security Overview

Unikernels offer a limited and unique attack surface by limiting the code base of deployed programmes, making them relatively secure [5].

1. Shell: Several Unikernels lack a native shell, rendering most payloads that rely on bash ineffective. By raising the complexity of the payloads, you can further prevent automated attacks or inexperienced attackers from successfully exploiting a vulnerability.

2. System Calls: Since system calls are frequently removed or not supported by Unikernels, malicious users must know the exact memory structure in order to invoke a function call like open() or write(). Randomised memory layouts are implemented at every build, significantly reducing the attack surface.

3. Hardware emulation: Possible breakouts, such as the 2015 Venom assault that damaged QEMU, can be avoided by not emulating hardware interfaces such as floppy drives, the Peripheral Component Interconnect (PCI) bus, or the Graphics Processing Unit (GPU).

3.4 Comparison of Virtual Machines, Container and Unikernel

The fundamental differences between the various types of virtualization, are summarised in Fig. 3.6. The layout of software assisted virtualization is shown in Fig. 3.6 (top left) and virtualization of the operating system on top of hypervisor creates virtualised hardware replaces the operating system, allowing for the development of virtual machines. Fig. 3.6 (top-right) shows hardware assisted virtualisation, and para-virtualisation. The hardware itself (sometimes assisted by the hypervisor) runs the virtual machines itself. Fig. 3.6 (bottom-left) shows how containers are implemented, while Fig. 3.6(bottom-right) presents an overview of the virtualization architecture used by Unikernels.Unikernels, as shown, do not require an operating system for the system to work properly.

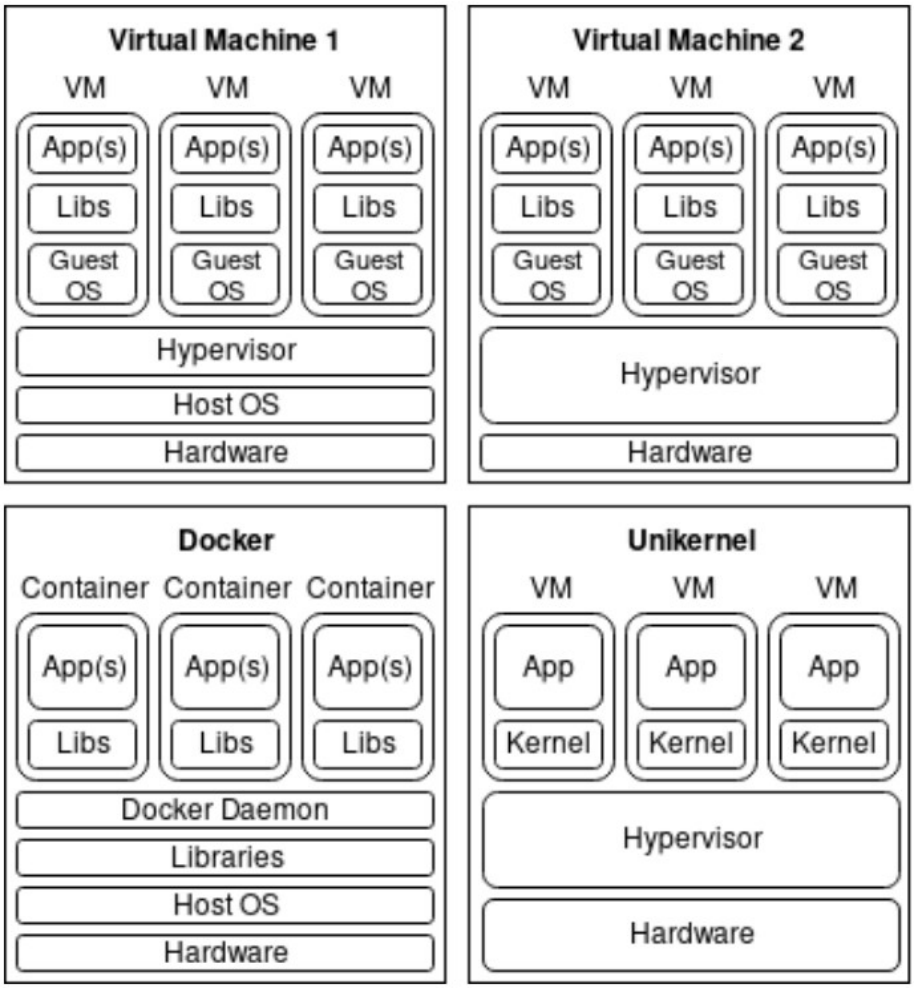


Figure 3.6: Layout of various virtualisation techniques[5]

After analysing the different types of virtualisation technologies we can compare their performance depending on factors like start up time, boot time, security, portability.The comparison has been described in the below Table. 3.1

	Virtual Machines	Container	Unikernels
Deployment Time	slow(∼5/10 s)	fast(∼800/100 ms)	very fast(∼<50 ms)
Booting time	slow	medium	fast
Image Size	large(∼1000 MBs)	smaller(∼50 MBs)	very small(∼<5 MBs)
Memory footprint	∼100 MBs	∼5 MBs	∼8 MBs
Programming Language	no	no	yes
Hardware Portability	high	high	medium
Security	high	medium	high
Live Migration Support	yes	yes	requires manual implementation
Multitenancy	yes	yes	yes

Table 3.1: Comparison of Virtual Machines, Containers and Unikernel

Chapter 4

Docker: Platform as a Service (PaaS) for Containers

Docker is a container-based platform for developing and running applications. It's an open source platform for developing, deploying, and managing distributed systems. For running apps in containerized settings, Docker containers have proven to be extremely useful. Despite the fact that they are not a complete substitute for virtual machines, they do provide separation for other container applications. Dockers are widely used for application development, testing, and deployment. By encapsulating dependencies, libraries, binaries, and other essential bits of software in a container, Docker applications are packed together for reliable testing of applications across various platforms. Docker containers have the following advantages:

1. Services can be scaled by removing or adding containers.
2. The Docker container's density can be increased to greater tasks to handle.
3. Because of their simplicity and speed, they can be constructed fast.The deployment time is really short.
4. Due to resource limitations and other resource management measures, Docker containers have a very high efficiency given resource use.

4.1 Docker Ecosystem

The Docker ecosystem is made up of several components. Docker is a container image and runtime specification that includes Dockerfiles for a reproducible build process (Fig. 4.1a). The Docker daemon, also known as the Docker engine, is used to implement this specification in Docker software. The Docker hub is a central repository that allows developers to store and share their images, as well as a trademark and third-party application bindings (Fig. 4.1b). Finally, the build process pulls code from remote sources and stores the packages that will be integrated in the images (Fig. 4.1c)

