Secure and Resource Effective Usage of Containers for Military and Critical Infrastructure Environments

Vetle Gjersøe, Mats Alexander Westad Asbjørnsen, Siv Hilde Houmb, and Kirsi Marjaana Helkala

Norwegian Defence University College / Norwegian Defence Cyber Academy, Lillehammer, Norway
vgjersoe/matsasbjornse/shoumb/khelkala@mil.no

Abstract. Virtualization is an essential part of modern ICT infrastructures and provides both resource and cost efficiency, as well as eases the operation and maintenance of such infrastructures. Currently, virtualization environments are most often based on virtual machines (VMs). However, compared to Containers, such as Kubernetes, VMs have overhead built into the architecture. This work examines the secure and resource efficient use of Containers for critical infrastructures such as Cyber Physical Systems (CPS). More specifically, this research assessed the suitability of Kubernetes as a lightweight alternative to virtual machines for military use and its compliance with the NATO CIS infrastructure requirements. Furthermore, this work examines the risk of container escape and how to setup a secure deployment of Kubernetes environments in practice.

Keywords: Critical Infrastructure · Cyber Physical Systems (CPS) · Cyber Security · Virtualization · Kubernetes.

1 Introduction

Virtualization has long contributed to improved efficiency and security for systems and applications[13], a development of great relevance for critical infrastructure, particularly within the military sector. In modern military systems, virtualization is considered state of the art because it enables rapid deployment of new capabilities, supports legacy software on modern hardware, and allows platforms to dynamically adapt to changing mission demands - all while reducing the attack surface through isolation and segmentation. This flexibility is essential in operational environments where mission parameters, threat landscapes, and hardware availability can shift in real time.

Although traditional use of virtual machines has been the norm, this approach presents challenges related to high costs and resource consumption [7]. By analyzing the security profile, performance, and scalability of containerized environments, this research assessed the suitability of Kubernetes as a lightweight alternative to virtual machines for military use and its compliance with the NATO Communication and Information System (CIS) infrastructure requirements.

Military applications impose stricter requirements than civilian applications due to many reasons. They must maintain operational capability under cyber attacks, function in degraded or disconnected networks, comply with classified data-handling rules, and meet real-time performance guarantees for mission critical systems. Civilian applications, while still concerned with security and uptime, typically operate in more predictable environments with less strict latency, survivability and interoperability constraints. Central to this analysis is the research question: How does the implementation of security mechanisms affect the performance of services running in containers in Kubernetes, and what level of hardening is possible to achieve while still ensuring adequate performance?

To address this research question, benchmark tests were conducted to compare the performance of containers and traditional virtual machines under identical workloads. To assess security risks, known vulnerabilities with Common Vulnerabilities and Exposures (CVE) numbers given from the National Vulnerability Database (NVD) were exploited. Subsequently, mitigation techniques were implemented to measure the performance impact of applied security mechanisms, with the aim of identifying an acceptable level of hardening for the use of Kubernetes within military-grade infrastructure. Expert evaluations from both the civilian and military sectors were carried out to contextualize the conclusions drawn on security and performance, as well as to provide recommendations for the implementation of containerization technology from multiple perspectives.

The primary delimitation of this study concerns the scope and testing requirements. We chose to focus on a single container operating within a cluster. This approach was a strategic decision to enable an in-depth analysis of performance and security implications under controlled conditions. It also provides insight into the interactions between the container and other components of the cluster. By concentrating on a single container, we can more precisely isolate and examine the factors that directly affect security and performance, without the additional variables and complexities introduced by large-scale testing.

The reminder of the paper is organized as follows: Section 2 discusses related work, while Section 3 describes the methodology followed in the work. Section 4 describes the results of the research, and Sections 5 discusses the results. Section 6 summarizes the main findings and points to future work.

