Comprehensive Journal of Science
Volume (11), Issue (41), (JUNE, 2026)
ISSN: 3014-6266

مجلة العلوم الشاملة
المجلد (11) العدد 41 يونيو 2026
ردمد: 3014-6266

A study on the effectiveness of read-only root filesystems in mitigating persistent malware threats

Nuha Omran Abokhdair
Dept.Computer Science.
Faculty of Science
University of Zawiya
abo.khdeir@zu.edu.ly

Thuraya Rajab
Dept.Computer science.
Faculty of Science
University of Sabratha
thuraya@ta.sabu.edu.ly

تاريخ الاستلام:2026/4/04 –تاريخ المراجعة:2026/05/04– تاريخ القبول: 2026/05/16– تاريخ للنشر2026/06/05

Abstract

Persistent malware poses a serious challenge to operating systems due to its ability to remain active within a system over extended periods, even after reboot. This persistence enables sustained unauthorized access to system resources and data. In most cases, such behavior depends on the ability to introduce lasting modifications to system files—an inherent characteristic of traditional mutable systems. This highlights the need for approaches that address persistence at the system design level, rather than relying solely on detection after compromise.

This paper investigates the effectiveness of enforcing a read-only root filesystem as a means of limiting persistence. To this end, a controlled experimental setup was implemented using a container-based environment. Two configurations were examined: a standard mutable system and a second configuration designed to emulate immutability by enforcing read-only access to the root filesystem. A simple persistence-oriented scenario was used, in which attempts were made to write files into system directories.

The observations were consistent across all trials. In the mutable configuration, write operations targeting system directories such as /etc were successful, allowing modifications to remain after execution. In contrast, the read-only configuration prevented all such attempts, effectively blocking the introduction of persistent changes.

These results suggest that restricting write access at the filesystem level can significantly limit the ability of malware to establish persistence. While this approach does not eliminate all possible attack vectors, it directly constrains one of the primary mechanisms used to maintain long-term presence within a system.

Keywords: Persistent Malware; Read-only Root Filesystem; Immutable Operating Systems; File System Security; Docker Containers

1. Introduction

With the rapid pace of digital transformation, and the growing reliance of organizations on information systems across various domains, operating system security has become a fundamental requirement for ensuring service continuity and data protection [1]As the core layer upon which applications and services depend, operating systems represent a primary target for cyberattacks, particularly those aiming to maintain long-term control over compromised systems [2]

Among these threats, persistent malware stands out as one of the most critical components of Advanced Persistent Threats (APTs), as it is specifically designed to establish a foothold within the system and remain active even after reboot, thereby enabling sustained access to system resources and sensitive data[2]. Such behavior largely depends on the system's ability to allow permanent modifications to its files—an inherent characteristic of traditional mutable operating systems [1]

Despite the availability of various security mechanisms, including antivirus solutions and intrusion detection systems, most existing approaches focus on detecting attacks after they occur, rather than preventing the underlying conditions that allow malware to achieve persistence in the first place [3].

This limitation highlights a research gap in the need for security models that go beyond detection, and instead aim to reduce the likelihood of malware establishing persistence at the system level.

In this context, the concept of immutable operating systems has emerged as a promising approach. These systems rely on enforcing a read-only root filesystem, thereby preventing any permanent modifications to system files during runtime. This design reduces the system's attack surface, and limits the ability of malicious software to maintain persistence [4].

This paper presents an experimental study to evaluate the effectiveness of a read-only root filesystem in mitigating persistent malware threats. To achieve this, a container-based test environment using Docker was developed to simulate two configurations: a conventional mutable system, and a second configuration designed to emulate immutability. The study examines the behavior of both environments under controlled conditions, by simulating attempts to write malicious files into core system directories, and assessing their ability to persist.

The main contribution of this work lies in providing a direct, practical evaluation of how filesystem design influences system security, particularly in relation to limiting persistence mechanisms in modern operating environments.

