SoK: A Comprehensive Analysis and Evaluation of Docker Container Attack and Defense Mechanisms

Md Sadun Haq[1], Thien Duc Nguyen[2], Ali Şaman Tosun[3],
Franziska Vollmer[2], Turgay Korkmaz[1], and Ahmad-Reza Sadeghi[2]

[1]*University Of Texas At San Antonio - {mdsadun.haq, Turgay.Korkmaz}@utsa.edu*
[2]*Technical University of Darmstadt - {ducthien.nguyen@trust, ahmad.sadeghi@trust, franziska.vollmer@stud}.tu-darmstadt.de*
[3]*University Of North Carolina at Pembroke - Ali.Tosun@uncp.edu*

2024 IEEE Symposium on Security and Privacy (SP) | 979-8-3503-3130-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/SP54263.2024.00268

***Abstract*—**

Container-based applications are increasingly favored for their efficiency in software development, deployment, and operation across various platforms. However, the growing number of security and privacy attacks poses significant concerns. Exploiting vulnerabilities within containers may compromise the entire host system, as both share the same operating system. Unfortunately, container defense mechanisms are inadequate due to the ever-evolving and dynamic attack landscape.

In this paper, we systematize container attacks and defense mechanisms. We systematically analyze the effectiveness of (i) static container scanning tools proposed for vulnerability detection and reveal their shortcomings, as well as (ii) existing run-time anomaly-based detection approaches. We then establish an evaluation framework and comprehensively re-evaluate cutting-edge anomaly detection techniques tailored for containers using an extensive dataset of 51 real-world vulnerabilities. We emphasize that existing defenses are ineffective in protecting containers against state-of-the-art attacks. While anomaly detection-based approaches show potential in addressing dynamic attack landscapes, their high false positive rates and limited training data hinder practicality. Therefore, our work highlights the urgent need for further research to enhance the security of container-based applications.

1. Introduction

Container-based technology is increasingly used in developing and deploying applications across various platforms since it provides several benefits [1]. Firstly, unlike Virtual Machines (VMs) that include an entire operating system and are more resource-intensive, containers are designed to be efficient and lightweight, bundling only essential libraries for operation. Secondly, containers offer portability as applications can be bundled into a single unit with all dependencies, simplifying deployment across different computing platforms. Thirdly, container technology provides isolation between various applications and their respective dependencies, ensuring that changes or updates in one container do not inadvertently impact other containers running on the same host machine. Furthermore, the encapsulation feature of containers guarantees isolated activities within a container, enhancing overall system stability. According to the Cloud Native Computing Foundation, container use in production has jumped by 300% from 2016 to 2020. Further, a study conducted by [2] estimates that by 2030, Global Container as a Service (CaaS) is expected to grow by 22.5% with a revenue of 10.75 billion USD and Docker will hold the majority market among the different container technologies. Tech giants such as Google, Amazon, IBM, Alibaba, and Microsoft provide container deployment and orchestration services used to develop and deploy applications. Furthermore, an estimated 96% of organizations currently use or plan to use container services [3].

However, recent works show that containers are vulnerable to various attacks, such as unauthorized access [4], [5], gaining privilege [6], [7], disclosing sensitive information [8], [9], bypassing authentication [10], [11], or Denial Of Service (DoS) attacks [12], [13]. Such attacks also affect DockerHub (the biggest official repository for hosting and sharing docker containers) [8], [14]–[17]. For example, the Mitre database lists 571 vulnerabilities exclusively associated with the DockerHub container engine [18]. Further, several tremendous attacks have taken place. For example, UpGuard reported over 10 million log4shell exploitation attempts every hour after Apache Log4j disclosed a vulnerability that affected distributed systems worldwide [19], including containers in cluster systems [20].

Various defense techniques have been proposed to mitigate such attacks to safeguard containers. Existing defense techniques can be categorized into four groups *Static Scanning, Image Hardening, Security Policies and Practices, Dynamic Anomaly Detection*. *Static Scanning-based* defenses regularly scan containers for vulnerabilities using tools like Clair, Trivy, and Snyk [21]–[23]. *Image Hardening-based* defenses enhance the security of containers by minimizing the attack surface, e.g., removing unnecessary dependencies [24], [25]. The defenses based on *Security Policies and Practices* use containers to ensure compliance with the industry standards and regulations [26]–[28], for example, not using a container as a root user or with a privileged flag and use AppArmor and Seccomp profiles, which restrict read access to sensitive files and filter system calls, respectively.

DOI 10.1109/SP54263.2024.00182

Dynamic Anomaly Detection-based approaches regularly monitor container activity and gather data regarding normal and anomalous activities [29], [30]. These approaches use Machine Learning (ML) models to profile the normal behaviors of the containers and detect any deviations, i.e., abnormal behaviors potentially caused by attacks.

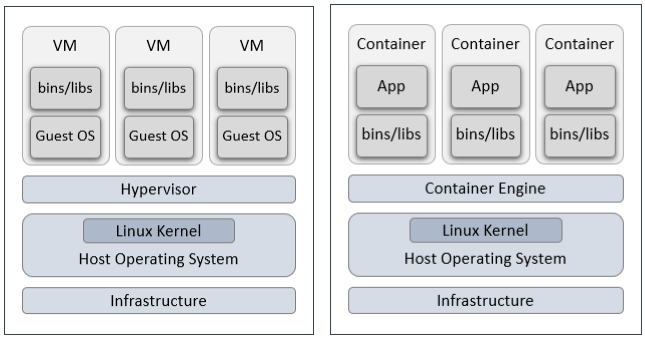
In this paper, we aim to provide a systematization of knowledge of state-of-the-art container attacks and defenses, discussing potential approaches to address the limitations of existing protection mechanisms. In summary, our contributions include:

- We systematize existing attacks and defenses and point out the deficiencies of existing defenses (Sect. 3 and 4). To the best of our knowledge, this is the first work that provides a comprehensive investigation and an extensive empirical evaluation of state-of-the-art attack and defense mechanisms on containers.
- We provide a framework to analyze and evaluate attacks and defenses and conduct a comprehensive empirical evaluation of existing protection mechanisms (Sect. 5 and 6).
- We provide an extensive dataset generated from state-of-the-art exploits used as a benchmark for evaluating the attacks and defenses on containers (Sect. 6). We will make our datasets and implementation available for research use.
- We point out the deficiencies of the existing defenses and discuss potential approaches to prevent state-of-the-art attacks effectively (Sect. 7).

2. Containers

Containers are a standalone and lightweight executable bundle of software encompassing code, system tools, and other dependencies needed to run an application. This ensures consistency and portability across diverse deployment platforms. The container image is the independent executable bundle encapsulating both application code and its dependencies. The container engine is responsible for deploying and managing containers during runtime on the host system, communicating with other components of the container and the internet through REST APIs. Different container technologies include Docker [31], Podman [32], LXC [33], and Kata Containers [34], with Docker and Podman being highly popular [35].

Virtual Machines (VMs) represent another virtualization technology enabling the execution of various applications on separate operating systems, known as 'Guest OS.' VMs include a hypervisor managing and deploying multiple VMs on a physical host system, performing resource management and isolation. VMs communicate with each other using LAN and Virtual Adapters. Containers and VMs are built on top of the Linux Kernel, which is the main component of the host operating system. The Linux Kernel provides isolation mechanisms such as namespace and cgroups, which both containers and VMs utilize to isolate and use resources effectively. Namespaces provide process isolation, ensuring that processes within the same namespace cannot directly access resources outside of their scope, whereas cgroups provide limitations on system resources to prevent processes from using up existing resources.



(a) (b)

Figure 1: Architecture of (a) Virtual Machines and (b) Docker Containers.

Figure 1 shows the difference between a traditional VM and a Docker container. Docker containers, renowned for their lightweight nature and portability, they use custom Docker Engines as daemons to deploy and run images. Containers are faster and lighter than VMs because they do not contain any operating system and depend on the host's operating system. Consequently, they require shorter boot times and require less resources. However, containers possess security risks due to their weak isolation from the host operating system compared to VMs. As containers depend on the host's operating system, any successful attack within a container might compromise the entire host system. In contrast, any successful exploit in a VM might be restricted inside the VM's operating system. Other VMs and the host system will not be affected because of the hypervisor.

Containers provide performance and deployment advantages over VMs, but their reliance on the host's operating system raises security concerns. Hence, there is a need for understanding diverse attack vectors to exploit containers and establish effective defenses to safeguard them.

3. Attack Scenarios

In this section, we systemize possible attack scenarios on containers. These include software vulnerabilities, system misconfigurations, and container engine and Linux kernel issues. The threat models of containers and VMs differ greatly due to their architecture. Containers share the underlying operating system kernel with the host system, which means that applications and their dependencies are executed directly on the host operating system. As a result, containers generally have weaker isolation compared to VMs. Exploiting a vulnerability in a container could break the isolation between the host and the container, potentially allowing an attacker to compromise the entire host system. In contrast,

virtual machines have their operating system and are isolated from the host and other VMs by the hypervisor. A successful attack within a VM usually remains isolated within the VM's environment. To compromise the entire system, an attacker must bypass the isolation mechanisms of both the VM's operating system and the hypervisor. Assuming that an attacker gains control of a container, either directly or indirectly, and extends the attack to the host system, we have created a threat model as shown in Fig. 2. Overall, there are several root causes of attacks:

- Downloading malicious container images and applications unknowingly by the user (Attack 1)
- Existing container vulnerabilities and their encapsulated applications due to lack of updates (Attacks 2-5, 7-8), such as log4j
- Misconfigurations in container deployment, e.g., using privileged flags or leaving ports exposed (Attacks 2-9)
- Limitations such as deficiencies or bugs in the container engine hinder both security measures and optimum functionality of containers, e.g., docker cp vulnerability (CVE-2019-14271) (Attacks 4-6, 8-9)

In the following, we will elaborate on each attack scenario in detail.

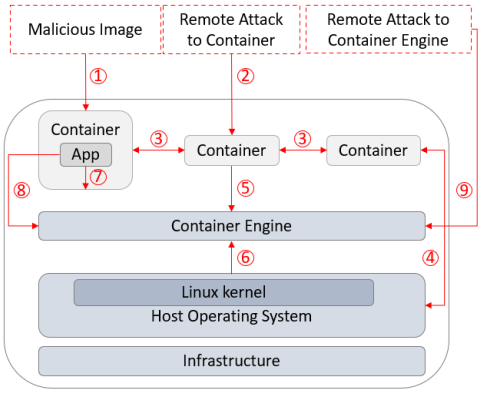


Figure 2: Overview of attack Scenarios on containers: ① Malicious Image Attacks, ② Remote Attacks to Container, ③ Container to Container Attacks, ④ Container to Host and Host to Container Attacks, ⑤ Container to Container Engine Attacks, ⑥ Host to Container Engine, ⑦ Application to Container Attacks, ⑧ Application to Container Engine Attacks, and ⑨ Remote to Container Engine.

