On the Security of Containers: Threat Modeling, Attack Analysis, and Mitigation Strategies

Ann Yi Wong[a], Eyasu Getahun Chekole[a], Martín Ochoa[b], Jianying Zhou[a]

[a] *Singapore University of Technology and Design, Singapore, Singapore*
[b] *Department of Computer Science, ETH Zurich, Zürich, Switzerland*

Abstract

Traditionally, applications that are used in large and small enterprises were deployed on "bare metal" servers installed with operating systems. Recently, the use of multiple virtual machines (VMs) on the same physical server was adopted due to cost reduction and flexibility. Nowadays, containers have become popular for application deployment due to smaller footprints than the VMs, their ability to start and stop more quickly, and their capability to pack the application binaries and their dependencies/libraries in standalone units for seamless portability. A typical container ecosystem includes a code repository (e.g., GitHub) where the container images are built from the codes and libraries and then pushed to the image registry (e.g., Docker Hub) for subsequent deployment as application containers. However, the pervasive use of containers also leads to a wide-range of security breaches such as attackers stealing credentials, source codes and sensitive data from image registry and code repository, carrying out DoS attacks on application containers, and gaining root access to misuse the underlying host resources, among others. In this paper, we first perform threat modeling on the containers ecosystem using the popular threat modeling framework, called STRIDE. Using STRIDE, we identify the vulnerabilities in each system component, and investigate potential security threats and their consequences. Then, we conduct a comprehensive survey on the existing countermeasures designed against the identified threats and vulnerabilities in containers. In particular, we assess the strengths and weaknesses of the existing mitigation strategies designed against such threats. We believe that this work will help researchers and practitioners to gain a deeper understanding of the threat landscape in containers and the state-of-the-art countermeasures. We also discuss open research problems, the research gaps and future research directions in containers security, which may ignite further research to be done in this area.

Keywords: Containers, Containerization, Containers Security, Docker, Threat Modeling, STRIDE Framework

1. Introduction

Many enterprises have started to deploy applications in containers. Some popular examples are Gmail, YouTube, Google Search [1], Netflix [2], and PayPal financial services [3], among others. Running an application in a container allows its binaries, libraries, and other dependencies to be abstracted from the operating environment and hence be portable from a developer notebook to the on-prem data centre and the public cloud. Therefore, containerization allows an application to be deployed efficiently and scaled easily. Gartner, a leading research and advisory company in information technology and cybersecurity forecasts that 15% of all applications will be running in containers by 2024, up from 5% in 2020 [4]. Gartner also forecasts that 75% of large enterprises globally will deploy production application in containers by 2022, up from less than 30% in 2020 [4]. The most widely employed container runtime is Docker at 79% share of the market [5].

Although containers are revolutionizing enterprises and other systems, they also have several weaknesses and vulnerabilities that expose them to a wide-range of cyberattacks. A recent report [6] revealed that about 51% of around 4 million images in Docker Hub have exploitable vulnerabilities of which 0.16% or 6,432 images had malicious software which were primarily cryptocurrency miner. The attackers could insert malicious images directly on misconfigured hosts [6], [7] or into Docker Hub due to the ease of pushing and pulling images to and from it without controls [6]. In another report [8], a cybersecurity team discovered through its regular monitoring that by the end of 2019, a hacker group scanned more than 59,000 IP networks on a large scale to find exposed Docker API endpoints. Most containers are also configured with default network settings, making it easy to establish remote connections. This was discovered by TeamTNT (a cybercrime group) and used it as a backdoor to run crypto-mining malware on the underlying system to generate cryptocurrencies [9]. As of the date of this paper, there are 516 container related security vulnerabilities listed in MITRE CVE [10].

Several real-world cyberattacks have also been reported on containers. In 2018, attackers hacked into Tesla's container orchestration console of Kubernetes and installed crypto-mining software to mine cryptocurrency using its cloud computing resources [11]. Consequently, the U.S. government National Security Agency (NSA) also alerted industries over a foreign-based cybercrime group APT28's massive attacks on containers that run in Kubernetes clusters [12]. In 2019, other attackers hacked into Docker Hub and gained access to usernames and passwords of 190,000 user accounts [13]. An attacker can then use the compromised Docker instance as a backdoor to spin the container, which will install the XMRig cryptocurrency miner for illegal mining [8]. There were also many other critical attacks that had been launched on containers and their subsystems [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [20], [26], [27]. These and other real-world examples show how security is a critical concern in container systems, beyond the conventional IT systems.

To alleviate the security concerns, several research works have been done on the security of containers, some focusing on vulnerability analysis [28], [29], [30], [31], [32], [33], [34], [35], and others on mitigation strategies [36], [37], [38], [39], [40], [41], [42], [43], [44], [45], [46], [20],[47], [48], [49]. However, most of the related works only focus on a specific vulnerability, threat, use-case or subsystem of containers. Hence, they do not provide a comprehensive security analysis on the entire container ecosystem (spanning image creation to distribution processes). In addition, most of the existing mitigation strategies already have certain flaws and limitations. For example, recent studies revealed that the existing Linux-based mitigation strategies used in containers, such as cgroups, namespaces and capabilities, are subjected to attacks resulting in resources exploitations and denials of services [26], [20]. Furthermore, some are probably outdated and may not reflect the latest threat landscape as shown in the example of [38], which suggests that the Docker container is fairly secure with the default configuration but it is in fact exploitable in today's context [50]. Therefore, the existing works might not provide a comprehensive security analysis and state-of-the-art information on the security landscape of the containers ecosystem.

In this work, we make a systematic and comprehensive survey on the security of containers, covering vulnerabilities, threats, threat consequences and existing mitigation strategies, to provide a comprehensive and state-of-the-art information on the security landscape of containers. To be able to specify the scope of our survey and map existing literature, we first perform threat modeling on the containers ecosystem. In particular, we study the threat landscape of the containers supply chain process – spanning code repository to image registry and then deployment processes – using the STRIDE [51] (Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege) threat modeling framework. We choose STRIDE as it is one of the most mature threat modeling frameworks, which has also been widely used in the Microsoft Secure Development Lifecycle [52, 53]. STRIDE has also been successfully adopted by several research works [54], [55], [56], [57]. Using STRIDE, we first design a data flow diagram (DFD) of the container system to map its components and their relationship via the flow of data. We then conduct a wide-range of security analysis on each component to discover the vulnerabilities, the associated threat actions and the resulting consequences.

After completing our threat modeling, we then conduct a comprehensive survey on the vulnerabilities and security threats identified. In particular, we analyze and discuss the effectiveness and limitations of existing mitigation strategies designed against the vulnerabilities and threats identified through our threat modeling. Furthermore, we highlight open security problems and future research directions in containers security, which may motivate the community to carry out further research in this area.

In sum, we believe that this work would provide a comprehensive and state-of-the-art information to researchers and practitioners on the latest security landscape of container systems. This can help the community to better understand the latest security issues in containers and the available mitigation strategies to counter them.

Organization: The rest of the paper is structured as follows: Section 2 provides relevant background information on containers, STRIDE framework, and related works on containers security. Section 3 discusses our threat modeling of the container ecosystem using the STRIDE framework. Section 4 investigates the existing mitigation solutions and analyses their limitations. In Section 5, we summarize the results of our survey, and highlight future research directions. Finally, Section 6 concludes our paper.

2. Background

2.1. Overview of Containers

A container is an independent, self-sufficient package for running an application or service. It includes the application binaries, the software libraries or dependencies, and the hardware requirements needed to run it, all combined into a self-contained unit. The key capabilities which enable a container to perform its function securely and efficiently (i.e., without resource constraints) are namespaces and control groups (cgroups). Namespaces provide process isolation and enable multiple application processes in containers to share a single host instance. On the other hand, cgroups allocate the host resources, such as CPU and memory, among the processes [58].

Containers are receiving high popularity and being widely adopted by various enterprises. This is mainly because of the following reasons: (a) a container is more lightweight than a virtual machine and therefore starts and stops much faster; (b) a container is portable as it includes the application and all its dependencies, libraries and binaries packaged into a runtime environment, therefore allowing it to run anywhere from a desktop to a datacentre; (c) a containerized application is scalable and can easily add or reduce the number of containers to meet varying demands.

The industry's main use of containers are often tied to microservices and the cloud. Containerization supports the microservices architecture very well [59]. Microservices structure an application into a set of loosely coupled software services that run in containers [60]. The entire container platform and the microservice architecture are typically deployed in the cloud infrastructure as it is scalable and resilient. IBM forecasts that within the next two years, 59% of all enterprise applications will be developed with microservices [61], further spurring the growth of container usage. There are many enterprise-level implementations of microservices on containers, and some prominent examples are Amazon, Netflix, The Guardian, Twitter, PayPal, Tencent, Baidu, Taobao, etc. [60].

2.2. Overview of STRIDE

Threat modeling is a process of identifying and evaluating threats and vulnerabilities in a particular system [62]. There are several threat modeling frameworks and methodologies designed for this purpose, including STRIDE [51], DREAD [63], PASTA [64], VAST [65], Trike [66], OCTAVE [67], and NIST [68], among others. In this paper, we use the STRIDE framework as it is amongst the most mature and popular threat modeling frameworks that has been widely used since 1999. It is also adopted by large tech companies, such as Microsoft [52, 53]. Moreover, its DFD-based approach suits well for the container environment as it allows to analyze the threat landscape of containers along its supply chain process.

STRIDE is developed by Microsoft to be used by its developers during the software development life cycle. More specifically, it is used to identify and analyze vulnerabilities and threats with respect to the authentication, authorization, confidentiality, integrity, non-repudiation and availability security properties. The STRIDE threat modeling can be performed using the STRIDE-per-element or STRIDE-per-interaction approaches [54]. The former is used to analyze threats on system components, and the latter is used to analyse threats on the interaction between a pair of components. A STRIDE threat modeling is performed using the data flow diagram (DFD) of the system. A DFD is a visual representation to show the flow of information or data through a process or system [69]. It uses four symbols to represent system components and their relationship with others: (a) interactors such as the developer, endpoints, attacker, servers represented by a rectangle, (b) processes such as the application, a functionality, represented by a circle, (c) data flows, which is the data over network connections, represented by a one way arrow, and (d) data store such as database, logs, and files, represented by two parallel horizontal lines [54] [70].

In general, threat modeling using the STRIDE framework involves the following main steps: (a) drawing the DFD of the system; (b) identifying vulnerabilities on each DFD component; (c) analyzing potential threats that exploit the vulnerabilities; (d) proposing mitigation strategies for the vulnerabilities and threats identified [54].

2.3. Literature Review on Security of Containers

As highlighted in the introduction, containers are vulnerable to a wide-range of cyber threats. The threats may target various attack surfaces in containers and their subsystems. The main attack surfaces of containers are user credentials, application codes, container images, container privileges, repositories, and network channels [71]. For example, stolen user credentials at the GitHub and Docker Hub can lead to user's account being hijacked or spoofed, resulting in malicious codes and images to be uploaded into these registries. An attacker may also use a compromised container as a backdoor to do illegal activities on other containers. This means that if the attacker gets access to the compromised container, it can penetrate to the host kernel and launch other containers for illegitimate purposes, e.g., crypto-mining [8].

The application code is another attack surface where bad coding practices can result in vulnerabilities like SQL injection, cross-site scripting, and server-side request forgery, among others. The Docker Hub is a popular registry for about four million of images and there are almost half which contain malware [6]. Some malicious images can stay online in Docker Hub for a year and while some have been installed for more than a million times [72]. Therefore, if a developer creates a multi-stage Dockerfile and uses multiple images without proper scanning, he may create a container with embedded vulnerabilities. An attacker can then gain access to a compromised container and raises its privilege to gain root access to the host kernel. Lastly, there are network-related threats in the virtual ethernet bridge connected between the containers and from the internet into the container.

There are a wide-range of related works on containers security. Below, we discuss the most relevant ones. To simplify our discussion, we categorize them as "vulnerability analysis" and "mitigation strategies".

2.3.1. Vulnerability analysis

There are several existing works focusing mainly on the investigation and analysis of threats and vulnerabilities around the container ecosystem. One research initiative [33] gathered 223 container related exploits from a public database[1] and classified them into a two-dimensional attack taxonomy. One dimension was the hierarchical layers of web app, server, library and kernel, and the other dimension was the consequences of attacks, such as sensitive information leakage, remote control, denial of service, and kernel privilege escalation. However, the main emphasis of this work was on the privilege escalation exploits and how to configure the kernel security mechanisms to defend against them.

Another study [28] was conducted on attacks that mainly target the Docker platform and the image distribution process. The study revealed that insecure configurations and weak access controls of the Docker platform can lead to unauthorised access to the host filesystems and network stack of the container. Automated builds and the use of webhooks during image distribution was shown to allow a tampered code to be deployed in a production server within minutes. However, this study only focuses on threats to the Docker platform in containers. Therefore, it is not comprehensive enough to cover the multifaceted threats facing the container ecosystem from image creation to image distribution.

MITRE recently released the Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK) for containers. It categorized the attacks techniques on containers and the orchestration manager (Kubernetes) under initial access, execution, persistence, privilege escalation, defense evasion, credential access, discovery and impact [34], [35]. However, the MITRE framework only focuses on the adversary techniques and does not trace the use-case of containers nor recommends context-relevant mitigation actions.

Another research was conducted on the security of the Docker platform by analyzing the vulnerabilities listed in Common Vulnerabilities and Exposure (CVE) [29]. In this work,

[1] https://www.exploit-db.com/

the authors used static code analysis (SCA) tools on the vulnerable and patched versions of the Docker's code-base in order to study the differences between the two and the effectiveness of SCA tools in detecting the vulnerabilities. This study primarily used static code analysis tools to analyze Docker's source-code and did not relate them to real use-cases nor recommend practical mitigation plans. A survey by Sultan et al. [32] was also conducted on the security of containers based on a four-dimensional risk analysis: risks from the application in the container, risks from another container, risks from a container to the host, and risks from the host to the container.

Wist et al. [30] scanned 2,500 Docker Hub images, mapped their vulnerabilities using the Common Vulnerability Scoring System (CVSS), and compared the vulnerabilities across the types of images, the types of scripting languages, and packages. In another research, Flauzac et al. [31] reviewed the native containers security by conducting a static comparison of 6 container runtime solutions, namely LXC (Linux Containers), LXD (an open-source container management extension for LXC), Singularity, Docker (runc), Kata-containers (kata-runtime) and gVisor(runsc), in terms of their abilities to isolate system resources such as storage, network, processor, and memory. However, this is carried out in the container's default and standalone state and therefore does not reflect a real operating environment that is used by a container.

2.3.2. *Mitigation strategies*

While there are several vulnerabilities and threats in the container ecosystem, there are also certain mitigation strategies developed against them. Some of the mitigation strategies, e.g., *namespaces* and *cgroups*, are built-in to the container's host operating system. The container's namespaces isolate the resources of inter-process communication (IPC), mount (or filesystems), process identifier (PID), network, user (User and Group IDs), and UTS (hostnames and domain names). The cgroups control the amount of resources (like the CPU, memory, disk I/O) a container can use so that other co-resident containers can obtain their fair share of the resources [73].

