Proceedings of the 10th World Congress on Electrical Engineering and Computer Systems and Sciences (EECSS'24)
Barcelona, Spain - August 19 - 21, 2024
Paper No. CIST 164
DOI: 10.11159/cist24.164

Mitigating Container Escape Threats Through Effective Countermeasures: A Survey

Nada Barnawi [1], Razan AITooq [2], Dr. Mohammed Almukaynizi [3]
Information Systems Department / King Saud University
Riyadh 11362, Saudi Arabia
First. nbarnawi@ksu.edu.sa ; Second raltoq@ksu.edu.sa .; Third. malmukaynizi@ksu.edu.sa

Abstract **-** The growing prevalence of container escape threats poses a significant challenge in the domain of cloud computing and containerized environments. This paper conducts a comprehensive review of existing literature to examine these threats and identify effective countermeasures. It underscores the criticality of profile misconfiguration and vulnerabilities in the container runtime as primary sources of risk within containerized environments. Profile misconfiguration arises when container settings are improperly configured, while vulnerabilities in the container runtime expose systems to exploitation. The paper also explores various countermeasures, including best practices for mitigating container escape threats and the utilization of tools and techniques for bolstering security. Ultimately, it emphasizes that by recognizing the potential risks and diligently implementing suggested countermeasures, organizations can substantially fortify the security posture of their containerized environments.

Keywords**:** Container security, Escape threats, Countermeasures, Cloud computing threats

1. Introduction

Cloud computing has revolutionized the way organizations handle data, offering enhanced scalability, flexibility, and cost efficiency. However, its adoption has also raised concerns related to security, privacy, and compliance, as organizations navigate the complexities of integrating cloud technology while ensuring data protection and adherence to regulations.

Cloud computing is vulnerable to various threats and risks, including data breaches due to the inability of network-based intrusion detection and prevention systems to monitor data traffic, as identified by NIST [1]. Another significant risk involves malicious insiders, particularly as data and applications transition to cloud environments, with the insider threat extending to cloud service provider employees, contractors, and customers [2]. Additionally, the adoption of container technology, a lightweight virtualization alternative that facilitates quicker application deployment and greater flexibility, has surged [3]. However, this technology also introduces new security challenges, notably the risk of container escapes, posing a growing concern within cloud security.

Containers are independent packages that efficiently run service applications using "Namespaces" and "cgroups" to ensure secure and efficient functionality, with a focus on protecting container images and orchestrators from security threats [4]. In cloud computing, container-based security solutions are crucial as they facilitate rapid and secure application deployment and scaling, providing isolation between the OS and applications to reduce configuration errors and improve reliability [5]. However, containers face numerous security challenges stemming from vulnerabilities in container images, misconfigurations, and resource abuse, compounded by their shared use of host system resources and the kernel [6][7][8]. These vulnerabilities can lead to complex security issues, notably container escapes, where attackers exploit these weaknesses to gain unauthorized access to the host system, posing significant risks to the integrity of container host systems [9][10].

The rest of this paper is organized as follows; Section 2 presents a briefing on the previous studies in the on-container threats. In Section 3, we discuss container escape threats and provide some vulnerabilities. Profile misconfiguration and container runtime vulnerabilities are the two main subtopics of container escape threats. Section 4 presents countermeasures for container escape threats with two subtopics: practices to avoid container escape threats and tools and techniques for counter-measuring container escape threats. Finally, Section 5 concludes this paper.

2. Literature Review

In [9], Reeves et al., studied the container runtime vulnerabilities by focusing on 59 CVEs for 11 different container runtimes and showed that applying user namespaces to containers may effectively prevent container escapes, while Abbas et al, [11] had worked to identify vulnerabilities in namespace isolation by applying a novel provenance-based container escape detection system called PACED, which detects the malicious entities responsible for bypassing the container boundary.

Both papers [12][6] address security concerns related to container usage, offering recommendations. Sultan et al.'s paper focuses on four generalized use cases, discussing security requirements within the host-container threat landscape. Meanwhile, NIST SP 800-190 provides practical suggestions for planning, implementing, and maintaining container technologies, tailored to specific components or tiers within the architecture. Additionally, paper [13] conducts a comprehensive study on security measures for safeguarding containerized applications, including vulnerability analysis and evaluation of measures like access controls and intrusion detection systems. The findings offer insights into best practices and guidance for organizations deploying containerized applications in the cloud.