① **Malicious Image**. This concerns the case where malware is located inside the container, for instance, when a regular user downloads malicious images from a public container repository, such as DockerHub, to a benign machine [17], [14], [8], [16]. The malicious image contains vulnerable packages that an adversary over the network can exploit. This vulnerability can lead to various exploits, such as reverse shell, crypto mining, and exposing credentials [14], [36].

② **Remote to Container**. The vulnerable component or application is located in the container and can be exploited from anywhere. The remaining elements in the system are benign. This differs from 'Malicious Image' as an adversary doesn't need to upload the image to the public repository. The vulnerability arises from preexisting software issues or misconfigurations within the container [37]. These types of vulnerabilities are often labeled 'remotely exploitable.'

③ **Container to Container.** This type of attack is initiated if one of the containers in the host system has already been compromised after being subjected to attack vectors ① and ②. In this attack, the adversary leverages a malicious container to compromise other benign containers within the host. Attacks such as Meltdown [38] can divulge vital kernel information from the host and all containers running in the system. This enables a malicious container to gain vital information regarding all the containers running within the system. Spectre [39] is another attack that is detrimental to containers. This attack tricks applications into accessing arbitrary locations within the memory. This causes a leak in kernel information, which the adversary then uses to gain information regarding other containers in the system. Both of these attacks [38], [39] attain vital information regarding the containers and can cause different types of attacks, such as DoS or Authentication Bypass.

④ **Container to Host and Host to Container.** The Container to Host attack scenario assumes that at least one of the containers in the system is malicious. The malicious container can access confidential host information and try to perform Authentication Bypass or Disclose Credential Information Attacks to take control of the benign host or containers. The malicious container can also perform a DoS attack by consuming resources and causing other containers and the host to stop functioning properly [40], [41]. Attacks, such as container breakouts, can bypass the container-host isolation and gain privileges equivalent to the root user [41], [42].

The Host to Container scenario assumes the host operating system is malicious. This scenario is relatable with the advent of Containers-As-A-Service (CaaS), where containers can be rented from the cloud and CaaS providers [43]. A malicious host can perform numerous attacks on the container. Moreover, the malicious host can gain access to the benign container using Return A Shell, Disclose Confidential Information, and Authentication Bypass attacks to monitor the activities of a container and discretely steal information for ransomware, which can lead to substantial financial loss [44].

⑤ **Container to Container Engine.** This scenario presumes that at least one of the containers in the system is malicious. The malicious container takes advantage of vulnerabilities and misconfigurations within the container engine and tricks the engine into gaining escalated privileges [45]. This allows the malicious container to perform, e.g., a DoS attack by manipulating resource isolation components such as namespaces and cgroups. In addition, the malicious container can masquerade as a benign container and steal

vital information from the entity using the containerized services. Since the container engine usually runs with root privileges, compromising it can grant superuser access.

⑥ **Host to Container Engine.** This scenario presumes that the host system is malicious. Given that the host has full privileges, performing an attack against the container engine becomes straightforward. Performing attacks via Docker Socket or Docker Volume Mount are just a few examples.

⑦ **Application to Container.** This scenario assumes that the application within the container is malicious. The adversary tries to take control of the container using Gain Privilege, Authentication Bypass, or Disclose Credential Information attacks. For example, a container hosting a database application can be impacted via attacks that can expose the username and password of the user and take full control of the container [11].

⑧ **Application to Container Engine.** In this scenario, there is a compromised application inside a container. However, the container itself is benign. The application attempts to gain control of the container engine by exploiting vulnerabilities in the container engine or its misconfiguration, such as defective Socket APIs [46], [47]. This can lead to privilege escalation, which can then be used to perform DoS attacks on the host and other containers. The difference between scenarios 7 and 8 is that application-to-container attacks can impact only a singular container or containers associated with the specific application. However, application to container-engine attacks can impact associated containers and orchestrators such as Kubernetes and Docker Swarm, hampering the lifecycle and deployment of other containers unrelated to the specific application and leading to broader service disruptions.

⑨ **Remote to Container Engine.** This setting assumes that the container engine contains vulnerabilities [47], [48]. The adversary scans for exposed ports, and if found, the adversary tries to gain privilege by exploiting the preexisting vulnerability.

4. A Taxonomy of Attack and Defense Mechanisms

This section systematically analyses existing attacks, distinguishing between attack types and corresponding attack techniques, before looking at defenses. We then highlight the shortcomings of existing defenses.

4.1. Attacks

There are numerous attacks on containers using various attack techniques [9]–[11], [41], [42], [49], [50], [54], [57]. To the best of our knowledge, the existing studies do not provide a comprehensive categorization and analysis where attack types are systematically categorized and their respective attack techniques are assigned. In our approach, we view the techniques as subcategories under the attack types for a more detailed and organized analysis. Since containers have gained immense popularity, we aim to provide deeper insights into the risks associated with containers and the effectiveness of potential mitigations. Our categorization methodology aligns with existing research [11], [41], [49], [50], [54], [73]. However, we observed that each of these works focuses on particular attacks. For example, [73] concentrates on DoS attacks, [10] covers DoS, privilege escalation, and authentication bypass, and [11] discusses DoS, remote code execution, and gain privileges. To address this inconsistency, we categorized the attacks into five distinct types: executing arbitrary code, gaining privileges, disclosing credential information, authentication bypass, and denial of service. Subsequently, we identified and categorized various attack techniques and their respective attack surfaces, as illustrated in Table 1. To ensure clarity, we refined the categorization of specific attack techniques such as 'Integer Overflow' (e.g., CVE-2014-01604) and 'crashing the application' (e.g., CVE-2015-5477, CVE-2016-7434) by referencing their CVE-IDs and descriptions from the National Vulnerability Database (NVD) [75] and assigning these attacks to Authentication Bypass and Denial Of Service categories, respectively. This standardized methodology has been consistently applied to all ad hoc attack reports, contributing to clarity in future research. A detailed overview of the CVE-IDs and their respective categorization into specific attack types can be found in Table 5. This table lists all considered CVE IDs associated with the defined attack types.

Moreover, attack techniques may not be mutually exclusively assigned to specific attack types. For example, SQL injection can serve Authentication Bypass (AB) in the form of *password bypass* (major objective), Disclose Credential Information (DCI) in the form of *gain login credentials* (major objective), or DoS attacks in the form of *brute-force* (minor objective). A similar statement can be made for Gain Privilege (GP) and DCI, where *escalation of privilege* of GP can overlap with *file system access and file name access* of DCI. While AB and DCI may sound similar, AB often refers to direct system access without authentication, whereas DCI exposes information like passwords or directory structures.

Further, specific attacks can only be successfully executed when certain applications are present. Therefore, we label the types of applications targeted by the attacks as attack surfaces, which align with existing research [9], [54]. The terms Web App, Server, and Database refer to distinct types of applications. The term Lib refers to any library that may cause a vulnerability. Kernel refers to an attack surface where the adversary takes advantage of a vulnerable Linux kernel from a compromised container.

Execute Arbitrary Code. This attack involves running arbitrary codes in the target system, focusing on web applications and servers. The attack aims to send arbitrary commands to the victim to gain unauthorized access or control of the system. One technique for this attack is to 'Return a shell'. In this attack technique, the adversary takes advantage of preexisting vulnerabilities within a container, opens a shell, and takes control of the container [4], [51]. After gaining access, the adversary can steal, delete informa-

Attack Types	Attack Techniques	Attack Surfaces					References
		Web App	Server	Database	Lib	Kernel	
Execute Arbitrary Code	return a shell	✓	✓	✓			[49], [50], [41]
			✓		✓		[51]
				✓			[52]
					✓		[53]
	return-oriented programming	✓	✓	✓			[49], [50], [41]
				✓			[52], [11]
		✓	✓	✓			[42]
					✓		[30]
		✓	✓	✓	✓		[8]
	remote code execution	✓	✓				[9]
		✓	✓	✓	✓		[54], [17]
			✓				[55]
Gain Privilege	escalation of privilege	✓	✓	✓			[49], [50], [41], [42], [56]
				✓			[52], [11], [57], [29], [10]
		✓	✓		✓		[58], [59], [60]
		✓	✓	✓	✓		[13], [5], [27]
		✓	✓				[9], [61]
					✓		[62]
		✓					[63]
	kernel escalation	✓	✓		✓	✓	[54]
		✓	✓	✓	✓	✓	[64], [65], [26], [6]
Disclose Credential Information	gain login credentials	✓	✓	✓			[49], [50], [41]
		✓	✓				[9]
		✓	✓		✓		[59]
		✓	✓	✓	✓		[54], [14], [8], [27]
		✓	✓	✓	✓	✓	[64]
	file system access			✓			[57], [29]
		✓	✓	✓	✓		[66], [67], [68]
		✓	✓		✓		[58]
		✓					[63]
	file name access	✓	✓				[9]
	channel attacks		✓		✓	✓	[69]
		✓	✓	✓	✓	✓	[26]
Authentication Bypass	password bypass	✓	✓	✓			[49], [50], [41], [8], [27]
		✓	✓				[9]
		✓	✓	✓	✓		[14], [67], [5], [68]
	reduced security attributes	✓	✓		✓		[58]
		✓	✓	✓	✓		[15]
	integer overflow			✓			[10], [11]
Denial Of Service	crash the application	✓	✓	✓			[49], [50], [41]
		✓	✓				[9]
		✓	✓		✓		[59]
		✓	✓	✓	✓		[54]
	consume excessive resources	✓	✓	✓			[70], [51], [71]
		✓	✓	✓	✓		[66], [72], [15], [68]
		✓	✓		✓		[60]
			✓			✓	[12]
		✓	✓	✓	✓		[13]
				✓			[57]
					✓		[30], [28]
		✓					[73]
		✓		✓			[74]
	brute-force			✓			[29], [52]
	spoofing	✓	✓	✓	✓	✓	[64]
		✓	✓	✓	✓		[72]
	DoS overflow			✓			[10], [11]
		✓	✓	✓	✓		[14], [27]
		✓	✓	✓			[42]
	redirect traffic	✓	✓		✓		[58]

TABLE 1: Table showing Attack Types, Attack Techniques, Attack Surfaces, and References.

tion, etc. For example, an adversary sends crafted commands to databases or web servers, which leads to opening a shell and creating a pathway for the adversary [42], [49], [50]. A similar attack technique over the network can be labeled as *remote code execution*. For instance, exploits that spawn a shell from attacks on cloud servers [9], [54]. If successful attacks are unable to return a shell, they may cause various issues, including memory corruption, denial of service, or authentication bypass. In such cases, this technique is called return-oriented programming (ROP). Although, in the

broader scope, ROP can lead to other attacks, in this paper, we will keep ROP exclusively for Execute Arbitrary Code. This is done because no existing literature mentions ROP as their attack technique for the remaining four attack types.