2 Related Work

Previous studies have compared benchmark performance and scalability between container-based virtualization and virtual machines [10]. Showing containers as a lightweight alternative that enables better performance and resource utilization [12]. However, they provide less security with weaker isolation [14]. Limited research has explored the trade-off between security and performance in the context of a defendable military information infrastructure (INI) [3]. Although the NATO doctrine [8] outlines ideal properties for INI, such as interoperability, flexibility, resilience, and security, there remains a gap addressing whether Kubernetes can fulfill these requirements, which is the topic of this research.

3 Method

This project used a comparative research approach[2] that combined literature studies, empirical evaluation, and expert evaluations. The goal was to investigate how Kubernetes security mechanisms affect performance in containerized environments. The literature study was conducted to enhance the knowledge level prior to the empirical evaluation and to identify relevant vulnerabilities to be exploited during the practical part of the research. Expert evaluations from military and civilian sectors contributed to contextualize the findings of this study and its conclusions. The research was divided into four phases.

Phase one consisted of comparing the baseline performance between a virtual machine (VirtualBox) and a container (Docker) hosted by a computer running Linux. The benchmark was performed in both containers and virtual machines under conditions as identical as possible. Performance was benchmarked using Sysbench[5] for CPU and RAM, and Flexible I/O Tester(Fio)[1] for disk I/O. For measuring CPU performance Sysbench makes the CPU calculate prime numbers. This represents a controlled, CPU-intensive and easily repeatable task that stresses the processor in a way that gives a clear measure of raw computational power. For measuring memory performance, Sysbench performs sequential memory read and write operations using a configurable block size and total data size. This generates a sustained, high-throughput workload that measures the available memory bandwidth and latency, while avoiding interference from disk I/O by operating entirely in RAM. For measuring disk I/O performance, Fio generates configurable read and write workloads directly on the storage device or file system. It supports both sequential and random access patterns, adjustable block sizes, queue depths, and I/O engines, allowing realistic simulation of different usage scenarios.

For phase two, the same tests were replicated in a Kubernetes cluster with one container to allow comparison between the bare-metal, virtual machine, standalone container and orchestrated container. The goal was to establish environments with the same amount of virtualization layers to create a foundation that was viable for comparison.

In phase three, the research focused on vulnerabilities with high CVSS scores (over 7) from the National Vulnerabilities Database (NVD). As a result of this, two documented Kubernetes vulnerabilities from NVD were reproduced in a controlled Kubernetes cluster. Specifically, CVE-2023-5044 and CVE-2024-21626 were reproduced to contribute to the evaluation of the Kubernetes security profile. Following this, in phase four, three security mechanisms from Kubernetes' best practices were incrementally implemented to measure their effectiveness and impact on performance. As a next step, the same performance tests as in phases 1 and 2 were performed to quantify the performance overhead caused by the security mechanisms.

Lastly, the findings were evaluated and discussed with professionals from both the civilian and military sectors to assess their practical use in military settings from a perspective of security and performance.

4 Results

This chapter describes the results from the empirical evaluation investigating the performance and security characteristics of containerized services in the Kubernetes environment.

4.1 Baseline performance

Sysbench measures CPU performance by running a script that calculates all prime numbers up to a given maximum value. Six different tests were performed on each of the systems where the variable 'cpuMaxPrime' varied between 10, 20 and 30 000, and it varied between being single- and double- threaded. Different parameters were used to provide insight on how the platforms may perform under diverse workloads. The command used is shown in the code listing 1.

```
sysbench cpu --cpu-max-prime="$variable-X" --threads="
    $variable-Y" --time=30 run
```

Code Listing 1: Sysbench command used for the CPU test.

Sysbench measures CPU performance by calculating all prime numbers up to a maximum value set with cpu-max-prime using 64-bit integers. The threads parameter decides if these calculations shall run in parallell or not, while the time parameter limits each test to 30 seconds.