In [4], it looked at the security of Linux containers and evaluated the effectiveness of various security measures in protecting them from attacks. The research provides a comprehensive overview of Linux container security and valuable insights into the effectiveness of various security measures. Some of the attacks were explained as privilege escalation, resource exhaustion, and denial of service. Lin et al., conducted several experiments to evaluate the security of containers, and assess the effectiveness of various security measures.

Wong et al. [14], have performed threat modelling on the container ecosystem by using STRIDE to identify the vulnerabilities in each system component. Also, they conducted a comprehensive survey on the existing countermeasures designed against the identified threats and vulnerabilities in containers.

Multiple studies delved into Docker's security mechanisms and identified potential threats. Huang et al. introduced a threat-detecting framework for Docker containers, addressing security concerns in both image and instance levels, with experimental validation [15]. Jian and Chen proposed a defense method based on namespace status inspection to detect abnormal processes and prevent escape behaviors [16]. Another study focused on Docker container security and outlined preventive measures against attacks on compromised containers [17]. MacLeod extensively analyzed these techniques, covering kernel exploits, privilege escalation, and network attacks.

3. Container Escape Threats

The security of a container service relies heavily on factors such as the execution environment, kernel version, permissions, and provider policies. Neglecting to manage the threats and attacks within this environment can jeopardize the confidentiality, integrity, and availability of both applications and infrastructure. Containerization users face increasing concerns over security threats and attacks. Attackers often target isolation vulnerabilities, as compromising the kernel or breaching container boundaries can lead to significant impacts, known as container escape [18] [19].

A container escape attack is a type of security breach that occurs when an attacker can compromise a container and exploit vulnerabilities in the container's runtime, container application, or host operating system [20]. It can use these vulnerabilities to defeat the isolation and security mechanisms constraining the container and gain unauthorized access to the host operating system or other containers on the same network [18].

Therefore, to increase container preservation from container escaper attackers, we first need to define and analyse the cause of the occurrence of this attack and then how they can exploit potential security threats and vulnerabilities in the container system components.

Container escapes can happen in several ways, such as by exploiting software in privileged mode and exploiting vulnerabilities like kernel bugs, weak access controls, or poor configuration to gain more privileges. Escalating privileges on the host can also give access to other containers. These vulnerabilities are especially hazardous as they can enable attackers to get sensitive data, affect other systems, and cause severe harm to the infrastructure at risk [21].

So, we will explain more about container escape through the two main categories of vulnerabilities accessible from within the container, and how the adversaries exploit to escape a container: Profile misconfiguration and container runtime. runtime.

3.1. Profile misconfiguration.

Misconfigurations in container security profiles can lead to container escapes, allowing attackers to breach the isolation isolation mechanisms and gain access to the host system [20]. These profiles dictate the security policies and limitations of a container, regulating its access to system resources and communication with other containers and the host [6]. Exploiting such misconfigurations, adversaries can launch various attacks, including network-based ones like man-in-the-middle or DNS spoofing attacks, from within the container [20]. Research shows that a significant percentage of exploits (56.82% out of 223) can succeed with default configurations [4].

A misconfigured profile can allow an attacker to escape the container and gain access to the underlying host system in several ways:

- **Allowing privileged access:** Privilege escalation occurs when attackers gain elevated access to a host system through misconfigured permissions or security profiles within containerized applications [22]. This can lead to complete control over the host, potentially compromising hardware resources and other networked hosts [23], [21]. Privileged containers serve as potential entry points for further attacks, enabling the spread of malware or unauthorized access to other containers and hosts [20]. Such attacks can result in severe consequences, including data breaches, container destruction, or manipulation of critical files [21].

A notable example is the CVE-2018-1002105 vulnerability in the Kubernetes API Server, which allows attackers to gain full admin privileges on a cluster. This vulnerability affects both authenticated users with specific pod privileges and unauthenticated remote users via the aggregated API server. By exploiting certain API calls, attackers can execute commands against the kubelet API on a specified node, posing a significant risk due to the default authorization settings allowing discovery API calls [24], [25].