The other mitigation strategies are the underlying security features of the host kernel. These include *capabilities*, *secure computing mode (seccomp)*, *security-enhanced Linux (SELinux)* and *AppArmor* [73]. The "capabilities" are list of privileges that can be enabled or disabled for a process, and they serve to limit a root-enabled process from getting more than the minimum permissions required for it to perform its function. The secure computing mode (seccomp) helps to filter the system calls to the kernel from the container [49]. It provides a finer control than capabilities and restricts the number of system calls an attacker may perform from the container to the kernel [37]. SELinux is integrated in Centos/RHEL/Fedora distros, and it provides mandatory access control (MAC) policy setting for the applications, processes, and files in a container such that it can prevent root-enabled process within a container to illegitimately access objects outside. AppArmor is integrated in Debian/Ubuntu distros, and it is an alternative MAC to SELinux. While SELinux applies security rules on files, AppArmor applies the rules on file paths.

However, recent studies revealed that most of the existing mitigation strategies of containers have certain flaws and limitations. For example, the Linux-based mitigation strategies used in containers, such as cgroups, namespaces and capabilities, are subjected to attacks resulting in resources exploitations, denials of services, and privilege escalation [26], [20], [50]. Thanh Bui [38] discovered that a container cannot achieve effective security by using only the built-in security features of the host operating system, such as namespaces and cgroups. But, it should also use firewall rules (e.g., ebtables), MAC measures (e.g., SELinux or AppArmor), and run in a "non-privileged" mode. A detailed discussion of other existing mitigation strategies is also provided in Section 4.

In general, most of the existing works (both in vulnerability analysis and mitigation strategies) focus only on certain security issues, and do not provide a comprehensive security analysis on the overall container ecosystem. As discussed above, some of the existing mitigation strategies have also their own limitations. Some of the related works are also likely outdated, and they might not show the current threat landscape in containers. Therefore, it would be difficult to get a comprehensive and state-of-the-art information on the security landscape of containers. In this work, we perform threat modeling and a systematic survey on the security of containers, covering vulnerabilities, threats, threat consequences and existing mitigation strategies, to provide a comprehensive and latest information on the threat and security landscape of containers.

3. Threat Modeling using STRIDE

This section discusses our STRIDE threat modeling for containers. As highlighted in the preceding sections, we first perform threat modeling using the STRIDE framework, particularly using the STRIDE-per-element approach, to identify potential vulnerabilities and threats that may exist in each component of the container ecosystem. The main purposes of doing the threat modeling are to specify the scope of our survey based on the threats identified, map existing literature to those threats and highlight missing research angles.

3.1. *Plotting the DFD of Containers*

As discussed in Section 2.2, plotting the DFD of the system is the first step in the STRIDE threat modeling. In the context of containers, a common use-case is that the developer develops his application code and upload it to a code repository, such as GitHub [74]. He will then build the app image from the source-code in GitHub by creating the Dockerfile [74] and pushes it to the Docker repositories in the Docker Hub registry. The image is finally pulled to a Docker Host and deployed as a container application. We plot the DFD of the above process in Figure 1, illustrating the container creation and deployment processes and its system components. More specifically, the developer (an external entity) performs the process of coding and Dockerfile creation (P-1). Then, the completed code and Dockerfile is committed and uploaded via dataflow path DF-1 to the code repository GitHub (DS-1). Thereafter, the code and its libraries are packaged into a docker image (P-2) which will be pushed via DF-3 to the Docker Hub registry (DS-2). The docker image will then be subsequently pulled and run (P-3) via DF-5 into the Docker Host and deployed in container.

In our example, the Docker Host comprises of 4 functional components with two containers P-4 and P-5. The container is a wrapped and controlled environment and contains the application and the dependent libraries and binaries The Docker engine or daemon component (P-6) is responsible for launching the containers and to control their isolation level, capabilities restrictions and security profiles. The host OS kernel component (P-7) controls the host hardware and manages functions such as memory, files system, network, and process management. The Docker engine communicates with the host OS kernel using system calls.

3.2. Identifying Vulnerabilities in Containers

Vulnerabilities are the weaknesses in a system that allow an attacker to gain access into it via malicious techniques. In containers, vulnerabilities can occur during image creation, in its push and pull connections, verification, during the registry storage process, communications between the container and the OS kernel, and during the communications between two different containers. Vulnerabilities can also occur because of misconfigurations of the Docker Host and the Linux kernel.

Using the STRIDE framework, we discovered several vulnerabilities on the DFD (cf. Figure 1) of the container systems. To save space and simplify our presentation, we only discuss the most relevant ones, as shown below.

V1: Docker Hub (DS-1) does not enforce stringent password policies other than the minimum password length restriction of 9 characters [75]. GitHub (DS-2) mandates an account password to be at least 8 characters long if it includes a number and a lowercase letter, or a 15 characters with any combination of characters [76]. Both Docker Hub and GitHub also do not enforce the additional protection of multi-factor authentication. Therefore, a determined attacker can deploy a variety of password attack techniques like brute-force, dictionary, password spraying, and many others [77] to steal account IDs and passwords or to gain root access. In fact, these vulnerabilities are reported in dozens of CVEs, such as CVE-2020-35467, CVE-2020-35466, 2020-35462, CVE-2020-35190, CVE-2020-35192, CVE-2022-2927, CVE-2022-2098, CVE-2022-1775, and CVE-2022-0777, among others.

V2: Docker Hub allows a developer to upload (or push) an image that is not signed. This allows an image to be downloaded (or pulled) without validating its authenticity [78]. This means that even tampered images can also be successfully stored in Docker Hub and used for deployment by unsuspecting developers. Certain related vulnerabilities are reported in CVE-2018-1277 and CVE-2014-8178.

V3: Both Docker Hub and GitHub store software images and codes as they are, and they do not scan them for sensitive parameters, such as hard-coded passwords, access keys and other credentials. Inexperienced developers may include such sensitive information within the images and codes. On the other hand, industry practitioners have developed open-source tools, e.g., Docker Images Explorer[2] and Whispers[3], to scan repositories and registries for passwords, API tokens, access keys, hashed credentials and others [79]. Hence, attackers may use these tools to discover exposed credentials. Related vulnerabilities are reported in several CVEs, e.g., CVE-2021-20537, CVE-2021-20537, and CVE-2022-25217.

V4: Docker images are not always safe and patched for use and Docker Hub does not check if the latest patches are applied. A recent study [6] was conducted on 4 million Docker Hub images and discovered that 51% of them had at least one critical vulnerability. Among them, about 6,400 were classified as malicious, of which 44% were related to cryptocurrency mining, 23% were due to flatmap-stream malware, and 20% were a variety of hacking tools. Another study of more than 2 million images from Docker Hub found that it took 181 days on average for a software originator to fix a software vulnerability, but it took an extra 422 days on average for the developer to patch the fix in the image containing the software [22]. Therefore, a software with security vulnerabilities can remain in an image for more than 600 days on average and has a high probability to be downloaded and potentially exploited by the attackers. Even for the prominent Log4j CVE-2021-44228 vulnerability, there are about 9 Docker images that remain unpatched [80]. Related vulnerabilities are also reported in recent CVEs, e.g., CVE-2022-20617, CVE-2022-29186, CVE-2022-42889 [80].

V5: The distribution of images from Docker Hub requires only the HTTP API [81]. This could allow an attacker to carry out a man-in-the-middle (MITM) attack. In fact, a recent CVE report, CVE-2017-18641, revealed that a critical vulnerability was detected on LXC (i.e., the Linux container namespace isolation technology used by Docker) that allowed a code to be download over cleartext HTTP and to omit digital-signature checks [82]. This vulnerability would allow a man-in-the-middle attacker to install malicious code into the container that will run as root. Other related vulnerabilities are also reported in CVE-2018-3834, CVE-2018-3833, CVE-2015-1843 and CVE-2021-22898.

V6: Container allows API endpoints to be publicly accessible on the internet, without any firewall or password protection. This can allow attackers to successfully scan the exposed APIs and access the containers to launch attacks [83]. Some related vulnerabilities are reported in CVE-2022-24829, CVE-2022-22152, CVE-2020-35197 and CVE-2022-31066.

V7: According to [84], 44% of developers use Continuous Integration/Continuous Delivery (CI/CD) process to deploy containers. The continuous integration stage pushes the application code through the commit, build and test phases to the code repository and subsequently to the image registry. The continuous delivery stage then deploys the application in a container with environment-specific parameters. The entire CI/CD process presents wider attack vectors for attackers to exploit. While the automatic CI/CD process yields efficiency, the speed and lack of manual oversight creates security risks. A successful exploit in any part of the pipeline will allow an attacker to

[2] https://github.com/matiassequeira/docker_explorer

[3] https://github.com/Skyscanner/whispers

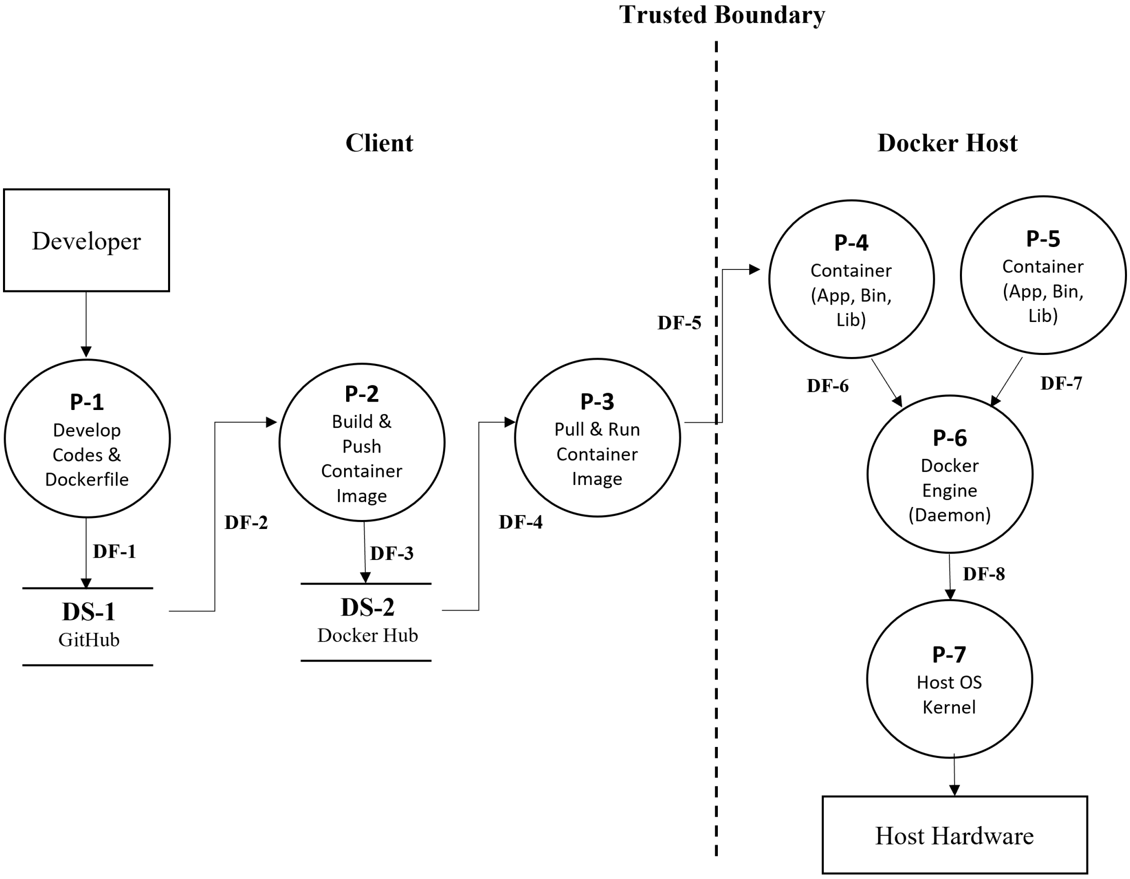

Figure 1: Data flow diagram of the container system

permeate its control to the rest of the pipeline. Some CI/CD related CVEs are CVE-2021-27024, CVE-2021-43832, CVE-2022-24768, CVE-2022-24878, CVE-2022-29184, and others.

V8: A container is immutable and when it is deployed and run, it cannot be changed or patched. A developer will need to ensure that the base image, application binaries and libraries are regularly updated to rebuild and redeploy the whole image. Some image related vulnerabilities are CVE-2021-3344, CVE-2021-3762, CVE-2021-32760, and CVE-2022-0552.

V9: Containers are typically stateless and not appropriate to store persistent data, hence the logs that record the containers' activities are stored in the local disk in the Docker host and in JSON file format. Each JSON log file contains only one container information [85]. Over time and as more logs are created, unless the old logs are cleared or log rotation is performed, the local disk will fill up and face exhaustion [86]. One such vulnerability is registered in CVE-2022-1708.

V10: One feature of the container is that it can directly connect with the host kernel, unlike a virtual machine (VM) which requires an application to bypass the VM kernel and hypervisor. Consequently, it is easier for an attacker to access the host kernel if it can breach into an application within a container that resides on the host [87]. This vulnerability has been exploited in CVE-2017-18509, CVE-2018-16884, CVE-2021-4154, and CVE-2022-0811.

V11: Container is reliance on Linux kernel and there are many vulnerabilities that are related to the Linux kernel that may affect the security of container, such as the vulnerability in runc module[4] that allows a malicious container to gain root-level access to the host machine [88]. To date, there are close to 3,000 Linux CVE vulnerabilities listed by MITRE [89]. However, there has not been much in-depth research done on the number and types of Linux vulnerabilities that directly impact containers. Some kernel-based container vulnerabilities are recently discovered and reported in CVE-2016-9191, CVE-2017-18509, CVE-2018-16884, CVE-2021-3669 and CVE-2021-44733.

V12: The efficient architecture design of multiple containers on a host and sharing its CPU, memory, network, UIDs and other resources from the same kernel is also a security risk and a vulnerability. This is because, if the kernel is attacked, malicious attackers can gain root privilege of the host and from there, they can attack other containers and the entire system [90]. Some of the vulnerabilities reported in CVE are CVE-2020-15257, CVE-2021-41103, CVE-2021-4154 and CVE-2022-31030.

The vulnerabilities and their associated CVEs are summarised in Table 1. It is observed that V1 is associated with the highest number of CVEs, in which most of them are rated

[4] https://www.cvedetails.com/cve/CVE-2019-5736/

Table 1: Vulnerabilities and their corresponding CVEs

Vulnerabilities	CVE(s)
V1	CVE-2020-35467, CVE-2020-35466, CVE-2020-35462, CVE-2020-35190, CVE-2020-35192, CVE-2022-2927, CVE-2022-2098, CVE-2022-1775, CVE-2022-0777
V2	CVE-2018-1277, CVE-2014-8178
V3	CVE-2021-20537, CVE-2022-25217
V4	CVE-2021-44228, CVE-2022-20617, CVE-2022-29186, CVE-2022-42889
V5	CVE-2017-18641, CVE-2018-3834, CVE-2018-3833, CVE-2015-1843, CVE-2021-22898
V6	CVE-2022-24829, CVE-2022-22152, CVE-2020-35197, CVE-2022-31066
V7	CVE-2021-27024, CVE-2021-43832, CVE-2022-24768, CVE-2022-24878, CVE-2022-29184
V8	CVE-2021-3344, CVE-2021-3762, CVE-2021-32760, CVE-2022-0552
V9	CVE-2022-1708
V10	CVE-2017-18509, CVE-2018-16884, CVE-2021-4154, CVE-2022-0811
V11	CVE-2016-9191, CVE-2017-18509, CVE-2018-16884, CVE-2021-3669, CVE-2021-44733
V12	CVE-2020-15257, CVE-2021-41103, CVE-2021-4154, CVE-2022-31030

critical. Note that V1 is related to weak password controls in Docker Hub and GitHub. The accessibility of Docker Hub and GitHub due to weak password controls could allow an attacker to gain root privileges and cause severe damages. V5, V7 and V11 are associated with the next highest number of CVEs, but their severity rate is mostly medium and high. V4 and V10 are associated with lesser number of CVEs, but their severity rate is mostly high and critical. V4 is related to the unsafe and infrequently patched Docker images that may carry malicious programs, and V10 is due to the container's direct access to the host kernel. Therefore, more emphasis would be placed on the vulnerabilities V1, V4 and V10, which are frequently exploited by attackers.