Gain Privilege. This attack aims to gain superuser privileges by exploiting, e.g., vulnerable web applications, misconfigured databases, and buffer overflows [11]. Modifications of memory and files can be used to escalate privilege by buffer overflow or modifying the superuser's password, respectively [11], [49], [50], [57]. Kernel escalation is a special type of privilege escalation that involves gaining root access by exploiting vulnerabilities in the kernel. The goal of kernel escalation is to use the vulnerable kernel to obtain all its restricted capabilities and gain root access [54], [64]. For example, successfully exploiting a vulnerable kernel with the help of 'SYS_ADMIN', 'NET_ADMIN', or other capabilities can compromise the entire host system and provide a pathway to affect other interconnected components.

Disclose Credential Information. This attack involves unauthorized access and exposure of credentials, such as usernames and passwords [49], [50], [14]. Further, these attacks can also aim to gain the underlying information of the systems, e.g., file names or directory structures, which can be used to steal information or cause a DoS attack [9], [57], [29], [58]. Several techniques allow the adversary to gather credential information. For example, having the contents of '/etc/shadow' or '/etc/passwd' files can give an adversary unlimited access. These attacks can exploit, e.g., vulnerable web applications and servers. This sensitive information can be leaked if users carelessly store credentials in plaintext in an unsecured repository [76]. Further, the adversary can also utilize side-channel attacks [69] to acquire sensitive information by analyzing CPU and memory usage patterns from the kernel message buffer [77].

Authentication Bypass. This attack involves exploiting vulnerabilities in authentication systems to gain unauthorized access without providing correct credentials [8], [27]. This can result from vulnerable web applications, exploitable database systems, and flawed design in authentication mechanisms [10], [41]. SQL injection and cross-site scripting attacks are primary examples of how an adversary can bypass the password authentication method and gain access to a person's online account or database [49], [50], [14]. This attack allows the adversary to, e.g., compromise a host and run further attacks (see Sect. 3). Integer overflow can bypass authentication by providing inappropriate data to specific input fields, causing the program to behave erratically and authenticate the adversary by mistake [10], [11].

Denial Of Service. This attack attempts to disrupt normal system functionalities, with the adversary focusing on disrupting applications, the host, or infrastructure by exhausting computing, communication, or memory resources. DoS attacks focus on three resources: CPU, Memory, and Network. A successful DoS attack on any of these three domains significantly affects the system's throughput and performance. Excessive resource consumption by one container can lead to starvation in others [70], [51], [66] and application crashes due to unavailable resources [49], [50], [9]. Brute-force attack techniques [29], [52] are implemented on database containers as infinitely long queries are created that occupy most of their processing capacity. Spoofing and redirecting traffic are attack techniques occupying the network bandwidth, causing them to go offline [64], [58]. DoS overflow [10], [11], [14] occurs when the adversary can cause memory corruption by triggering buffer or heap overflow, causing the system to crash [41], [59], [54].

4.2. Defenses

We have extensively reviewed state-of-the-art defense mechanisms developed to safeguard containers against various attacks. These defenses are categorized into four distinct groups: *Static Scanning, Image Hardening, Security Policies and Practices, Dynamic Anomaly Detection*. Table 2 illustrates existing research that employs different container scanning tools to identify known vulnerabilities, detailing the total number of containers analyzed and summarizing their findings. Meanwhile, Table 3 categorizes existing works based on defense type, method, and performance metrics, including Precision (P), Recall (R), and F1-Score (F1). True Positive (TP) is utilized for works lacking these metrics to assess their performance. The table also includes a list of abbreviations used throughout the discussion, ranging from machine learning algorithms to security mechanisms. In the subsequent sections, each defense category will be explored in depth.

Static Scanning. These defenses use scanning tools to seek known vulnerabilities in containers [22], [23]. Scanning tools rely on data from different vulnerability databases [75], [80] and utilize different algorithms for detecting and assigning severity ratings [81]. These approaches can detect vulnerabilities in non-updated images, poisoned images, malicious images, and even insecure configurations, as some tools provide additional features to scan code for potential vulnerabilities and information leaks [82]. The difference between poisoned and malicious images is that the former involves legitimate images from a trusted source that has been tampered with, whereas the latter is intentionally crafted by attackers for malicious purposes and is spread via various free, public container registries and forums. Table 2 shows previous works that perform static analysis, the tools used, the number of investigated containers, and a summary of the findings. Clair [21] is the most used tool as it has been around the longest. We also observe an inverse relationship between the number of tools used and the number of containers investigated as the experiment becomes more complex with the involvement of more tools. Shu et al. [14] and Socchi et al. [15] scan more than 350,000 containers using only one tool. Berkovich et al. [78], Javed et al. [67], Brady et al. [51] and Kaur et al. [24] use multiple tools but scan less than 30 images. Haq et al. [5] use four different tools, more than any existing work, but do not specify the number of containers they scan.

Image Hardening. These defenses remove unnecessary packages to reduce the attack surface [25] or build an application on top of a minimal base image to prevent the

Approach	Static Analysis Tools					Containers Investigated	Findings
	Clair	Trivy	Snyk	Anchore	JFrog		
Shu et al. [14]	✓					356,218	180 vulnerabilities on average, not updated in 200 days
Socchi [15]	✓					440,524	Security features of DockerHub are mostly ineffective
Berkovich et al. [78]	✓	✓		✓		3	Different tools are better suited to scan particular images
Katrine et al. [8]				✓		2500	JavaScript and Python packages have the most vulnerabilities
DIVDS [79]	✓					4	Checks for bugs when downloading from DockerHub
Brady et al. [51]	✓			✓		7	Uses scanning tools and virus scanners for anomaly detection
Javed et al. [67]	✓	✓		✓		55	Tools are inaccurate, vulnerable OS packages are more prevalent
Zerouali et al. [16]			✓			3000	JavaScript, Python and Ruby packages have the most vulnerabilities
Kaur et al. [24]	✓			✓		26	Image update and minification, removes most vulnerabilities.
Chen et al. [68]	✓					132850	A tree-based detection approach performs better than Clair.
Haq et al. [17]	✓	✓	✓		✓	Unspecified	Tools produce mismatches for the same set of images

TABLE 2: Existing static analysis approaches.

Defense Type	Approach	Method	Performance Metrics			
			P	R	F1	TP
Image Hardening	Kaur et al. [24]	Image update and minimizing size	N/A	N/A	N/A	N/A
	Rastogi et al. [25]	Remove unnecessary packages	N/A	N/A	N/A	N/A
	Haq et al. [17]	Trim down larger packages	N/A	N/A	N/A	N/A
Security Policies & Practices	Mattetti et al. [65]	Restricts container operations	N/A	N/A	N/A	1.000
	Chelladhurai et al. [72]	Restricting use of memory using cgroups	N/A	N/A	N/A	1.000
	Luo Yang et al [26]	Limited CAP, SELinux, AppArmor	N/A	N/A	N/A	1.000
	Jian et al. [6]	namespace and process inspection	N/A	N/A	N/A	1.000
	Lin et al [54]	reinforcement of commit_creds()	N/A	N/A	N/A	1.000
	Sun et al. [13]	Reinforcement of namespace and MAC	N/A	N/A	N/A	1.000
	Gantikow et al. [63]	Ruleset of Sysdig & Falco	N/A	N/A	N/A	0.857
	Lu et al [70]	Graph Comparison	N/A	N/A	N/A	0.847
	Brady et al. [51]	Anchore, Clair, Virus Total Scanner	N/A	N/A	N/A	1.000
	Ghavamnia et al. [27]	Filter and limit system calls	N/A	N/A	N/A	0.967
	Wenhao et al. [28]	Limited CAP, Seccomp, AppArmor, MAC	N/A	N/A	N/A	N/A
	Lopes et al. [55]	Automatic generation of Seccomp Profiles	N/A	N/A	N/A	0.333
	Zhu et al [9]	AppArmor and LicSec profiling	N/A	N/A	N/A	0.250
	Chen et al. [68]	Global Relationship Tree	N/A	N/A	N/A	N/A
	He et al [59]	Fine-grained eBPF access control	N/A	N/A	N/A	1.000
	Xiao et al [60]	Jail container runtime, prevent file sharing	N/A	N/A	N/A	1.000
	Jiao et al. [56]	Enhances eBPF program to filter system calls	N/A	N/A	N/A	1.000
Dynamic Supervised	Abed et al [29]	BoSC and STIDE	0.980	N/A	N/A	1.000
	Srinivasan et. al [57]	Probabilistic Approach (MLE and SGT)	0.987	0.998	0.992	0.858
	Huang et al. [30]	Clam Antivirus and Random Forest	N/A	N/A	N/A	0.667
	Flora et al. [11]	STIDE, BoSC, HMM	0.903	0.971	0.936	0.998
	Marcos et al. [10]	AB, DT, GNB, MLP, MNB, RF, KNN, SVM.	0.999	0.997	0.998	1.000
Dynamic Unsupervised	Zou et al. [74]	Isolation Forest	98.04	N/A	N/A	1.000
	Onadele et al [49]	Clustering (KNN, K-Means)	0.909	N/A	N/A	0.979
	Onadele et al [41]	AutoEncoder, SVM	0.987	N/A	N/A	0.811
	Zhang et al. [52]	One-Class-SVM	0.937	N/A	N/A	1.000
Dynamic Self-Supervised	Lin et al. [50]	AutoEncoder, Isolation Forest, Random Forest	0.990	N/A	N/A	0.841

TABLE 3: Existing defenses excluding static scanning approaches. P: Precision, R: Recall, F1: F1-Score, TP: True Positive, AB: AdaBoost, DT: Decision Tree, GNB: Gaussian Naive Bayes, MLP: Multi-layer-Perceptron, MNB: Multinominal Naive Bayes, RF: Random Forest, KNN: K-Nearest-Neighbors, SVM: Support-Vector Machine, MLE: Maximum Likelihood Estimator, SGT: Simple Good Turing, HMM: Hidden Markov Model, BoSC: Bag of System Calls, STIDE: Sequence Time-Delay Embedding, MAC: Mandatory Access Control and CAP: Capabilities

buildup of unnecessary applications in the form of bloatware [24], [5]. However, removing unnecessary packages is not straightforward and can hamper the execution of applications [24].