- **Allowing access to sensitive resources:** Containers that are given access to critical system resources, such as the host's file system or network, can be exploited by attackers to infiltrate the underlying host system, steal sensitive data, or launch further attacks. For instance, attackers could manipulate the host's file system access to alter important files or access sensitive data like user information or configuration files. Similarly, granting a container access to the host's network could facilitate additional attacks on the host. An example of such vulnerability is CVE-2016-5195, also known as the Copy-On-Write (COW) vulnerability in Docker, where attackers could gain unauthorized read and write access to read-only files and directories on the host system. This flaw, originating from a flaw in Docker's implementation of copy-on-write functionality, enabled attackers to manipulate the copy-on-write layer, granting them access to the underlying read-only layer and potentially allowing them to modify sensitive files, replace system binaries, and execute arbitrary code with elevated privileges [26][27].

- **Allowing communication with other containers:** Adversaries exploit misconfigurations to enable communication between containers by delving into their internal networking mechanisms. Containers utilize tools like Docker or Kubernetes for internal communication [28], each having its own virtual interface and IP address within a designated network [12]. They share a network namespace, facilitating communication akin to a single network [29], exchanging network packets through this virtual interface.

The default bridge network in many containers poses network risks due to default settings, potentially allowing unrelated containers or services to communicate, thus creating security vulnerabilities. Attackers could exploit this by gaining access to a container and opening a listening port to other containers on the same network [30]. For instance, vulnerabilities in Kubernetes permitted attackers to escape a container and access the host system through a misconfigured network policy [31].

Containerization involves sharing virtual instances and resources within a single operating system [29], relying on the host system to provide OS, hardware resources, and system-level services [6]. Communication between containers relies heavily on the host, often through shared volumes for data exchange and collaboration [32]. However, security depends on proper host configuration to prevent vulnerabilities, such as running vulnerable software or improperly configuring ports, which could lead to unauthorized access or arbitrary code execution [33]. Exploiting vulnerabilities in the host system can compromise other containers and the host itself [34], such as the CVE-2015-3630 vulnerability in Docker, where containers could manipulate shared resources to modify host kernel parameters, potentially accessing sensitive information. Separating hosts and containers with restricted resources can mitigate such risks, as certain shared directories like /proc and /sys can be exploited by malicious containers to gather information about others [35].

The host plays a crucial role in enabling communication among containers through network and storage resources. To safeguard containers, it refrains from making assumptions and focuses on specific aspects of the host OS, network setups, or other elements within the container runtime environment that could undergo frequent changes [6].

3.2. Container Runtime.

A container runtime manages the creation and operation of containers on a host computer, facilitating interaction between containers and the host OS [6]. It handles tasks like environment setup, networking, storage, and security, and oversees container lifecycle stages such as initialization, termination, and removal. Common container runtimes, like runC, containerd, and cri-o, adhere to the Open Container Initiative (OCI) specification [36]. Vulnerabilities in the container runtime can be exploited by attackers to execute arbitrary code, potentially leading to container escapes and unauthorized access to the host system. For example, CVE-2019-5736 exposes a weakness in the runC container runtime, allowing attackers to overwrite the runtime binary and execute malicious code on the host, thereby achieving a successful container escape [9].

Runtime software vulnerabilities can be particularly dangerous as they can enable "container escape" scenarios, allowing malicious software to target resources in other containers and the host's operating system. Attackers can exploit these vulnerabilities to compromise the runtime software, modify it to gain access to other containers, monitor container-to-container communications, and perform other unauthorized actions [6]. Moreover, if an application requires root privileges or parts of full root access, it could potentially gain control over the container manager and target the host system and other containers within the system, or specifically target a vulnerable container. CVE-2017-5123 mentions a vulnerability in the Docker runtime that allows applications to modify the capabilities [12].

Container runtimes offer numerous configurable options that, if not set properly, can significantly decrease the security of the system. For instance, widening the default set of allowed system calls on Linux container hosts can increase the risk of a compromised container affecting other containers and the host OS. Running a container in privileged mode can also allow it to act as part of the host OS, thereby impacting all other containers running on it. Another example of an insecure runtime configuration is allowing containers to mount sensitive directories on the host, which could allow a compromised container to elevate privileges and attack the host and other containers [6].

4. Countermeasure for Container Escape Threats

To address emerging threats in container environments, it's crucial to prioritize the security of containerized applications. This involves implementing effective countermeasures to mitigate risks, such as container escapes. Best practices and frameworks are available to help organizations achieve this efficiently [32].

4.1. Some tools and techniques for countermeasure containers escape threats.

I. **Container isolating mechanisms:** A defence method against Docker escape attacks involves using Linux namespaces and relevant security mechanisms to isolate containers from the host system.