2. Literature Review

A growing body of recent research has examined the role of operating system design in strengthening resilience against diverse forms of malicious software, particularly in environments that require continuous and reliable operation. In the study by Guihard (2025) [5], an experimental evaluation was conducted to compare mutable systems, immutable systems, and container-based environments. The findings indicate that immutable systems tend to provide higher levels of stability and reliability, especially under failure conditions such as system crashes and power interruptions. This improvement is largely attributed to

characteristics such as the use of a read-only root filesystem and isolation mechanisms. However, the study primarily focused on operational and performance aspects and did not directly address the impact of such designs on malware persistence within the system.

Similarly, Böhm and Wirtz (2023) [6] presented a conceptual analysis of immutable operating systems, highlighting their reliance on a read-only root filesystem as a means of limiting exploitable points within the system and preventing unauthorized modifications to core components. While this work provides a clear theoretical foundation, it remains limited to a conceptual level and does not include experimental validation under realistic attack scenarios.

From a different perspective, Caporaso et al. (2024) [7] proposed a practical filesystem-level approach through the VaultFS system, which is based on a “write-once” principle. This design prevents data modification even under elevated privilege conditions, offering strong protection against ransomware and data tampering. Nevertheless, the focus of this work is primarily on protecting stored (cold) data, and it does not directly examine the behavior of traditional mutable systems in scenarios that simulate attempts to establish persistence within system environments.

Despite the contributions of these studies, there remains a lack of work that combines a direct experimental evaluation of read-only root filesystem behavior with an analysis of malware persistence mechanisms. Existing research has largely addressed performance, system architecture, or data protection in isolation, without establishing a practical link between these aspects in the context of controlled attack scenarios targeting persistence.

Accordingly, a clear research gap emerges in the absence of experimental studies that explicitly evaluate the effectiveness of read-only root filesystem configurations in limiting malware persistence. This study seeks to address this gap by designing a controlled experimental environment and analyzing system behavior under simulated persistence-oriented attack **scenarios.**

3. Methodology

This study adopts a comparative experimental approach to evaluate the effectiveness of immutable operating system configurations in mitigating persistent malware threats. The experimental setup is based on the comparison between two distinct environments: a conventional writable Linux environment and a second configuration designed to exhibit immutable behavior, implemented within a Docker container with a read-only root filesystem enabled. The framework was specifically designed to assess a single security property—namely, whether restricting write access at the root filesystem level can prevent attempts to modify system files. The overall structure of the proposed experimental framework is illustrated in **Figure (1)**.

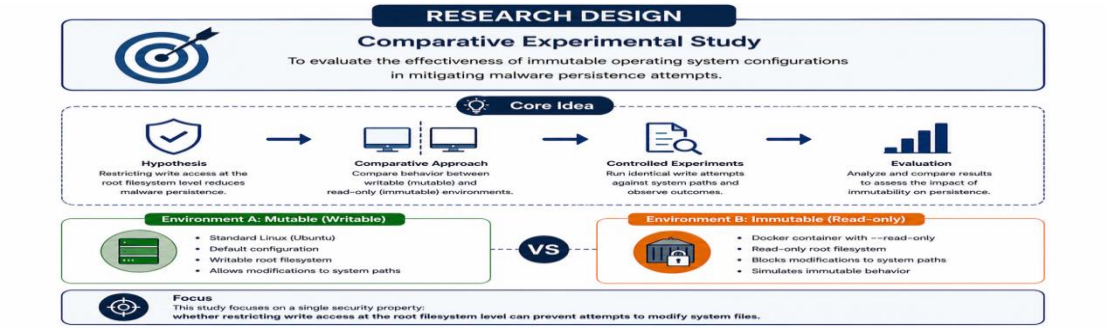

Figure (1): Overall Research Design and Comparative Experimental Framework

- **Experimental Design**

Two environments were prepared for execution, as illustrated in **Figure (2)**. The first represents a standard Linux system with default write permissions enabled. The second serves as the experimental configuration, designed to emulate immutability by running a container with the read-only option enabled, thereby enforcing a non-writable root filesystem [8]. This setup allows for the direct observation of the effect of immutability on write operations targeting system directories, through a controlled comparison between mutable and non-mutable environments, while monitoring whether executed commands leave persistent changes within the system [6].