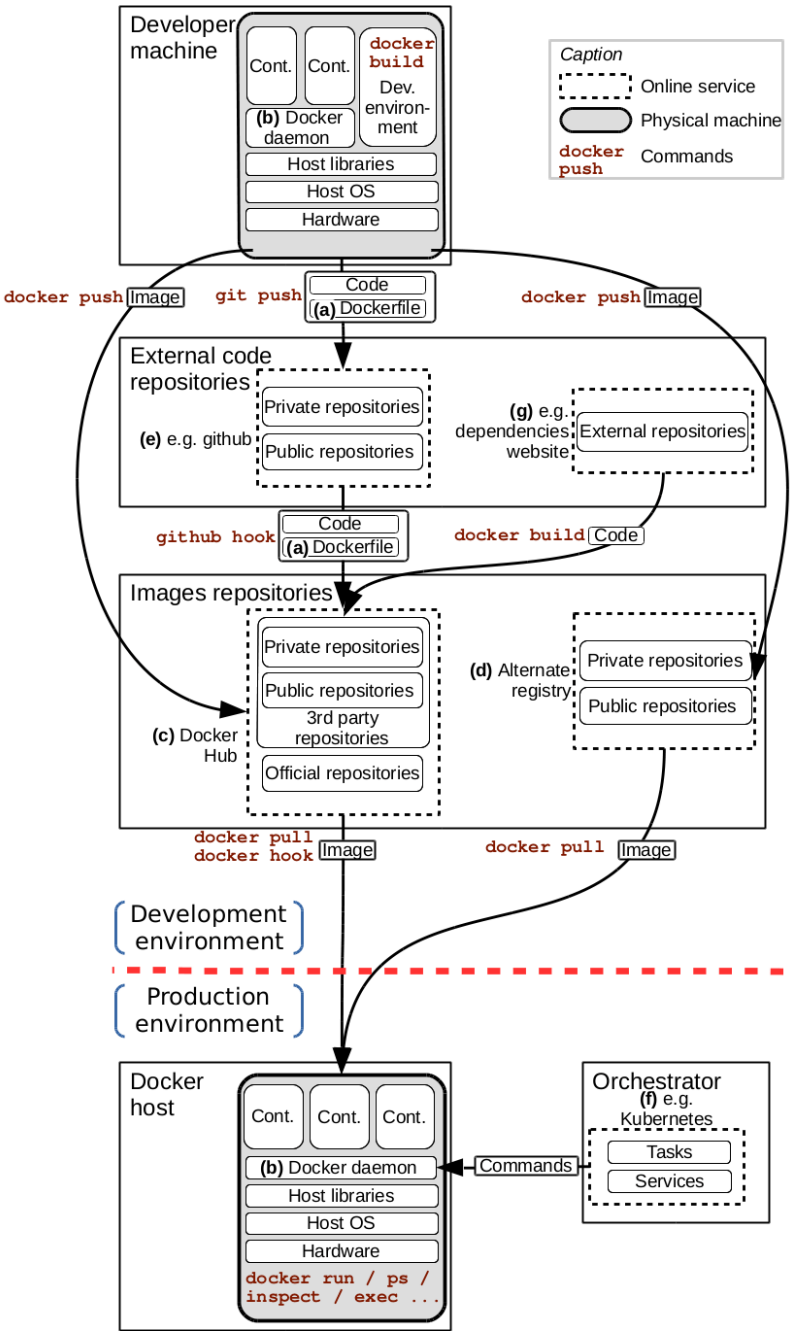


Figure 4.1: The Docker ecosystem. (a) Docker specifies container images and runtime, including Dockerfiles that enable a reproducible building process. (b) The Docker repositories. (c) The build process. Arrows show the code path and associated commands (docker action)[6]

4.1.1 Docker Specification

The scope of the specification is container images and runtime. Layers and metadata in the JavaScript Object Notation (JSON) standard are included in Docker disc images. The pictures are saved in the directory /var/lib/docker/driver¿/.where driver¿ denotes the storage driver in use.Advanced multi-layered unification, for example, is a technique that has been employed.B-tree file system (Btrfs), filesystem (aufs), virtual filesystem switch (VFS), device mapper, or OverlayFS are all examples of filesystem switches.Starting with a base image, each layer includes the filesystem modifications relative to the previous layer (typically, a lightweight Linux distribution).The images are organised by a lightweight Linux distribution.Except for the root, each image in a tree has a parent.The roots of the trees are represented by images. This structure enables Docker to only include in an image the modifications that are directly related to it.

Docker has two methods for building images. It can start a container from an existing image (docker run), make changes and installations inside it, then stop the container and save its state as a new image (docker commit).This procedure is similar to traditional VM installation, but it must be repeated at each image rebuild; the sequence of instructions is identical because the base image is standardised.Dockerfiles automate this process by allowing users to define a base image and a command sequence to construct the image, as well as other image-specific variables like exposed ports. The docker build command is then used to create the image.

4.1.2 Docker Internals

Docker containers surround the host computer in a controlled environment where arbitrary code can execute securely. This isolation is accomplished in two ways.Kernel namespaces,a feature of the kernel as well as control groups (cgroups)—that have been combined since the Version 2.6.24 of the Linux kernel. Namespaces are used to separate the processes' perspectives on the system. Currently, the kernel has six different namespaces—PID,IPC, NET, MNT, UTS, and USER—all of these terms refer to different ways of isolating different aspects of the system. Each of these namespaces has its own set of rules,has its own type-specific kernel internal objects and each provides a local instance of some routes in the /proc and /sys filesystems.Cgroups are a kernel mechanism for limiting a process's or a group of processes' resource utilisation. Their purpose is to prevent a process from using all of the host's resources, starving other processes and containers. CPU shares, RAM, network bandwidth, and disc I/O are all controlled resources.

4.1.3 The Docker Daemon

On the host machine, the Docker software runs as a daemon. It can start containers, manage their isolation (cgroups, namespaces, capability limitations, and SELinux/Apparmor profiles), monitor them for actions (like restarting), and spawn shells inside running containers for administration. The software can modify the host's iptables rules and create network interfaces. It's also in charge of container image management, which includes pulling and pushing images from a remote registry (like the Docker hub), creating images from Dockerfiles, and signing images. The daemon operates on the host as root (with full privileges) and is controlled remotely through a Unix socket. The daemon can also listen on a traditional TCP socket.

4.1.4 The Docker Hub

Developers can upload their Docker images to the Docker hub online repository, and consumers can download them. Developers can sign up for a free account that allows them to build public repositories or a premium account that allows them to create private repositories. Developer repositories have the name "developer/repository" in their name. Official repositories, offered directly by Docker Inc., are also available; these are referred to as "repository." Docker's daemon, hub, and repositories work in a similar way to a package manager, with a local daemon installing software on both the host and distant repositories.

4.2 Docker Containers Security Architecture

Docker security is focused around three main components:

1. Userspace process isolation maintained by the Docker daemon
2. Kernel level enforcement of the stated technique
3. Network operations security

4.2.1 Isolation

Namespaces, cgroups, hardening, and capabilities are all used exclusively by Docker containers in the Linux kernel. Cgroups constraints must be configured on a per-container basis using the -a -c options on container launch. Namespace isolation and capabilities drop are enabled by default, but cgroups limitations are not.The isolation setup is rather severe by default. The sole weakness is that all containers use the same network bridge, making ARP poisoning attacks between containers on the same host possible.

However, Docker's overall security can be decreased via choices that are triggered at container launch and provide containers expanded access to select sections of the host. Additionally, security configurations can be specified worldwide using Docker daemon options.This includes security-lowering options like –insecure-registry, which disables the Transport Layer Security (TLS) certificate check on a certain registry. Options that improve security, such as the –icc=false argument, is forbidden although network connectivity between containers and mitigation of the ARP poisoning attack are available, they are rarely used since they prohibit multicontainer applications from functioning effectively.

4.2.2 Host Hardening

Security-related limiting constraints imposed on containers are enforced by host hardening via Linux kernel security modules (such as breaching a container and escaping to the host operating system). SELinux, Apparmor, and Seccomp are currently supported by default profiles. These are not limiting profiles, but rather generic ones. For example, the dockerdefault Apparmor profile grants full access to the filesystem, network, and all Docker container features. Similarly, by default, SELinux groups all Docker objects into a single domain. As a result, while default hardening protects the host against containers, it does not protect containers from each other. This aspect of security can be addressed by writing separate profiles that are dependent on the containers individually.

4.2.3 Network Security

Docker makes use of network resources for image distribution and Docker daemon remote control. Docker uses a hash to verify images downloaded from a remote source before distributing them and the registry connection is secured by TLS. Moreover,Developers may now sign their images before publishing them to Docker Hub,a storage facility thanks to the Docker Content Trust architecture.TUF11 is a framework that was created expressly for this purpose to fix problems in the package manager.12 TUF can recover from a key breach and reduce replay attacks by a factor of two.Embedding timestamps for expiration in signed pictures,and so forth.The expense is complex key management; TUF uses a public-key architecture in which each developer has their own root key ("offline key"), which is used to sign "signing keys" that are used to sign Docker images[6].

The Docker daemon can be controlled remotely through a socket, allowing any Docker command to be executed from another host. The socket used to administer the daemon is by default a Unix socket;owned by root and located at /var/run/docker.sock.It can be converted to a TCP port via root:docker.socket. With access to this socket, attackers can pull and push data.Any container can be executed in privileged mode, giving it root access to the host.

4.3 Docker Use Cases: Security Challenges

Containers are frequently compared to virtual machines in security talks (VMs). Some container technologies, like OpenVZ, which is used to create virtual private servers, were created with very different goals in mind. It's critical to define Docker's most common applications and explore their security implications.We can distinguish 3 forms of Docker usages:

1. Recommended use-case, i.e., the usages Docker was designed for, as defined withinside the legitimate documentation;
2. Wide-spread use-case, i.e., the common usages done through application developers and system administrators;
3. Cloud provider's CaaS use-case, i.e., the usages guided through the Cloud vendors implementations to manage with both security and integration inside their infrastructure.

4.3.1 Recommended use-case

Since there are no tools to manage Docker containers, they are not considered virtual machines. Instead, Docker developers advise using a micro-services approach, which requires each container to host only one service in a single process. All administrative actions (container stop, restart, backups, updates, builds, etc.) must be conducted through the host computer, implying that the containers admin has root access.The key benefit of Docker is the simplicity with which applications can be deployed. It was created to keep the data plane and the code plane separate. A generic build file can be used to create Docker images everywhere. This generic approach to image production makes the process essentially host-agnostic, relying only on the kernel and not installed libraries.