- Namespaces mechanism**:** The study highlights the significance of namespaces in Linux, particularly in container security [28]. It explains how user namespaces ensure secure isolation by mapping UIDs and GIDs inside containers to different ones outside, effectively preventing escape attempts. Reeves et al. found user namespaces effective in thwarting exploits, with seven out of nine attempts blocked in various runtime environments [9]. However, they note that user namespaces may offer only a partial solution in some scenarios. Another study proposes a dynamic defense mechanism against Docker escape attacks, utilizing process state examination within namespaces, implemented via Shell Scripting Language and the PSTREE tool for process enumeration [16].

- Cgroup mechanism: Control groups, or Cgroups, are a Linux kernel technology that organizes task processes into hierarchical groups to restrict and control resource usage [37]. The Cgroup mechanism focuses on performance isolation by by limiting resources such as memory, CPU, and devices [4]. Sultan et al. suggested using Cgroup as one of the isolation techniques to address vulnerabilities like CVE-2017-5123 and CVE-2016-5195, along with periodic vulnerability scanning scanning for images, applications, and container runtime [12].

II. **Kernel security mechanisms:** Containers are gaining more favor than VMs due to their decreased virtualization overhead and quicker deployment, highlighting the importance of secure kernel sharing for container security. Critical kernel security features such as Capability, Seccomp, and MAC are essential for preventing privilege escalation, operating in tandem. Lin et al. proposed a defense strategy to counter privilege escalation attacks by limiting access to the kernel function commit-creds(). Exploiting vulnerabilities in the Linux kernel could allow attackers to utilize commit-creds() to gain root privileges, thus gaining full control over the system [4].

- Capability in Linux, a security mechanism, restricts process privileges by assigning specific privileges within the kernel. It's crucial for processes within containers, aiding tasks like configuring IP addresses and firewalls, thus reducing the risk of exploits in compromised API implementations. However, an abundance of capabilities can make strict Seccomp configuration impractical [37].

- Seccomp, a Linux kernel feature, filters system calls to bolster security. Ghavamnia et al. proposed a Seccomp policy for containers, enabling effective countermeasures against attacks by configuring allow/deny lists of system calls. Leveraging eBPF, this policy facilitates efficient tracing and monitoring in Docker containers, enhancing security by limiting kernel resource access [38] [4].

- Mandatory Access Control (MAC) on Linux, enforced via frameworks like SELinux and AppArmor, regulates resource access through administrator-defined policies. These policies fortify kernel and container security [4].

III. **Linux Security Modules (LSM):** A collection of kernel-level security frameworks safeguards Linux, addressing operating system-level threats with flexibility. Various modules within the LSM framework offer interchangeable security primitives, including SELinux, AppArmor, and Integrity Measurement Architecture (IMA) [39], [40].

- AppArmor, a Linux security module, safeguards both the OS and applications from security threats by employing path-based profiles and supporting both application and complaint mode profiles. It offers Mandatory Access Control (MAC) and simplifies development through file utilization, making it more accessible compared to other MAC systems [41] [42].

- SELinux, another robust framework, enables Role-Based Access Control (RBAC) or Mandatory Access Control (MAC) by enforcing policies and implementing a file labeling system. It ensures resource policies dictate access levels for users, programs, and services, and it isolates containers to prevent interference from root processes outside the container [43][37].

- Integrity Measurement Architecture (IMA), also a Linux security module, focuses on monitoring system processes and file integrity during execution. By calculating hash values for specific files and verifying them against expected values, it ensures system integrity. This involves creating unique hash values for images or data, providing a digital fingerprint for each [40][44].

Linux IMA, endorsed by the Trusted Computing Group (TCG), is deemed a convenient technology for runtime integrity checking. The Trusted Platform Module (TPM), a hardware chip prevalent in modern processors, aids IMA by providing integrity verification at runtime. Antonio Lioy et al. highlighted the significance of integrity measurement architecture [45][44].

IV. **Countermeasure in misconfiguration:** Countermeasures against misconfigurations involve employing tools like "Cilium" for container networking and security. Cilium offers a centralized management console to easily configure security policies, reducing the risk of misconfigurations and enhancing overall security in containerized environments [4] [44]. Additionally, Aqua's "kube-hunter" tool is recommended as a countermeasure. This open-source tool detects Kubernetes nodes and conducts automated penetration testing, identifying misconfigurations and vulnerabilities such as the Kubernetes Privilege Escalation vulnerability (CVE-2018-1002105) [24].