Sysbench was also used to test read and write operations in memory for the different technologies.

```
sysbench memory --memory-block-size=128M --memory-total-size
    =1G --memory-oper=[read/write] --threads=1 run
```

Code Listing 2: Command used to test read/write operations in memory.

Memory-block-size sets the size for each memory block used during testing and was chosen to simulate large data processing typical of big data applications. The total memory size was set to 1GB with the memory-total-size parameter to limit the amount of data read or written per test. The memory-oper parameter defined the operation mode for testing and threads was set to 1 to measure raw memory performance without the benefits of virtualization.

Lastly, it was desirable to evaluate how effective the different systems could handle read and write operations to disk. In that context, the FIO tool was used because of its opportunity to simulate different workloads. The developed command is shown in the code listing 3.

```
fio --name=seq_"$modus"_test --ioengine=sync --rw="$modus" --
    bs=1M --direct=1 --iodepth=2 --size=2G --numjobs=1 --
    runtime=60 --filename=/filepath/filename --
    group_reporting
```

Code Listing 3: FIO command to test read and write operations to disk.

The name parameter was used to label the test results. The ioengine=sync setting ensures each I/O operation completes before the next begins. This was used to simulate applications that require immediate I/O feedback - such as real-time systems used in the military. The rw=mode parameter specifies whether the test performs read or write operations. The block size for each I/O operation was set to 1MB with the bs=1M parameter, while the file size was set to 2GB using the size parameter.

Each combination of these tests was performed 10 times each to be able to take the standard deviation into account. In table 1, two of the six different tests for the CPU is presented along with the read and write operations to memory and disk.

Test	Container	Kubernetes	VM	Host
CPU (Single-Thread)	$8852.3 \pm 5.43\%$	$8264.9 \pm 4.72\%$	$8738.1 \pm 4.82\%$	$8735.5 \pm 5.49\%$
CPU (Multi-Thread)	$14108.4 \pm 2.81\%$	$15870.4 \pm 0.17\%$	$14133.3 \pm 3.13\%$	$14561.0 \pm 3.47\%$
RAM (Read)	$11182.7 \pm 3.09\%$	$11174.6 \pm 4.07\%$	$9959.6 \pm 3.99\%$	$11105.9 \pm 4.16\%$
RAM (Write)	$11953.5 \pm 3.25\%$	$11186.5 \pm 11.50\%$	$11117.0 \pm 6.68\%$	$11946.0 \pm 5.04\%$
Disk (Read)	$421.2 \pm 1.30\%$	$447.0 \pm 0.95\%$	$288.4 \pm 6.34\%$	$448.3 \pm 1.68\%$
Disk (Write)	$304.7 \pm 9.40\%$	$300.1 \pm 6.05\%$	$302.7 \pm 6.16\%$	$314.5 \pm 6.29\%$

Table 1: Performance comparison between a container hosted with Docker, a container hosted in Kubernetes-distro MiniKube, a VM and the host machine. CPU tests ran with CPU max prime = 30 000 along with the results from the read and write operation tests of RAM and disk.

The units in table 1 represent the total amount of operations the system made within the specified time limit. For CPU it was the amount of prime number calculations made, for RAM and DISK it was the total amount of read or write operations made. Each benchmark was performed 10 times, the number shown in each cell is the average for each benchmark, and the standard deviation is shown.

4.2 Vulnerabilities

This part of the results describes two Kubernetes vulnerabilities that the research was able to exploit. They were selected due to their severity and current relevance, and the security mechanisms examined later demonstrate how such exploits could potentially be discovered or mitigated.

CVE-2023-5044. CVE-20230-5044 is a vulnerability (CVSS 7.6) in the ingress-NGINX controller, affecting versions prior to 1.9.0. If exploited, it allows authenticated users to inject OS commands via the "permanent-redirection" annotation and extract the controller's service account token, granting a user the same permission as the ingress controller. With default permissions, this token grants

read access to all secrets in the cluster, potentially compromising the entire environment.