Use cases for Docker that follow the official recommendations : Code pipeline management; developer productivity; app isolation; server consolidation; debugging capabilities; multi-tenancy; and rapid deployment

4.3.2 Wide-spread use-case

Docker is used by some system administrators and developers to ship full virtual environments. This reduces the number of system management duties to a bare minimum (such as docker pull). Application developers and system administrators are two examples of widespread uses. It's tempting to execute administrative operations from within a container because it has enough software to run a full system but is completely opposed to Docker's design and has several security implications.

4.3.3 Cloud provider‘s CaaS use-case

PaaS implementations advise platform as a service (PaaS) usages to deal with security and infrastructure integration challenges. Docker was implemented into the major PaaSs by the end of 2015. The market leaders are Amazon Web Services and Google Container Engine. Both approaches are similar: a VMor cluster of VMs is established, with an orchestrator tool available to manage the containers inside the VMs. The container's admin has complete orchestrator access in this model.

4.4 Docker vulnerability-oriented analysis

Containers, host OS, co-located containers, code repositories, image repositories, and management network are among the targets we've identified for attack.

4.4.1 Vulnerabilities identification,exploitation and mitigation

The primary components of the Docker ecosystem are studied independently in this section, exposing (part of) their threat surface. Five categories of vulnerabilities have been recognized[26], each one associated to a distinct layer of the ecosystem. In terms of a production system running Docker containers, these levels are categorised from the most remote to the most local:

1. **Insecure system configuration:** Docker's default configuration is fairly secure because it isolates containers and limits their access to the host. A container has its own namespaces and cgroup, and only a few capabilities.Some options, either given to the Docker daemon at startup or given to the command deploying a container, might provide containers with more access to the host; for example, Containerization of sensitive host directories, Docker control socket permissions, options directly allowing containers with greater access to the host (–net=host, –uts=host, –privileged, additional capabilities).The –uts=host option sets the container in the same UTS namespace as the host. As a result, the container can see and change the host's name and domain.For instance, with some storage drivers (e.g.,AUFS), Docker does not limit containers disk usage.Along with these runtime container options, various host parameters can open the door to assaults. Even simple characteristics might cause a denial of service.

   The Center for Internet Security created a Docker Benchmark to prevent these potentially hazardous options that could result in the container accessing the host.It offers two lists of options: those that should be used and those that should not be used when using Docker to run containers as isolated applications.These options are divided into six categories: host configuration, Docker daemon configuration, Docker daemon configuration files, Container images and build files, Container runtime, and Docker security operations.

2. **Vulnerabilities in the image distribution, verification, decompression, and storage processes:**Since the code contained in these images will be run on the host, the distribution of images through the Docker Hub and other registries is a source of vulnerabilities in Docker.

   The Docker Hub has a package repository-like architecture, with the Docker daemon functioning as a package manager on the host. As a result, it's prone to the same vulnerabilities that affect package managers. The Docker daemon with root privileges is responsible for the processing, storage, and uncompression of potentially untrusted code. This code can be tampered with at the source (malicious image) or during transmission.If an attacker possesses control of a portion of the network between the Docker host and the repository, package manager attacks are feasible. If the exploit is successful, attacker will be able to get her image downloaded on docker hosts. As a result, images are compromised, and vulnerabilities in the extraction process can be exploited. The only security prior to release 1.8 was the use of TLS on the registry connection, which could be disabled. Docker 1.8 introduces Content Trust, an architecture for signing images in the same way that package managers sign packages.

   Account hijacking, tampering with network communications (depending on the use of TLS), and insider attacks are all possible compromise methods in the image deployment architecture. This configuration adds numerous external intermediary steps to the code route, each with its own authentication and attack surface, resulting in a larger attack surface overall.

3. **Vulnerabilities within the images:** When in production, the code accessible for download on the Docker Hub (used to build images) is directly exposed to attackers.While making it easier for developers to upload and pull images to and from Docker Hub, it has also made it easier for malicious actors to transmit malware. Image vulnerability scanning is embedded into several cloud systems, including GCP, AWS, and Azure. Although Docker Hub offers a scanning tool, many studies have identified an increase in the number and extent of vulnerabilities in public repositories like Docker Hub over time. According to a 2015 study, 30% of official photos (statically assessed) contained high-priority vulnerabilities. Based on a review of 2500 Docker Hub images, a group of academics found earlier in 2020 that "the number of newly discovered vulnerabilities on Docker Hub is constantly increasing" and that "certified images are the most vulnerable."Prevasio, a cyber security firm, has conducted an investigation[**?**] of about 4 million Docker Hub images and discovered that 51% of the images have exploitable vulnerabilities.

   When an attack comes from the outside, exploiting vulnerabilities is critical (i.e., not a malicious image).Traditional application vulnerabilities can be exploited using traditional ways if the container exposes an entry point (network port, input data, etc.). Docker Security Scanning can be used to scan images in private repositories to ensure that they are free of known security issues. For a limited period, this feature is provided as a free preview for private repository subscribers. The scan goes through each layer of the image, finding and indexing the software components' SHAs(secure hashing algorithm). These SHAs are compared to the CVE database to get information on already known security vulnerabilities.

4. **Vulnerabilities directly connected to Docker or libcontainer:** Vulnerabilities encountered in Docker and libcontainer mostly impact file-system isolation: chroot escapes (CVE-2014-9357, CVE-2015-3627), path traversals (CVE-2014-6407, CVE-2014-9356, CVE-2014-9358), access to special file systems on the host (CVE-2015-3630, CVE-2016-8867, CVE-2016-8887), container escalation (CVE-2016-3697). The majority of these individual vulnerabilities

have all been addressed as of Docker version 1.6.1. CVE2016-3697 is also patched as of Docker version 1.11.0, whereas CVE2016-8867 and CVE2016-3697 are patched as of Docker version 1.12.3. Because container processes typically operate with PID 0 (i.e., root privileges), when they escape they have read and write access to the whole host file system.As a result, they can overwrite host binaries, resulting in delayed arbitrary code execution with root capabilities.

The docker-default profile included with the docker.io package grants containers full access to network devices, file systems, and a full set of capabilities while containing only a minimal range of deny directives, effectively a blacklist.

5. **Vulnerabilities in the Linux kernel:** Containers are vulnerable to kernel attacks since they run on the same kernel as the host. An exploit that grants full root privileges to a container allows an attacker to break out of the container and compromise the host (for example, triggering isolation and integrity breaches, as well as data disclosure).

Chapter 5

Proposed Work

Computer networks today face increasingly complex and numerous threats. It is normal practise to use vulnerability scanners to detect the number, type, and location of vulnerabilities that exist on our networks in order to analyse such threats. Such tools, however, analyse vulnerabilities independently and do not illustrate how they link to one another to expose combinations of them that may pose serious threats to our networks. To defend such networks, each attack path through a network must be identified and any malicious access through those paths must be blocked. To analyse a network's overall vulnerability, vulnerabilities must be grouped based on their multi-step and multi-host nature. A variety of attack graph generating algorithms have been presented, each with varying levels of maturity and usefulness. Prerequisite/postcondition models are among the most intuitive and straightforward. Prerequisites are the circumstances that must be met in order to exploit vulnerabilities. Postconditions are the effects and capabilities achieved by the attackers as a result of the exploits.

We intend to define a prerequisite and postcondition classification scheme for attacker privileges specific to Docker, in order to generate attack graphs. We distinguish between real and virtual machine privileges when defining attacker privileges. Then, using the National Vulnerability Database (NVD[1]), we provide a rule-based strategy for producing these categories. A vulnerability is a weakness in a system that can be exploited by a malicious user using the right set of tools. Many vulnerabilities are publicly known (e.g., those on the Common Vulnerabilities and Exposures (CVE[2]) list) and catalogued in databases like the National Vulnerability Database (NVD). CVE is a collection of publicly known cybersecurity vulnerabilities that includes a number, a description, and at least one public reference for each entry. The NVD repository organises this list of publicly known vulnerabilities, allowing for automated vulnerability management and security measurement.

The generation of attack graphs is significantly reliant on privileges. Mainly, privileges are organised as a hierarchy with different levels of access (User, Admin) and access scope (virtual machine VOS, host machine OS). None, VOS(User), VOS(Admin), OS(User), and OS(Admin) are the privileges used in this thesis. VOS refers to a privilege that is only granted to a virtual machine and does not affect the host machine. However, unlike hosts on a network, these privileges apply to images rather than virtual machines in our situation. The OS keyword indicates that a user with this permission has the ability to control the host machine. VOSs are at the bottom of the hierarchy

[1] https://nvd.nist.gov/
[2] https://cve.mitre.org/

since they are isolated from host machines and their exploitation does not imply exploitation of the host machine. None indicates that no privilege is granted, User indicates that only a subset of user level privileges is granted, and Admin indicates that complete management of the system is granted. To capture the semantics of the vulnerabilities, the defined rules analyze both taxonomy-based and textual data. In the following section, we provide a description of the CVSS data fields [27] used in our rules.