3.3. Analyzing Threats in Containers

Before we perform the threat analysis, we first outline the possible threat consequences as we will refer them in the threat analysis sections below. A threat consequence is a security violation that happens as a result of an attack. This includes unauthorized disclosure, deception, disruption and usurpation [91]. "Unauthorized disclosure" is when an unauthorized entity gains access to the data. "Deception" is when the victim believes that a false data is true. "Disruption" is when an normal operation is disrupted and cannot carry on. "Usurpation" is when an unauthorized entity takes control of the system and operation. To simplify our presentation as well as to easily refer them in other sections, we assign short notations for the threat consequences as follows: TC-1 for "unauthorized disclosure", TC-2 for "deception", TC-3 for "disruption" and TC-4 for "usurpation". To provide a quick and easy reference to readers, a description of the notations is also provided in Table 2.

Table 2: Notations of threat consequences

TC-1	TC-2	TC-3	TC-4
Unauthorized Disclosure	Deception	Disruption	Usurpation

3.3.1. Spoofing

Spoofing identity is an attack in which the attacker impersonates the victim (which can be a user, file, process, or role) to gain access into a system without the rightful consent. This attack compromises the authenticity security property, and the threat consequence is primarily TC-2 (or deception). Below, we discuss a list of potential spoofing threats in the containers ecosystem.

Spoofing the user's GitHub account: By exploiting vulnerability V1 listed in section 3.2, the attacker can gain access to a developer's credential in the GitHub repo at DS-1 and to embed malware into the code. Some techniques to "steal" credentials are through spearphishing email, password-spraying, brute force, scraping published credentials in repositories [92], [93]. Applying the automated deployment pipeline, the malicious code will be built into a container image at P-2. The image is then pushed into Docker Hub at DS-2 and automatically pulled and deployed at P-3 as container into the user's docker host. The entire process can take place within minutes and may infect many other machines [20]. The threat consequences are TC-2 followed by TC-1 (or unauthorized disclosure).

Spoofing the GitHub or Docker Hub: The GitHub repository can be spoofed by an attacker and may mislead the victim to upload his code to the attacker's repository. The attacker can then add malicious elements into the code and upload it to the real GitHub repository. The threat consequence is TC-2. The techniques can be in the form of DNS server spoofing where the attacker diverts the victim's traffic to a malicious IP address [94] and this is achieved by using DNS cache poisoning, Kaminsky attack, or DNS hijacking (DNSpionage) [95]. The same spoofing technique can be used on the Docker Hub (DS-2) and can lead to a malicious image being pulled to the Docker Host. So far, we have not found any article that reports about this attack vector in GitHub or Docker Hub.

Spoofing the Docker Account: A Docker account in Docker Hub at DS-2 can be spoofed by an attacker and lead developers to go to a “fake” account to download a malicious image. The investigation team from security firm Aqua Security found that a cybercrime group created an account called “portaienr” in order to masquerade a legitimate account called “pontainer” [16]. The idea was to exploit typosquatting when a victim mistyped the account name and be transferred to the attacker's account to pull malicious images [16], resulting in the threat consequence of TC-2. Due to vulnerability V5, a Docker image is not scanned for vulnerability nor verified for legitimacy, hence the attack can be successful.

Spoofing the Docker Image: A Docker image can be spoofed by an attacker and lead to an incorrect image being pulled to the Docker Host. Security firm Trend Micro discovered that attackers uploaded two malicious images and labelled them as "alpine" and "alpine2" to fake it with the popular Alpine Linux and trick unsuspecting developers [17]. Due to vulnerability V5, the image was successfully pulled without scanning. Running these images resulted in spawning of containers that installed the XMRig crypto-mining applications. The attackers could tap on the victim's computing resources to mine cryptocurrency [17], resulting in threat consequence TC-4 (or usurpation).

Spoofing the DNS responses to a cluster of containers: Most application containers are deployed in Kubernetes clusters (RedHat's survey shows that 88% of customers use Kubernetes to manage the containers [96]) and reside in pods. Each pod communicates with each other via a bridge that runs in the root network namespace. This is made possible due to the default enablement of the capability NET_RAW, which allows traffic (e.g., ICMP, ARP, DNS) to flow between containers. This is a characteristic of vulnerability V12 where multiple containers share the same host. An attacker can launch a DNS spoofing attack from a compromised container in a pod and return fake answers to DNS queries sent from a co-located victim container pod. Subsequently, the attacker can execute MITM attack on the network traffic between the containers [18], [19], resulting in threat consequence TC-2.

3.3.2. Tampering

Tampering is an attack in which the attacker modifies the data, memory space, or network and violates the security property of integrity. The main tampering threats in containers are discussed as below.

Tampering the network between Docker Hub and Docker Host: Due to vulnerability V5, an attacker can tamper DF-4 and DF-5 (Figure 1), which are the data flow channels between Docker Hub and Docker Host. The attacker can insert his malicious images and be downloaded on the docker host. For example, an attacker can craft an image to contain a large file filled with garbage and when it is extracted, it would fill the host storage to cause a consequence of TC-3 (or disruption) [20]. In another example, when the malicious image is extracted on the host filesystem, path traversals can allow the attacker to replace binaries on the host with binaries from the image [20] causing the consequences of TC-1 and TC-2.

Tampering the CI/CD pipeline: This threat is due to vulnerability V7. CI/CD pipelining is a popular software development and deployment pattern used by many enterprises. The two distinct processes automate the entire flow of software build to deployment. It starts with code build, test and commit to the code repository (GitHub), to building container image based on the code, tags and pushes the container image to the container registry (Docker Hub), and finally to deploy the image as a container in the Docker host. Attacks to the network in each "pipeline" situated in DF-1, DF-2, DF-3, DF-4, and DF-5 can result in tampered software artifact and image. There are limited in-depth studies of the threats and attacks that can occur during the transportation of the codes and images along the pipelines in an automated CI/CD workflow. Martin et al. [20] did a comprehensive study in the vulnerability analysis of container in three use-cases - microservices architecture, virtual environment deployment, and cloud provider using it as container-as-a-service. Somya Garg and Satvik Garg [97] described the mechanism of CI/CD using Docker and listed some common container security best practices in the use of namespaces, cgroups, and Linux capabilities. There is an opportunity for more research works around the security aspect of the entire CI/CD process. The consequences of this threat are TC-1, and potentially TC-3 if the network connection of any of the pipelines is disrupted.

Tampering application codes at Docker Hub/Github: The application code on DS-2 may be tampered with by attackers to include vulnerabilities. Docker Hub was attacked in such way before, and the usernames and hashed passwords of 190,000 users were exposed [21]. The breach can result in the attacker accessing a user's application image and tamper with its codes. If the image is not signed, the change will not be detected during download. In addition, Docker images may contain inherent vulnerabilities which the developers are not aware of until they are deployed in a production environment. Furthermore, Docker Hub hosts many third-party applications used by organizations. For example, the SolarWinds hacking [98] incident showed that the APT group successfully tampered with the codes of Orion which was used to monitor and to manage network resources. The tempered patch was delivered to many organizations resulting in the sensitive information disclosure and loss of confidentiality [99]. A study has shown that the official and community images contain an average of 180 vulnerabilities and 50% of these images have not been updated [100]. It takes an average of 181 days to fix the vulnerability and an additional 422 days on average to update the image [22], and this presents a window for an attacker to exploit the vulnerability. This threat is attributed to vulnerabilities of V2 and V4. The consequences are TC-1 and TC-2.

Tampering image during image build: Due to the vulnerabilities of V2 and V3 where an image is freely uploaded without any checks and controls, it can be tampered without being discovered. During the image build at P2, an attacker may inject malicious commands or vulnerable components into the image file. The image may continue to be signed and appear legitimate to the developer [23]. The tampered image can cause the deployed containers at P-4 and P-5 to perform malicious acts to the host or other containers residing in the same host causing the consequences of TC-1, TC-2 or even TC-4. Developers rely on open source libraries when developing their applications. A commercial study finds that seven in ten applications use at least one open-source library with a security flaw [101], and that the library vulnerabilities increase by 88% over a two year period [102]. Palo Alto Networks conducted a study which found that 96% of third-party container applications deployed in the cloud contain known vulnerabilities [103]. The most recent and prominent library vulnerability is the Apache Log4J Java-based library whose vulnerability to log4shell allows an attacker to perform a remote code execution when it is used for logging function [104]. The attack surface is further expanded if the libraries have their own dependencies on codes from other libraries. The malicious libraries in a deployed container at P-4 and P-5 will interact via the Docker daemon at P-6 to gain unauthorised access to the OS kernel. These threats will lead to the consequence of TC-1, TC2, and even TC-4 when the host kernel is under control.

3.3.3. Repudiation

Repudiation is associated with an attacker claiming that something which is done is not performed by him. This attack violates the security property of non-repudiation. In the following, we discuss the main repudiation threats in container systems.

Disabling logging functions: An experienced attacker will cover his track to avoid detection and attribution. The attacker may attack the audit mechanism and attempt to delete or modify the logs stored in element P7 in Figure 1. He may disable the logging function using "Auditpol" in Windows systems or "auditctl" in Linux systems. He may delete the logs with clear-logs.exec in Windows systems and shred tools in Linux systems [105]. The consequence is the disruption of logging activity.

Modifying log data: The log files in Docker can be found in /var/lib/docker/containers directory on the host system [106] and they can be modified by the attacker. This threat is possible due to the vulnerability of V9 and V10 as the container is dependent on the Linux host for logging activities and storage. At this point we have not found any reports that describe a real attack event on Docker logs. The consequence is deception by modifying the log data.

Overwriting log disk space: A container utilizes the memory and storage space of the host and this vulnerability is aligned with V9 and V10. A container is enabled with the capability CAP_AUDIT_WRITE to record activities and events into the kernel audit log [107]. The kernel audit log is stored on the disk in the host at P-7 and the attacker container can write massive amount of junk data onto the disk and overwrite the valid logs recorded by the victim container [24]. This attack can cover the tracks of a malicious action and prevent the victim from accessing valid logs to perform investigation. This threat will result in deception as the real logs are overwritten.

3.3.4. Information Disclosure

Information disclosure is allowing unauthorized entity to access data, information, processes or networks which he is not allowed to. This attack compromises the security property of confidentiality, and the following are a list of information disclosure threats in containers.

Weak access control of GitHub and Docker Hub: Weak access control of GitHub (DS-1) and Docker Hub (DS-2) allows an attacker to access information which he is not authorised to do so. There have been several security breaches in GitHub where identity keys and data information have been stolen. For example, developers from Starbucks expose API keys in GitHub, which can allow an attacker to access its active directory management platform [108]. Starbucks later removed the repository and revoked the API keys. Another attacker got access into CircleCI's user data which include their GitHub's usernames, emails, repo URLs, branch names, organization names and repo owners [109]. This prompted CircleCI to enforce two-factor authentication (2FA) for their account holders. Another attack involved gaining access into all the Git hosting services including GitHub, GitLab, etc. to steal source-codes and demanding ransoms from the owners [110]. Some of the victims had admitted to using weak passwords and forgetting to remove access tokens for old apps. Recently, millions of Brazilian COVID-19 patients' personal private information (including the Brazil's President, ministers and provincial governors) were exposed when a spreadsheet which stored the login credentials of the government healthcare systems were exposed by a GitHub user [111]. The source codes of Nissan were leaked and exposed from a Git server when its developer secure it with its default username and password combo of admin/admin and they were easily cracked by attackers [112]. Mercedes Benz's smart car components source code were leaked when an outsider successfully signed up for an account in its Git web portal using a non-existent Daimler corporate email [113]. In addition to the easy access into GitHub account, an inexperienced developer may make a change in a source code file and unknowingly commit and upload all other files (which include sensitive ones) in the same folder into GitHub. An attacker who breaches a Github account can access these sensitive files. It was also found that SolarWinds developers' credentials were possibly leaked from GitHub that could have resulted in attackers gaining access to the codes and leading to the infamous SolarWinds Orion hack [114]. The access control of Docker Hub at DS-2 can be exploited and sensitive data be exposed. In 2019, a database of 190,000 users' usernames and their hashed passwords in Docker Hub was hacked into by attackers [13]. On separate occasions, attackers managed to steal the credentials from the cloud providers and took control of the container instances which were owned by Aviva, Gemalto and Tesla and used them for crypto-currency mining [25]. This threat can be attributed to vulnerability V1 which is due to a non-stringent credential and access control measures. The consequence is the unauthorized disclosure of sensitive information.

Sensitive parameters to access the host: The run-commands used in P-3 to run a container may contain sensitive parameters which allows an attacker that develops the container image to gain access to the user's host and its data. These parameters are not usually detected by the security scanner as they are not malicious in nature. For example, a user may run a container command with "- -privileged" to access certificate on the host to spawn a container [15]. The use of such "sensitive" parameter will allow the container to gain root access to the host and this can be exploited by an attacker [22]. Another example is the use of "- -volume" and "-v src:dest" that allows a container to gain access to "src", which is a volume in the host and as a result allows an attacker to upload data in the host to a online repository [22], resulting in threat consequence of TC-1. In some instances, there may be a need to configure the parameter of "- -pid=host" within a container in order to run debugging tools, like strace or gdb [115]. Such configuration allows the container to share the host's PID (process ID) namespace. If an attacker gains control of the container, he will be able to view all the other processes running on the host. Armed with info of the PID, along with "owner" and path of the executable file, the attacker can conduct attack to other containers and the host [22]. This threat is attributed to vulnerabilities V10 and V12 which is due to the common Linux kernel shared by multiple containers. Due to vulnerability V10, the configuration options of the Docker engine/daemon at P-6 can provide access to the host OS kernel. This can be achieved with the options of "-net=host", "-uts=host", "-privileged", and additional "capabilities". The option "-uts=host" can allocate the same UTS namespace for the container and the host which allows the container to see and change the host's name and domain [20]. The capability "-cap-add=SYS_ADMIN" can enable a container to remount /proc and /sys sub-directories in read/write mode, and change the host's kernel parameters [20], leading to potential threat consequences of TC-1 and TC-4.

Leakage of information between containers: Containers that reside in the same Linux host and share the OS kernel (P-7) can leak information to each other via storage path mapping, port mapping, layer-2 network connection, and covert channels. This can enable an attacker of one container to gain access into another co-locating container [24]. Some of the methods include exploiting the openly observed globally used memory (GUM), which an attacker can obtain visibility of the victim container's memory information [24]; accessing the global variable of inode number (or index node) allows an attacker container to know the metadata of a victim container's process file [24]; and an attacker container can read into the kernel message buffer (KMB) which is written into by a victim container with the CAP_SYSLOG enabled [24]. This is again due to the vulnerabilities V10 and V12 and the consequence is the leakage of unauthorized information (TC-1).

3.3.5. Denial of Service (DoS)

The denial of service causes a service to be disrupted or degraded such that users cannot access the service. This attack violates the security property of availability. Most of the threats listed below are attributed to the vulnerability V10 which is the close connection between the container and the host kernel unlike a virtual machine which is separated by the VM kernel and the hypervisor. The attack involves abnormally consuming resources such as CPU, memory, storage, networks, etc. The threat consequence is mainly TC-3. Below, we discuss the main DoS-related threats in the containers context.