Security Policies & Practices. These defenses provide certain policies and practices to prevent breaches through containers. Firstly, using the container as *root* or in *privileged* mode is highly discouraged, as breaking out from such a container will provide the adversary with root access. Secondly, several security tools should be applied, e.g., Seccomp [83], AppArmor [84], SeLinux [85], Capability filters [86], before launching a container. Table 3 provides extensive use of those tools to strengthen container security see [72], [26], [6], [54], [13], [9], [65]. These security components provide rigid access to system calls, capabilities, and privileges. For example, AppArmor restricts applications from accessing specific files, such as the file containing credentials. Seccomp allows users to set up a filter to prevent using certain system calls to break out of a container. Dropping specific capabilities will prevent the container from starting up with elevated privileges. SELinux creates rules and security policies regarding which files can be used and accessed. Further, [63] describes how different rule sets of Sysdig [87] and Falco [88] can prevent exploits

	Static Scanning	Image Hardening	Security Policies & Practices	Dynamic Anomaly Detection
Return A Shell	[30], [8], [17] [79], [16]	[17]	[42], [54], [55]	[49], [50], [41], [11], [30]
Gain Privilege	[17], [79]	[17]	[42], [54], [56], [59], [63] [60], [13], [27], [64], [65] [26], [6]	[49], [50], [41], [52], [11] [10], [57]
Disclose Credentials	[8], [14], [66] [67], [24], [16]	[24]	[42], [54], [27], [63], [64] [26], [68], [69]	[49], [50], [41], [57], [29]
Authentication Bypass	[8], [14], [15], [17]	[17]	[27], [64], [68]	[11]
Denail Of Sevice	[51], [30], [14] [66], [15], [24] [79], [16]	[25], [24]	[54], [27], [13], [64], [68] [70], [72], [28]	[49], [50], [41], [52], [64] [30], [57], [11], [51] [74], [29], [10], [60]
Limitations	Cannot mitigate zero-day and run-time attacks	Difficult to detect and remove	incur overhead	Difficult to train and detect due to lack of training data
	Inconsistencies high false positives	Can break software dependency	limits functionalities due to limited syscalls and capabilities	High positive rates
Ease of Use	Simple	Moderate	Complex	Complex

TABLE 4: Overview of applying defenses against attacks and limitations

as these tools can trace system calls. [70] use graph comparison whereas [51] uses scanning tools and virus total scanners [89] to detect anomalies in a container system.

Dynamic Anomaly Detection. These defenses adopt machine learning algorithms to detect anomalies in the runtime. The detection model looks for subtle changes in certain features, such as system calls, or performance metrics (CPU, memory, and network) generated by benign and malicious workloads. For instance, Bag and Sequence of System Calls (BoSc and STIDE) are used as supervised anomaly detection by [29], [11]. Srinivasan et al. [57] use the Maximum Likelihood Estimator (MLE) and Simple Good Turing (SGT) to detect anomalies. Onadele [49] employs clustering techniques for the detection of anomalies. Huang [30] uses the Random Forest (RF) and the Clam Antivirus [90] for anomaly detection. Marcos et al. [10] perform anomaly detection using eight supervised models. Further, several defenses use unsupervised approaches such as Isolation Forest, AutoEncoder, and Support Vector Machine [74], [41], [52]. Lin et al. [50] improve AutoEncoder-based unsupervised approach [41] by introducing a self-supervised approach. The self-supervised approach uses an Isolation Forest for outlier detection in combination with an AutoEncoder to reduce the false positive rate.

4.3. Limitations of existing defenses

Table 4 outlines the defense mechanisms applicable to specific attack types and their limitations. *Static Scanning* based defenses cannot detect run-time attacks and suffer from inconsistencies and false positives [17], [67]. Moreover, static scanning tools cannot detect zero-day attacks, and there is a significant delay from when a zero-day vulnerability is detected until it is updated in scanning tools. Among the defenses, this method is the simplest to use. *Image Hardening* defenses face a critical challenge as it is hard to pinpoint unnecessary packages, resulting in additional complications, such as breaking the software dependency if a package is incorrectly removed. Little work has been done on this mechanism compared to other defenses, as shown in Table 4. Further, this method is more challenging to use than Static Scanning. Although many works exist on *Security Policies & Practises* based defenses, this mechanism introduces an increase in overhead. Further, some techniques like filtering system calls [59], [60], [27] are hard to implement in practice since restrictions of the system call are complex and may prevent a container from functioning correctly. *Dynamic Anomaly Detection* is more sophisticated than the other defense mechanisms. Selecting an effective detection model is challenging since existing works show varying performance for different ML algorithms. Moreover, building a robust detection model is difficult due to the scarcity of training data. Also, there is no dataset on which a model may be trained nor a benchmark against which the results of a model may be compared. Further, detection models are often trained on a dataset generated from specific deployment settings that might not generally be effective when applied in other settings, as we show in Sect. 6.

5. Evaluation Framework

We propose an evaluation framework that combines both static and dynamic evaluations, as shown in Fig. 3. Existing works [17], [51], [78] have demonstrated that combining multiple scanning tools results in a higher detection rate of vulnerabilities compared to relying on a single tool. Therefore, our static evaluation employs a variety of scanning tools to scan container images and assess their performance. The dynamic evaluation compares the effectiveness of multiple machine learning and deep learning algorithms for anomaly detection. This evaluation is essential as existing literature [10], [74], [41], [30], [49], [52], [90] lacks systematic approaches and relies on a disparate dataset, casting doubt on the readability of their experimental findings. Lastly, we evaluate the effectiveness of each method in detecting vulnerabilities and determine whether it is preferable to utilize a combination of both methods for anomaly detection. In the following, we will discuss these components in detail.

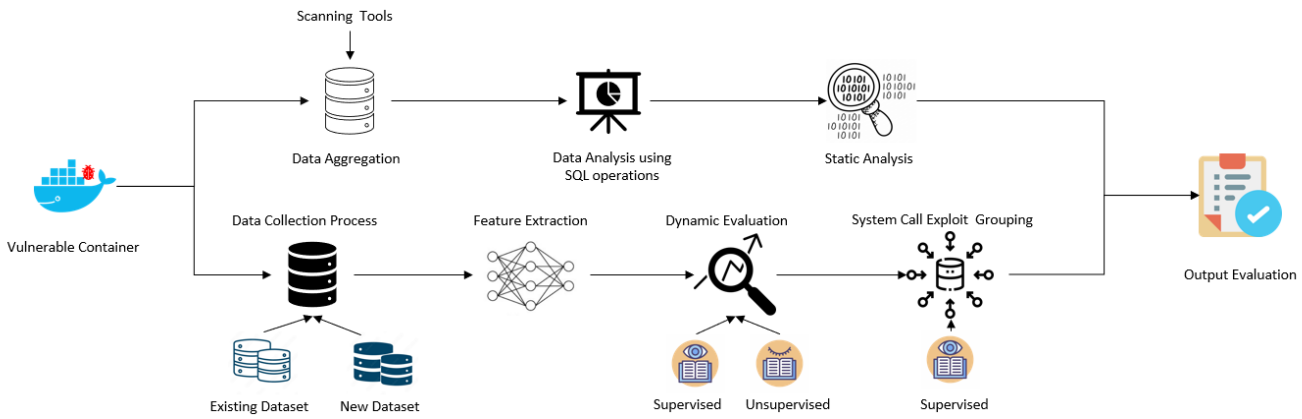


Figure 3: Overview of Evaluation Framework.

Static evaluation. This component utilizes various scanning tools like Trivy [22], Snyk [23], Clair [21], and Grype [91] to identify preexisting vulnerabilities in containers. Anchore [92] is excluded due to its discontinuation and replacement with Grype. The scanning results are merged and compared to address inconsistencies found in previous research. The outcomes of the tools are aggregated in CSV format. Then, the data acquired shall be inserted into a database for further analysis using basic SQL operations. The additional analysis serves a different purpose than merely detecting the exploit. It involves processing the collected data and comparing the outcomes of various tools that evaluate the same set of images.

Dynamic Analysis. This component evaluates and compares the performance of multiple machine learning and deep learning algorithms for detecting attacks on containers. *Data Collection & Feature Extraction* component gathers traces of containerized applications and is based on the works of Lin and Onadele [41], [49], [50] in combination with their dataset and the dataset collected by ourselves. We employ Sysdig [87] (the most widely used open-source tool), which can be attached to containers to record the system calls generated. System calls are the backbone of kernel operations and can provide insight into the tasks and performance within a container. Sysdig returns the system call used along with a timestamp, making it ideal as a logging tool. The system calls are then accumulated every 0.1s and generated as a frequency vector. To be consistent with previous works and handle the diversity of system calls, the dimensions of the frequency vector are expanded to include all 555 Linux system calls. To simulate the application running, several workloads are applied to the containers via Apache Jmeter [93] for 7 minutes. The exploits are carried out at the fourth minute, and the time is recorded till the exploit succeeds. Each experiment is repeated four times with an increasing workload of 1X, 2X, 4X, and 8X. The data collected in the first three minutes is labeled as benign, while the rest is labeled as malicious. *Supervised Learning* consists of algorithms that will learn with the help of labeled data. As each exploit is repeated 4 times, we repeat the experiment using stratified K-Fold cross-validation (k=4) for supervised learning. All algorithms (Random Forest, AdaBoost, Decision Trees, K-Nearest Neighbor, and Multi-Layer Perceptron) mentioned in Tab. 3 will be evaluated. *Unsupervised Learning* will consist of algorithms that learn without labeled data. We use the first three minutes of data for training and the last four minutes for testing the model. This is different from supervised learning as supervised learning needs both attack and benign data for training. The typical defenses using AutoEncoder and K-Means mentioned in Tab. 3 will be evaluated. *System Call Exploit Grouping* involves taking the system calls for each categorized exploit and combining them together. Each category will be trained and validated separately by different anomaly detection algorithms. This will provide deep insight into how well the anomaly detection algorithms work for different types of exploits and whether a specific model can maintain high accuracy across all types of exploits. Additional details about the exploits are given in Sect. 6. *Output Evaluation* component assesses the static and dynamic analyses. Firstly, we shall compare the results of the static analysis to those of the dynamic analysis and deduce which scanning tools and anomaly detection algorithms perform the best.