5.1 CVE Metrics for Rule Generation

For defining the rules in order to find the precondition and postcondition we have considered certain metrices. In the NVD for every vulnerabilities there is a Common Vulnerability Scoring System(CVSS) which is a published standard used by organizations worldwide. The Common Vulnerability Scoring System (CVSS) is a method for capturing a vulnerability's key characteristics and generating a numerical score that reflects its severity. The numerical score can then be converted to a qualitative representation (low, medium, high, and critical) to enable organisations in correctly assessing and prioritising their vulnerability management activities. For calculating these scores NVD have identified certain metrices. For our analysis we have chosen few of those metrices mainly the ones that are used for impact score calculation. The description of these metrices have been presented below.

5.1.1 Privileges Required

This metric expresses the level of privilege an attacker must possess and it takes the values of None, Low and High.

1. None: The attacker is unauthorized prior to attack, and therefore does not require any access to settings or files to carry out an attack.
2. Low: The attacker is authorized with (i.e., requires) privileges that provide basic user capabilities that could normally affect only settings and files owned by a user. Alternatively, an attacker with Low privileges may have the ability to cause an impact only to non-sensitive resources.
3. High: The attacker is authorized with (i.e., requires) privileges that provide significant (e.g., administrative) control over the vulnerable component that could affect component -wide settings and files.

5.1.2 User Interaction

This metric indicates the requirement for an user other than the attacker to actively compromise the vulnerable component successfully. This measure defines whether the vulnerability may be exploited solely by the attacker or if a separate user (or user-initiated activity) is required to participate in some way and it takes the values of None and Required.

1. None: The vulnerable system can be exploited without interaction from any user.
2. Required: Successful exploitation of this vulnerability requires a user to take some action before the vulnerability can be exploited, such as convincing a user to click a link in an email.

5.1.3 Scope

The Scope metric measures whether a vulnerability in one vulnerable component has an impact on resources in other components. A security authority is formally defined as a mechanism (e.g., an application, an operating system, firmware, or a sandbox environment) that defines and enforces access control in terms of how certain subjects/actors (e.g., human users, processes) can access certain restricted objects/resources (e.g., files, CPU, memory) in a controlled manner. A Scope change happens when a vulnerability in a vulnerable component can affect a component that is in a different security scope than the vulnerable component. A Scope change occurs when the impact of a vulnerability crosses a security/trust border and affects components outside the security scope in which the vulnerable component resides.

1. Unchanged: An exploited vulnerability can only affect resources managed by some authority. In case the vulnerable component and the impacted component are the same.

2. Changed: An exploited vulnerability can affect resources beyond the authorization privileges intended by the vulnerable component. In this case the vulnerable component and the impacted component are different.

5.1.4 Confidentiality Impact

This metric assesses the impact of a successfully exploited vulnerability on the confidentiality of the information resources maintained by a software component. Confidentiality refers to restricting information access and disclosure to just authorised users while also preventing unauthorised access and disclosure.

1. None: There is no loss of confidentiality within the impacted component.

2. Low: There is some lose of confidentiality. Access to some restricted information is obtained, but the attacker does not have control over what information is obtained, or the amount or kind of loss is constrained or the amount or kind of loss is constrained. The information disclosure does not cause a direct, series loss to the impacted component.

3. High: There is total loss of confidentiality, resulting in all resources within the impacted component being divulged to the attacker. Alternatively, access to only some restricted information is obtained, but the disclosed information presents a direct, serious impact.For example, an attacker steals the administrator's password, or private encryption keys of a web server.

5.1.5 Integrity Impact

This metric measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of information.

1. None: There is no loss of integrity within the impacted component.

2. Low: Modification of data is possible , but the attacker does not have control over the consequence of a modification, or the amount of modification is constrained. The data modification does not have a direct, serious impact on the impacted component.

3. High: There is total loss of integrity, or a complete loss of protection. For example, an attacker is able to modify any/all files protected by the impacted component. Alternatively, only some files can be modified, but malicious modification would present a direct , serious consequence to the impacted component.

5.1.6 Availability Impact

This metric assesses the impact of a successfully exploited vulnerability on the availability of the impacted component. This measure relates to the loss of availability of the impacted component itself, such as a networked service, rather than the loss of confidentiality or integrity of data (e.g., information, files) used by the impacted component (e.g., web, database, email). Attacks that use network bandwidth, CPU cycles, or disc space all affect the availability of an effected component, because availability refers to the accessibility of information resources.

1. None: There is no impact to availability within the impacted component.

2. Low: There is reduced performance or interruptions in resource availability. Even if repeated exploitation of the vulnerability is possible , the attacker does not have the ability to completely deny service to legitimate users. The resources in the impacted component are either partially available all of the time, or fully available only some of the time, but overall there is no direct, serious consequence to the impacted component.

3. High: There is total loss of availability, resulting in the attacker being able to fully deny access to resources in the impacted component ; this loss is either sustained (while the attacker continues to deliver the attack) or persistent (the condition persists even after the attack has completed). Alternatively, the attacker has the ability to deny some availability, but the loss of availability presents a direct, serious consequence to the impacted component (e.g., the attacker cannot disrupt existing connections, but can prevent new connections; the attacker can repeatedly exploit a vulnerability that, in each instance of successful attack, leaks a only small amount of memory, but after repeated exploitation causes a service to become completely unavailable.

5.1.7 Impact

The Impacts data for the three pillars of information security (Confidentiality, Integrity, and Availability) shows the damage caused to the victim and takes the values None, Partial, and Complete depending on the impact score provided by NIST.

5.1.8 Common Platform Enumeration (CPE)

Data from the Common Platform Enumeration (CPE) offers a list of vulnerable products for each known vulnerability. Operating systems, firmwares, and applications are the three types of platforms that are vulnerable.
The NVD uses natural language descriptions in addition to the taxonomy of data described above to explain the vulnerabilities. A number of single terms or sequences of phrases to be more specific we call them **vocabulary** detected during our manual study of the description are also used in our rules, and they show to be quite valuable in assessing attacker privileges.

There are more than 1 lakh vulnerabilities in NVD. For this project we have selected only the vulnerabilities with the keyword 'Docker' published in the years between 2021 and 2014. We have found 115 vulnerabilities but have excluded 15 of them the reason being impact score not yet provided or the vulnerabilities are undergoing re-analysis. So finally we have considered 100 vulnerabilities and have defined rules manually by analysing the experimental data. The rules have been presented in the below given tables.

Attack Vector	Authentication	Privilege	CPE	Pre-condition
-	None	-	NONE	NONE
Local	-	Low	Only OS	OS(user) or VOS(user)
Local	-	High	Only OS	OS(admin) or VOS(admin)
Local	! None	Low	APP or HW	APP(user)
Local	! None	High	APP or HW	APP(admin)
Local	None	Low	APP or HW	OS(user) or VOS(user)
Local	None	High	APP or HW	OS(admin) or VOS(admin)
! Local	! None	Low	Only OS	OS(user) or VOS(user)
! Local	! None	High	Only OS	OS(admin) or VOS(admin)
! Local	! None	Low	APP or HW	APP(user)
! Local	! None	High	APP or HW	APP(admin)
Vocabulary				
allow ... guest OS user , allow ... PV guest user , user on a guest operating system			-	VOS(user)
allow ... guest OS admin , allow ... PV guest admin , allow ... guest OS kernel admin			-	VOS(admin)
allows local users , allowing local users , allow local users , allows the local user			-	OS(user) or VOS(user)
allows local administrators , allow local administrators , allows the local administrator			-	OS(admin) or VOS(admin)
remote authenticated user AND ! remote authenticated users with administrative privileges			APP or HW	APP(user)
remote authenticated admin , remote authenticated users with administrative privileges			APP or HW	APP(admin)
remote authenticated users			Only OS	OS(user) or VOS(user)
remote authenticated admin			Only OS	OS(admin) or VOS(admin)

Table 5.1: Previous Rules For Producing Attacker Privilege Prerequisites as presented by [7]

Tables 5.1 and 5.1 has been referred from [7]. This table presents the rules that can be used to find out the preconditions that is the privilege level that must be met in order to exploit vulnerabilities.
Table 5.1 presents the rules that can be used to find out the post-conditions that is the privilege level achieved by the attackers as a result of the exploits.