4.2. Practices to avoid container escape threats.

There are several practices that can be followed to mitigate container escape attacks. In the following sections, the most important practices that reduce threats will be mentioned.

- Image Countermeasures emphasize the importance of secure container image usage in container security [46]. Container images contain all necessary files, tools, and dependencies for running a container [6]. Enhancing security involves analyzing these images with vulnerability identification tools and applying security features and controls [1]. Careful image integration management is necessary to enforce security policies, including securing vulnerabilities across all image layers and monitoring the entire creation process [6][45]. Compliance verification ensures adherence to security standards and continuous monitoring for risks and weaknesses, with trusted base layers recommended for minimizing attack surfaces [6].
- Secure container runtime configurations, exemplified by the Open Container Initiative (OCI) established in 2015, standardize container creation and runtime for industry uniformity [12]. It defines requirements for runtimes and images, with runC as the default runtime, allowing for targeted security measures on each component [20]. Emphasizing security, recommendations include using mandatory access control (MAC) techniques and following consensus-based benchmarks like the Center for Internet Security Docker Benchmark [6]. These standards are developed through collaboration across sectors to represent best practices [47][41].
- Unauthorized access Countermeasures involve implementing strong authentication measures, such as multi-factor authentication, for administrative accounts to prevent unauthorized access to containers [6]. Mechanisms like SCONE secure containers from external attacks [38]. Risks of resource sharing among containers on the same host can be mitigated through multi-factor authentication and network segmentation [12].

6. Conclusion

In conclusion, container escape threats pose a significant security risk in container environments, with profile misconfiguration and container runtime vulnerabilities being the two main subtopics. Profile misconfiguration can enable attackers to escape containers and gain privileged access, sensitive resource access, and communication with other containers. Container runtime vulnerabilities can lead to container escapes, allowing attackers to compromise runtime software and access other containers. Poorly configured runtime options, such as privileged mode and mounting sensitive directories, can increase the risk of a security breach. To mitigate container escape threats, organizations must utilize appropriate tools and techniques, like container isolating mechanisms, kernel security mechanisms, Linux security mechanisms, Linux Security Modules (LSM), and countermeasures for misconfiguration. A protective measure is to utilize Linux namespaces and appropriate security mechanisms to isolate containers and the host system. Kernel security mechanisms such as Capability, Seccomp, and MAC are essential parts of privilege escalation prevention as they work concurrently with each other. CPU protection mechanisms are also commonly used to prevent attacks on the Linux kernel. Linux security modules (LSM) are a set of kernel-level security frameworks that protect Linux and mitigate issues at the operating system level.

There have some practices and roles for a container environment, such as making sure to use secure container images and secure container runtime configurations, and there are several measures that can be taken to prevent unauthorized access to containers. Ultimately, container escape threats are a major security concern in cloud computing environments. However, by implementing best practices and using appropriate tools and technologies, organizations can mitigate the risks associated with container escape threats and ensure the security of their container environments.

References

[1] Jansen, Wayne, and Tim Grance. "Guidelines on security and privacy in public cloud computing." (2011).

[2] Kandias, M., Virvilis, N., & Gritzalis, D. (2013). The insider threat in cloud computing. In *Critical Information Infrastructure Security: 6th International Workshop, CRITIS 2011, Lucerne, Switzerland, September 8-9, 2011, Revised Selected Papers 6* (pp. 93-103). Springer Berlin Heidelberg.

[3] Joy, Ann Mary. "Performance comparison between Linux containers and virtual machines." *2015 international conference on advances in computer engineering and applications*. IEEE, 2015.

[4] Lin, X., Lei, L., Wang, Y., Jing, J., Sun, K., & Zhou, Q. (2018, December). A measurement study on linux container security: Attacks and countermeasures. In *Proceedings of the 34th annual computer security applications conference* (pp. 418-429).

[5] Kozhirbayev, Zhanibek, and Richard O. Sinnott. "A performance comparison of container-based technologies for the cloud." *Future Generation Computer Systems* 68 (2017): 175-182.

[6] M. Souppaya, J. Morello, and K. Scarfone, “Application container security guide,” National Institute of Standards and Technology, Tech. Rep., 2017.