Dataset. The *existing dataset* includes the system calls for 41 exploits. The time taken for the exploit to succeed is also recorded. This dataset is published by CDL [41] and includes some of the most common exploits published over the years. Previously they implemented 33 exploits [49], which was later expanded to include 41. The exploits range over various applications from the front to the back end encapsulated within a container. *The new dataset* comprises 9 exploits that we have generated by ourselves. This dataset includes some of the most recent and up-to-date exploits published as listed in Table 5. To collect this data, four docker containers were set up for each vulnerability under Ubuntu 16.04 LTS. Different workloads were delivered to

Scenario	CVE ID	Severity	Applications	Attack Types					Scanning Tools				
				1	2	3	4	5	N/A	A	B	C	D
Dataset d1	CVE-2012-1823	High	PHP			✓			✓				
	CVE-2014-0050	High	Apache Commons	✓				✓					✓
	CVE-2014-0160	Medium	OpenSSL			✓				✓	✓		✓
	CVE-2014-3120	Medium	Elasticsearch	✓			✓	✓				✓	
	CVE-2014-6271	Critical	Bash	✓			✓			✓	✓		✓
	CVE-2015-1427	High	Elasticsearch	✓			✓					✓	✓
	CVE-2015-2208	High	phpMoAdmin	✓			✓						
	CVE-2015-3306	Critical	ProFTPd			✓	✓	✓	✓				
	CVE-2015-5477	High	BIND					✓		✓	✓		✓
	CVE-2015-5531	Medium	Elasticsearch				✓		✓				
	CVE-2015-8103	High	JBoss			✓			✓				
	CVE-2015-8562	High	Joomla	✓			✓						
	CVE-2016-3088	High	Apache ActiveMQ	✓			✓			✓			✓
	CVE-2016-3714	Critical	ImageMagick	✓									
	CVE-2016-6515	High	OpenSSH					✓					
	CVE-2016-7434	Medium	NTP					✓					
	CVE-2016-9920	Medium	Roundcube	✓					✓				
	CVE-2016-10033	High	PHPMailer	✓									
	CVE-2017-5638	Critical	Apache Strutus 2	✓						✓		✓	✓
	CVE-2017-7494	Critical	Samba	✓									
	CVE-2017-7529	Medium	Nginx			✓							
	CVE-2017-8291	Medium	Ghostscript				✓		✓				
	CVE-2017-8917	High	Joomla			✓							
	CVE-2017-11610	Critical	Supervisor	✓						✓			✓
	CVE-2017-12149	High	JBoss			✓							
	CVE-2017-12615	Medium	Apache Tomcat	✓									
	CVE-2017-12635	Critical	CouchDB	✓	✓								
	CVE-2017-12794	Medium	Django				✓						
	CVE-2018-11776	Critical	Apache Strutus 2	✓						✓		✓	✓
	CVE-2018-15473	Medium	OpenSSH				✓						
	CVE-2018-16509	Critical	Ghostscript		✓		✓						
	CVE-2018-19475	Critical	Ghostscript				✓						
	CVE-2018-19518	High	PHP	✓						✓		✓	✓
	CVE-2019-5420	High	Rails				✓			✓			✓
Dataset d2	CVE-2019-6116	Medium	Ghostscript				✓						
	CVE-2019-10758	Critical	VM	✓									
	CVE-2020-1938	Critical	Apache Tomcat	✓						✓		✓	✓
	CVE-2020-17530	Critical	Apache Strutus 2	✓						✓		✓	✓
	CVE-2021-28164	Medium	Eclipse Jetty			✓				✓		✓	✓
	CVE-2021-28169	Medium	Eclipse Jetty			✓				✓		✓	✓
	CVE-2021-34429	Medium	Eclipse Jetty				✓			✓		✓	✓
	CVE-2021-41773	Medium	Apache HTTP Server				✓			✓			
	CVE-2021-44228	Critical	Apache Solr	✓						✓		✓	✓
Dataset d3	CVE-2022-0847	High	Linux Kernel		✓		✓						
	CVE-2022-21449	High	Oracle Java SE				✓						
	CVE-2022-22963	Critical	Spring Cloud	✓									
	CVE-2022-22965	Critical	Spring MVC	✓						✓		✓	✓
	CVE-2022-26134	Critical	Confluence Server	✓									
	CVE-2022-42889	Critical	Apache Commons	✓			✓			✓		✓	✓
	CVE-2023-23752	Medium	Joomla				✓						
	CVE-2021-42013	Critical	Apache HTTP Server				✓			✓			
Total Counts			51	25	3	9	22	6	6	19	3	13	19

TABLE 5: The effectiveness of different scanning tools in detecting vulnerabilities.(1-Execute Arbitrary Code, 2-Gain Privilege, 3-Disclose Credential Information, 4-Authentication Bypass, 5-Denial Of Service, A-Grype, B-Clair, C-Snyk, D-Trivy).

each of the four containers using Apache JMeter, namely 1x, 2x, 4x and 8x.

Training And Testing Scenarios. Due to the presence of data collected at different times and to perform extensive and proper evaluation, we split the data collected into three separate parts. The split of the dataset is done according to when the subset of the dataset was published. Dataset **d1** includes the data for the first 34 exploits published in 2019 [41]. Dataset **d2** includes the data for the following 9 exploits published in 2022 [50]. Dataset **d3** includes the data for the last 8 exploits that we performed for our experiments. Table 5 contains the details for the three different datasets. To test the robustness and validity of different models, we propose three different scenarios to train and test the models. *Scenario S1:* In this scenario, we train and test the detection model with the same dataset using Stratified K-Fold Cross-Validation, where k=4. *Scenario S2:* In this scenario, we train and test the model with different datasets. This scenario

aims to test whether a model can perform well with unseen data. This is the reason why we will not be using Stratified K-Fold Cross-Validation. Instead, we will use three traces of one CVE from one dataset to train the model and one trace of another CVE from the other dataset to test the model. This process will be repeated four times for each CVE to include all the traces until both datasets have been used to train and test the models. *Scenario S3:* In this scenario, we train and test the model with the entire dataset using Stratified K-Fold Cross-Validation, where k=4.

6. Evaluation of Existing Defense Mechanisms

While Sect. 4 analyzes the advantages and disadvantages of existing attacks and defenses, this section exposes further limitations of the defenses in practices that cannot be seen without extensive empirical evaluations in various settings. For example, most existing approaches have high accuracy but are evaluated on specific settings [10], [11], [41], [49], [51]. In this section, we extensively and systematically evaluate attack and defense mechanisms in a robust manner and suggest potential defense mechanisms.

Following our evaluation framework presented in Sect. 5, we conduct statistic and dynamic analyses on 51 real-world vulnerabilities archived in the National Vulnerability Database (NVD) (see Tab. 5). To assess the effectiveness of defenses, we use two key metrics: (1) True Positive Rate (TPR) or Detection Rate (DR) indicates the ratio of the correct number of vulnerabilities detected to the total number of vulnerabilities, and (2) False Positive Rate (FPR) indicates the ratio of the incorrect number of vulnerabilities detected to the total number of benign samples.

6.1. Static Analysis

We evaluate all four scanning tools, Clair [21], Trivy [22], Snyk [23], and Grype [91] as outlined in Sect. 4.2 since they are widely adopted in industry and extensively discussed in academia [17], [51], [67], [78]. Table 5 compares their results, including a separate *N/A* column for attacks that couldn't be implemented due to a lack of vulnerable software, and these are excluded from the preceding calculation. We observe that Clair has the lowest detection rate of 6.67%. Snyk has the second-highest detection rate of 28.9% and both Trivy and Grype achieve the highest detection rate at 42.2%. Unfortunately, none of the tools provide a detection rate above 50%, and only two vulnerabilities are detected by the tools individually.

Figure 4 shows the severity ratings detected by the tools when scanning the same sets of container images containing the exploits. The higher the severity rating, the more severe the vulnerability. The *Unknown* tag is given to a vulnerability, that still does not have a valid severity rating. Clair exhibits the lowest detection rate across all levels of severity, except for the Low category. Snyk presents a higher detection rate for the Low category, whereas Grype demonstrates a higher detection rate for the Medium, High, and Critical categories. The detection rate of Trivy is similar to that of Snyk for Medium and Critical severities. It has a higher detection rate than Snyk for the High category and does not detect anything at all in the Negligible category.

Existing works evaluate these tools with commonly known vulnerabilities, resulting in a high detection rate [8], [17], [51], [78]. We aim to highlight their deficiencies when identifying state-of-the-art vulnerabilities, as the detection rate falls below 50% even when combining multiple scanning tools. The inconsistencies in container scanning results, combined with low detection rates, emphasize the necessity for enhanced detection tools, whether as standalone tools or in conjunction with current scanning technology.

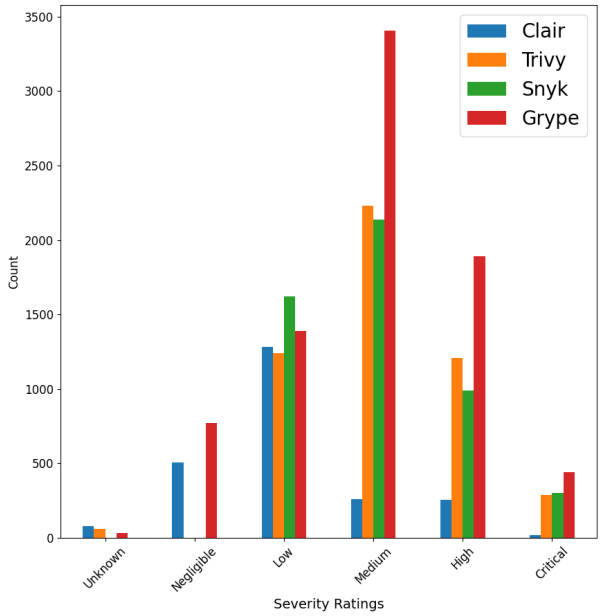
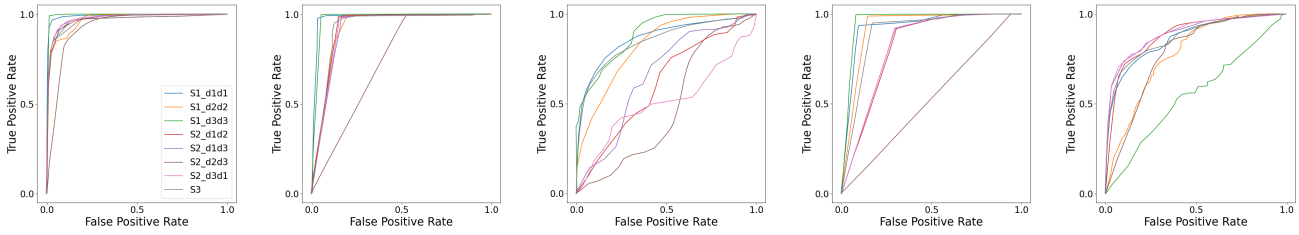


Figure 4: Severity Ratings of the scanning tools

6.2. Anomaly Detection

In this section, we evaluate the state-of-the-art anomaly detection algorithms divided into two groups, supervised and unsupervised learning, as presented in Sect. 5.