Inaccessibility of GitHub or Docker Hub: The attacker may cause GitHub (DS-1) or Docker Hub (DS-2) to be inaccessible to developers for code updates and container deployments. While the infrastructure facilities of GitHub and Docker Hub are not publicly known, it is assumed that they are highly resilient, secured and are distributed across multiple sites like the commercial cloud computing services, such as AWS, Microsoft Azure or Google Cloud. Therefore, at this point there is little evidence to show that the services from GitHub or Docker Hub have been disrupted due to attacks on their server infrastructures. An article was written that painted a scenario where a DDoS attack targeted at the control traffic between the Network Operations Center (NOC) and the data center's Heating, ventilation, and airconditioning (HVAC) could potentially result in overheating and to cause a data center outage [116]. However, in reality, there had also been data center outages that resulted from overheating due to equipment failures [117], service component failures such as the DNS outage in Azure [118], Kinesis disruption in AWS [119], and other non-attack related causes.

Service disruption at host via OS kernel: From Figure 1, the container via the Docker engine (P6) communicates with host OS kernel (P7) via a series of system calls. By default, each container has access to the host's CPU cycles and memory without limit [120]. An attack on the OS kernel will cause the disruption of services to the host's computing resources like the CPU, memory, storage, and others resulting in the threat consequence TC-3. Attacks utilizing exceptions handling, logs writing, and disk write-backs can impact CPU, disk I/Os and memory performances. The Linux kernel will trigger an exception handler when exceptions such as faults (e.g., divide error) and traps (e.g., overflow) occur. When one of them happens, the kernel will send a signal to the process which generates it, and it will take steps to recover or to abort [121]. The exception will trigger the core dump kernel function to generate a core dump file which is used for debugging. It is shown that when a container keeps raising exceptions (example div 0) and triggers the core dump, the host system CPU and memory performances are reduced by 95% [26]. Therefore, an attacker can create a DoS attack on a host using this exploitation and thereby impacting the performance of all containers which run on this host. System logging in Linux at P-7 is typically performed by journald which is a part of systemd, an init system and system manager [122]. As a system service, journald not only collects system and kernel log messages, but it also collects three types of log messages in a container. They are switch user (su), add user/group, and exception [26]. As journald is a system service, its resource utilization will be taxed on the host and is not controlled by the container cgroups. It is shown that the three container logging operations performed by journald can cost up to 20% extra CPU utilization and an average of 2MB/s IO throughput [26]. Therefore, this is an exploit which an attacker can use in a container to overwhelm the host resources which in turn impact the performance of the other containers causing a TC-3 consequence. To improve performance, the Linux kernel writes data in the cache memory and later performs a disk writeback of the data into the disk at the host. However, data may be lost or corrupted when the system crashes, and one way for a user to invoke a writeback is to run a system call "sync", which writes any data stored in the cache memory out to the disk [123]. It is shown that when a malicious container keeps calling "sync" while another victim container performs write operations, it leads to high CPU wait time due to the combination of sync and write operation. The victim I/O performances (such as sequential read /write and random read/write) are reduced to almost 1% [26]. This shows that an attacker can launch a DoS attack on the host and hence on another container by exploiting the data writeback mechanism to the disk.

Inaccessibility of the data flows: As shown in prior threats targeting the CI/CD automated integration and deployment process (due to vulnerability V7), an attack in any of the data flow connections at DF-1, DF-2, DF-3, DF-4, and DF-5 will cause disruptions to one or more of the processes of code commits, images build and upload, images download and containers deployment.

3.3.6. Elevation of Privilege

Elevation of privilege increases the level of authorization of an attacker such that he can perform operations or access information which he is not allowed to do so. This attack violates the authorization property of security and leads to the consequence of TC-4. A container is vulnerable to a host take-over attack because of vulnerabilities V10, V11, and V12. This is due to its tight integration with the Linux kernel, sharing it with other containers, and inheriting vulnerabilities that frequently discovered in the Linux operating system. Below, we discuss the specific elevation of privilege threats in containers.

Run container as root: At P-2 in Figure 1, there are some considerations when configuring the Dockerfile to build a Docker image. By default, the Docker container runs as root since the Docker daemon needs root privileges to modify the host filesystems to run [124], unless a developer intentionally configures it otherwise. As such, an inexperience developer may pull and deploy container at P-3 in a root privilege mode. This allows an attacker to copy files from the host to the container and access them, and launch a remote command execution (RCE) attack [27].

Gain root access via misconfigured networking: A newly created container will be configured with the default bridge network at the Docker daemon networking stack at P-6. The default bridge network allows other unrelated containers or services to communicate with it remotely [125]. An attacker can exploit this container and open a listening port to other containers in the same network. When it discovers an open port, it will connect to its Docker daemon and instruct it to download and run a malicious script [126]. The malicious script can potentially disable the security system of the host, create a root user, and download and install a malicious program such as a crypto-miner to perform crypto currency mining [126].

Use of system calls to gain privilege: During the starting and running of application containers, system calls are made from the containers to the host kernel at P-7. It is noted that 331 system calls are allowed by default, but an experiment with a MySQL database container show that only 116 system calls are needed in the booting phase and 58 system calls are used in the running phase [49]. In another experiment using the Apache web server, 47 unnecessary system calls are enabled in the container, and they are found to be vulnerable to exploitation e.g., prctl() in CVE-2020-10768 and setsockopt() in CVE-2021-20239. Therefore, a high default number of system calls increases the attack surface and the unnecessary system calls can be used by malicious processes to gain elevated privilege in the host.

Kernel privilege escalation attack: A study has shown that an attacker can make use of a compromised container to launch attack on the host kernel at P-7 to escalate its privilege. Exploits contained in CVE-2017-7308, CVE-2017-5123, CVE-2016-8655 (or Exploit-DB IDs of 41994, 43127, 43029 and 40871), CVE-2021-3344, CVE-2021-4154, CVE-2022-29179 show that privilege escalation exploits can overcome the default security mechanisms in “Namespace”, “Cgroup”, “Capability”, “Seccomp” and “MAC” to launch a malicious shellcode in the kernel and in supervisor mode [33]. This is carried out by bypassing the KASLR (Kernel Address Space Layout Randomization) to obtain the address of the critical kernel static functions, and to launch attacks like “use after free”, race condition, buffer flow etc., to enable the overwriting of the pointers of the kernel functions. The attacker then overwrites the kernel functions’ pointers to disable the CPU protections of SMEP (Supervisor Mode Execution Protection) and SMAP (Supervisor Mode Access Protection) and to point to a malicious user space function or shellcode, which invokes a kernel function commit_creds() to apply for root credential [33]. Another attack leverages the "time of check to time of use" (TOCTOU) vulnerability to gain root access to the host. This happens when a user executes a "docker cp" command to copy contents from the container to the host filesystem and the attacker adds a symlink component to the path after the resolution and before the operation. This results in resolving the symlink path component on the host as root allowing it to read and write to any path on the host [127].

The DFD diagram with the overlay of the vulnerabilities and the potential STRIDE threats is shown in Figure 2.

4. Existing Mitigation Strategies and Their Limitations

In Section 3, we discussed the potential threats and vulnerabilities we have identified using the STRIDE framework. This helped us to also explore the respective mitigation strategies mentioned in the literature as well as to identify research areas that have not yet been explored. Below, we will discuss the identified mitigation strategies and their limitations to address the corresponding security threats in containers.

4.1. Multi-Factor Authentication Systems

One of the practices to harden access to an account is using multi-factor authentication (MFA) systems. It is found that 99.9% of the accounts that were breached before did not use MFA, and that a basic 2FA using SMS could stop 100% of automated attacks and 96% of phishing attacks [128]. Docker Hub offers 2FA using mobile phone authenticator application (e.g., Google Authenticator) or Yubico Authenticator with a Yubikey [39], while GitHub offers 2FA with applications like Authy, Duo Mobile, Google Authenticator, Microsoft Authenticator, etc. [129]. In addition, we should enforce policies like strong passwords and regular rotation of passwords. Microsoft has listed useful password guidelines like banning common passwords, not to re-use organization passwords for non-work related purposes, enable risk-based MFA, and others [130]. These measures can help prevent attackers from using stolen credentials to access codes and images in containers.

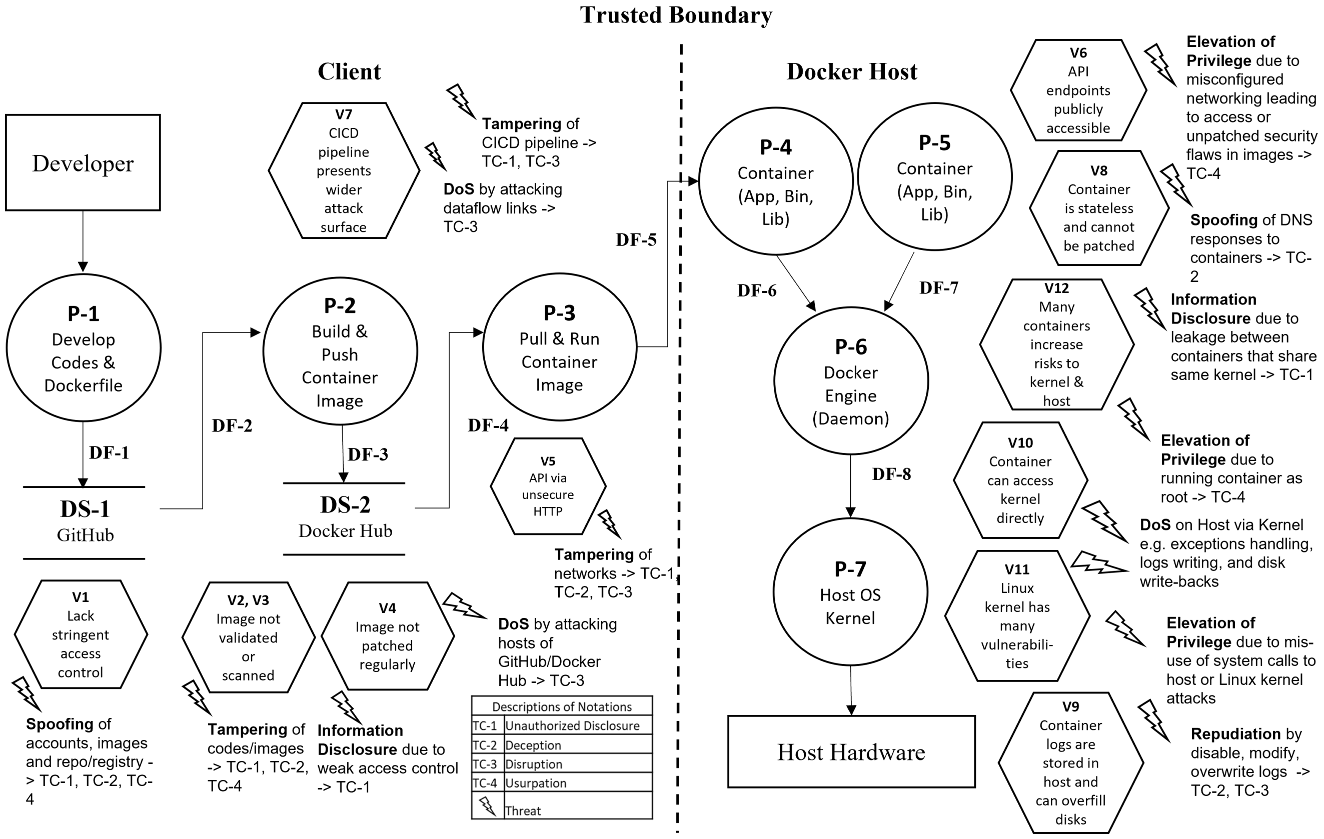
Trusted Boundary
Client
Docker Host
Developer
P-1 Develop Codes & Dockerfile
DF-1
DS-1 GitHub
DF-2
P-2 Build & Push Container Image
DF-3
DS-2 Docker Hub
DF-4
P-3 Pull & Run Container Image
DF-5
P-4 Container (App, Bin, Lib)
P-5 Container (App, Bin, Lib)
DF-6
DF-7
P-6 Docker Engine (Daemon)
DF-8
P-7 Host OS Kernel
Host Hardware
V1 Lack stringent access control
Spoofing of accounts, images and repo/registry -> TC-1, TC-2, TC-4
V2, V3 Image not validated or scanned
Tampering of codes/images -> TC-1, TC-2, TC-4
V4 Image not patched regularly
Information Disclosure due to weak access control -> TC-1
DoS by attacking hosts of GitHub/Docker Hub -> TC-3
V5 API via unsecure HTTP
Tampering of networks -> TC-1, TC-2, TC-3
V7 CICD pipeline presents wider attack surface
Tampering of CICD pipeline -> TC-1, TC-3
DoS by attacking dataflow links -> TC-3
Descriptions of Notations
TC-1 Unauthorized Disclosure
TC-2 Deception
TC-3 Disruption
TC-4 Usurpation
Threat
V6 API endpoints publicly accessible
Elevation of Privilege due to misconfigured networking leading to access or unpatched security flaws in images -> TC-4
V8 Container is stateless and cannot be patched
Spoofing of DNS responses to containers -> TC-2
V12 Many containers increase risks to kernel & host
Information Disclosure due to leakage between containers that share same kernel -> TC-1
V10 Container can access kernel directly
Elevation of Privilege due to running container as root -> TC-4
DoS on Host via Kernel e.g. exceptions handling, logs writing, and disk write-backs
V11 Linux kernel has many vulnerabilities
Elevation of Privilege due to misuse of system calls to host or Linux kernel attacks
V9 Container logs are stored in host and can overfill disks
Repudiation by disable, modify, overwrite logs -> TC-2, TC-3

Figure 2: DFD of the container system with vulnerabilities and potential threats

Limitations: While 2FA improves the security by adding a layer of authentication to the password controls, it does have several disadvantages. 2FA increases the time and cost to access the accounts and this can be significant if an organization has thousands of employees [131]. By default, 2FA uses SMS to text the verification code to a user's phone. An attacker can easily perform SMS attack on a compromised phone or the messaging center to retrieve the verification code that is not encrypted [131]. While using a mobile authenticator app is safer than 2FA with SMS, there is a report that shows attackers stealing one-time passcodes generated by Google Authenticator on a mobile phone [132].

4.2. Image Security

Securing container images is one of the existing mitigation strategies against threats in containers. Below, we discuss the main image security strategies applicable in container systems.

Reducing attack surfaces: It is recommended that an image be kept minimal so that the attack surfaces can be reduced. A couple of best practices in this regard include using multi-stage build feature that enables the developer to create an intermediate container with the required tools, and selectively copy the artifacts to the final image with only the minimal required binaries and dependencies [40]. The other practice is to use distroless[5] images as they do not contain package managers, shells, and others so that the image is kept minimal [41].

Signing images: It is advised that a developer digitally signs his image with Docker Content Trust [42] that is attached to the Notary server,which is used for validating the integrity of the images [133]. Consequently, it is a good practice for developers to verify the authenticity of the images before pulling them by enabling Docker Content Trust [42]. In addition, the developer should ensure that the hash of the image is the same at the Docker Hub as well as when it is deployed to the Docker host. Another effective mitigation method is to enable the Linux Integrity Measurement Architecture (IMA), which would validate the file signatures against pre-installed certificates and denies unauthorized file from being executed. It is shown that IMA can prevent a code that is not signed or signed with unknown key, or a modified code with an invalid signature [43].