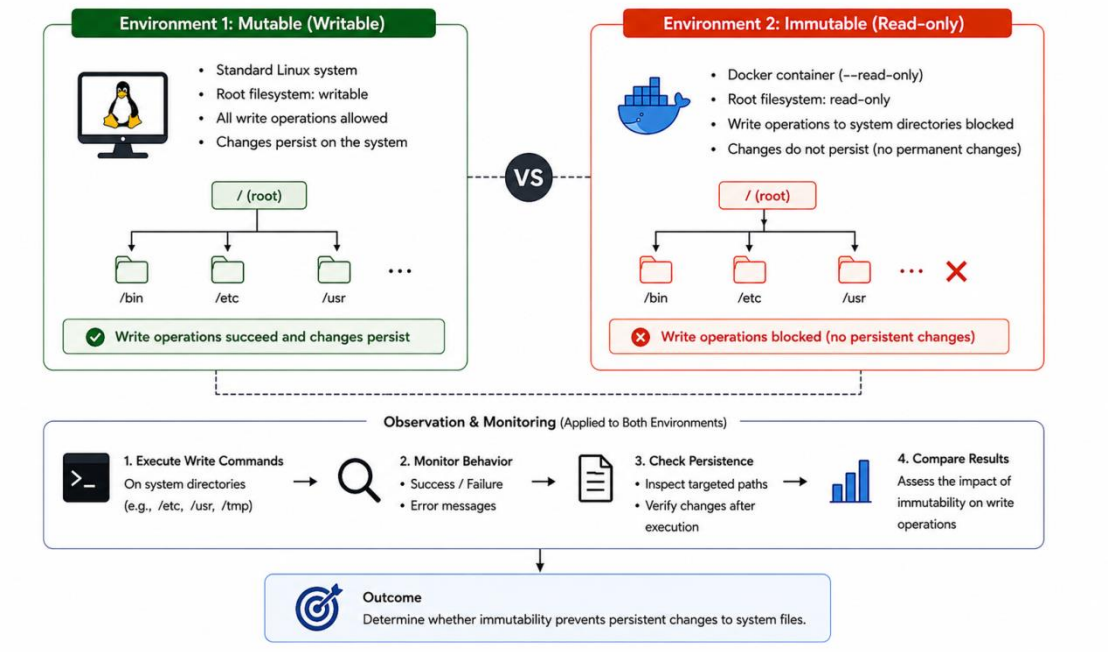

Figure (2): Comparative Experimental Setup Between Mutable and Immutable Environments

- **Tools Used**

The study was conducted using a set of software tools and technologies that support controlled and reproducible experimentation:

- **Container Platform:** Docker was used to create and run isolated environments. Two configurations were tested: a mutable system and a configuration that simulates immutability through a read-only root filesystem [8]. It is worth noting that certain container configurations may influence the level of isolation and potentially increase exposure to attacks if not properly managed [4].
- **Operating System:** A Linux distribution (Ubuntu) was used within the containers to execute all experimental procedures, due to its widespread use in server environments and security testing.
- **Development Environment:** Visual Studio Code was used as the primary interface for interacting with the system, including terminal access and file management.
- **Command-Line Interface:** All operations were executed using Bash, relying on standard Linux commands such as `echo` and `cp` to simulate file creation and persistence attempts at the filesystem level, which reflects common behaviors observed in malware operations [2].
- **Host System:** The experiments were conducted on a Windows host using Docker Desktop, which leverages WSL2 to run Linux containers, providing a compatible environment for isolation-based testing [8].
- **Execution Procedure**

  The process began with setting up a dedicated workspace containing the theoretical framework, test scripts, and output data. A baseline test was first conducted within a writable environment to verify that write operations to system paths succeed under normal conditions, reflecting the expected behavior of mutable systems [5].