6.2.1. Supervised Algorithms. To choose the optimum parameters for each supervised algorithm, we used the parameters from [10] along with GridSearchCV [94] and experimented with some of the most common parameters of each algorithm. Details are given in Tab. 6 in Sect. 9. For scenario S2, there are six possible experiments with three distinct datasets (d1d2, d1d3, d2d1, d2d3, d3d1, d3d2). For example, d1d2 means training with dataset d1 and testing with dataset d2. However, we focus on only four of those (d1d2, d1d3, d2d3, and d3d1) that provide the best results. We plot the ROC Curves for each scenario in Fig. 5. An ROC curve (receiver operating characteristics curve) is a



(a) (b) (c) (d) (e)

Figure 5: ROC Curves of (a) Random Forest, (b) Decision Tree, (c) AdaBoost, (d) K-Nearest-Neighbors, and (e) Multi-Layer-Perceptron for all eight scenarios

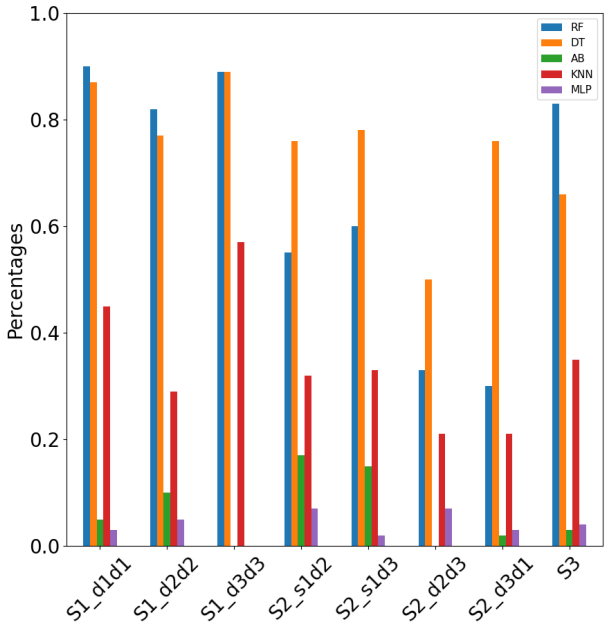


Figure 6: True Positive Rates For Anomaly Class

graph plot of the FPR against the TPR that is typically used as a performance metric for the model [95]. Figure 6 shows the detection rates for all scenarios. The xlabels are given as *Scenario_trainDatasetTestDataset*. *Random Forest (RF)* performs well for scenario S1 as shown in Fig. 5a and Fig. 6. For S2, the TPR falls under 60% due to testing with unseen data. The TPR further falls when using d2 and d3 to train the model, as these have lower training data than d1. The TPR rises for S3 with more data. *Decision Tree (DT)* performs well except for S2_d2d3 as shown in Fig. 5b, and Fig. 6, where it achieves TPR and FPR of around 50%. For other scenarios, the TPR is above 70% with an FPR of 20%. For S3, the performance slightly falls, but it is still above 60%. The increased data of S3, might be causing the DT to overfit, hence the fall in TPR. *AdaBoost (AB)* provides the worst results compared to all evaluated algorithms (see Fig. 5c). Figure 6 shows that AB archives the lowest TPR across almost all scenarios, accompanied by an increasing FPR. *K-Nearest-Neighbor (KNN)* does not perform well for any of the scenarios as shown in Fig. 5d and Fig. 6. Apart from S1_d3d3, all other scenarios have TPRs of less than 50% and the FPR is around 20% for the best and 90% for the worst scenario. *Multi-Layer-Perceptron (MLP)* does not perform well for any scenarios. Figure 6 shows TPRs below 10% across all scenarios, and Fig. 5e confirms this with an FPR of above 60% for all scenarios.

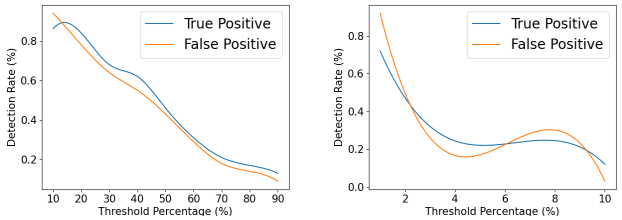
In summary, the supervised algorithms do not work well in detecting anomalies. RF and DT seem to classify the anomaly class better than other algorithms. However, there is room for improvement, as neither RF nor DT provide consistent TPRs across the different scenarios, with results falling in S2. From Fig. 6, we see that DT outperforms RF for the S2 scenario. This may occur due to the RF overfitting or the DT performing better due to feature prioritization. The results can be improved by performing dimensionality reduction before conducting the classification. Separating the scenarios based on the application and fine-tuning the algorithms might also yield better results.

Combining both static analysis and dynamic anomaly detection increases the detection rate from 88% to 92% while the individual rates are 50% and 88% respectively.

6.2.2. Unsupervised Algorithms. In this section, we analyze the detection accuracy of the widely used unsupervised algorithms such as AutoEncoder and K-Means [96], [97]. As unsupervised training does not need labeled data, we picked a different approach for testing these models as we used only benign data (the first three-minute traces of the workload) to train and the last four-minute traces to test the models. We only experiment with S3 scenario.

To choose the optimum parameter for each unsupervised algorithm, we used a combination of GridSearchCV and *threshold* value. The *threshold* value is used for validation purposes. Changing the threshold may increase the accuracy (TPR), but may also cause high FPR as well.

AutoEncoders (AE): Autoencoders consist of neural networks and contain an encode and decode region. The model tries to replicate the input data by compressing and decompressing the data with the help of neural networks. The model during training is fine-tuned to minimize the



(a) (b)

Figure 7: Detection rate (DR) of (a) AutoEncoders and (b)K-Means across all thresholds

difference between input and output. After adequate training, the model can produce the output with minimum reconstruction error compared to the original input data. The reconstruction error can then be used for anomaly detection. For example, when the autoencoder produces results with very high reconstruction errors, we can assume they came from abnormal data inputs.

Our autoencoder consists of six layers apart from the input and output layers, with the ReLu activation function in the hidden layers and the Sigmoid activation function at the output layer. The model was trained for ten epochs with the input data. The training was done to minimize mean squared error (MSE). Backpropagation is performed with the root mean square with a learning rate of 0.05.

To determine the proper threshold, we use an intuitive approach. We choose the threshold with the highest difference between true positive and false positive rates. This value was found to be at the 40th percentile. Figure 7a shows the change in true positive and false positive due to a change in threshold. Choosing a lower threshold increases the true positive rate, but it also increases the false positive rate. The opposite is true when the threshold is increased. At the 40th percentile, the true positive rate is 62%, and the false positive rate is 54%. The entire autoencoder has been implemented with Tensorflow.

K-Means (KM): K-means is a clustering algorithm that tries to group data into different sub-groups or clusters. After finding the optimum number of clusters, the algorithm assigns each data point to the nearest cluster based on some distance metric, such as Euclidean distance. Once the clustering is complete, we set a threshold to check for anomalies. The concept is that the data points situated far from the center of the cluster, i.e., have the distances to the center exceeding the threshold value are deemed to be anomalies. We evaluate this approach with various values of cluster k and found out that $k = 20$ is the optimum value via the elbow method [98]. Any data points that did not belong to any of the 20 clusters were identified as anomalies. According to Figure 7b, the greatest difference between TPR and FPR comes at the 4th percentile. At this threshold value, the TPR is 25%, and the TPR is 18%. Raising the threshold reduces both the TPR and FPR.

The unsupervised algorithms perform very poorly in detecting anomalies due to their lack of labeled data. Even though they perform better than some supervised algorithms, they also give rise to FPR. The fact that the FPR trails slightly behind the TPR means that neither of the algorithms can accurately separate the anomaly class, from the benign class. Repeating the experiment for other scenarios yields similar or poorer results.

6.3. Specific Attack Type Detection

In this part, we evaluate the detection of different attack types using supervised algorithms. We aimed to determine if testing each attack type individually produces better results, as different attacks have distinct characteristics. We group the CVEs into specific attack types by manually going through their description from NVD [75]. Table 5 shows the different exploits and their attack types. Due to the complex nature of each exploit, some CVEs encompass multiple attack types. The supervised algorithms chosen for this evaluation are the same as before (RF, DT, AB, KNN, MLP). Figure 8 shows the detection rate for each algorithm, for specific attack types.

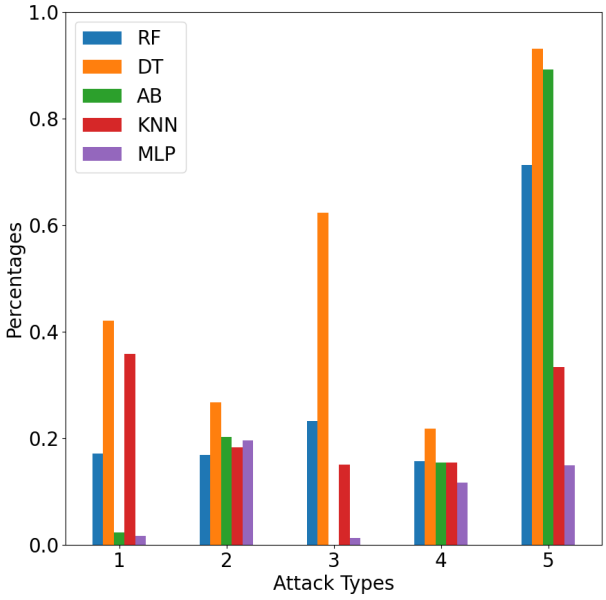


Figure 8: Performance of supervised detection models on different attack types. 1: Execute Arbitrary Code, 2: Gain Privilege, 3: Disclose Credential Information, 4: Authentication Bypass, 5: Denial Of Service

Execute Arbitrary Code: None of the models can accurately detect this type of attack. DT performs best with a TPR of around 40%, and AB and KNN perform worst with a TPR of around 5%. *Gain Privilege:* None of the models can accurately detect Gain Privilege attacks. DT performs best with a TPR of around 30%. The other algorithms detect around 20%. *Disclose Credential Information:* DT detects

around 60% for Disclose Credential Information attacks. The other algorithms perform poorly, with AB having a TPR of 0%. *Authentication Bypass:* None of the models can accurately detect Authentication Bypass attacks. DT performs best with a TPR of around 20%. The other algorithms detect less than 20%. *Denial Of Service:* RF, DT, and AB can detect Denial Of Service attacks with high TPRs of approximately 70%, 95%, and 80%, respectively. MLP performs the worst with a TPR of less than 20%.

The fact that DT performs best for all attack types corresponds with our previous experiments. Perhaps prioritizing a few features over others leads to better detection, but it does not truly explain why there are such vast differences between Disclose Credential Information and Denial Of Service attacks. Perhaps the data distribution for these two attacks makes it favorable for DT. Both Execute Arbitrary Code and Authentication Bypass have higher amounts of data, which may cause overfitting. It is also not clear why AB performs significantly well for Denial Of Service attacks but fails considerably for the other attack types.

7. Discussion of Potential Improvements

None of the scanning tools achieve a detection rate greater than 50%. In fact, the best-performing tools, Grype and Trivy, both have a detection rate of 42.2%. Combining the detection results of Grype, Snyk, and Trivy can increase the detection rate to 46.7%. Combining the results of multiple scanning tools would increase the detection rate and reduce false positives. Nevertheless, relying solely on scanning tools, even after combining multiple scanning reports is inadequate to safeguard container systems.

All the anomaly detection algorithms have poor performance. Out of the supervised algorithms, RF and DT perform better than the others as they provide a detection rate of around 80%, on the dataset that part of the data has been used for training. The performance falls by 20% for both algorithms when tested with a new dataset. Unsupervised algorithms such as AutoEncorders and K-Means suffer from poor performance. This is probably due to the lack of labels and the fact that they are unable to separate the anomaly class from the benign class. Dimensionality reduction might provide better results for both supervised and unsupervised algorithms, as this might reduce the chance of overfitting. Classifying the data into specific application domains might yield better results, as data tend to be similar, for corresponding applications. This might favor unsupervised algorithms more, as this might help them to distinguish the benign and anomalous data better.