Vulnerability scanning: After building the image and before a developer pushes the image to Docker Hub, one good practice is to scan the image by baking a scanning command in the Dockerfile or running a script at P-2. Another good practice is to scan the images before deploying them. Docker Hub provides vulnerability scanning but only to paid subscribers under the Pro or Team plan [134]. However, there are several open-source container scanners in the market and these are Anchore, Clair, Dagda, OpenSCAP, Sysdig Falco, and others [44]. In addition to static scanning, we can also perform dynamic analysis by running the container in a Docker-in-Docker sandbox mode and scanning it with tools such as VirusTotal (a collection of anti-

[5] https://github.com/GoogleContainerTools/distroless

virus tools) and examining the collected tcpdump/log files for file changes, network traffic, and list of processes [45]. For application code scanning, GitHub offers CodeQL [135] and integration to third-party code scanning tools, such as Checkmarx, Synopsys Intelligent Security Scan, Veracode Static Analysis, and others [136] for identifying vulnerabilities in the codes.

Limitations: While signing the image is an important safety measure, the private keys used for signing can be stolen. There have been several instances and methods deployed to steal private keys [137], [138], [139], and therefore more research can be done to protect them. With respect to container scanners, Javed and Toor [140] used Claire, Anchore, and Microscanner to investigate the quality of the container scanning and found that they were at most 65% accurate in the detection rate, leaving about 34% of the vulnerabilities being undetected and passed through and being deployed in production environment. The container scanners depend on the CVE data from public databases such as the National Vulnerability Database from the National Institute of Standards and Technology (NIST), Red Hat Enterprise Linux, Debian, and others to check if an image has vulnerabilities. As such, the scanners are not able to detect security flaw that has not been publicly disclosed or if the image is rebuilt from an open-source software package and given a version number which is not tracked in the vulnerability databases [141]. Another limitation is the disparate processes and tools across the container scanning workflow and there is no one integrated tool which can perform static and dynamic scans.

4.3. Security Patching

It is advisable for developers to use verified and official images from trusted repositories and providers. A study [30] shows that "official" images are the most secure among image types, which include "verified", "certified" and "community". Both the "official" and "verified" images are the most updated, while the "community" and "certified" images are the least updated ones. The developers should update their images with the latest security patches and rebuild the images periodically. NIST recommends the following scenarios and the urgency of patching [46]. Routine patching is the standard procedure to patch on a regular release cycle (e.g. Patch Tuesday). Emergency patching is carried out quickly to address extreme severity vulnerabilities and exploits. Emergency workaround is performed prior to the vendor releasing a patch and it may include roll back exercises. Lastly, it can involve the isolation of unpatchable assets if the systems cannot be easily patched [46].

IBM researchers Araujo and Taylor [142] developed a just-in-time (JIT) patching framework called "Insider" for patching running legacy application processes. This was done by injecting and compiling the code inside the running processes while sandboxing malicious processes for threat investigations. However, it is not developed for a containerized environment. A containerized application self-patch framework was developed by Tunde-Onadele et al. [143] that performed attack detection by using machine learning methods on the system calls; attack classification by comparing it to the CVE database; and finally patch execution by downloading the latest files to update the image and spinning new application container.

Limitations: At this point, there is no known automatic or JIT patching mechanism developed for the container. While rapid patching is important to address vulnerability in the container before an attacker gets into it, it may cause compatibility issue with the application without first testing it in a lab environment. Therefore, a reliable and rapid patching framework for containerized application is a gap which should be tackled quickly. However, the SolarWinds hacking incident has shown that patching and updating the software can have an adverse effect as the latest version of software can contain vulnerabilities [98].

4.4. Minimise Administrative Privileges

One way to mitigate against attacks on sensitive parameters is to design a mechanism to detect sensitive parameters and to alert the user of the risks before he executes the run command. As far as we know, there is no such mechanism available to perform this function in containers. There are recommendations by Center of Internet Security (CIS) to limit harmful docker run options and some examples are, hardening host configuration, limit file permissions, configure TLS for Docker Hub and control socket, and many others [20],[47]. There are also methods to configure a container to run in a "rootless mode" and some of these are proposed by Docker [48], Bitnami [144], Redhat [145],[146], and others.

System calls related vulnerabilities could lead to privilege escalation attacks. Almost 17% privilege escalation attacks listed in the Exploit Database maintained by Offensive Security were due to system calls [147]. The threats of mis-using of the system calls in the containers can be mitigated using the following methods. The SPEAKER mechanism developed by Lei et al. [49] traces the systems calls needed in the booting and running phases of a container and then dynamically modifies the security filter to reduce the number of system calls in each phase, thereby reducing the attack surface which is exposed by the system calls.

Another method, called Classified Distributed Learning (CDL), which is developed by Lin et al. [148], uses the machine learning algorithm to detect anomalous behaviour of the system calls and to raise an alert if it differs from the normal pattern. The system calls are collected from running containers and they are classified by application class using the random forest technique and subsequently grouped together. The autoencoder neural network is then used to train on the system calls data set and the model is applied to new system calls flow to detect anomalous behaviour [148]. The accuracy rate is 74% when applied to 24 commonly used applications with 33 known vulnerabilities.

Another method of anomaly detection developed by Abed et al. [149], uses the Bag of System Calls (BoSC) technique. This method is first introduced in 2005 to improve the then widely used fixed-length contiguous subsequence models in intrusion detection systems (IDS) [150]. It is subsequently applied onto the Linux containers to detect anomaly in system calls [149].

The method collects “bags of system calls” (BoSC) in a normal container operation and stores them in a database. In a new container operation, the new bags of systems calls are compared against the database of BoSC and if there are mismatches which exceed a certain threshold, an anomaly is assumed. Each BoSC consists of an array of distinct system calls’ frequency of occurrences [149]. The method is shown to be accurate to detect anomaly but it is only tested on a MySQL container using SQL injection attacking tool. It has not been proven to work in other use-cases.

Rastogi et al. [151] developed a method called Cimplifier, which applies the principle of privilege separation and it aims to partition a container into smaller containers which isolate from each other and only equip with the necessary resources and they communicate with each other when needed. Lastly, it is also a good practice to limit the permissions of capabilities in the container to those which are necessary so that attackers do not take advantage to exploit them to gain control of the host [152].

Limitations: The system calls anomaly detection techniques proposed are either not highly accurate or only tested on a specific use-case. There is a need to develop higher accuracy anomaly detection method which can apply to most use-cases and applications.

4.5. Proper Isolation

The cgroups of the Linux kernel are primarily functioned to control and limit the underlying host resources for each container. Within the cgroups, there is the cpuset subsystem which a developer can configure to bind a container to a set of CPU cores so that the CPU resources are protected from DoS attack [153]. It was also demonstrated that the use of Linux memory bandwidth management module MemGuard can limit the CPU access to the memory and can thus prevent a DoS attack on the memory [153]. There are numerous security best practices that can mitigate DoS attacks, e.g., using read-only filesystems, limiting kernel calls, restricting networking and inter-container communication, not expose Docker daemon socket, limit resoucres, and others [87], [152].

Limitations: The use of cgroups and namespace isolation methods in containers have several limitations. A recently discovered CVE vulnerability[6] showed that a use-after-free flaw can occur in the cgroupv2 subsystem during system reboot. This flaw would crash the system or escalate its privileges [154]. The other limitation of container isolation is that the current isolation measures do not truly sandboxed containers that share the same host [155]. Consequently, numerous container escape vulnerabilities have been discovered, such as CVE-2016-5195, CVE-2016-9962, CVE-2017-5123, CVE-2018-6552, CVE-2019-5736, CVE-2020-3514, and CVE-2022-0811. Gao et al. [26] also presented several exploiting strategies to escape the resource protection set up by the cgroups. Furthermore, other researches [20], [38] showed that the current container isolation system cannot effectively isolate the network as the same network bridge is shared by the containers, causing ARP poisoning and MAC flooding attacks on the containers.

4.6. Prevent Confidential Data Leaks

To mitigate against credentials exposure, it is a good practice not to store unencrypted secrets in Git repositories, but to use tool like git-secret to encrypt passwords, secret keys and sensitive data [156]. Within Docker Hub, developers can store secrets in credential stores such as D-Bus Secret, Apple macOS keychain, Microsoft Windows Credential Manager and "pass" [157]. The recommendations to strengthen passwords and protect access control as described in section 4.1 are applicable here.

When committing and uploading modified files into GitHub, one good practice is to use ".gitignore" feature to specifically exclude certain files from being "committed" into GitHub [158]. This will prevent sensitive files which reside in the same folder as the program code to be uploaded into GitHub. Another practice is to use ".gitignore" to whitelist the files (instead of exclude) to commit [159].

Limitations: Credential storage secrets manager or vault is not bullet-proof. CyberArk had tested a method to steal credentials stored in Local Security Authority (LSA) Secrets registry and to achieve lateral movement throughout the system [160]. Despite having solid vaults, confidential data and credentials can be leaked if the user share credentials such as committing access keys, passwords, and secrets to source control repositories. A compromised user’s endpoint devices such as notebook, desktop, and mobile device will also allow an attacker to find secretive credentials. MITRE has listed a number of credentials dumping methods that can be exploited by attackers [161].

4.7. Implement Network Controls

In order to prevent DNS spoofing attacks, it is a good practice not to use Docker’s default bridge docker0 but to use Docker’s user-defined network [47]. The developer using the end point device should encrypt the network with a virtual private network (VPN) and to regularly flush the device’s DNS cache [162]. The VPN is also important to secure the communication between the containers [163]. To protect the network connectivity from DoS attack, it is a good practice to turn on the intrusion detection and prevention systems (IDS and IPS) to detect and prevent such attacks. Lastly, it is recommended not exposing the Docker daemon socket (the main entry point for Docker API) [152] and other unnecessary ports (e.g., SSH Port 22).

Limitations: The use of VPN can increase network latency and introduces delays that are bad for ad-hoc transient container applications such as event-triggered serverless functions or Internet of Things (IoT) containers communicating many small packets rapidly. Therefore, additional research is needed in network protection for such use-case. IDS and IPS use rule or signature-based packet evaluation and therefore not effective against unknown attacks or against an attacker that poses as admin to "legitimately" log into the system [164]. IDS which yields many false alarms can lead to "alert fatigue" while IPS can consume much network bandwidths.

[6] https://nvd.nist.gov/vuln/detail/CVE-2020-25220#vulnCurrentDescriptionTitle

4.8. Robust Log Monitoring

The mitigation measures need to enable the logging system to be robust and immutable. One method is the use of message authentication codes (MACs) and digital signatures to produce the secure logs, and to apply Bitcoin blockchain technique to produce a distributed log immutabilization solution [165], thus ensuring the logs' authenticity and non-repudiation. To resolve the log storage problem, one practice is to use logging drivers to read the data directly from the Docker container's stdout and stderr ouput and to forward the logs to host machine or other endpoints such as syslog, journald, gelf, and others [166].

Limitations: When running the blockchain operation, there is transaction fees (at 0.00016 BTC/KB or USD6.83/KB as of 25th Sep 2021[7]) and is not sustainable in the long run. Other limitations when using the logging drivers are that the capacity limit of the local storage will determine the size of the log file [86]. If the logs are sent remotely, a network failure will cause the lost of the logs [167].

5. Summary of Results and Future Research Directions

5.1. Summary of Results

The overall containers security analysis we conducted using the STRIDE framework is summarized in Table 3. It is observed that each of the STRIDE threat occurs in several DFD elements and results in multiple consequences with the aim to deceive, disrupt, disclose information, or to usurp control of the system. Spoofing is about using a fake identity to gain access into the system. GitHub (DS-1), Docker Hub (DS-2) and the containers (P-4, P-5) are the obvious targets for attackers to exploit and to introduce malicious contents in order to deceive (TC-2), retrieve info (TC-1) and to control the systems (TC-4). The efficient and ease-of-use characteristics of the container systems turn out to be the vulnerabilities for the threat to be successful. The ease of access into the code repository and image registry, unrestricted push and pull of the images, and the efficient sharing of host resources by several co-locating containers become the vulnerabilities.

Tampering aims to modify the system or data with the intention to deceive (TC-2) the victim, steals the info (TC-1), disrupts the service (TC-3), and to gain control of the system (TC-4) via the tainted images. This threat has the widest impact to the DFD elements including the data stores of DS-1 and DS-2, all the data flow (DF) links, and the process of image build (P-2). In addition to the vulnerabilities listed earlier, the lack of container image governance is another vulnerability. Docker Hub is an open registry which is accessible by a private (paid membership) or community user. The images are freely uploaded and stored with no patch management or threats scanning rigor. Its integration into the automated CI/CD pipeline process further increases the attack surface.

Repudiation occurs when an attacker denies an action which he has performed. The logs of a container is not stored in itself as the container is stateless and therefore the kernel will store the logs in the host storage (vulnerability V9). Due to the shared resources characteristic of co-locating containers (V12), an attacker can use a compromised container to access the kernel (P-7) to disable, modify or overwrite logs at the host storage.

Information disclosure causes information to be revealed to attackers. The attackers will attempt to gain access to data stores at GitHub (DS-1) and Docker Hub (DS-2) to steal information about accounts, source codes, sensitive data, configuration files, etc. A skilled attacker can exploit the sensitive parameters used during the container configuration (P-3) to gain access to files in the host. He can also use the common shared network at the kernel (P-7) to connect two co-locating containers and to exchange unauthorised information.

Denial of service (DoS) makes the system inaccessible for use. DoS can occur when an attack happens at each of the connecting "pipe" (DF-1,2,3,4,5) that links the elements in the container DFD system. A breakage in a connection will result in a change or patch in the application code not being updated in the final image and not deployed or updated in the application container. Proven tactics targeted at the resource isolation measures in the kernel can cause the host resources (eg. CPU, storage) to be inaccessible.

Elevation of privilege grants the attacker access and control of the system. This is a serious threat which allows the attacker to take control (TC-4) of the host and carry out further damages. The tight integration of the container with the Linux kernel is a critical vulnerability (V10, V11, V12). Therefore, an attacker with access to a compromised container can utilize the Docker daemon (P-6) via exposed network ports and privilege system calls to attack the kernel (P-7) to obtain root control of the host.

5.2. Future Research Directions

Based on the above analysis, there are some areas which are open for further research. In our STRIDE threat modeling exercise, we focus on the "supply chain" from the code repository (using GitHub), to the image registry (Docker Hub), and finally to the Docker host with emphasis on the six elements of STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege). However, the wider and more holistic container ecosystem is connected and overlapped with the cloud and IoT ecosystems. Containers are used to build the cloud and IoT systems, and at the same time the cloud and IoT are using containers to run applications [168].

Vulnerabilities in IoT containers Gartner predicts that the number of IoT devices will double every five years and it will reach 15 billion IoT devices by 2029, and they pose security risks to the enterprise infrastructures [169]. Therefore, this is an important field to study the expanded attack vectors presented by the relationships between the container, cloud and IoT systems.

Enhancement of container engine security In this paper,

[7] https://bitinfocharts.com/comparison/bitcoin-transactionfees.html

we use Docker as the representative container engine for security survey as it is the most popular and pervasively used by enterprises and businesses. However, a couple of reports state that an alternative container engine called Kata[8] container which is developed by IBM and Hyper.sh can offer better security isolation while maintaining efficiency and performance and it has a strong reference customer in the form of Baidu AI Cloud [170], [31], [171]. Therefore, another direction of study is a comprehensive comparison of the security and performance between Kata container and Docker container and investigate the possibility of a Docker substitute or areas for Docker's security enhancements.