  The same test was then repeated within a Docker container configured with the `read-only` option to enforce root-level immutability [8], thereby simulating an immutable system configuration [6].

  The test scenarios focused on write operations targeting core system directories such as `/etc`, given their role in storing configuration files that directly affect system behavior and service execution. These directories are commonly targeted by persistence mechanisms in malware [2].

  Additionally, the use of the `--rm` option ensures that each container instance is ephemeral, meaning it is automatically removed upon termination and replaced with a new instance upon re-execution. Therefore, “restarting” in this context refers to launching a new container using `docker run`, rather than restarting an existing instance. All experiments were conducted in clean and isolated environments to ensure consistency and reliability of results, which is a key requirement in experimental research [5].
- **Simulation of Persistent Threats**

Rather than executing full malware samples, the study employed controlled write attempts to represent the core behavior of persistence. These attempts involved creating or modifying files within system directories commonly used by persistence mechanisms. Such an approach ensures safety, reproducibility, and direct alignment with the research objective. Consequently, the success or failure of these write operations serves as a practical indicator of the system's ability to resist persistence-oriented modifications, consistent with studies linking system mutability to security exposure [2], [9].

- **Data Collection and Analysis**

Experimental results were recorded manually and, when needed, organized into CSV format for comparison between the two environments. The primary metric considered was the success or failure of write operations in each configuration. Additional observations included file persistence across sessions, system state changes after container relaunch, and the impact of runtime options such as `--rm`. Given the focused and controlled nature of the experiment, the analysis remains descriptive and comparative rather than statistically intensive [5].

- **Privilege Management**

Installation and configuration steps on the host system were performed with elevated privileges when necessary (e.g., using `sudo` for Docker setup). Within the container environment, commands were typically executed with default root privileges unless otherwise specified. This separation between host-level and container-level permissions helps demonstrate that write restrictions were not merely the result of insufficient privileges, but rather a direct consequence of enforcing a read-only root filesystem [8], highlighting the role of filesystem-level control as a fundamental security mechanism [6].

- **Methodology Limitations**

This methodology focuses on a specific security property, namely the ability of a read-only root filesystem to prevent persistence at the file level [6]. It does not cover all forms of malware, such as memory-resident attacks, those relying on temporary writable paths, or those using external command-and-control channels.

Furthermore, this approach does not replace other security mechanisms such as secure boot, kernel hardening, permission control, or software supply chain protection, which have been addressed in other studies within the broader context of system security[10], [11] . Nevertheless, the results provide clear experimental evidence that enforcing immutability at the filesystem level can significantly reduce the likelihood of persistent threats maintaining a foothold within the system [9].

5.1 Results

The experiment consisted of 20 test attempts distributed equally between a writable (mutable) environment and a non-modifiable (immutable) environment. The results showed that write operations targeting temporary paths under `/tmp` were successful in both environments. However, the created files did not persist after restart, which is consistent with the temporary nature of these directories.

In contrast, when write attempts targeted sensitive system paths under `/etc`, all operations succeeded in the mutable environment, whereas all corresponding attempts failed in the immutable configuration, as summarized in **Tables 1(a) and 1(b)**

Table 1(a): Experimental Results for Write Attempts on Temporary Paths (/tmp)

No.	Environment	Test Type	Command	Actual Result	After Restart	Observation
1	Mutable	Write to `/tmp`	Echo test> /tmp/malwar1.txt	Success ✔	Not persistent ✕	Temporary storage
2	Immutable	Write to `/tmp`	Echo test> /tmp/malwar1.txt	Success ✔	Not persistent ✕	Temporary only
3	Mutable	Write to `/tmp`	Echo test> /tmp/malwar2.txt	Success ✔	Not persistent ✕	Temporary storage
4	Immutable	Write to `/tmp`	Echo test> /tmp/malwar2.txt	Success ✔	Not persistent ✕	Temporary only