Integrating both static and dynamic defenses would provide better security for the container ecosystem. Static defense can be used to check for preexisting vulnerabilities within a container image and are addressed accordingly. Dynamic defense can be used on top of scanning tools, to look for patterns, analyze historical data, and detect and stop attacks during the runtime. Combining both static and dynamic defenses can protect against both existing and evolving threats with the help of continuous monitoring and automation. This approach is essential for organizations attempting to fortify their containerized applications.

Future Directions. In summary, there are several research directions to tackle the limitations of existing approaches: (1) standardizing best security practices for container deployment, e.g. secure configurations, minimizing packages required, using minimal privileges, employing function filtering, securing network interface, and ensuring the integrity of images; (2) improving effectiveness and reducing the delay of scanning tools by utilizing collaborative learning for real-time vulnerability updates; (3) improving existing anomaly detection using dynamic approaches by training and tuning different models with diverse and up-to-date data; (4) exploring federated learning for anomaly detection to enhance user data privacy and model efficacy.

8. Conclusion

This paper presents a systematic and comprehensive study of existing attacks and defense mechanisms for containers. We point out the advantages and shortcomings of the existing defenses and evaluate both defense mechanisms namely *Static Scanning* and *Dynamic Anomaly Detection*. We also contribute new datasets for recent state-of-the-art attacks to evaluate existing dynamic anomaly detection models. According to our findings, neither of the defenses can fully protect containers against state-of-the-art attacks. Dynamic anomaly detection is the most sought-after defense technique, as it is the only one that can protect against zero-day vulnerabilities. However, the nature of an exploit is so complex, that it is very difficult to implement each specific attack. In addition, a lack of training data aggravates the problem further. This led to the creation of simpler defense mechanisms like scanning tools, but they also have their shortcomings. This urgently raises the need for further work to secure container-based applications.

9. Acknowledgments

Our research work was partially and jointly funded by the Horizon program of the European Union under the grant agreements No. 101093126 (ACES) and No. 101070537 (CrossCon), Deutsche Forschungsgemein-schaft (DFG) – SFB 1119 – 236615297, OpenS3 Lab and the United States Army Research Office under the grant number W911NF-23-1-0187.

References

[1] IBM, "The true benefits of moving to containers," Web page (accessed 3rd Sep 2023), https://developer.ibm.com/articles/true-benefits-of-moving-to-containers-1/.

[2] Polaris, "Survey by polaris," Webpage (accessed 19th July 2023)https://www.prnewswire.com/news-releases/global-container-as-a-service-market-share-will-surpass-usd-10-75-billion-valuation-at-22-5-cagr-growth-by-2030-polaris-market-research-301723218.html.

[3] GOOGLE, "Survey by google cloud," Webpage (accessed 19th July 2023)https://cloud.google.com/blog/transform/what-19th-century-railroad-wars-can-teach-us-about-cloud-containers.

[4] Aquasec, "Reverse shell," Webpage (accessed 19th July 2023), https://www.aquasec.com/cloud-native-academy/cloud-attacks/reverse-shell-attack/.

[5] O. I. Alqaisi, M. S. Haq, and A. S. Tosun, "Security of containerized computer vision applications," in *2022 2nd International Conference on Computing and Information Technology (ICCIT)*, 2022.

[6] Z. Jian and L. Chen, "A defense method against docker escape attack," in *Proceedings of the 2017 International Conference on Cryptography, Security and Privacy*, ser. ICCSP '17, 2017. [Online]. Available: https://doi.org/10.1145/3058060.3058085

[7] S. Flag, "Privilege escalation," Webpage (accessed 19th July 2023), https://www.aquasec.com/cloud-native-academy/cloud-attacks/reverse-shell-attack/.

[8] K. Wist, M. Helsem, and D. Gligoroski, "Vulnerability analysis of 2500 docker hub images," 2020.

[9] H. Zhu and C. Gehrmann, "Lic-sec: an enhanced apparmor docker security profile generator," *Journal of Information Security and Applications*, 2021.

[10] M. Cavalcanti, P. Inacio, and M. Freire, "Performance evaluation of container-level anomaly-based intrusion detection systems for multi-tenant applications using machine learning algorithms," in *Proceedings of the 16th International Conference on Availability, Reliability and Security*, 2021. [Online]. Available: https://doi.org/10.1145/3465481.3470066

[11] J. Flora, P. Gonçalves, and N. Antunes, "Using attack injection to evaluate intrusion detection effectiveness in container-based systems," in *2020 IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC)*, 2020.

[12] X. Gao, Z. Gu, Z. Li, H. Jamjoom, and C. Wang, "Houdini's escape: Breaking the resource rein of linux control groups," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19, New York, NY, USA, 2019. [Online]. Available: https://doi.org/10.1145/3319535.3354227

[13] Y. Sun, D. Safford, M. Zohar, D. Pendarakis, Z. Gu, and T. Jaeger, "Security namespace: Making linux security frameworks available to containers," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/sun

[14] R. Shu, X. Gu, and 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*, ser. CODASPY '17, 2017. [Online]. Available: https://doi.org/10.1145/3029806.3029832

[15] E. Socchi, "A deep dive into docker hub's security landscape - a story of inheritance?" 2019.

[16] A. Zerouali, T. Mens, and C. De Roover, "On the usage of javascript, python and ruby packages in docker hub images," *Science of Computer Programming*, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167642321000460

[17] M. S. Haq, A. Tosun, and T. Korkmaz, "Security analysis of docker containers for arm architecture," in *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*, 2022.

[18] Mitre, "Container vulnerabilities in mitre," Github page (accessed 19th July 2023)https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=container.

[19] "Log4j vulnerability," Web page (accessed 19th October 2023), https://www.cisa.gov/uscert/apache-log4j-vulnerability-guidance.

[20] "Upgaurd statistics for log4shell," Web page (accessed 19th October 2023), https://www.upguard.com/blog/apache-log4j-vulnerability.

[21] Quay, "Clair," Github page (accessed 19th July 2023), https://github.com/quay/clair.

[22] Aqua, "Aquasecurity," Github page (accessed 19th July 2023)https://github.com/aquasecurity/trivy.

[23] "Snyk vulnerability scanning tool," Webpage (accessed 19th July 2023), https://snyk.io/.

[24] B. Kaur, M. Dugré, A. Hanna, and T. Glatard, "An analysis of security vulnerabilities in container images for scientific data analysis," *GigaScience*, 2021, giab025. [Online]. Available: https://doi.org/10.1093/gigascience/giab025

[25] V. Rastogi, D. Davidson, L. De Carli, S. Jha, and P. McDaniel, "Cimplifier: Automatically debloating containers," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017. [Online]. Available: https://doi.org/10.1145/3106237.3106271

[26] Y. Luo, W. Luo, X. Sun, Q. Shen, A. Ruan, and Z. Wu, "Whispers between the containers: High-capacity covert channel attacks in docker," in *2016 IEEE Trustcom/BigDataSE/ISPA*, 2016.

[27] S. Ghavamnia, T. Palit, A. Benameur, and M. Polychronakis, "Confine: Automated system call policy generation for container attack surface reduction," in *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, 2020. [Online]. Available: https://www.usenix.org/conference/raid2020/presentation/ghavanmnia

[28] J. Wenhao and L. Zheng, "Vulnerability analysis and security research of docker container," in *2020 IEEE 3rd International Conference on Information Systems and Computer Aided Education (ICISCAE)*, 2020.

[29] A. S. Abed, C. Clancy, and D. S. Levy, "Intrusion detection system for applications using linux containers," in *Security and Trust Management*, 2015. [Online]. Available: https://doi.org/10.1007%2F978-3-319-24858-5_8

[30] D. Huang, H. Cui, S. Wen, and C. Huang, "Security analysis and threats detection techniques on docker container," in *2019 IEEE 5th International Conference on Computer and Communications (ICCC)*, 2019.

[31] "Definition of docker," Web page (accessed 19th June 2023), https://www.docker.com/.

[32] Podman, "The best free and open source container tools," Web page (accessed 19th Nov 2023), https://podman.io/.

[33] L. Containers, "Linux containers," Web page (accessed 19th Nov 2023), https://linuxcontainers.org/.

[34] K. Containers, "Kata containers," Web page (accessed 19th Nov 2023), https://katacontainers.io/.

[35] I. Cloud, "Podman vs docker: What are the differences," Web page (accessed 19th Nov 2023), https://www.imaginarycloud.com/blog/podman-vs-docker/.

[36] Sysdig, "Supply chain attacks," https://sysdig.com/blog/analysis-of-supply-chain-attacks-through-public-docker-images/.

[37] I. Tragick, "Image magick vulnerability," Github page (accessed 19th October 2023)https://github.com/ImageTragick/PoCs.

[38] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/lipp

[39] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019.

[40] L. Catuogno, C. Galdi, and N. Pasquino, "Measuring the effectiveness of containerization to prevent power draining attacks," in *2017 IEEE International Workshop on measurement and Networking (MN)*, 2017.

[41] Y. Lin, O. Tunde-Onadele, and X. Gu, "Cdl: Classified distributed learning for detecting security attacks in containerized applications," in *Annual Computer Security Applications Conference*, 2020. [Online]. Available: https://doi.org/10.1145/3427228.3427236

[42] L. Lei, J. Sun, K. Sun, C. Shenefiel, R. Ma, Y. Wang, and Q. Li, "Speaker: Split-phase execution of application containers," in *International Conference on Detection of intrusions and malware, and vulnerability assessment*, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:20448694

[43] S. Sultan, I. Ahmad, and T. Dimitriou, "Container security: Issues, challenges, and the road ahead," *IEEE Access*, 2019.

[44] Zerto, "Why containers are susceptible to ransomware," Web page (accessed 3rd Jan 2024), https://www.zerto.com/blog/ransomware-recovery/why-containers-are-susceptible-to-ransomware-how-zerto-can-help/.

[45] Snyk, "Top 5 docker security vulnerabilities," Web page (accessed 19th June 2023), https://snyk.io/learn/docker-security/top-5-vulnerabilities/.

[46] HackTricks, "Abusing docker socket for privilege escalation," Web page (accessed 19th Nov 2023), https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security/abusing-docker-socket-for-privilege-escalation.

[47] Datadog, "Dipty pipe container escape," Github page (accessed 19th July 2023)https://github.com/DataDog/security-labs-pocs/tree/main/proof-of-concept-exploits/dirtypipe-container-breakout.

[48] P. A. Networks, "Docker copy vulnerability," Github page (accessed 19th October 2023)https://unit42.paloaltonetworks.com/docker-patched-the-most-severe-copy-vulnerability-to-date-with-cve/-2019-14271/.