The PoC is adapted from CovertSwarm[4] and r0binak[11]. For this execution a HTTP-echo service was deployed in a Kubernetes cluster consisting of two pods, Ingress-NGINX to route traffic and an embedded secret. The exploit relies on a user with the privilege of editing and deploying an ingress in the cluster. Code listing **??** shows the implemented yaml configuration to enable the ingress exploit.

```
nginx.ingress.kubernetes.io/permanent-redirect: https://www.
    google.com;}location ~* "^/exploit(/|$)(.*)" {
    content_by_lua 'ngx.say(io.popen("cat /var/run/secrets/
    kubernetes.io/serviceaccount/token"):read("*a"))';}
    location ~* "^/exploit(/|$)(.*)" { content_by_lua 'os.
    execute("touch /you")'
```

Code Listing 4: Configuration used to enable exploitation of ingress-NGINX. label

When deployed, this annotation injects malicious Lua code allowing "cat" to run at the OS-level, and retrieval of the ingress controller's service account token when requesting the specified URL as shown in figure 1.

```
ubuntu@ubuntu-vm:~$ curl localhost/exploit
eyJhbGciOiJSUzI1NiIsImtpZCI6ImVzdkdpRmVpdkNPSGhWcFpKNFJaTHg4U3JoWmEzWllZV1FSdDI1Y3dLNmsifQ.eyJhdWQiOlsiaHR0cHM6Ly9rdWJlcm5ldGVzLmRlZ
mF1bHQuc3ZjLmNsdXN0ZXIubG9jYWwiXSwiZXhwIjoxNzQ0MDU4NzQ3LCJpYXQiOjE3MTI1MjI3NDcsImlzcyI6Imh0dHBzOi8va3ViZXJuZXRlcy5kZWZhdWx0LnN2Yy5jb
HVzdGVyLmxvY2FsIiwia3ViZXJuZXRlcy5pbyI6eyJuYW1lc3BhY2UiOiJpbmdyZXNzLW5naW54IiwicG9kIjp7Im5hbWUiOiJpbmdyZXNzLW5naW54LWNvbnRyb2xsZXItN
TViYmQ3NGI1Zi12MjVxNiIsInVpZCI6ImQ0NDhiYjE4LWFmOTAtNGZjYi05NmM3LTFjOGM4NWEwZDM0OCJ9LCJzZXJ2aWNlYWNjb3VudCI6eyJuYW1lIjoiaW5ncmVzcy1uZ
2lueCIsInVpZCI6IjdkNzU5YTk2LWIwYTktNDYwZi04NmY2LTVjMDQzYzMyZmM4NiJ9LCJ3YXJuYWZ0ZXIiOjE3MTI1MjYzNTR9LCJuYmYiOjE3MTI1MjI3NDcsInN1YiI6I
nN5c3RlbTpzZXJ2aWNlYWNjb3VudDppbmdyZXNzLW5naW54OmluZ3Jlc3MtbmdpbngifQ.o7SeX1DapMkU7lHhxk4AMuLp9iuGgqdehhyim4WmW2O5eHmZ2HGfNeIKmFRG5_
rTFtSYEV07QHfJLEKUBYkoLf9Ne4pXVK0YJDK0qNoVnCF_p0fRPOGI5NUWXVxtLEbkzrtxdFm-CvUwC4-GLz9wi2dB1s-JSkCcec95Wq78KDQ9DD1WB_MNiMib6EBcpc6o8V
WQfemYCuXUvx_jKpolkCvfpbO61e25GvFrB98y0iTuA58VLbifqgsuiJGOs_u6ddrneSMpohYQWmV3WRt0q8Aw15A8AoDx9-24bRYbD8iqypxhREEFTsJJoe2VtnUPBID1OA
Ywj4XSyap5Uq3qVA
```

Fig. 1: Using the code injection to extract ingress-NGINX service account token.