5	Mutable	Write to `/tmp`	Echo test> /tmp/malwar3.txt	Success ✔	Not persistent ✕	Temporary storage
6	Immutable	Write to `/tmp`	Echo test> /tmp/malwar3.txt	Success ✔	Not persistent ✕	Temporary only
7	Mutable	Write to `/tmp`	Echo test> /tmp/malwar4.txt	Success ✔	Not persistent ✕	Temporary storage
8	Immutable	Write to `/tmp`	Echo test> /tmp/malwar4.txt	Success ✔	Not persistent ✕	Temporary only
9	Mutable	Write to `/tmp`	Echo test> /tmp/malwar5.txt	Success ✔	Not persistent ✕	Temporary storage
10	Immutable	Write to `/tmp`	Echo test> /tmp/malwar5.txt	Success ✔	Not persistent ✕	Temporary only

Table 1(b): Experimental Results for Write Attempts on System Directories (/etc)

No.	Environment	Test Type	Command	Actual Result	After Restart	Observation
1	Mutable	Write to `/etc`	Echo test> /etc/malwar1.txt	Success ✔	Persistent ✔	Security risk
2	Immutable	Write to `/etc`	Echo test> /etc/malwar1.txt	Failed ✕	Not present	Write protection
3	Mutable	Write to `/etc`	Echo test> /etc/malwar2.txt	Success ✔	Persistent ✔	Security risk
4	Immutable	Write to `/etc`	Echo test> /etc/malwar2.txt	Failed ✕	Not present	Write protection
5	Mutable	Write to `/etc`	Echo test> /etc/malwar3.txt	Success ✔	Persistent ✔	Security risk

6	Immutable	Write to `/etc`	Echo test> /etc/malwar3.txt	Failed ✗	Not present	Write protection
7	Mutable	Write to `/etc`	Echo test> /etc/malwar4.txt	Success ✔	Persistent ✔	Security risk
8	Immutable	Write to `/etc`	Echo test> /etc/malwar4.txt	Failed ✗	Not present	Write protection
9	Mutable	Write to `/etc`	Echo test> /etc/malwar5.txt	Success ✔	Persistent ✔	Security risk
10	Immutable	Write to `/etc`	Echo test> /etc/malwar5.txt	Failed ✗	Not present	Write protection

Quantitative Summary

- Total number of trials: 20
- Write attempts targeting /tmp: 10
- Write attempts targeting /etc: 10
- Successful write operations to /tmp in both environments: 10/10
- Successful write operations to /etc in the mutable environment: 5/5
- Failed write operations to /etc in the immutable environment: 5/5

As illustrated in **Figure (3)**, the experimental results reveal a clear contrast between the two environments. All write attempts targeting system directories succeeded in the mutable configuration, whereas the read-only configuration completely blocked write operations directed at sensitive system paths. This distinction further highlights the effectiveness of enforcing a read-only root filesystem in limiting persistence-oriented modifications within the system.