Vocabulary	Impacts	CPE	Post-Condition
gain root /gain unrestricted, root shell access /obtain root	All Complete	-	OS(ADMIN)
gain privilege /gain host OS privilege /gain admin /obtain local admin /obtain admin /gain unauthorized access /to root /to the root /elevate the privilege /elevate privilege /root privileges via buffer overfow	All Complete	-	OS(ADMIN)
unspecified vulnerability /unspecified other impact / unspecified impact / other impacts	All Complete	-	OS(ADMIN)
	Partial	Only OS	OS(USER)
gain privilege / gain unauthorized access	Partial	Only OS	OS(USER)
gain admin / obtain admin	Partial	-	APP(ADMIN)
hijack the authentication of admin / hijack the authentication of super admin / hijack the authentication of moderator	-	-	APP(ADMIN)
hijack the authentication of users / hijack the authentication of arbitrary users / hijack the authentication of unspecified victims	-	-	APP(USER)
obtain password / obtain credential / sniff ... credentials / sniff ... passwords / steal ... credentials / steal ... passwords	All Complete	Only OS	OS(ADMIN)
	Partial	Only OS	OS(USER)
	Partial	APP/ HW	APP(ADMIN)
cleartext credential / cleartext password / obtain plaintext / obtain cleartext / discover cleartext / read network traffic / un-encrypted / unencrypted / intercept transmission / intercept communication / obtain and decrypt passwords / conduct offline password guessing / bypass authentication	All Complete	Only OS	OS(ADMIN)
	Partial	Only OS	OS(USER)
	Partial	APP/ HW	APP(ADMIN)
buffer overflow / command injection / write arbitrary, file / command execution / execute command / execute root command / execute commands as root / execute arbitrary / execute dangerous / execute php / execute script / execute local / execution of arbitrary / execution of command / remote execution / execute code & !execute arbitrary SQL	All Complete	-	OS(ADMIN)
	Partial .	-	OS(USER)
SQL injection	-	APP/HW	APP(ADMIN)
-	Any None	-	NONE

Figure 5.1: Previous Rules For Producing Attacker Privilege Postconditions as presented by [7]

The manually developed rules in the above two tables are static in the sense that they do not alter or expand in size as the amount of vulnerability data imported from the NVD continues to expand. As a result the prerequisites and the postconditions that we get using those rules results default for majority of the vulnerabilities that we have considered.Out of the 100 vulnerabilities that were collected as experimental data, for 30 vulnerabilities precondition and postcondition could be figured out using the rules presented in tables 5.1 and 5.1. But for 70 vulnerabilities the result was default So, we have developed another set of rules for those vulnerabilities where none of the rules from the above table applies.

Vocabulary	Privileges required	User interaction	Scope	Confident -iality	Integrity	Avail- ability	Impact Score	Impact	CPE	Pre- condition
Denial of service	none	none	unchanged	none	none	high	3.6	partial	App/HW	none
	low	none	unchanged	none	none	high	3.6	partial	App/HW	VOS(user)
	high	none	unchanged	none	none	high	3.6	partial	App/HW	VOS(user)
restriction bypass	none	none	unchanged	none	high	none	3.6	partial	App/HW	none
	none	none	changed	none	high	none	4	partial	App/HW	none
	none	none	unchanged	high	none	none	3.6	partial	App/HW	none
log highly sensitive information/ decrypt highly sensitive information/ obtain highly sensitive information/ access to critical data	low/none	none	unchanged	high	none	none	3.6	partial	only os	VOS(user)
	high	required	unchanged	none	high	none	3.6	partial	App/HW	VOS(user)
	high	none	unchanged	high	none	none	3.6	partial	App/HW	VOS(user)
	high	required	changed	low	low	none	2.7	partial	App/HW	VOS(user)
visit a specially crafted Web site/Image/ Certificate	high	required	unchanged	none	high	none	3.6	partial	App/HW	VOS(user)
	low	none	unchanged	none	none	high	3.6	partial	App/HW	VOS(user)
	low	none	unchanged	high	none	none	3.6	partial	App/HW	VOS(user)
	none	none	unchanged	none	high	none	3.6	partial	App/HW	none
	none	required	unchanged	none	high	none	3.6	partial	App/HW	none
	none	required	unchanged	high	high	high	5.9	partial	App/HW	VOS(user)
	high	none	unchanged	high	none	none	3.6	partial	App/HW	none
escalate or gain privileges/ impersonate privileges	high	none	unchanged	high	high	high	5.9	partial	App/HW	VOS(user)
	low	none	unchanged	high	high	high	5.9	partial	App/HW	VOS(user)
	none	required	unchanged	high	high	high	5.9	partial	App/HW	VOS(user)

Figure 5.2: New Rules For Producing Attacker Privilege Prerequisites(a)

Vocabulary	Privileges required	User interaction	Scope	Confident -iality	Integrity	Avail- ability	Impact Score	Impact	CPE	Pre- condition
write/read in the root filesystem of the host or inject new attributes/ code	low	none	changed	high	high	high	6	partial	App/HW	VOS(user)
	low	required	changed	high	high	high	6	partial	only OS	VOS(user)
	low	none	unchanged	high	high	high	5.9	partial	App/HW	VOS(user)
	low	none	unchanged	high	none	none	3.6	partial	App/HW	VOS(user)
	high	required	changed	low	low	none	2.7	partial	App/HW	VOS(user)
	none	none	unchanged	none	high	none	3.6	partial	App/HW	none
	none	none	unchanged	high	none	none	3.6	partial	App/HW	VOS(user)
	none	none	unchanged	high	high	high	5.9	partial	only OS	VOS(user)
	none	required	unchanged	high	high	high	5.9	partial	App/HW	VOS(user)
	none	required	changed	high	high	high	6	partial	App/HW	VOS(user)
Contain a blank Password	none	none	unchanged	high	high	high	5.9	partial	App/HW	none
Gain root access	low	none	unchanged	high	high	high	5.9	partial	App/HW	
	none	none	unchanged	high	high	high	5.9	partial	App/HW	none
Allow a Remote attcker/ remote un- authenticated attacker	low	none	unchanged	none	none	high	3.6	partial	App/HW	VOS(user)
	low	none	unchanged	high	high	none	5.2	partial	App/HW	
	high	required	unchanged	none	high	none	3.6	partial	App/HW	VOS(user)
	high	none	unchanged	high	none	none	3.6	partial	App/HW	none

Figure 5.3: New Rules For Producing Attacker Privilege Prerequisites(b)

Vocabulary	Privileges required	User interaction	Scope	Confidentiality	Integrity	Availability	Impact Score	Impact	CPE	Post-condition
Denial of service	none	none	unchanged	none	none	high	3.6	partial	App/FW	VOS(user)
	low	none	unchanged	none	none	high	3.6	partial	App/FW	VOS(admin)/ OS (user/admin)
	high	none	unchanged	none	none	high	3.6	partial	App/FW	VOS(admin)
restriction bypass	none	none	unchanged	none	high	none	3.6	partial	App/FW	VOS (user/admin)
	none	none	changed	none	high	none	4	partial	App/FW	VOS/OS(admin)
	none	none	unchanged	high	none	none	3.6	partial	App/FW	VOS(user)
log highly sensitive information/ decrypt highly sensitive information/ obtain highly sensitive information/ access to critical data	high	required	unchanged	none	high	none	3.6	partial	App/FW	VOS/OS(admin)
	low/none	none	unchanged	high	none	none	3.6	partial	only os	VOS ADMIN
	high	none	unchanged	high	none	none	3.6	partial	App/FW	VOS/OS(admin)
	high	required	changed	low	low	none	2.7	partial	App/FW	VOS/OS(admin)
visit a specially crafted Web site/Image/Certificate	high	required	unchanged	none	high	none	3.6	partial	App/FW	VOS(admin)
	low	none	unchanged	none	none	high	3.6	partial	App/FW	VOS(admin)/ OS (user/admin)
	low	none	unchanged	high	none	none	3.6	partial	App/FW	VOS(admin)
	none	none	unchanged	none	high	none	3.6	partial	App/FW	VOS (user/admin)
	none	required	unchanged	none	high	none	3.6	partial	App/FW	VOS (user/admin)
	none	required	unchanged	high	high	high	5.9	partial	App/FW	VOS(admin)
	high	none	unchanged	high	none	none	3.6	partial	App/FW	VOS(user)/ OS (user)
escalate or gain privileges/ impersonate privileges	high	none	unchanged	high	high	high	5.9	partial	App/FW	VOS(admin)
	low	none	unchanged	high	high	high	5.9	partial	App/FW	VOS (admin/user)
	none	required	unchanged	high	high	high	5.9	partial	App/FW	VOS(admin)

Figure 5.4: New Rules For Producing Attacker Privilege Postconditions(a)