[49] O. Tunde-Onadele, J. He, T. Dai, and X. Gu, "A study on container vulnerability exploit detection," in *2019 IEEE International Conference on Cloud Engineering (IC2E)*, 2019.

[50] Y. Lin, O. Tunde-Onadele, X. Gu, J. He, and H. Latapie, "Shil: Self-supervised hybrid learning for security attack detection in containerized applications," in *2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, 2022.

[51] K. Brady, S. Moon, T. Nguyen, and J. Coffman, "Docker container security in cloud computing," in *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, 2020.

[52] L. Zhang, R. Cushing, C. d. Laat, and P. Grosso, "A real-time intrusion detection system based on oc-svm for containerized applications," in *2021 IEEE 24th International Conference on Computational Science and Engineering (CSE)*, 2021.

[53] F. Loukidis-Andreou, I. Giannakopoulos, K. Doka, and N. Koziris, "Docker-sec: A fully automated container security enhancement mechanism," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018.

[54] X. Lin, L. Lei, Y. Wang, J. Jing, K. Sun, and Q. Zhou, "A measurement study on linux container security: Attacks and countermeasures," in *ACSAC 2018*, 2018.

[55] N. Lopes, R. Martins, M. E. Correia, S. Serrano, and F. Nunes, "Container hardening through automated seccomp profiling," in *Proceedings of the 2020 6th International Workshop on Container Technologies and Container Clouds*, 2021. [Online]. Available: https://doi.org/10.1145/3429885.3429966

[56] J. Jia, Y. Zhu, D. Williams, A. Arcangeli, C. Canella, H. Franke, T. Feldman-Fitzthum, D. Skarlatos, D. Gruss, and T. Xu, "Programmable system call security with ebpf," 2023.

[57] S. Srinivasan, A. Kumar, M. Mahajan, D. Sitaram, and S. Gupta, "Probabilistic real-time intrusion detection system for docker containers," in *Security in Computing and Communications*, S. M. Thampi, S. Madria, G. Wang, D. B. Rawat, and J. M. Alcaraz Calero, Eds., 2019.

[58] A. Martin, S. Raponi, T. Combe, and R. Di Pietro, "Docker ecosystem – vulnerability analysis," *Computer Communications*, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366417300956

[59] Y. He, R. Guo, Y. Xing, X. Che, K. Sun, Z. Liu, K. Xu, and Q. Li, "Cross container attacks: The bewildered eBPF on clouds," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/he

[60] J. Xiao, N. Yang, W. Shen, J. Li, X. Guo, Z. Dong, F. Xie, and J. Ma, "Attacks are forwarded: Breaking the isolation of MicroVM-based containers through operation forwarding," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/xiao-jietao

[61] Prevasio, "Docker escape," Web page (accessed 3rd Sep 2023), https://www.prevasio.io/blog/kinsing-punk-an-epic-escape-from-docker-containers.

[62] Duo, "Docker bug allows root access," Web page (accessed 3rd Sep 2023), https://duo.com/decipher/docker-bug-allows-root-access-to-host-file-system.

[63] H. Gantikow, C. Reich, M. Knahl, and N. Clarke, "Rule-based security monitoring of containerized workloads," 2019.

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

[65] M. Mattetti, A. Shulman-Peleg, Y. Allouche, A. Corradi, S. Dolev, and L. Foschini, "Securing the infrastructure and the workloads of linux containers," in *2015 IEEE Conference on Communications and Network Security (CNS)*, 2015.

[66] P. Liu, S. Ji, L. Fu, K. Lu, X. Zhang, W.-H. Lee, T. Lu, W. Chen, and R. Beyah, "Understanding the security risks of docker hub," in *Computer Security – ESORICS 2020*, L. Chen, N. Li, K. Liang, and S. Schneider, Eds., 2020.

[67] O. Javed and S. Toor, "Understanding the quality of container security vulnerability detection tools," 2021.

[68] L. Chen, Y. Xia, Z. Ma, R. Zhao, Y. Wang, Y. Liu, W. Sun, and Z. Xue, "Seaf: A scalable, efficient, and application-independent framework for container security detection," *Journal of Information Security and Applications*, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S221421262200196X

[69] S. P. Mullinix, E. Konomi, R. D. Townsend, and R. M. Parizi, "On security measures for containerized applications imaged with docker," 2020.

[70] C. Lu, K. Ye, W. Chen, and C.-Z. Xu, "Adgs: Anomaly detection and localization based on graph similarity in container-based clouds," in *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, 2019.

[71] J. Franco, A. Acar, A. Aris, and S. Uluagac, "Forensic analysis of cryptojacking in host-based docker containers using honeypots," in *ICC 2023 - IEEE International Conference on Communications*, 2023, pp. 4860–4865.

[72] J. Chelladhurai, P. R. Chelliah, and S. A. Kumar, "Securing docker containers from denial of service (dos) attacks," in *2016 IEEE International Conference on Services Computing (SCC)*, 2016.

[73] A. Tomar, D. Jeena, P. Mishra, and R. Bisht, "Docker security: A threat model, attack taxonomy and real-time attack scenario of dos," in *2020 10th International Conference on Cloud Computing, Data Science Engineering (Confluence)*, 2020.

[74] Z. Zou, Y. Xie, K. Huang, G. Xu, D. Feng, and D. Long, "A docker container anomaly monitoring system based on optimized isolation forest," *IEEE Transactions on Cloud Computing*, 2022.

[75] "National vulnerability database," Web page (accessed 19th October 2023)https://nvd.nist.gov/.

[76] R. Feng, Z. Yan, S. Peng, and Y. Zhang, "Automated detection of password leakage from public github repositories," in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 175–186. [Online]. Available: https://doi.org/10.1145/3510003.3510150

[77] "Kernel message buffer," https://www.baeldung.com/linux/kernel-buffer-add-messages.

[78] "UBCIS: Ultimate benchmark for container image scanning," in *13th USENIX Workshop on Cyber Security Experimentation and Test (CSET 20)*, 2020. [Online]. Available: https://www.usenix.org/conference/cset20/presentation/berkovich

[79] S. Kwon and J. Lee, "Divds: Docker image vulnerability diagnostic system," *IEEE Access*, 2020.

[80] "Vuldb vulnerability database," Web page (accessed 19th October 2023), https://vuldb.com/.

[81] Snyk, "Snyk vulnerability database," Web page (accessed 12th Dec 2023), https://docs.snyk.io/scan-using-snyk/snyk-open-source/manage-vulnerabilities/snyk-vulnerability-database.

[82] ——, "Scan and fix misconfigurations," Web page (accessed 12th Dec 2023), https://docs.snyk.io/scan-using-snyk/scan-infrastructure/scan-your-iac-source-code/scan-terraform-files/scan-and-fix-security-issues-in-terraform-files-current-iac.

[83] "Seccomp," Web page (accessed 19th June 2023), https://man7.org/linux/man-pages/man2/seccomp.2.html.

[84] Linix, "apparmor," Web page (accessed 19th June 2023), https://apparmor.net/.

[85] "Selinux," Web page (accessed 19th June 2023), https://www.redhat.com/en/topics/linux/what-is-selinux.

[86] "Linux capabilities," Web page (accessed 19th June 2023), https://man7.org/linux/man-pages/man7/capabilities.7.html.

[87] Sysdig, "Sysdig," Web page (accessed 3rd March 2023), https://sysdig.com/.

[88] Sysdig and Falco, "Falco," Web page (accessed 3rd March 2023), https://falco.org/.

[89] V. Total, "Virus total-intelligence overview," Web page (accessed 3rd Sep 2023), https://www.virustotal.com/gui/intelligence-overview/.

[90] CLAM, "Clam av," Web page (accessed 3rd Sep 2023), https://www.clamav.net//.

[91] Anchore, "Grype scanning tool," Web page (accessed 19th June 2023), https://github.com/anchore/grype.

[92] "Anchore-engine," Web page (accessed 3rd March 2023), https://anchore.com//.

[93] Apache, "Jmeter," Web page (accessed 19th June 2023), https://jmeter.apache.org/.

[94] sklearn, "Gridsearchcv," Web page (accessed 3rd Sep 2023), https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.

[95] G. Developers, "Classiifcation: Roc curve and auc," Web page (accessed 19th June 2023), https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc.

[96] Sklearn, "K-means," Web page (accessed 12th Dec 2023), https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html.

[97] Tensorflow, "Intro to autoencoders," Web page (accessed 12th Dec 2023), https://www.tensorflow.org/tutorials/generative/autoencoder.

[98] A. Vidhya, "Choosing the optimum number of clusters," Web page (accessed 3rd March 2023), https://www.analyticsvidhya.com/blog/2021/05/k-mean-getting-the-optimal-number-of-clusters/.

Appendix A.
Model parameters

Table 6 shows the parameters used for the different supervised algorithms. The parameters were chosen using GridSearchCV.

Algorithm (Abbv)	Parameters
Random Forest (RF)	n_estimators=200, bootstrap=False, class_weight='None' max_depth=4, max_features='sqrt', min_samples_leaf=1, min_samples_split=2
Decision Tree (DT)	max_features=200, class_weight=None, max_depth=None min_samples_leaf=1, min_samples_split=2
AdaBoost (AB)	n_estimators=200,
K-Nearest Neighbor (KNN)	algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None,n_neighbors=5, p=2, weights='uniform'
Multi-Layer Perceptron (MLP)	solver='adam', alpha=1e-5, max_iter=1000,activation='relu', hidden_layer_sizes=(5, 2), random_state=1

TABLE 6: Displaying parameters for different algorithms and their abbreviations

Appendix B.
Meta-Review

The following meta-review was prepared by the program committee for the 2024 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

B.1. Summary

This paper provides a comprehensive analysis and systematization of knowledge of security vulnerabilities in container-based applications, a critical concern given their reliance on a shared OS kernel. It systematically categorizes container attacks, assesses current defense mechanisms, and highlights their limitations. It introduces a comprehensive evaluation framework using a dataset of 51 real-world vulnerabilities and tests various static and dynamic defense strategies, including anomaly detection methods.

B.2. Scientific Contributions

- Independent Confirmation of Important Results with Limited Prior Research
- Creates a New Tool to Enable Future Science Provides a Valuable Step Forward in an Established Field
- Creates a New Tool to Enable Future Science Provides a Valuable Step Forward in an Established Field

B.3. Reasons for Acceptance

1) Important topic: The Systematization of Knowledge (SoK) on Container Security provides significant benefits to the research community, considering the widespread use of containers and the escalating security threats.
2) The paper offers a detailed and high-quality Systematization of Knowledge (SoK) container threats, attack, and defense mechanisms.
3) Comprehensive evaluation framework with new data
4) The re-evaluation of existing defenses provides highly useful insights about the pros and cons of these techniques and suggestions to enhance container security.
5) The paper is well-written