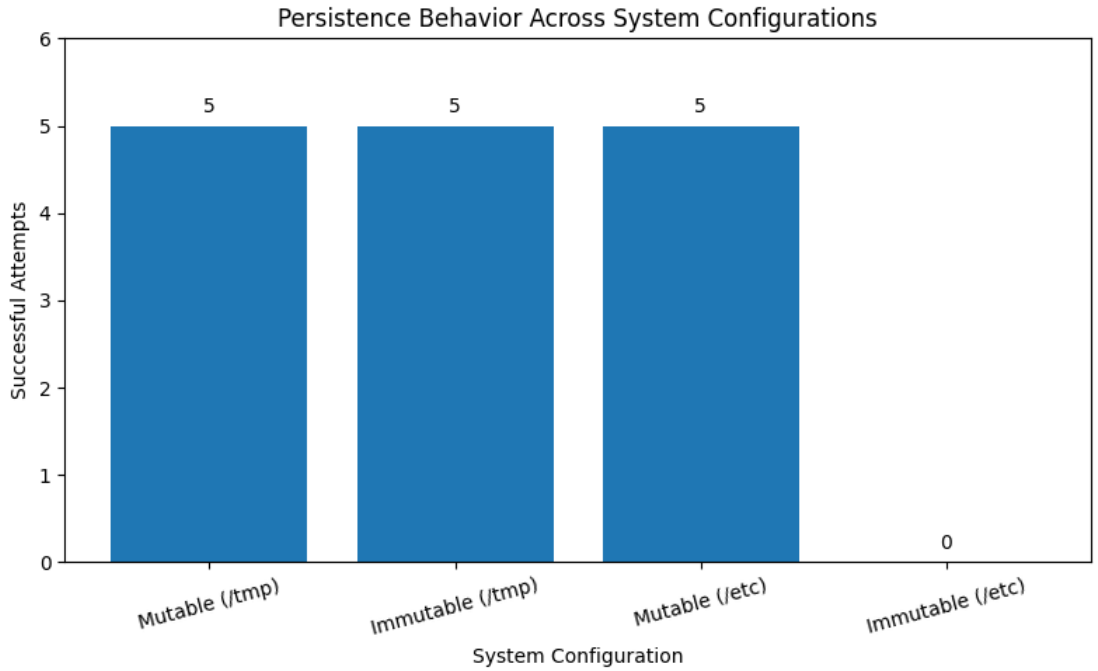

Figure (3): Persistence Behavior Across Mutable and Immutable System Configurations

The results indicate that the immutable environment did not prevent write operations targeting temporary paths such as /tmp, which is expected given that such locations do not typically represent persistence points within the system. In contrast, the same environment completely blocked write attempts directed at sensitive system paths under /etc, demonstrating that enforcing a **read-only root filesystem** effectively disrupts one of the primary file-based persistence mechanisms commonly used by malicious software.

Furthermore, the successful write operations to /etc in the mutable environment, compared with their complete failure in the immutable configuration, represent a significant distinction that directly supports the research hypothesis. These findings suggest that immutability does not eliminate all forms of write activity within the system; rather, it substantially increases the difficulty of establishing persistence through modifications to critical system files [6].

5.2 Discussion

The findings of this study demonstrate that filesystem behavior plays a critical role in determining a system's ability to resist persistence-oriented malware activity. Within the mutable environment, the experiments confirmed that files could be created and modified inside core system directories such as `/etc`, reflecting the ability of such systems to accept persistent modifications that may later be exploited to establish long-term presence within the system. This observation is consistent with existing literature, where many persistence mechanisms rely on altering configuration files or executable system components [2].

In contrast, the environment configured with a read-only root filesystem blocked all write attempts targeting core system directories, even when executed with root privileges. This indicates that restricting write access at the filesystem level represents an effective mechanism for preventing unauthorized modifications and limiting the ability of malware to establish

persistence within the system [6]. These observations are also aligned with recent approaches that employ immutable filesystem designs to prevent modifications even under elevated privilege conditions [7].

The results further suggest that the key distinction between the two configurations lies not merely in the execution of the attack itself, but rather in the ability to maintain persistence after execution. While the mutable environment permits permanent modifications to system components, the immutable configuration fundamentally prevents such changes from occurring [6], [9].

It should also be noted that this study relies on a container-based simulation, in which immutable behavior was approximated through the enforcement of read-only mode within Docker containers [8]. Consequently, the results represent a simplified model when compared with fully immutable operating systems. In addition, the absence of persistent storage mechanisms, such as Docker volumes, limits the evaluation of persistence within long-term containerized environments.

Furthermore, insecure container configurations—such as the use of `privileged` mode or exposing the `docker socket`—may reduce isolation boundaries and increase the system's exposure to attacks. This suggests that isolation alone is insufficient to achieve comprehensive security without additional filesystem-level restrictions [4].

6 Conclusion and Future Work

6.1 Conclusion

This study demonstrates that filesystem design constitutes a fundamental factor in strengthening operating system security, particularly in relation to limiting malware persistence. The experimental results showed that mutable systems allow modifications to be introduced into core system directories, thereby enabling the establishment of persistent malicious changes. In contrast, enforcing a read-only root filesystem effectively prevented such modifications, significantly limiting the ability of attacks to maintain persistence within the system [6].