Vocabulary	Privileges required	User interaction	Scope	Confidentiality	Integrity	Availability	Impact Score	Impact	CPE	Post-condition
write/read in the root filesystem of the host or inject new attributes/code	low	none	changed	high	high	high	6	partial	App/FW	VOS(admin)/ OS (admin)
	low	required	changed	high	high	high	6	partial	only OS	VOS(admin)
	low	none	unchanged	high	high	high	5.9	partial	App/FW	OS(admin)/ VOS (Admin)
	low	none	unchanged	high	none	none	3.6	partial	App/FW	VOS(admin)
	high	required	changed	low	low	none	2.7	partial	App/FW	VOS(admin)
	none	none	unchanged	none	high	none	3.6	partial	App/FW	VOS (user/admin)
	none	none	unchanged	high	none	none	3.6	partial	App/FW	VOS(admin)
	none	none	unchanged	high	high	high	5.9	partial	only OS	VOS(admin)/ OS (user)
	none	required	unchanged	high	high	high	5.9	partial	App/FW	VOS(admin)
	none	required	changed	high	high	high	6	partial	App/FW	VOS(admin)/ OS (admin)
Contain a blank Password	none	none	unchanged	high	high	high	5.9	partial	App/FW	VOS/OS(admin)
Gain root access	low	none	unchanged	high	high	high	5.9	partial	App/FW	VOS/OS(admin)
	none	none	unchanged	high	high	high	5.9	partial	App/FW	VOS/OS(admin)
Allow a Remote attcker/ remote unauthenticated attacker	low	none	unchanged	none	none	high	3.6	partial	App/FW	VOS(admin)/ OS (user/admin)
	low	none	unchanged	high	high	none	5.2	partial	App/FW	VOS (admin/user)
	high	required	unchanged	none	high	none	3.6	partial	App/FW	VOS(admin)
	high	none	unchanged	high	none	none	3.6	partial	App/FW	VOS(user)/ OS (user)

Figure 5.5: New Rules For Producing Attacker Privilege Postconditions(b)

We can see in Table. 5.4 and 5.5 that for many vocabulary multiple rules apply so we decided to keep only the privilege that is highest in level. In the same manner When multiple rules apply to the same vulnerability, the rule with the highest privilege level takes precedence.And the revised table has been presented in Table 5.6 and 5.7.

Vocabulary	Privileges required	User interaction	Scope	Confident-iality	Integrity	Availability	Impact Score	Impact	CPE	Post-condition
Denial of service	none	none	unchanged	none	none	high	3.6	partial	App/FW	VOS(user)
	low	none	unchanged	none	none	high	3.6	partial	App/FW	OS(admin)
	high	none	unchanged	none	none	high	3.6	partial	App/FW	VOS(admin)
restriction bypass	none	none	unchanged	none	high	none	3.6	partial	App/FW	VOS(admin)
	none	none	changed	none	high	none	4	partial	App/FW	OS(admin)
	none	none	unchanged	high	none	none	3.6	partial	App/FW	VOS(user)
log highly sensitive information/ decrypt highly sensitive information/ obtain highly sensitive information/ access to critical data	high	required	unchanged	none	high	none	3.6	partial	App/FW	OS(admin)
	low/none	none	unchanged	high	none	none	3.6	partial	only os	VOS(admin)
	high	none	unchanged	high	none	none	3.6	partial	App/FW	OS(admin)
	high	required	changed	low	low	none	2.7	partial	App/FW	OS(admin)
visit a specially crafted Web site/Image/ Certificate	high	required	unchanged	none	high	none	3.6	partial	App/FW	VOS(admin)
	low	none	unchanged	none	none	high	3.6	partial	App/FW	OS(admin)
	low	none	unchanged	high	none	none	3.6	partial	App/FW	VOS(admin)
	none	none	unchanged	none	high	none	3.6	partial	App/FW	VOS(admin)
	none	required	unchanged	none	high	none	3.6	partial	App/FW	VOS(admin)
	none	required	unchanged	high	high	high	5.9	partial	App/FW	VOS(admin)
	high	none	unchanged	high	none	none	3.6	partial	App/FW	OS(user)
escalate or gain privileges/ impersonate privileges	high	none	unchanged	high	high	high	5.9	partial	App/FW	VOS(admin)
	low	none	unchanged	high	high	high	5.9	partial	App/FW	VOS(admin)
	none	required	unchanged	high	high	high	5.9	partial	App/FW	VOS(admin)

Figure 5.6: Revised New Rules For Producing Attacker Privilege Postconditions(a)

Vocabulary	Privileges required	User interaction	Scope	Confidentiality	Integrity	Availability	Impact Score	Impact	CPE	Post-condition
write/read in the root filesystem of the host or inject new attributes/ code	low	none	changed	high	high	high	6	partial	App/FW	OS(admin)
	low	required	changed	high	high	high	6	partial	only OS	VOS(admin)
	low	none	unchanged	high	high	high	5.9	partial	App/FW	OS(admin)
	low	none	unchanged	high	none	none	3.6	partial	App/FW	VOS(admin)
	high	required	changed	low	low	none	2.7	partial	App/FW	VOS(admin)
	none	none	unchanged	none	high	none	3.6	partial	App/FW	VOS(admin)
	none	none	unchanged	high	none	none	3.6	partial	App/FW	VOS(admin)
	none	none	unchanged	high	high	high	5.9	partial	only OS	OS(user)
	none	required	unchanged	high	high	high	5.9	partial	App/FW	VOS(admin)
	none	required	changed	high	high	high	6	partial	App/FW	OS(admin)
Contain a blank Password	none	none	unchanged	high	high	high	5.9	partial	App/FW	OS(admin)
Gain root access	low	none	unchanged	high	high	high	5.9	partial	App/FW	OS(admin)
	none	none	unchanged	high	high	high	5.9	partial	App/FW	OS(admin)
Allow a Remote attcker/ remote un-authenticated attacker	low	none	unchanged	none	none	high	3.6	partial	App/FW	OS(admin)
	low	none	unchanged	high	high	none	5.2	partial	App/FW	VOS(admin)
	high	required	unchanged	none	high	none	3.6	partial	App/FW	VOS(admin)
	high	none	unchanged	high	none	none	3.6	partial	App/FW	OS(user)

Figure 5.7: Revised New Rules For Producing Attacker Privilege Postconditions(b)

In terms of rule application, Logical AND is utilised for reasoning, such that the privilege level of a given rule is assigned to that vulnerability as a prerequisite or postcondition only if all of the fields of that rule satisfy for that vulnerability. If none of the rules apply to a given vulnerability, the privilege level is set to default.

There have been several additions to the listings of vulnerabilities in NVD specific to Docker after we collected data for our analysis. So, we have checked for few of those newly added vulnerabilities with our set of rules to figure out the preconditions and postconditions and the result have been presented in the Table 5.2

CVE-Id	Description	Pre-Condition	Post-Condition
CVE-2021-34079	OS Command injection vulnerability in Mintzo Docker-Tester through 1.2.1 allows attackers to execute arbitrary commands via shell metacharacters in the 'ports' entry of a crafted docker-compose.yml file.	VOS(user)	VOS(admin)/ OS(user)
CVE-2022-31245	mailcow before 2022-05d allows a remote authenticated user to inject OS commands and escalate privileges to domain admin via the –debug option in conjunction with the —PIPEMESS option in Sync Jobs.	VOS(user)	OS(admin)
CVE-2021-46440	Storing passwords in a recoverable format in the DOCUMENTATION plugin component of Strapi before 3.6.9 and 4.x before 4.1.5 allows an attacker to access a victim's HTTP request, get the victim's cookie, perform a base64 decode on the victim's cookie, and obtain a cleartext password, leading to getting API documentation for further API attacks.	None	App(admin)
CVE-2022-24799	wire-webapp is the web application interface for the wire messaging service. Insufficient escaping in markdown "code highlighting" in the wire-webapp resulted in the possibility of injecting and executing arbitrary HTML code and thus also JavaScript. If a user receives and views such a malicious message, arbitrary code is injected and executed in the context of the victim. This allows the attacker to fully control the user account. Wire-desktop clients that are connected to a vulnerable wire-webapp version are also vulnerable to this attack.	None	OS(user)
CVE-2022-26659	Docker Desktop installer on Windows in versions before 4.6.0 allows an attacker to overwrite any administrator writable files by creating a symlink in place of where the installer writes its log file. Starting from version 4.6.0, the Docker Desktop installer, when run elevated, will write its log files to a location not writable by non-administrator users.	None	OS(admin)
CVE-2021-45414	A Remote Code Execution (RCE) vulnerability exists in DataRobot through 2021-10-28 because it allows submission of a Docker environment or Java driver.	None	OS(admin)
CVE-2022-23774	Docker Desktop before 4.4.4 on Windows allows attackers to move arbitrary files.	None	OS(user)
CVE-2022-20617	Jenkins Docker Commons Plugin 1.17 and earlier does not sanitize the name of an image or a tag, resulting in an OS command execution vulnerability exploitable by attackers with Item/Configure permission or able to control the contents of a previously configured job's SCM repository.	VOS(user)	OS(admin)

Table 5.2: Application of the rules previously given

Chapter 6

Conclusion and Future Work

6.1 Conclusion

First, we reviewed the different types of virtualization techniques, which including Virtual Machines, Containers, and Unikernels, and compared them in terms of deployment time, booting time, security, and other factors. From past research works, we conducted a detailed analysis of the container virtualization technology, focusing primarily on the Docker ecosystem, its security, vulnerabilities, and use cases. We conducted a thorough survey into the vulnerabilities and security risks specific to Docker reported in the National Vulnerability Database and listed there (NVD) and defined a prerequisite and postcondition classification scheme for attacker privileges that can be used for attack graph generation.