[7] T. Shaffer, T. S. Phung, K. Chard, and D. Thain, “Landlord: Coordinating dynamic software environments to reduce container sprawl,” IEEE Transactions on Parallel and Distributed Systems, vol. 34, no. 5, pp. 1376–1389, 2023.

[8] S. Mart ́ınez-Magdaleno, V. Morales-Rocha, and R. Parra, “A review of security risks and countermeasures in containers,” International Journal of Security and Networks, vol. 16, no. 3, pp. 183–190, 2021.

[9] M. Reeves, D. J. Tian, A. Bianchi, and Z. B. Celik, “Towards improving container security by preventing runtime escapes,” in 2021 IEEE Secure Development Conference (SecDev). IEEE, 2021, pp. 38–46.

[10] “The route to root container escape using kernel exploitation,” https://www.cyberark.com/resources/threat-research-blog/ the-route-to-root-container-escape-using-kernel-exploitation, 10 2020, (undefined 13/5/2023 7:16).

[11] M. Abbas, S. Khan, A. Monum, F. Zaffar, R. Tahir, D. Eyers, H. Irshad, A. Gehani, V. Yegneswaran, and T. Pasquier, “Paced: Provenance-based automated container escape detection,” in 2022 IEEE International Conference on Cloud Engineering (IC2E). IEEE, 2022, pp. 261–272.

[12] S. Sultan, I. Ahmad, and T. Dimitriou, “Container security: Issues,\ challenges, and the road ahead,” IEEE access, vol. 7, pp. 52 976–52 996, 2019.

[13] B.-C. Tak, C. Isci, S. S. Duri, N. Bila, S. Nadgowda, and J. Doran, “Understanding security implications of using containers in the cloud.” in USENIX annual technical conference, 2017, pp. 313–319.

[14] A. Y. Wong, E. G. Chekole, M. Ochoa, and J. Zhou, “On the security of containers: Threat modeling, attack analysis, and mitigation strategies,” Computers & Security, vol. 128, p. 103140, 2023.

[15] 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). IEEE, 2019, pp. 1214–1220.

[16] Z. Jian and L. Chen, “A defense method against docker escape attack,” in Proceedings of the 2017 International Conference on Cryptography, Security and Privacy, 2017, pp. 142–146.

[17] M. MacLeod, “Escaping from a virtualised environment: An evaluation of container breakout techniques,” 2021.

[18] Y. Wu, L. Lei, Y. Wang, K. Sun, and J. Meng, “Evaluation on the security of commercial cloud container services,” in Information Security: 23rd International Conference, ISC 2020, Bali, Indonesia, December 16–18, 2020, Proceedings 23. Springer, 2020, pp. 160–177.

[19] H. Perera, B. Reza, H. De Silva, A. Karunarathne, B. Ganegoda, and A. Senarathne, “Docker container security orchestration and posture management tool,” in 2022 13th International Conference on Computing Communication and Networking Technologies (ICCCNT). IEEE, 2022, pp. 1–6.

[20] M. J. Reeves, “Investigating escape vulnerabilities in container run- times,” Ph.D. dissertation, Purdue University Graduate School, 2021.

[21] “Stop container escape and prevent privilege escalation,” https: //goteleport.com/blog/stop-container-escape-privilege-escalation/, (Ac- cessed on 05/10/2023).

[22] “Why running a privileged container is not a good idea cloud native now,” https://cloudnativenow.com/topics/cloudnativesecurity/why-running-a-privileged-container-is-not-a-good-idea/, (undefined 13/5/2023 7:20).

[23] “Why running a privileged container is not a good idea - cloud native now,” https://cloudnativenow.com/topics/cloudnativesecurity/ why-running-a-privileged-container-is-not-a-good-idea/, (Accessed on 05/10/2023).