Security of alternative container technology In recent years, there have been studies on Unikernel and its advantages of small footprint, speed and a reduced attack surface [172],[173],[174],[175],[176]. This technology presents a useful area of study to determine the feasibility of replacing the container technology in order to reduce the vulnerabilities faced by the current container technology.

Vulnerabilities of containers using different kernels There are no comparison studies of container security between one which is based on Linux vs one based on Windows. Both the Linux and Windows kernels are designed differently and there is a large Windows application installed base and therefore it is of interest to know the comparative security strengths and weaknesses between the two. So far, most of the security analysis of the Windows and Linux operating systems were carried out several years ago and were considered out-dated [177],[178],[179],[180],[181].

Evaluation of container scanning tools There is little study about container vulnerability and threats detection tools and the evaluations of their performances. To date, there are many container image scanning tools such as Clair, Anchore, Trivy, etc. [182] but few research into their effectiveness, their gaps and their impacts to the container's security. Javed and Toor of [140] evaluated three scanners of Clair, Anchore and Microscanner in terms of the detection coverage and detection hit ratio for only 59 Docker Java-based images. Tunde-Onadele et al. [183] compared the detection accuracy of a static scanner (Clair) and a dynamic runtime detection scheme which analyzed the system call features using machine learning methods, like K-means, Self-Organizing Map and others to detect anomaly. Therefore, there is a need to study the available vulnerability detection methods and tools and to carry out a comprehensive evaluation of them.

Vulnerabilities relevant to language-based application containers The top three programming languages used in Docker container application images in 2020 are Python, Go and Javascript and they have been growing steadily since 2015 [184]. There is no known study of the types of threats and vulnerabilities that occur in the container and their attribution to the programming languages. Therefore, it is useful to the industry for researches to be carried out on the vulnerabilities of the different language-based applications when deployed in a container architecture.

An end-to-end practical guide to securing containers There is no one structured and integrated approach for container security. Today, each security tool or process only targets a specific area and to address it independently. For example, in the code build phase a developer will need to remember to scan the image, keep credentials in the secret vaults, verify the image signature and to sign it when pushing it to the registry, and all of these steps require different tools and processes. During the pull and deployment phase, a developer will need to scan the image for new vulnerabilities and to configure least privileges, network segmentation and least kernel interaction (e.g., minimal system calls) in runtime. The developer will then need to ensure the integrity of the images (e.g., patch and re-image) throughout the lifecycle of the container and to run monitoring and logging mechanisms to keep the container and its users safe. The National Institute of Standards and Technology (NIST) published a comprehensive container security guide in 2017 [185] and it contained recommendations of best practices for specific components in a container architecture but did not provide working level details and its application in practical use-cases (e.g., via code repo, image registry, deployment, etc). Therefore, there is a need for the research community to produce industry relevant and practical guides for container security.

[8] https://katacontainers.io/

Table 3: Summary of our STRIDE analysis

STRIDE	Affected DFD Components	Vulnerabilities	Threat Actions	Threat Consequences	The Existing Mitigation Strategies	Limitations of the Mitigation Strategies
Spoofing	DS-1	V1	Spoof Github account by stealing credentials to gain access to GitHub account and to upload malicious codes.	TC-1, TC-2	MFA to protect account, scan image for vulnerabilities.	2FA increases time and cost, SMS verification code can be attacked, and one-time passcodes generated by phone authenticator app can be stolen.
	DS-1, DS-2	V2	Spoof GitHub or Docker Hub by using DNS hijack & others.	TC-2	Protect network, use VPN, sign image, and scan image.	VPN introduces delay, and private keys for signature can be stolen.
	DS-2	V2	Spoofing of Docker Hub account and image by exploiting typo squatting and "almost-similar name" image.	TC-2, TC-4	Use official or verified image, and scan image before upload to registry or download for deployment.	Container scanner is not foolproof and 34% of vulnerabilities are undetected.
	P-4, P-5	V12	Spoofing of DNS responses to all the container applications running on the Kubernetes cluster and to execute MITM attack on the network traffic between the containers.	TC-2	Protect network, use VPN, and limit capabilities permissions in container (e.g. NET_RAW).	Same as above.
Tampering	DF-4, DF-5	V5	MITM by attacker to insert malicious image in the connection between Docker Hub and the Docker host.	TC-1, TC-2, TC-3	Encrypt network, check for signature in image, verify hash, and scan image.	Private keys for signature can be stolen, and container scanners are not highly accurate.
	DF-1, DF-2, DF-3, DF-4, DF-5	V7	During the auto CI/CD pipeline, attacker can insert tampered and malicious images into any stage of the pipeline.	TC-1, TC-3	Protect network pipeline, scan code/image at each stage, sign image and verify it during deployment.	Container scanners are not highly accurate.
	DS-2	V2, V4	Images in Docker Hub being tampered after attackers hacked into accounts. A vulnerability in an image takes an average of 181 days for it to be fixed and an extra 422 days to be updated.	TC-1, TC-2	Scan image, sign image, and verify it during deployment, verify hash, and regular patching of image.	Same as above. Patching is manual, no testing for app compatibility before patch.
	P-2	V2, V3	When the image is built, malicious commands are injected into the image or tampered libraries are used in the application.	TC-1, TC-2, TC-4	Same as above. Keep image minimal, use multistage build, and use distroless images.	Same as above.
Repudiation	P-7	V9, V10	Disable logging, and modify logs.	TC-2, TC-3	Use message authentication code (MAC) and signature. Apply blockchain to distribute and immutabilize logs.	Transaction fees in blockchain is costly in the long term.

	P-7	V9, V10	Overwrite log disk space with junk.	TC-2	Use log drivers to store logs locally or to remote endpoints.	Local storage limit log size, and network failure causes logs to be lost.
Information Disclosure	DS-1, DS-2	V1	API and identity keys are exposed for attacker to take control of accounts in Github and Docker Hub.	TC-1	Do not store credentials and secrets in clear, keep them in "vaults". Use .gitignore to avoid uploading sensitive info during commit.	Credentials in "vaults" and compromised endpoints can be stolen, and user's negligence in sharing credentials
	P-3	V10, V12	Include sensitive parameters in the run command when deploying container.	TC-1, TC-4	Exercise diligence in not exposing sensitive parameters, scan for sensitive parameters and to raise alerts.	No significant limitation is reported or observed.
	P-7	V10, V12	Leakage of information between containers on the same host.	TC-1	Same as above. Harden host configuration, limit file permissions, and configure TLS for connections.	No significant limitation is reported or observed.
Denial of Service	P-6, P-7	V10	Service disruption at the Host via kernel due to exception handling, disk write-back, logging, and others.	TC-3	Hardened configuration of cgroups to limit host resources usages eg. read-only filesystems, limit kernel calls, limit network communications, and use memory management module like MemGuard.	Cgroups and namespaces isolation are subjected to container escape and network attacks.
	DF-1, DF-2, DF-3, DF-4, DF-5	V7	Any of the data flow pipes become disrupted to perform transmission of codes/images.	TC-3	Install intrusion detection (IDS) and prevention systems (IPS) to protect the network connectivity.	IDS can result in "alert fatigue" and IPS takes up network bandwidth.
Elevation of Privilege	P-2	V10, V11, V12	Run container as root when it is not necessary.	TC-4	Harden container configuration to just-enough privileges or run as "non-root mode".	No significant limitation is reported or observed.
	P-6	V10, V11, V12	Misconfiguration with network ports open.	TC-4	Scan network ports, do not expose Docker daemon socket and other unnecessary ports.	No significant limitation is reported or observed.
	P-6	V10, V11, V12	Enabling excessive systems calls	TC-4	Trace system calls and reduce unnecessary ones, analyse systems calls traffic and use Machine Learning techniques to detect anomaly, apply principle of privilege separation and partition container to smaller isolating containers.	Current ML techniques are not highly accurate and not tested for most use-cases.
	P-7	V10, V11, V12	Memory attack by overcoming security of Linux and using TOCTOU techniques.	TC-4	Same as above.	Same as above

6. Conclusion

The advancement of containers has helped enterprises and organizations to improve their processes and enable new business models. However, its full utilization has been daunted by the various security risks posed in the containers ecosystem. In this paper, we first assessed the security landscape in containers. In particular, we used the STRIDE framework to identify vulnerabilities, threats and threat consequences on the entire container ecosystem. From our study, we found that many of the vulnerabilities are due to the containers' shared access to the host operating system's kernel. While there were isolation measures (e.g., namespaces) and resource control mechanisms (e.g., cgroups) in place, these could be breached when misconfigurations and liberal use of system calls and capabilities happened. From the ecosystem perspective, the numerous external entities who involved in writing the code, building the image, configuring the installation, setting up the network connectivities, and eventually deploying the application in production containers greatly increased the attack surfaces.

Then, we conducted a systematic survey on the existing works on containers security. In particular, we assessed the strengths and weaknesses of existing mitigation strategies against the identified security threats in containers. Based on our assessment, most of the existing mitigation strategies have certain limitations and not sufficient to address the security risks posed to the container systems. Therefore, we have also outlined several areas of future research directions to enhance the security of containers. We hope this paper will help practitioners and researchers to be aware of the current threat landscape and security gaps in containers, and open up areas for further explorations and studies.

Acknowledgment

We would like to thank our peers and colleagues for their valuable feedback. Any opinions, conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the universities.

References

[1] Google, Containers at google, online (Apr 2021).
URL https://cloud.google.com/containers#:~:text=Containers%20give%20developers%20the%20ability,runtimes%20and%20other%20software%20libraries..

[2] C. Hall, Netflix's container management system is now open source, online (Apr 2018).
URL https://www.datacenterknowledge.com/cloud/netflixs-container-management-system-now-open-source

[3] J. Armstrong, The journey to 150,000 containers at paypal, online (Dec 2017).
URL https://m-square.com.au/the-journey-to-150000-containers-at-paypal/

[4] S. Williams, Gartner: Strong revenue growth forecast for container management software and services, online (Jun 2020).
URL https://datacenternews.asia/story/gartner-strong-revenue-growth-forecast-for-container-management-software-and-services

[5] M. Vizard, Sysdig report shines light on container usage patterns, online (Oct 2019).
URL https://containerjournal.com/topics/container-ecosystems/sysdig-report-shines-light-on-container-usage-patterns/

[6] H. Barua, Half of 4 million public docker hub images found to have critical vulnerabilities, online (Dec 2020).
URL https://www.infoq.com/news/2020/12/dockerhub-image-vulnerabilities/

[7] R. Field, Attackers found building malicious container images directly on host, online (Sep 2020).
URL https://www.infoq.com/news/2020/09/Malicious-Container-Images/

[8] C. Cimpanu, A hacking group is hijacking docker systems with exposed api endpoints, online (Nov 2019).
URL https://www.zdnet.com/article/a-hacking-group-is-hijacking-docker-systems-with-exposed-api-endpoints/

[9] M. Vizard, Latest docker container attack highlights remote networking flaws, online (Aug 2020).
URL https://containerjournal.com/topics/container-security/latest-docker-container-attack-highlights-remote-networking-flaws/

[10] MITRE, Cve, online (Nov 2022).
URL https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=container

[11] T. Seals, Tesla falls to crypto-jackers, online (Feb 2018).
URL https://www.infosecurity-magazine.com/news/tesla-falls-to-cryptojackers/

[12] S. Nichols, Russia using kubernetes cluster for brute-force attacks, online (Jul 2021).
URL https://searchsecurity.techtarget.com/news/252503482/Russia-using-Kubernetes-cluster-for-brute-force-attacks

[13] K. Matthews, Incident of the week: Impact of docker security breach, online (May 2019).
URL https://www.cshub.com/attacks/articles/incident-of-the-week-impact-of-docker-security-breach

[14] K. Townsend, Attacks against container infrastructures increasing, including supply chain attacks, online (Jun 2021).
URL https://www.securityweek.com/attacks-against-container-infrastructures-increasing-including-supply-chain-attacks

[15] M. Jarvis, Privileged docker containers—do you really need them?, online (Nov 2020).
URL https://snyk.io/blog/privileged-docker-containers/

[16] A. Morag, G. Singer, Threat alert: Market-first container image built to attack kubernetes clusters, online (Nov 2020).
URL https://blog.aquasec.com/kubernetes-vulnerability-security-threat

[17] A. Remillano, II, Malicious docker hub container images used for cryptocurrency mining, Trend MicroOnline (Aug 2020).
URL https://www.trendmicro.com/vinfo/us/security/news/virtualization-and-cloud/malicious-docker-hub-container-images-cryptocurrency-mining

[18] D. Sagi, Dns spoofing on kubernetes clusters, online (Aug 2019).
URL https://blog.aquasec.com/dns-spoofing-kubernetes-clusters

[19] N. Chako, Attacking kubernetes clusters through your network plumbing: Part 1, online (May 2020).
URL https://www.cyberark.com/resources/threat-research-blog/attacking-kubernetes-clusters-through-your-network-plumbing-part-1

[20] A. Martin, S. Raponi, T. Combe, R. Pietro, Docker ecosystem – vulnerability analysis, Computer Communications 122 (2018) 30–43,.

[21] S. Kerner, Build and ship any application anywhere, online (Apr 2019).
URL https://www.eweek.com/security/docker-hub-breached-impacting-190-000-accounts/

[22] P. Liu, S. Ji, L. Fu, K. Lu, X. Zhang, W.-H. Lee, T. Lu, W. Chen, R. Beyah, Understanding the security risks of docker hub, in: European Symposium on Research in Computer Security – ESORICS 2020, 2020.

[23] Y. Shen, X. Yu, Docker container hardening method based on trusted computing, Journal of Physics: Conference Series 1619 (2020) 012014. doi:10.1088/1742-6596/1619/1/012014.
URL https://doi.org/10.1088/1742-6596/1619/1/012014

[24] Y. Luo, W. Luo, X. Sun, Q. Shen, A. Ruan, Z. Wu, Whispers between the containers: High-capacity covert channel attacks in docker, in: 2016 IEEE Trustcom/BigDataSE/ISPA, 2016, pp. 630–637. doi:10.1109/TrustCom.2016.0119.

[25] RedLock CSI Team, Lessons from the cryptojacking attack at tesla, on-

line (Feb 2018).
URL https://redlock.io/blog/cryptojacking-tesla.

[26] X. Gao, Z. Gu, Z. Li, H. Jamjoom, C. Wang, Houdini's escape: Breaking the resource rein of linux control groups, in: 2019 ACM SIGSAC Conference on Computer and Communications, London, 2019.

[27] V. Pavišić, User privileges in docker containers, online (Apr 2019).
URL https://medium.com/jobteaser-dev-team/docker-user-best-practices-a8d2ca5205f4

[28] T. Combe, A. Martin, R. Pietro, To docker or not to docker: A security perspective, IEEE Cloud Computing 3 (5) (2016) 54–62.

[29] A. Duarte, N. Antunes, An empirical study of docker vulnerabilities and of static code analysis applicability, in: 2018 Eighth Latin-American Symposium on Dependable Computing (LADC), 2018, pp. 27–36. doi:10.1109/LADC.2018.00013.