6.2 Future Work

Every year several vulnerabilities are reported and listed in the NVD. As a result the rules that we generated which are static might not always recognise preconditions/postconditions for newly reported vulnerabilities.Thus in future we can think of an automated method for recognising preconditions/postconditions. Also in future we can use the result of our survey to generate attack graphs in order to secure networks.

Bibliography

[1] A. U. Rehman, R. L. Aguiar, and J. P. Barraca, "Network functions virtualization: The long road to commercial deployments," *IEEE Access*, vol. 7, pp. 60439–60464, 2019.

[2] F. Sierra-Arriaga, R. Branco, and B. Lee, "Security issues and challenges for virtualization technologies," *ACM Computing Surveys*, vol. 53, pp. 1–37, 05 2020.

[3] M. Souppaya, J. Morello, and K. Scarfone, "Application container security guide," tech. rep., National Institute of Standards and Technology, 2017.

[4] L. Rice, *Container Security: Fundamental Technology Concepts That Protect Containerized Applications*. O'Reilly Media, Incorporated, 2020.

[5] J. Talbot, P. Pikula, C. Sweetmore, S. Rowe, H. Hindy, C. Tachtatzis, R. Atkinson, and X. Bellekens, "A security perspective on unikernels," in *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, pp. 1–7, 2020.

[6] T. Combe, A. Martin, and R. Pietro, "To docker or not to docker: A security perspective," *IEEE Cloud Computing*, vol. 3, pp. 54–62, 09 2016.

[7] M. Aksu, K. Bicakci, M. H. Dilek, M. Ozbayoglu, and E. Tatlı, "Automated generation of attack graphs using nvd," pp. 135–142, 03 2018.

[8] A. Wong, E. Chekole, M. Ochoa, and J. Zhou, "Threat modeling and security analysis of containers: A survey," 11 2021.

[9] X. Lin, L. Lei, Y. Wang, J. Jing, K. Sun, and Q. Zhou, "A measurement study on linux container security: Attacks and countermeasures," in *Proceedings of the 34th Annual Computer Security Applications Conference*, ACSAC '18, (New York, NY, USA), p. 418–429, Association for Computing Machinery, 2018.

[10] A. Duarte and N. Antunes, "An empirical study of docker vulnerabilities and of static code analysis applicability," pp. 27–36, 10 2018.

[11] T. Kurek, "Unikernel network functions: A journey beyond the containers," *IEEE Communications Magazine*, vol. 57, no. 12, pp. 15–19, 2019.

[12] T. Goethals, M. Sebrechts, A. Atrey, B. Volckaert, and F. De Turck, "Unikernels vs containers: An in-depth benchmarking study in the context of microservice applications," in *2018 IEEE 8th International Symposium on Cloud and Service Computing (SC2)*, pp. 1–8, 2018.

[13] K. Ingols and K. Piwowarski, "Practical attack graph generation for network defense," in *In Proc. 22nd IEEE Annual Computer Security Applications Conference (ACSAC*, pp. 121–130, 2006.

[14] A. Singhal and X. Ou, "Security risk analysis of enterprise networks using probabilistic attack graphs," *Quantitative Security Risk Assessment of Enterprise Networks*, 01 2012.

[15] A. Ibrahim, S. Bozhinoski, and A. Pretschner, "Attack graph generation for microservice architecture," pp. 1235–1242, 04 2019.

[16] M. Inokuchi, Y. Ohta, S. Kinoshita, T. Yagyu, O. Stan, R. Bitton, Y. Elovici, and A. Shabtai, "Design procedure of knowledge base for practical attack graph generation," pp. 594–601, 07 2019.

[17] G. Stergiopoulos, P. Dedousis, and D. Gritzalis, "Automatic analysis of attack graphs for risk mitigation and prioritization on large-scale and complex networks in industry 4.0," *International Journal of Information Security*, vol. 21, pp. 1–23, 02 2022.

[18] S. Bardhan and A. Battou, "Security metric for networks with intrusion detection systems having time latency using attack graphs," in *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, pp. 1107–1113, 2021.

[19] D. Hyde, "A survey on the security of virtual machines," 01 2009.

[20] R. Anand, S. Sarswathi, and R. Regan, "Security issues in virtualization environment," in *2012 International Conference on Radar, Communication and Computing (ICRCC)*, pp. 254–256, 2012.

[21] L. Chen, M. Xian, J. Liu, and H. Wang, "Research on virtualization security in cloud computing," *IOP Conference Series: Materials Science and Engineering*, vol. 806, p. 012027, 05 2020.

[22] V. Kumar and R. S. Rathore, "Security issues with virtualization in cloud computing," in *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, pp. 487–491, 2018.

[23] S. Luo, Z. Lin, X. Chen, Z. Yang, and J. Chen, "Virtualization security for cloud computing service," in *2011 International Conference on Cloud and Service Computing*, pp. 174–179, 2011.

[24] C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi, "Cloud container technologies: A state-of-the-art review," *IEEE Transactions on Cloud Computing*, vol. 7, no. 3, pp. 677–692, 2019.

[25] S. Sultan, I. Ahmad, and T. Dimitriou, "Container security: Issues, challenges, and the road ahead," *IEEE Access*, vol. 7, pp. 52976–52996, 2019.

[26] A. Martin, S. Raponi, T. Combe, and R. Di Pietro, "Docker ecosystem – vulnerability analysis," *Computer Communications*, vol. 122, pp. 30–43, 2018.

[27] "Cvss specification." `https://www.first.org/cvss/v3.1/specification-document`.

[28] N. Technology, *Guide to Data-Centric System Threat Modeling: NiST SP 800-154*. CreateSpace Independent Publishing Platform, 2016.

[29] "Container security user guide," tech. rep., Qualys,Inc., September 22,2021.

Appendix

Here we describe the steps that we followed for building a set of rules based on the experimental data that would produce preconditions and postconditions for Docker vulnerabilities.

Denial of Service											
CVE-id	Privileges required	User interaction	Scope	Confidentiality	Integrity	Availability	Impact Score	Impact	CPE	Pre-condition	Post- condition
CVE-2020-26213	none	none	unchanged	none	none	high	3.6	partial	App/FW	None	vos(user)
CVE-2017-11468	none	none	unchanged	none	none	high	3.6	partial	App/FW	None	vos(user)
CVE-2017-14992	low	none	unchanged	none	none	high	3.6	partial	App/FW	vos(user)	Vos(admin)/ os (user/admin)
CVE-2018-20699	high	none	unchanged	none	none	high	3.6	partial	App/FW	vos(user)	Vos(admin)

restriction bypass										
	Privileges required	User interaction	Scope	Confidentiality	Integrity	Availability	Impact Score	Impact	CPE	Pre-condition	Post- Condition
CVE-2019-16884	none	none	unchanged	none	high	none	3.6	partial	App/HW	None	vos(user/admin)
CVE-2014-8179	none	none	unchanged	none	high	none	3.6	partial	App/HW	None	vos(user/admin)
CVE-2014-9356	none	none	changed	none	high	none	4	partial	App/HW	None	vos(user/admin)
CVE-2016-8867	none	none	unchanged	high	none	none	3.6	partial	App/HW	Vos(User)	vos(admin)

log/decrypt/obtain highly sensitive information/access to critical data											
	Privileges required	User interaction	Scope	Confidentiality	Integrity	Availability	Impact Score	Impact	CPE	PreCondition	Post-Condition
CVE-2021-20497	none	none	unchanged	high	none	none	3.6	partial	App/HW	NONE	VOS USER/ ADMIN
CVE-2018-3213	none	none	unchanged	high	none	none	3.6	partial	App/HW	none	VOS USER
CVE-2021-45449	low	none	unchanged	high	none	none	3.6	partial	only os	VOS USER	VOS ADMIN
CVE-2021-20534	high	required	unchanged	none	high	none	3.6	partial	App/HW	vos(user)	Vos(admin)
CVE-2021-20500	high	none	unchanged	high	none	none	3.6	partial	App/HW	Vos(User)	vos(admin)
CVE-2021-20524	high	required	changed	low	low	none	2.7	partial	App/HW	Vos(User)	vos(admin)