[24] “Severe privilege escalation vulnerability in kuber- netes (cve-2018-1002105),” https://blog.aquasec.com/ kubernetes-security-cve-2018 1002105, (Accessed on 05/10/2023).
[25] S. Qi, S. G. Kulkarni, and K. Ramakrishnan, “Assessing container network interface plugins: Functionality, performance, and scalability,” IEEE Transactions on Network and Service Management, vol. 18, no. 1, pp. 656–671, 2020.\
[26] D. Alam, M. Zaman, T. Farah, R. Rahman, and M. S. Hosain, “Study of the dirty copy on write, a linux kernel memory allocation vulnerability,” in 2017 International Conference on Consumer Electronics and Devices (ICCED). IEEE, 2017, pp. 40–45
[28] C. Abdelmassih, “Container orchestration in security demanding envi- ronments at the swedish police authority,” 2018.
[29] K. Suo, Y. Zhao, W. Chen, and J. Rao, “An analysis and empirical study of container networks,” in IEEE INFOCOM 2018-IEEE Conference on Computer Communications. IEEE, 2018, pp. 189–197.
[30] A. Y. Wong, E. G. Chekole, M. Ochoa, and J. Zhou, “Threat mod eling and security analysis of containers: A survey,” arXiv preprint arXiv:2111.11475, 2021.
[31] G. Budigiri, C. Baumann, J. T. M¨uhlberg, E. Truyen, and W. Joosen, “Network policies in kubernetes: Performance evaluation and security analysis,” in 2021 Joint European Conference on Networks and Com- munications & 6G Summit (EuCNC/6G Summit). IEEE, 2021, pp. 407–412.
[32] R. K. Barik, R. K. Lenka, K. R. Rao, and D. Ghose, “Performance analysis of virtual machines and containers in cloud computing,” in 2016 international conference on computing, communication and automation (iccca). IEEE, 2016, pp. 1204–1210.
[33] “2018 nsfocus technical report on container security - nsfocus, inc., a global network and cyber security leader, protects enterprises and carriers from advanced cyber attacks.” https://nsfocusglobal.com/company-overview/resources/ 2018-nsfocus-technical-report-container-security/, (Accessed on 05/10/2023).
[34] “Advances in cyber security: Third international conference, aces 2021, penang, malaysia, august 24–25, 2021, revised selected papers — springerlink,” https://link.springer.com/book/10.1007/ 978-981-16-8059-5, (Accessed on 05/11/2023).
[35] T. Combe, W. Mallouli, T. Cholez, G. Doyen, B. Mathieu, and E. Montes de Oca, “An sdn and nfv use case: Ndn implementation and security monitoring,” Guide to Security in SDN and NFV: Challenges, Opportunities, and Applications, pp. 299–321, 2017.
[36] “3 types of container runtime and the kubernetes connection,” https://www.aquasec.com/cloud-native-academy/container-security/container-runtime/#:∼:text=What%20Is%20a%20Container% 20Runtime,on%20a%20host%20operating%20system, (Accessed on 05/11/2023).
[37] M. Garrett, “Container security with selinux and coreos,” 2015.
[38] S. Ghavamnia, T. Palit, A. Benameur, and M. Polychronakis, “Confine: Automated system call policy generation for container attack surface reduction,” in International Symposium on Research in Attacks, Intrusions and Defenses (RAID), 2020.
[39] C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman, “Linux security modules: General security support for the linux kernel,” in 11th USENIX Security Symposium (USENIX Security 02), 2002.
[40] M. B´elair, S. Laniepce, and J.-M. Menaud, “Leveraging kernel security mechanisms to improve container security: a survey,” in Proceedings of the 14th international conference on availability, reliability and security, 2019, pp. 1–6.
[41] “Apparmor security profiles for docker docker documentation,” https://docs.docker.com/engine/security/apparmor/, 03 2016, (undefined 29/5/2023 6:19).
[42] “Apparmor ubuntu wiki,” https://wiki.ubuntu.com/AppArmor, 11 2006, (undefined 29/5/2023 6:26).
[43] 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). IEEE, 2016, pp. 856–859.
[44] M. De Benedictis and A. Lioy, “Integrity verification of docker containers for a lightweight cloud environment,” Future generation computer systems, vol. 97, pp. 236–246, 2019.

[45] R. Sailer, X. Zhang, T. Jaeger, and L. Van Doorn, “Design and implementation of a tcg-based integrity measurement architecture.” in USENIX Security symposium, vol. 13, no. 2004, 2004, pp. 223–238.
[46] B. Kaur, M. Dugr ́e, A. Hanna, and T. Glatard, “An analysis of security vulnerabilities in container images for scientific data analysis,” GigaScience, vol. 10, no. 6, p. giab025, 2021.
[47] “What is selinux,” https://www.redhat.com/en/topics/linux/ what-is-selinux, 11 2019, (undefined 29/5/2023 6:32).