Using this token, it was possible to retrieve all secrets in the Kubernetes cluster. This could then be base 64 decoded to plaintext, demonstrating the potential of a full cluster compromise.

This vulnerability was exploitable under default settings when using the latest Ingress-NGINX and NGINX versions at the time of execution, v1.10.0 and 1.25.3, but can be mitigated by enabling annotation validation (from v1.9.0). It is also preventable by applying the least-privilege principle through RBAC. General security measures such as access control, monitoring, network policies, and patching help minimize the risk of exposure.

CVE-2024-21626. CVE-2024-21626[9], is a vulnerability (CVSS 8.6) in the Runc lightweight container runtime. Runc is a command line tool that is used to start and run containers. It serves as a fundamental component in many

containerization technologies, including Docker, and enables containers to operate on Linux-based systems. The vulnerability affects Runc versions prior to 1.1.12 due to improper handling of file descriptors, which allowed newly created container processes to set their working directory on the host's file system rather than staying confined within the container's file system. The Vulnerability requires initial access to the system and privileges to create new container processes. Given the ability to escape the isolation layer that comes with containerization, this vulnerability constitutes a huge security risk, as it gives an adversary the opportunity to perform lateral movement on the host system.

To recreate the exploitation of this vulnerability, the PoC published on EthicalHacking, from James McGill[6] was used. The exploit can be divided into three main steps. Firstly, by using the bash script "verify.sh", an adversary can locate which file descriptor has access to read files on the host system. When located, the attacker can create a new container image where the working directory is set to the file descriptor that has the privileges to read files from the host system. Additionally, the container image also attempts to get access to /etc/passwd on the host system through a traversal from the current working directory. Lastly, the attacker can create the vulnerable container based on the created image, which runs the bash script "poc.sh", which allows you to create a reverse shell to the host machine as seen in figure 2.

```
tropisk@tropisk-VirtualBox:~/CVE-2024$ docker run -it --rm cve2024 bash /poc.sh
shell-init: error retrieving current directory: getcwd: cannot access parent di
rectories: No such file or directory
job-working-directory: error retrieving current directory: getcwd: cannot acces
s parent directories: No such file or directory
job-working-directory: error retrieving current directory: getcwd: cannot acces
s parent directories: No such file or directory
job-working-directory: error retrieving current directory: getcwd: cannot acces
s parent directories: No such file or directory
Listening on 0.0.0.0 1337
Connection received on tropisk-Lenovo-V330-14IKB 51952
root@tropisk-VirtualBox:/bin# ls
ls
[
aa-enabled
aa-exec
aconnect
acpi_listen
add-apt-repository
addpart
alsabat
alsaloop
```

Fig. 2: Establishing a connection and initiating a reverse shell that grants the container access to the host machine's root file system.

The vulnerability can be mitigated by updating Runc to version $\geq$ 1.1.12. Additionally, implementing robust security measures such as regular vulnerability

scanning of running containers and applying the principle of least privilege can help to protect against such vulnerabilities.

4.3 Post-hardening performance

This chapter presents how different security mechanisms affect the performance of containers orchestrated in Kubernetes. The implemented security mechanisms were Role-Based Access Control (RBAC), Pod Security Admission (PSA), and Auditing. Because it is possible to perform RBAC, PSA and Auditing in many different ways, it was necessary to focus on specific scenarios. For RBAC a role with specific permissions was created which was then bound to a job. When implementing PSA, a namespace was linked with baseline PSA, restricting auditing and notifications. An auditing policy that audited at the metadata level was established, and Grafana and Prometheus were used to visualize these results. The "Hardened" column in Table 2 shows the performance after all of the aforementioned security mechanisms were implemented. To evaluate the impact of these mechanisms, the same benchmark tests as described in section 4.1 were repeated using Sysbench for CPU and memory, and Fio for disk I/O.