[30] K. Wist, M. Helsem, D. Gligoroski, Vulnerability analysis of 2500 docker hub images, CoRR abs/2006.02932 (2020). arXiv:2006.02932.
URL https://arxiv.org/abs/2006.02932

[31] O. Flauzac, F. Mauhourat, F. Nolot, A review of native container security for running applications, Procedia Computer Science 175 (2020) 157–164, the 17th International Conference on Mobile Systems and Pervasive Computing (MobiSPC),The 15th International Conference on Future Networks and Communications (FNC),The 10th International Conference on Sustainable Energy Information Technology. doi:https://doi.org/10.1016/j.procs.2020.07.025.
URL https://www.sciencedirect.com/science/article/pii/S187705092031704X

[32] S. Sultan, I. Ahmad, T. Dimitriou, Container security: Issues, challenges, and the road ahead, IEEE Access 7 (2019) 52976–52996. doi:10.1109/ACCESS.2019.2911732.

[33] X. Lin, L. Lei, Y. Wang, J. Jing, K. Sun, Q. Zhou, A measurement study on linux container security: Attacks and countermeasures, in: Proceedings of the 34th Annual Computer Security Applications Conference, 2018.

[34] J. Burns, Att&ck® for containers now available!, online (Apr 2021).
URL https://medium.com/mitre-engenuity/att-ck-for-containers-now-available-4c2359654bf1

[35] "MITRE", Containers matrix att&ck, online (Sep 2021).
URL https://attack.mitre.org/matrices/enterprise/containers/

[36] D. Oh, 10 layers of linux container security, online (Oct 2017).
URL https://opensource.com/article/17/10/10-layers-container-security#:~:text=Containers%20are%20Linux%20processes%20with,and%20still%20the%20best%20practice..

[37] Red Hat, Linux capabilities and seccomp, online (May 2021).
URL https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html/container_security_guide/linux_capabilities_and_seccomp

[38] T. Bui, Analysis of docker security, CoRR abs/1501.02967 (2015).

[39] Docker, Enable two-factor authentication for docker hub, online (Sep 2021).
URL https://docs.docker.com/docker-hub/2fa/#:~:text=To%20enable%20two-factor%20authentication%2C%20log%20in%20to%20your,authenticator%20app.%20Click%20Set%20up%20using%20an%20app.

[40] Docker, Use multi-stage builds, online (Aug 2021).
URL https://docs.docker.com/develop/develop-images/multistage-build/

[41] A. Iradier, Top 20 dockerfile best practices, online (Mar 2021).
URL https://sysdig.com/blog/dockerfile-best-practices/

[42] Docker, Content trust in docker, online (Jun 2021).
URL https://docs.docker.com/engine/security/trust/

[43] Y. Sun, D. R. Safford, M. Zohar, D. Pendarakis, Z. Gu, T. Jaeger, Security namespace: Making linux security frameworks available to containers, in: USENIX Security Symposium, 2018.

[44] S. Bhat, 5 open source tools for container security, online (Aug 2018).
URL https://opensource.com/article/18/8/tools-container-security

[45] K. Brady, S. Moon, T. Nguyen, J. Coffman, Docker container security in cloud computing, in: 2020 10th Annual Computing and Communication Workshop and Conference (CCWC), 2020. doi:10.1109/ccwc47524.2020.9031195.

[46] M. Souppaya, K. Stine, M. Simos, K. Scarfone, Critical cybersecurity hygiene: Patching the enterprise, online (Mar 2020).
URL https://www.nccoe.nist.gov/projects/building-blocks/patching-enterprise

[47] CIS, Cis docker benchmark, online (May 2021).
URL https://www.cisecurity.org/benchmark/docker/

[48] Docker, Run the docker daemon as a non-root user (rootless mode), online (Sep 2021).
URL https://docs.docker.com/engine/security/rootless/

[49] L. Lei, J. Sun, K. Sun, C. Shenefiel, R. Ma, Y. Wang, Q. Li, Speaker: Split-phase execution of application containers, in: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, 2017.

[50] N. Stoler, G. Reti, The strange case of how we escaped the docker default container, online (Mar 2021).
URL https://www.cyberark.com/resources/threat-research-blog/the-strange-case-of-how-we-escaped-the-docker-default-container

[51] M. Howard, S. Lipner, The Security Development Lifecycle, Microsoft Press, USA, 2006.

[52] N. Shevchenko, T. Chick, P. O'Riordan, T. Scanlon, C. Woody, Threat Modeling: A Summary of Available Methods, Carnegie Mellon University, Software Engineering Institute, Pittsburgh, 2018.

[53] R. Scandariato, K. Wuyts, W. Joosen, A descriptive study of microsoft's threat modeling technique, Requirements Engineering 20 (2) (2015) 163–180. doi:10.1007/s00766-013-0195-2.

[54] R. Khan, K. McLaughlin, D. Laverty, S. Sezer, Stride-based threat modeling for cyber-physical systems, in: 2017 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe), Turin, Italy, 2017.

[55] L. Sion, K. Yskout, D. Van Landuyt, W. Joosen, Solution-aware data flow diagrams for security threat modeling, Proceedings of the 33rd Annual ACM Symposium on Applied Computing (2018).

[56] N. Mead, F. Shull, The hybrid threat modeling method, online (Apr 2018).
URL https://resources.sei.cmu.edu/asset_files/TechnicalNote/2018_004_001_516627.pdf

[57] A. Karahasanovic, P. Kleberger, M. Almgren, Adapting threat modeling methods for the automotive industry, Proceedings of Escar Europe conference | Embedded Security in Cars (2017).

[58] IBM, Docker, online (Jun 2021).
URL https://www.ibm.com/cloud/learn/docker

[59] B. Golden, 3 reasons why you should always run microservices apps in containers, online (May 2021).
URL https://techbeacon.com/app-dev-testing/3-reasons-why-you-should-always-run-microservices-apps-containers

[60] G. Liu, B. Huang, Z. Liang, M. Qin, H. Zhou, Z. Li, Microservices: architecture, container, and challenges, in: 2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C), 2020, pp. 629–635. doi:10.1109/QRS-C51114.2020.00107.

[61] IBMCloud Education, Microservices, online (Mar 2021).
URL https://www.ibm.com/cloud/learn/microservices

[62] X. Lin, P. Zavarsky, R. Ruhl, D. Lindskog, Threat modeling for csrf attacks, in: International Conference on Computational Science and Engineering, 2009.

[63] T. UcedaVelez, M. M. Morana, Risk Centric Threat Modeling: process for attack simulation and threat analysis, John Wiley & Sons, 2015.

[64] Nick, Pasta threat modeling, online (July 2022).
URL https://threat-modeling.com/pasta-threat-modeling/

[65] Threatmodeler, Threat modeling methodologies: What is vast?, online (Oct 2018).
URL https://threatmodeler.com/threat-modeling-methodologies-vast/

[66] P. Saitta, B. Larcom, M. Eddington, Trike v.1 methodology document [draft], White Paper, available at http://www.octotrike.org/ (July 2005).

[67] C. Alberts, A. Dorofee, J. Stevens, C. Woody, Introduction to the octave approach, online (August 2003).
URL https://resources.sei.cmu.edu/asset_files/UsersGuide/2003_012_001_51556.pdf

[68] M. Souppaya, K. Scarfone, Guide to data-centric system threat modeling, online (March 2016).
URL https://csrc.nist.gov/publications/detail/sp/800-154/draft

[69] H. von Scheel, M. von Rosing, M. Hove, M. Fonseca, U. Foldager,

Phase 2: Process concept evolution, in: M. von Rosing, A.-W. Scheer, H. von Scheel (Eds.), The Complete Business Process Handbook, Morgan Kaufmann, Boston, 2015, pp. 11–35. doi:https://doi.org/10.1016/B978-0-12-799959-3.00002-1.
URL https://www.sciencedirect.com/science/article/pii/B9780127999593000021

[70] S. Hernan, S. Lambert, T. Ostwald, A. Shostack, Threat modeling:uncover security design flaws using the stride approach, online (July 2019).
URL https://docs.microsoft.com/en-us/archive/msdn-magazine/2006/november/uncover-security-design-flaws-using-the-stride-approach

[71] W. Gamage, Common container security threats, online (Nov 2019).
URL https://www.wwt.com/article/common-container-security-threats

[72] C. Cimpanu, 17 backdoored docker images removed from docker hub, online (Jun 2018).
URL https://www.bleepingcomputer.com/news/security/17-backdoored-docker-images-removed-from-docker-hub/

[73] A. Grattafiori, Understanding and hardening linux containers, online (June 2016).
URL https://research.nccgroup.com/wp-content/uploads/2020/07/ncc_group_understanding_hardening_linux_containers-1-1.pdf

[74] J. Cito, G. Schermann, J. E. Wittern, P. Leitner, S. Zumberi, H. C. Gall, An empirical analysis of the docker container ecosystem on github, in: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), 2017, pp. 323–333. doi:10.1109/MSR.2017.67.

[75] Docker, Docker id accounts, online (Sep 2021).
URL https://docs.docker.com/docker-id/

[76] G. Docs", Creating a strong password, online (Jul 2021).
URL https://docs.github.com/en/github/authenticating-to-github/keeping-your-account-and-data-secure/creating-a-strong-password

[77] SailPoint, 8 types of password attacks, online (Feb 2021).
URL https://www.sailpoint.com/identity-library/8-types-of-password-attacks/

[78] Docker, Repositories, online (Sep 2021).
URL https://docs.docker.com/docker-hub/repos/

[79] M. Sequeira, Low-hanging secrets in docker hub and a tool to catch them all, online (Nov 2020).
URL https://ioactive.com/guest-blog-docker-hub-scanner-matias-sequeira/

[80] docker, Docker official images impacted by log4j 2 cve, online (May 2022).
URL https://docs.docker.com/security/

[81] Docker, Docker registry http api v2, online (Jul 2021).
URL https://docs.docker.com/registry/spec/api/

[82] CVE, Vulnerability details : Cve-2017-18641, online (Feb 2020).
URL https://www.cvedetails.com/cve/CVE-2017-18641/

[83] Docker, Docker security, online (Sep 2021).
URL https://docs.docker.com/engine/security/

[84] Docker, Best practices for using docker hub for ci/cd, online (Aug 2021).
URL https://docs.docker.com/ci-cd/best-practices/

[85] sematext, Docker logging: A complete guide, online (Jul 2021).
URL https://sematext.com/guides/docker-logs/

[86] Docker, Configure logging drivers, online (Jul 2021).
URL https://docs.docker.com/config/containers/logging/configure/

[87] J. Chelladhurai, P. R. Chelliah, S. A. Kumar, Securing docker containers from denial of service (dos) attacks, in: 2016 IEEE International Conference on Services Computing (SCC), 2016, pp. 856–859. doi:10.1109/SCC.2016.123.

[88] Red Hat, runc - malicious container escape - cve-2019-5736, online (Apr 2020).
URL https://access.redhat.com/security/vulnerabilities/runcescape

[89] CVE Details, Linux kernel: Vulnerability statistics, online (Aug 2021).
URL https://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendorid=33

[90] Z. Jian, L. Chen, A defense method against docker escape attack, in: Proceedings of the 2017 International Conference on Cryptography, Security and Privacy, ICCSP '17, Association for Computing Machinery, New York, NY, USA, 2017, p. 142–146. doi:10.1145/3058060.3058085.
URL https://doi.org/10.1145/3058060.3058085

[91] R. Shirey, Internet security glossary, online. IETF Request for comments 4949 (Aug 2007).
URL https://datatracker.ietf.org/doc/html/rfc4949

[92] J. Sirkin, Github repositories leak thousands of secrets, study shows, online (Nov 2019).
URL https://www.cyberark.com/resources/blog/github-repositories-leak-thousands-of-secrets-study-shows

[93] R. A. Sandvik, Attackers scrape github for cloud service credentials, hijack account to mine virtual currency, online (Jan 2014).
URL https://www.forbes.com/sites/runasandvik/2014/01/14/attackers-scrape-github-for-cloud-service-credentials-hijack-account-to-mine-virtual-currency/?sh=5ee6e5f83196

[94] P. Ramesh, D. Bhaskari, Ch.Satyanarayana, A comprehensive analysis of spoofing, International Journal of Advanced Computer Science and Applications 1 (6) (2010). doi:10.14569/ijacsa.2010.010623.

[95] T. H. Kim, D. Reeves, A survey of domain name system vulnerabilities and attacks, Journal of Surveillance, Security and Safety (2020). doi:10.20517/jsss.2020.14.

[96] Red Hat, Kubernetes adoption, security, and market trends report 2021, online (Jul 2021).
URL https://www.redhat.com/en/resources/kubernetes-adoption-security-market-trends-2021-overview

[97] S. Garg, S. Garg, Automated cloud infrastructure, continuous integration and continuous delivery using docker with robust container security, in: 2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR), 2019, pp. 467–470. doi:10.1109/MIPR.2019.00094.

[98] F. Massacci, T. Jaeger, S. Peisert, Solarwinds and the challenges of patching: Can we ever stop dancing with the devil?, IEEE Security Privacy 19 (2) (2021) 14–19. doi:10.1109/MSEC.2021.3050433.

[99] C. for Internet Security, The solarwinds cyber-attack: What you need to know, online (Mar 2021).
URL https://www.cisecurity.org/solarwinds

[100] R. Shu, X. Gu, W. Enck, A study of security vulnerabilities on docker hub, in: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, CODASPY '17, Association for Computing Machinery, New York, NY, USA, 2017, p. 269–280. doi:10.1145/3029806.3029832.
URL https://doi.org/10.1145/3029806.3029832

[101] F. Y. Rashid, Most applications contain vulnerable open source libraries, online (May 2020).
URL https://duo.com/decipher/most-applications-contain-vulnerable-open-source-libraries

[102] L. Tal, 88% increase in application library vulnerabilities over two years, online (Feb 2019).
URL https://snyk.io/blog/88-increase-in-application-library-vulnerabilities-over-two-years/

[103] J. Greig, 96% of third-party container applications deployed in cloud infrastructure contain known vulnerabilities: Unit 42, online (Sep 2021).
URL https://www.zdnet.com/article/96-of-third-party-container-applications-deployed-in-cloud-infrastructure-contain-known-vulnerabilities-unit-42/

[104] D. Everson, L. Cheng, Z. Zhang, Log4shell: Redefining the web attack surface, NDSS Symposium - Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb) (2022).
URL https://dx.doi.org/10.14722/madweb.2022.23010

[105] G. Belding, Ethical hacking: Log tampering 101, online (Sep 2019).
URL https://resources.infosecinstitute.com/topic/ethical-hacking-log-tampering-101/

[106] A. Rahic, Where are docker container logs stored?, online (Apr 2020).
URL https://sematext.com/blog/docker-logs-location/

[107] D. Fiser, A. Oliveira, Why a privileged container in docker is a bad idea, online (Dec 2019).
URL https://www.trendmicro.com/en_sg/research/19/l/why-running-a-privileged-container-in-docker-is-a-bad-idea.html

[108] I. Ilascu, Starbucks devs leave api key in github public repo, online (Dec 2019).
URL https://www.bleepingcomputer.com/news/security/starbucks-devs-leave-api-key-in-github-public-repo/

[109] A. Joshi, Circleci reports of a security breach and malicious database in a third-party vendor account, online (Sep 2019).