Figure 1: Data collected from NVD for CVEs having the vocabulary "Denial of Service, restriction bypass and logsensitive information" in their description

For every vulnerabilities that we have found in NVD specific to the keyword to Docker we assembled CVE metrices namely privileges required, user interaction, scope, confidentiality, integrity, availability, impact score. We then selected certain vocabulary/vocabularies from the description of each and every vulnerabilities we have chosen.

visit a specially crafted Web site/Image/Certificate											
	Privileges required	User interaction	Scope	Confidentiality	Integrity	Availability	Impact Score	Impact	CPE	PreCondition	Post-Condition
CVE-2019-10340	none	required	unchanged	high	high	high	5.9	partial	App/HW	Vos(User)	vos(admin)
CVE-2014-8179	none	none	unchanged	none	high	none	3.6	partial	App/HW	None	vos(user/ admin)
CVE-2014-8178	none	required	unchanged	none	high	none	3.6	partial	App/HW	None	vos(user/ admin)
CVE-2017-14992	low	none	unchanged	none	none	high	3.6	partial	App/HW	vos(user)	Vos(admin)/ os (user/admin)
CVE-2019-10341	low	none	unchanged	high	none	none	3.6	partial	App/HW	Vos(User)	vos(admin)
CVE-2021-20511	high	none	unchanged	high	none	none	3.6	partial	App/HW	None	vos(user)/ os (user)
CVE-2021-20534	high	required	unchanged	none	high	none	3.6	partial	App/HW	vos(user)	Vos(admin)

escalate or gain privileges/ impersonate privileges											
	Privileges required	User interaction	Scope	Confidentiality	Integrity	Availability	Impact Score	Impact	CPE	PreCondition	Post-Condition
CVE-2019-15752	none	required	unchanged	high	high	high	5.9	partial	App/HW	Vos(User)	vos(admin)
CVE-2021-35497	low	none	unchanged	high	high	high	5.9	partial	App/HW	Vos(User)	vos(admin)
CVE-2021-3162	low	none	unchanged	high	high	high	5.9	partial	App/HW	None	vos(admin/ user)
CVE-2020-11492	low	none	unchanged	high	high	high	5.9	partial	App/HW	Vos(User)	Vos(admin)
CVE-2016-3697	low	none	unchanged	high	high	high	5.9	partial	App/HW	Vos(User)	vos(admin)
CVE-2020-8907	low	none	unchanged	high	high	high	5.9	partial	App/HW	Vos(User)	os(admin)/ Vos (Admin)
CVE-2021-20500	high	none	unchanged	high	none	none	3.6	partial	App/HW	Vos(User)	vos(admin)
CVE-2020-10665	high	none	unchanged	high	high	high	5.9	partial	App/HW	VOS USER	VOS ADMIN

Figure 2: Data collected from NVD for CVEs having the vocabulary "visit a specially crafted image,gain privileges" in their description

write/read in the root filesystem of the host or inject new attributes/code											
	Privileges required	User interaction	Scope	Confidentiality	Integrity	Availability	Impact Score	Impact	CPE	PreCondition	Post-Condition
CVE-2014-8179	none	none	unchanged	none	high	none	3.6	partial	App/HW	None	vos(user/ admin)
CVE-2016-8867	none	none	unchanged	high	none	none	3.6	partial	App/HW	Vos(User)	vos(admin)
CVE-2016-0761	none	none	unchanged	high	high	high	5.9	partial	only OS	Vos(User)	vos(admin)/ os (user)
CVE-2019-10340	none	required	unchanged	high	high	high	5.9	partial	App/HW	Vos(User)	vos(admin)
CVE-2019-5736	none	required	changed	high	high	high	6	partial	App/HW	Vos(User)	Vos(admin)/ os (admin)
CVE-2020-26278	low	none	changed	high	high	high	6	partial	App/HW	Vos(User)	Vos(admin)/ os (admin)
CVE-2018-15664	low	required	changed	high	high	high	6	partial	only OS	Vos(User)	Vos(admin)
CVE-2020-8907	low	none	unchanged	high	high	high	5.9	partial	App/HW	Vos(User)	os(admin)/ Vos (Admin)
CVE-2019-10341	low	none	unchanged	high	none	none	3.6	partial	App/HW	Vos(User)	vos(admin)
CVE-2021-20524	high	required	changed	low	low	none	2.7	partial	App/HW	Vos(User)	vos(admin)

Figure 3: Data collected from NVD for CVEs having the vocabulary "read/write arbitrary code" in their description

After we have organised our data we create separate table as shown in Figures 1, 2, 3, 4, 5, 6, 7 for every vocabulary.Then we analyse each of the vulnerabilities to recognize the precondition and the postcondition.In this way we noticed that many of the vulnerabilities were redundant i.e., present in more than one table. So, while making the rules as desribed in Figures 5.2, 5.3, 5.4,5.5 we chose the precondition/postcondition that has the highest privilege.Regarding the implementation of the rules, logical AND is to be performed for reasoning, so that the privilege level of the rule is only assigned to a vulnerability as a prerequisite or postcondition if all of its fields are satisfied for that vulnerability. The rule with the greatest privilege level takes precedence when many rules are applicable to a certain vulnerability.

Contain a blank Password											
CVE-id	Privileges required	User interaction	Scope	Confidentiality	Integrity	Availability	Impact Score	Impact	CPE	Pre-Condition	Post- Condition
CVE-2020-35197	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35196	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35195	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35192	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35191	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35190	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35186	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35189	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35187	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35185	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35184	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35469	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35468	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35465	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35466	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35467	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35462	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35463	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35464	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-29591	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-29602	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-29601	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-29581	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-29580	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-29576	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-29577	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-29579	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-29564	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-29578	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-29389	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)

Figure 4: Data collected from NVD for CVEs having the vocabulary "contain a blank password" in their description

Gain root access											
CVE-id	Privileges required	User interaction	Scope	Confidentiality	Integrity	Availability	Impact Score	Impact	CPE	Pre-Condition	Post-Condition
CVE-2020-35197	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35196	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35195	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35192	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35191	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35190	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35186	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35189	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35187	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35185	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35184	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35469	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35468	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35465	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35466	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35467	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35462	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35463	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-35464	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-29591	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-29602	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-29601	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-29581	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-29580	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-29576	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-29577	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-29579	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-29564	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-29578	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2020-29389	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS(admin)
CVE-2018-9862	low	none	unchanged	high	high	high	5.9	partial	App/HW	Vos(user)	Vos(Admin)

Figure 5: Data collected from NVD for CVEs having the vocabulary "gain root access" in their description

Allow a Remote attcker/ remote unauthenticated attacker											
CVE-id	Privileges required	User interaction	Scope	Confidentiality	Integrity	Availability	Impact Score	Impact	CPE	pre condition	post condition
CVE-2021-20534	high	required	unchanged	none	high	none	3.6	partial	App/HW	vos(user)	Vos(admin)
CVE-2021-20511	high	none	unchanged	high	none	none	3.6	partial	App/HW	None	vos(user)/ os (user)
CVE-2017-14992	low	none	unchanged	none	none	high	3.6	partial	App/HW	vos(user)	Vos(admin)/ os (user/admin)
CVE-2014-5282	low	none	unchanged	high	high	none	5.2	partial	App/HW	None	Vos(admin/ user)
CVE-2017-11468	none	none	unchanged	none	none	high	3.6	partial	App/HW	None	vos(user)
CVE-2014-9356	none	none	changed	none	high	none	4	partial	App/HW	None	vos(user/ admin)
CVE-2020-35197	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-35196	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-35195	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-35192	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-35191	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-35190	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-35186	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-35189	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-35187	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-35185	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-35184	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-35469	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)

Figure 6: Data collected from NVD for CVEs having the vocabulary "allow remote attacker" in their description(a)

Allow a Remote attcker/ remote unauthenticated attacker											
CVE-id	Privileges required	User interaction	Scope	Confidentiality	Integrity	Availability	Impact Score	Impact	CPE	pre condition	post condition
CVE-2020-35468	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-35465	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-35466	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-35467	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-35462	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-35463	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-35464	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-29591	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-29602	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-29601	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-29581	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-29580	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-29576	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-29577	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-29579	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-29564	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-29578	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2020-29389	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)
CVE-2016-9223	none	none	unchanged	high	high	high	5.9	partial	App/HW	none	VOS/ OS (admin)

Figure 7: Data collected from NVD for CVEs having the vocabulary "allow remote attacker" in their description(b)