The results, shown in Table 2, indicate that CPU performance was reduced by approximately 8–12% after the security mechanisms were applied. Memory performance showed a decrease in throughput of around 6–9%, while disk I/O experienced a smaller drop, typically between 4–7%. All tests were performed under conditions as identical as possible, and each value represents the average of 10 consecutive runs.

Test	Standard	Auditing	PSA	RBAC	Hardened
CPU (Single-Thread)	$7251.2 \pm 4.52\%$	$6916.2 \pm 5.28\%$	$6653.3 \pm 3.00\%$	$7023.4 \pm 4.09\%$	$6663.8 \pm 3.27\%$
CPU (Multi-Thread)	$13692.8 \pm 2.30\%$	$12650.3 \pm 1.69\%$	$12705.7 \pm 1.44\%$	$13468.4 \pm 1.44\%$	$12483.3 \pm 1.09\%$
RAM (Read)	$11124.6 \pm 2.32\%$	$11415.7 \pm 2.03\%$	$10854.6 \pm 3.15\%$	$10924.9 \pm 2.19\%$	$11285.0 \pm 1.71\%$
RAM (Write)	$10987.3 \pm 7.79\%$	$11506.5 \pm 8.24\%$	$11553.3 \pm 6.23\%$	$11863.1 \pm 3.28\%$	$11357.2 \pm 6.61\%$
Disc (Read)	$440.0 \pm 1.67\%$	$440.6 \pm 1.18\%$	$438.9 \pm 1.47\%$	$440.1 \pm 0.85\%$	$440.9 \pm 1.06\%$
Disc (Write)	$283.8 \pm 6.75\%$	$291.1 \pm 5.38\%$	$308.2 \pm 5.90\%$	$288.7 \pm 5.62\%$	$297.1 \pm 3.38\%$

Table 2: Performance comparison of a container hosted in Kubernetes-distro MiniKube with gradually adding security mechanisms. The standard refers to the default security settings that comes with MiniKube, and the hardened column refers to the benchmark results after adding all of the security mechanisms. The units shows the total amount of operations the system made within the specified time limit.

5 Discussion

The results of this study demonstrate the significant performance advantages that come with containers versus virtual machines (VMs), even when hardened with the best practice security measures provided by Kubernetes. This performance advantage is mainly architectural. VMs virtualize hardware through a

hypervisor and run a full guest Operating System (OS) per workload, whereas containers virtualize at the OS level and share the host kernel, isolating processes with namespaces and cgroups. By avoiding VM kernel emulation, containers eliminate duplicate OS overhead, pack more services per node, start in seconds rather than minutes - explaining why they outperform VMs even under security hardening.

Although the implementation of the security mechanisms RBAC, PSA and auditing presented some overhead, the performance level remained within acceptable ranges. In practice, these mechanisms affect different parts of the system. RBAC adds a per-request authorization check in the API server, which is lightweight and becomes noticeable first at high API call rates. The PSA configuration runs evaluations on object create/update and emits warnings - adding milliseconds of latency at the control plane. Auditing at the metadata level serializes API requests to log storage, increasing control-plane CPU and disk I/O, which affect the request volume and log verbosity. Because the study focused on a single container with few API interactions, the overhead was small. Whereas in larger, multi-node clusters with frequent deployments and controller activity, the same mechanisms could amplify the overhead shown in this study.

The CPU reduction of approximately 10% is noticeable, but the margin is still adequate for real-time operations with command and control. The 6-9% reduction in RAM is well under critical level, which allows for unseen processes during operations. When it comes to the 4–7% drop in disk I/O throughput, it would not affect auditing, sensor data, or replication to other critical systems. This indicates the possibility of gaining strong security without losing performance advantages. However, the study was limited to the scope of a single container environment within a Kubernetes cluster. Because of this, it does not replicate the complexity that comes with an operational real-world deployment, where factors such as orchestration of multiple nodes and dynamic scaling would introduce additional variables that could affect performance and security. In line with expert feedback, civilian operators report few performance issues in well-tuned Kubernetes clusters, whereas the military operator notes higher initial overhead during setup but potential efficiency gains as the number of services scales.