URL https://hub.packtpub.com/circleci-reports-of-a-security-breach-and-malicious-database-in-a-third-party-vendor-account/

[110] C. Cimpanu, A hacker is wiping git repositories and asking for a ransom, online (May 2019).
URL https://www.zdnet.com/article/a-hacker-is-wiping-git-repositories-and-asking-for-a-ransom/

[111] Cimpanu, Catalin, Personal data of 16 million brazilian covid-19 patients exposed online, online (Nov 2020).
URL https://www.zdnet.com/article/personal-data-of-16-million-brazilian-covid-19-patients-exposed-online/

[112] C. Cimpanu, Nissan source code leaked online after git repo misconfiguration, online (Jan 2021).
URL https://www.zdnet.com/article/nissan-source-code-leaked-online-after-git-repo-misconfiguration/

[113] Cimpanu, Catalin, Mercedes-benz onboard logic unit (olu) source code leaks online, online (May 2020).
URL https://www.zdnet.com/article/mercedes-benz-onboard-logic-unit-olu-source-code-leaks-online/

[114] S. Breach, Solarwinds leaked ftp credentials through a public github repo "mib-importer" since 2018, online (Dec 2020).
URL https://savebreach.com/solarwinds-exposed-ftp-credentials-back-in-2018-says-security-researcher-vinoth/

[115] Docker, Docker run reference, online (Jul 2021).
URL https://docs.docker.com/engine/reference/run/

[116] Z. Anwar, A. W. Malik, Can a ddos attack meltdown my data center? a simulation study and defense strategies, IEEE Communications Letters 18 (7) (2014) 1175–1178. doi:10.1109/LCOMM.2014.2328587.

[117] L. Tung, Google: This is what caused cpu throttling at our cloud data center, online (Mar 2020).
URL https://www.zdnet.com/article/google-this-is-what-caused-cpu-throttling-at-our-cloud-data-center/

[118] Tung, Liam, Microsoft: Here's what caused our azure cloud-computing outage, online (Apr 2021).
URL https://www.zdnet.com/article/microsoft-heres-what-caused-our-recent-azure-cloud-computing-services-outage/

[119] L. Tung, Amazon: Here's what caused the major aws outage last week, online (Nov 2020).
URL https://www.zdnet.com/article/amazon-heres-what-caused-major-aws-outage-last-week-apologies/

[120] Docker, Runtime options with memory, cpus, and gpus, online (Aug 2021).
URL https://docs.docker.com/config/containers/resource_constraints/

[121] D. Bovet, C. Marco, Understanding the Linux Kernel, O'Reilly Media Inc, 2007.

[122] R. Gheorghe, Tutorial: Logging with journald, online (Apr 2020).
URL https://sematext.com/blog/journald-logging-tutorial/

[123] J. Haas, A step-by-step guide to using the linux 'sync' command, online (Sep 2020).
URL https://www.lifewire.com/sync-linux-command-4091818

[124] L. Rice, Boosting container security with rootless containers, online (Jan 2021).
URL https://blog.aquasec.com/rootless-containers-boosting-container-security

[125] Docker, Use bridge networks, online (Apr 2021).
URL https://docs.docker.com/network/bridge/#differences-between-user-defined-bridges-and-the-default-bridge

[126] S. Shevchenko, Kinsing punk: An epic escape from docker containers, online (Aug 2020).
URL https://www.prevasio.io/blog/kinsing-punk-an-epic-escape-from-docker-containers

[127] D. Fisher, Docker bug allows root access to host file system, online (May 2019).
URL https://duo.com/decipher/docker-bug-allows-root-access-to-host-file-system

[128] S. Nahari, Best defense? our red team lead reveals 4 mfa bypass techniques, online (Jun 2021).
URL https://www.cyberark.com/resources/threat-research-blog/mfa-bypass-techniques-from-red-team-research

[129] Github Docs, Two-factor authentication, online. Accessed on 26 Jun 2021.
URL https://docs.gitlab.com/ee/user/profile/account/two_factor_authentication.html

[130] Microsoft 365, Password policy recommendations, online (Jul 2021).
URL https://docs.microsoft.com/en-us/microsoft-365/admin/misc/password-policy-recommendations?view=o365-worldwide

[131] Z. Doffman, Why you should stop using sms security codes—even on apple imessage, online (Oct 2020).
URL https://www.forbes.com/sites/zakdoffman/2020/10/11/apple-iphone-imessage-and-android-messages-sms-passcode-security-update/?sh=203086bc2ede

[132] C. Cimpanu, Android malware can steal google authenticator 2fa codes, online (Feb 2020).
URL https://www.zdnet.com/article/android-malware-can-steal-google-authenticator-2fa-codes/

[133] Github, Notary, online (Jul 2021).
URL https://github.com/theupdateframework/notary

[134] Docker, Pricing & subscriptions, online (Jun 2021).
URL https://www.docker.com/pricing?utm_source=docker&utm_medium=webreferral&utm_campaign=docs_driven_upgrade

[135] G. Docs", About code scanning with codeql, online (Sep 2021).
URL https://docs.github.com/en/code-security/code-scanning/automatically-scanning-your-code-for-vulnerabilities-and-errors/about-code-scanning-with-codeql

[136] J. Palafox, Announcing third-party code scanning tools: static analysis & developer security training, online (Oct 2020).
URL https://github.blog/2020-10-05-announcing-third-party-code-scanning-tools-static-analysis-and-developer-security-training/

[137] DRD, Crack ssh private key passwords with john the ripper, online (Jul 2020).
URL https://null-byte.wonderhowto.com/how-to/crack-ssh-private-key-passwords-with-john-ripper-0302810/

[138] AppViewX, All you need to know about securing your private keys, online (Sep 2019).
URL https://www.appviewx.com/eguide/all-you-need-to-know-about-securing-your-private-keys/

[139] D. Goodwin, Hackers steal secret crypto keys for nordvpn. here's what we know so far, online (Oct 2019).
URL https://arstechnica.com/information-technology/2019/10/hackers-steal-secret-crypto-keys-for-nordvpn-heres-what-we-know-so-far/

[140] O. Javed, S. Toor, Understanding the quality of container security vulnerability detection tools, arXiv:2101.03844v1, (2021).

[141] G. Avner, Docker image security scanning: What it can and can't do, online (Apr 2021).
URL https://www.whitesourcesoftware.com/resources/blog/docker-image-security-scanning/

[142] F. Araujo, T. Taylor, Improving cybersecurity hygiene through jit patching, in: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2020, Association for Computing Machinery, New York, NY, USA, 2020, p. 1421–1432. doi:10.1145/3368089.3417056.
URL https://doi.org/10.1145/3368089.3417056

[143] O. Tunde-Onadele, Y. Lin, J. He, X. Gu, Self-patch: Beyond patch tuesday for containerized applications, in: 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS), 2020, pp. 21–27. doi:10.1109/ACSOS49614.2020.00022.

[144] R. C. Godoy, Why non-root containers are important for security, online (Nov 2018).
URL https://engineering.bitnami.com/articles/why-non-root-containers-are-important-for-security.html

[145] D. Walsh, Running rootless podman as a non-root user, online (Oct 2019).
URL https://www.redhat.com/sysadmin/rootless-podman-makes-sense

[146] S. McCarty, Understanding root inside and outside a container, online (Dec 2019).
URL https://www.redhat.com/en/blog/understanding-root-inside-and-outside-container

[147] G. Provelengios, A. Pouraghily, R. Tessier, T. Wolf, A hardware monitor to protect linux system calls, in: 2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2018, pp. 551–556. doi:10.1109/ISVLSI.2018.00106.

[148] Y. Lin, O. Tunde-Onadele, X. Gu, Cdl: Classified distributed learning

for detecting security attacks in containerized applications, in: ACSAC '20: Annual Computer Security Applications Conference, Austin, 2020.

[149] A. Abed, T. Clancy, D. Levy, Applying bag of system calls for anomalous behavior detection of applications in linux containers, in: 2015 IEEE Globecom Workshops, San Diego, CA, USA, 2015.

[150] D.-K. Kang, D. Fuller, V. Honavar, Learning classifiers for misuse detection using a bag of system calls representation, Intelligence and Security Informatics 3495 (2005) 511–516,.

[151] V. Rastogi, D. Davidson, L. De Carli, S. Jha, P. McDaniel, Cimplifier: Automatically debloating containers, in: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Association for Computing Machinery, New York, NY, USA, 2017, p. 476–486. doi:10.1145/3106237.3106271.
URL https://doi.org/10.1145/3106237.3106271

[152] OWASP, Docker security cheat sheet, online (Aug 2021).
URL https://cheatsheetseries.owasp.org/cheatsheets/Docker_Security_Cheat_Sheet.html

[153] J. Chen, Z. Feng, J. Wen, B. Liu, L. Sha, A container-based dos attack-resilient control framework for real-time UAV systems, CoRR abs/1812.02834 (2018). arXiv:1812.02834.
URL http://arxiv.org/abs/1812.02834

[154] R. H. Bugzilla", Bug 1868453 (cve-2020-14356) - cve-2020-14356 kernel: Use after free vulnerability in cgroup bpf component, online (Jun 2021).
URL https://bugzilla.redhat.com/show_bug.cgi?id=1868453

[155] J. Chen, Making containers more isolated: An overview of sandboxed container technologies, online (Jun 2019).
URL https://unit42.paloaltonetworks.com/making-containers-more-isolated-an-overview-of-sandboxed-container-technologies/

[156] J. Wallen, How to install and use git-secret, online (Jan 2020).
URL https://www.techrepublic.com/article/how-to-install-and-use-git-secret/

[157] Docker, Docker login, online (Jul 2021).
URL https://docs.docker.com/engine/reference/commandline/login/

[158] G. Docs", Ignoring files, online (Jul 2021).
URL https://docs.github.com/en/get-started/getting-started-with-git/ignoring-files

[159] G. Kuizinas, .gitignore mistake that everyone makes, online (Sep 2020).
URL https://dev.to/gajus/gitignore-mistake-that-everyone-makes-44kb

[160] Y. B. Naim, Cyberark labs research: Stealing service credentials to achieve full domain compromise, online (Nov 2016).
URL https://www.cyberark.com/resources/blog/cyberark-labs-research-stealing-service-credentials-to-achieve-full-domain-compromise

[161] M. . ATT&CK", Os credential dumping, online (Sep 2021).
URL https://attack.mitre.org/techniques/T1003/

[162] Kaspersky, What is dns cache poisoning and dns spoofing?, online (Aug 2021).
URL https://www.kaspersky.com/resource-center/definitions/dns

[163] T. Goethals, D. Kerkhove, B. Volckaert, F. D. Turck, Scalability evaluation of vpn technologies for secure container networking, in: 2019 15th International Conference on Network and Service Management (CNSM), 2019, pp. 1–7. doi:10.23919/CNSM46954.2019.9012673.

[164] L. Dwyer, 5 things an ids/ips can't do, online (Mar 2018).
URL https://blog.cygilant.com/blog/5-things-an-ids/ips-cant-do

[165] J. Cucurull, J. Puiggalí, Distributed immutabilization of secure logs, in: Security and Trust Management, Vol. 9871, Springer International Publishing, 2016, pp. 122–137. doi:10.1007/978-3-319-46598-2_9.

[166] Solarwinds, Docker logging strategies, online (Aug 2021).
URL https://documentation.solarwinds.com/en/success_center/loggly/content/admin/strategies-for-docker-logging.htm

[167] Docker, Use docker logs with remote logging drivers, online (Sep 2021).
URL https://docs.docker.com/config/containers/logging/dual-logging/

[168] M. Syed, E. Fernandez, A reference architecture for the container ecosystem, in: Proceedings of the 13th International Conference on Availability, Reliability and Security, 2018.

[169] K. Costello, Gartner predicts the future of cloud and edge infrastructure, online (Feb 2021).
URL https://www.gartner.com/smarterwithgartner/gartner-predicts-the-future-of-cloud-and-edge-infrastructure/

[170] H. Li, The road to kata containers 2.0, online (Jul 2020).
URL https://thenewstack.io/the-road-to-kata-containers-2-0/

[171] R. Kumar, B. Thangaraju, Performance analysis between runc and kata container runtime, in: 2020 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT), 2020, pp. 1–4. doi:10.1109/CONECCT50063.2020.9198653.

[172] C. Shichao, M. Zhou, Evolving container to unikernel for edge computing and applications in process industry, Processes 9 (2021) 351. doi:10.3390/pr9020351.

[173] H.-C. Kuo, D. Williams, R. Koller, S. Mohan, A linux in unikernel clothing, in: Proceedings of the Fifteenth European Conference on Computer Systems, EuroSys '20, Association for Computing Machinery, New York, NY, USA, 2020. doi:10.1145/3342195.3387526.
URL https://doi.org/10.1145/3342195.3387526

[174] P. Olivier, D. Chiba, S. Lankes, C. Min, B. Ravindran, A binary-compatible unikernel, in: Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE 2019, Association for Computing Machinery, New York, NY, USA, 2019, p. 59–73. doi:10.1145/3313808.3313817.
URL https://doi.org/10.1145/3313808.3313817

[175] A. Bratterud, A. Happe, R. Duncan, Enhancing cloud security and privacy: The unikernel solution, in: Eighth International Conference on Cloud Computing, GRIDs, and Virtualization, 19 February 2017 - 23 February 2017, Athens, Greece, Cloud Computing IARIA, Curran Associates, 2017, pp. 79–86, the Eighth International Conferences on Cloud Computing, GRIDs, and Virtualization, CLOUD COMPUTING 2017 ; Conference date: 19-02-2017 Through 23-02-2017.

[176] A. Bratterud, A.-A. Walla, H. Haugerud, P. E. Engelstad, K. Begnum, Includeos: A minimal, resource efficient unikernel for cloud services, in: 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom), 2015, pp. 250–257. doi:10.1109/CloudCom.2015.89.

[177] L. Zeng, Y. Xiao, H. Chen, B. Sun, W. Han, Computer operating system logging and security issues: a survey, Security and Communication Networks 9 (17) (2016) 4804–4821. doi:10.1002/sec.1677.
URL https://dx.doi.org/10.1002/sec.1677

[178] L. Thomeczek, Security Analysis of Linux Kernel Features for Embedded Software Systems in Vehicles, in: CARS 2015 - Critical Automotive applications: Robustness & Safety, Paris, France, 2015.
URL https://hal.archives-ouvertes.fr/hal-01193025

[179] K. Salah, J. M. Alcaraz Calero, J. B. Bernabé, J. M. Marín Perez, S. Zeadally, Analyzing the security of windows 7 and linux for cloud computing, Computers & Security 34 (2013) 113–122. doi:https://doi.org/10.1016/j.cose.2012.12.001.
URL https://www.sciencedirect.com/science/article/pii/S0167404812001800

[180] Y. Bassil, Windows and linux operating systems from A security perspective, CoRR abs/1204.0197 (2012). arXiv:1204.0197.
URL http://arxiv.org/abs/1204.0197

[181] Y. Zhang, B. Fang, Y. Chi, X. Yun, Assessment of windows system security using vulnerability relationship graph, in: Y. Hao, J. Liu, Y.-P. Wang, Y.-m. Cheung, H. Yin, L. Jiao, J. Ma, Y.-C. Jiao (Eds.), Computational Intelligence and Security, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 415–420.

[182] B. Doerrfeld, 17 open-source container security tools, online (Mar 2021).
URL https://techbeacon.com/security/17-open-source-container-security-tools

[183] O. Tunde-Onadele, J. He, T. Dai, X. Gu, A study on container vulnerability exploit detection, in: 2019 IEEE International Conference on Cloud Engineering (IC2E), 2019, pp. 121–127. doi:10.1109/IC2E.2019.00026.

[184] C. Lin, S. Nadi, H. Khazaei, A large-scale data set and an empirical study of docker images hosted on docker hub, in: 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2020, pp. 371–381. doi:10.1109/ICSME46990.2020.00043.

[185] M. Souppaya, J. Morello, K. Scarfone, Application container security guide (2017-09-25 2017). doi:https://doi.org/10.6028/NIST.SP.800-190.