Another critical point is the operational complexity of securing Kubernetes. Although Kubernetes as a platform includes powerful security features, they are not enabled by default and require technical expertise to configure correctly. This could make the platform vulnerable to misconfigurations, which could possibly lead to security breaches such as discussed in this paper. When evaluated against NATO requirements for military CIS infrastructure which emphasize security, interoperability, scalability, and resilience, Kubernetes have both strengths and limitations. On one hand, its flexibility and scalability align well with the dynamic needs of modern military operations. However, its reliance on shared kernel architecture and the complex configuration landscape might not make it sufficiently reliable to meet the high assurance demands of a defendable critical infrastructure (CI). While Kubernetes may form the foundation for a defendable CI, it must be augmented with strict access control, continuous mon-

itoring, and automated security enforcement to meet NATO-grade assurance levels. Expert assessments as part of this research reinforced this - both sectors (civilian and military) consider Kubernetes as a mature technology but still hard to secure. The civilian organization mitigates risk through private clusters with trusted base images, systematic image scanning, least-privilege containers and Zero Trust controls tailored to IaaS/PaaS/SaaS. On the other hand, the military sector prioritizes physical and network isolation and proceeds cautiously, seeking guidance from national security authorities before adopting new platform features.

A major challenge identified in this study highlights the difficulty in defining a universally sufficient hardening baseline from a military perspective. With a threat landscape that is constantly evolving, the question of what is "secure enough" will be context-dependent, and therefore, has to be reassessed continuously. Feedback from domain experts reinforced this view. Although the potential of containers in military contexts is widely acknowledged, concerns remain about their maturity and resilience in environments that require high assurance. Crucially, competence and organizational maturity are limiting factors - civilian actors are further along a cloud transition by leveraging PaaS and IaaS, which require specialized skills. The military sector operates more driven by the supplier with limited use of containers, where skill gaps risk under-utilization of the technology. Both sectors accept some performance tradeoffs from security controls. The civilian actor typically scales resources to compensate, while the military sometimes works around heavy endpoint controls to preserve operational effect. Taking this into consideration, the most viable path for the military appears to be an expansion of containerization, lead by competence, in preferably controlled private/air-gapped clusters. In this way it would be possible to pair continuous training with policy enforcement and periodic reassessment of the hardening baseline.

6 Conclusion

The findings of this research show that containers outperform virtual machines in terms of resource utilization and scalability, particularly in CPU- and I/O intensive workloads. Even when hardened with security mechanisms such as RBAC, PSA and auditing, the Kubernetes orchestrated containers maintained their performance. From a military perspective, Kubernetes has several capabilities that align with NATO's vision for secure and scalable CIS infrastructure, including flexibility, automation, and high resource efficiency. However, the results also highlight the complexity of securing Kubernetes, and the limitations related to applying Kubernetes in an operational environment. The research was carried out in a controlled laboratory environment consisting of a single container and therefore does not address the added challenges associated with a fully operational distributed system. Lastly, the study underscores that it is not feasible to define a fixed hardening baseline that ensures sufficient security for military applications. The main reason for this is the recently discovered vulnerabilities

which remained exploitable until the most recent patches were applied. This is a huge challenge as it causes uncertainty regarding whether there are other similar undocumented vulnerabilities of concern, such as zero-day vulnerabilities.

7 Recommendations and Future Work

As a result of this research, the recommendation is to introduce container technology, and Kubernetes in particular, gradually into military environments through controlled pilot projects. These projects should prioritize minimal exposure to external networks and focus on establishing a secure and stable operating environment. The time horizon for such projects should extend up to five years, in order to capture long-term vulnerability trends and contribute to greater predictability within a constantly evolving threat landscape in the Cyber domain.

Future work should also investigate how Kubernetes-based systems perform and behave in a fully operational environment with multiple nodes, where scaling, traffic handling and various API functions are enabled. This would provide a deeper understanding of how the security mechanisms affect the overall performance of the system. In addition, future initiatives should explore the integration of machine learning (ML) to predict and dynamically adapt resource usage, enabling more efficient infrastructure utilization. The development of ML-driven tools capable of learning from known vulnerabilities could further allow real-time monitoring and threat detection. Such proactive solutions would enhance the ability to prevent attacks before they occur, strengthening the resilience of containerized military infrastructure.

Acknowledgments. We would like to express our thanks to the faculty and staff at the Norwegian Defence Cyber Academy, who provided access to necessary resources during the practical phases of this research.

We are also grateful to the individuals from both civilian and military sectors who contributed with their perspectives through expert interviews. Their insights provided important context and helped ground the findings in real-world applications.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Axboe, J.: fio - flexible i/o tester rev. 3.36. https://fio.readthedocs.io/en/latest/fio_doc.html (2024), accessed: 2024-02-09
2. Collier, D.: The comparative method. Political Science: The State of Discipline II, Ada W. Finifter, ed., American Political Science Association (1993)
3. Flydahl, E.: Kompendie for militær informasjonsinfrastruktur (ini) (2023)
4. Kleinke, M.: Exploiting cve-2023-5044 and cve-2023-5043 to overtake a kubernetes cluster. https://www.covertswarm.com/post/exploiting-cve-2023-5044-and-cve-2023-5043-to-overtake-a-kubernetes-cluster, accessed: 14.05.2024
5. Kopytov, A.: sysbench. https://github.com/akopytov/sysbench, accessed: 08.02.2024
6. McGill, J.: Cracking containers: Understanding cve-2024-21626 in runc. https://ethicalhacking.uk/cracking-containers-understanding-cve-2024-21626-in-runc/#gsc.tab=0, accessed: 14.05.2024
7. Nanda, S., Chiueh, T.C.: A survey on virtualization technologies. Tech. Rep. TR-179, Stony Brook University (2005), http://132.248.181.216/MV/CursoMaquinasVirtuales/Bibliograf%C3%ADaMaquinasVirtuales/VirtualizationSurveyTR179.pdf
8. NATO: Allied joint doctrine for communication and information systems. https://www.coemed.org/files/stanags/01_AJP/AJP-6_EDA_V1_E_2525.pdf (2017), accessed: 2024-02-10
9. NVD: Cve-2024-21626 detail. https://nvd.nist.gov/vuln/detail/CVE-2024-21626, accessed: 16.05.2024
10. Potdar, A.M., Narayan, D.G., Kengond, S.: Performance evaluation of docker container and virtual machine. Procedia Computer Science **171**, 1419–1428 (2020). https://doi.org/https://doi.org/10.1016/j.procs.2020.04.152, https://www.sciencedirect.com/science/article/pii/S1877050920311315, third International Conference on Computing and Network Communications (CoCoNet'19)
11. r0binak: Cve-2023-5044. https://github.com/r0binak/CVE-2023-5044, accessed: 14.05.2024
12. Seo, K.T., Hwang, H.S., Moon, I.Y., Kwon, O.Y., Kim, B.J.: Performance comparison analysis of linux container and virtual machine for building cloud. Advanced Science and Technology Letters **66**, 105–111 (2014)
13. Sigvartsen, J.: Spennende og kostnadseffektivt med virtualisering (del 1) (2005), https://www.tek.no/artikkel/i/MRqEer/spennende-og-kostnadseffektivt-med-virtualisering-del-1
14. Xavier, M.G., Neves, M.V., Rossi, F.D., Ferreto, T.C., Lange, T., De Rose, C.A.F.: Performance evaluation of container-based virtualization for high performance computing environments. In: 2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing. pp. 233–240 (2013). https://doi.org/10.1109/PDP.2013.41