The findings further indicate that adopting read-only filesystem configurations represents an effective approach for mitigating persistence-oriented attacks that rely on permanent modifications to system files. This observation is consistent with previous studies that associate reduced system mutability with improved protection against malicious activity[6], [9] . Nevertheless, this approach should not be viewed as a standalone security solution, as achieving stronger protection in real-world environments still requires integration with additional defensive mechanisms.

6.2 Future Work

This work can be extended by evaluating fully immutable operating systems that rely on more advanced implementation techniques, as well as by applying more sophisticated attack scenarios involving system configuration manipulation or service exploitation. Future studies may also investigate the impact of persistent storage mechanisms, such as Docker volumes, on

malware persistence, given their potential to bypass certain immutability constraints within containerized environments [3].

In addition, further research may explore the integration of this approach with complementary protection mechanisms, including kernel hardening and process isolation, in order to enhance the overall security posture of modern operating environments [10].

References

[1] Z. Zhang, Y. Cheng, S. Nepal, D. Liu, Q. Shen, and F. Rabhi, "KASR: A Reliable and Practical Approach to Attack Surface Reduction of Commodity OS Kernels," in Research in Attacks, Intrusions, and Defenses, M. Bailey, T. Holz, M. Stamatogiannakis, and S. Ioannidis, Eds., Cham: Springer International Publishing, 2018, pp. 691–710. doi: 10.1007/978-3-030-00470-5_32.

[2] B. A. S. Al-Rimy et al., "A Pseudo Feedback-Based Annotated TF-IDF Technique for Dynamic Crypto-Ransomware Pre-Encryption Boundary Delineation and Features Extraction," IEEE Access, vol. 8, pp. 140586–140598, 2020, doi: 10.1109/ACCESS.2020.3012674.

[3] E. Reshetova, J. Karhunen, T. Nyman, and N. Asokan, "Security of OS-Level Virtualization Technologies," in Secure IT Systems, K. Bernsmed and S. Fischer-Hübner, Eds., Cham: Springer International Publishing, 2014, pp. 77–93. doi: 10.1007/978-3-319-11599-3_5.

[4] Y. S. Putta, "Enhancing Docker Container Security," masters, Dublin, National College of Ireland, 2024. Accessed: Jun. 12, 2026. [Online]. Available: https://norma.ncirl.ie/7145/

[5] Guihard, F., "Impact of operating systems on edge device: Benchmarking performance, reliability, and post-quantum readiness," Master's thesis, University of Turku, Turku, Finland, 2025.

[6] Böhm, Sebastian and Wirtz, Guido, "Immutable Operating Systems: A Survey," Proceedings of the 15th ZEUS Workshop (ZEUS 2023), CEUR-WS.org, 2023.

[7] P. Caporaso, G. Bianchi, and F. Quaglia, "VaultFS: Write-once Software Support at the File System Level Against Ransomware Attacks," Oct. 29, 2024, arXiv: arXiv:2410.21979. doi: 10.48550/arXiv.2410.21979.

[8] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and Linux containers," in 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Mar. 2015, pp. 171–172. doi: 10.1109/ISPASS.2015.7095802.

[9] Y. Zhang, A. Rajimwale, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "End-to-end Data Integrity for File Systems: A ZFS Case Study".

[10] M. Momeu et al., "ISLAB: Immutable Memory Management Metadata for Commodity Operating System Kernels," in Proceedings of the 19th ACM Asia Conference on Computer and Communications Security, Singapore Singapore: ACM, Jul. 2024, pp. 1159–1172. doi: 10.1145/3634737.3644994.

[11] S. Sultan, I. Ahmad, and T. Dimitriou, "Container Security: Issues, Challenges, and the Road Ahead," IEEE Access, vol. 7, pp. 52976–52996, 2019, doi: 10.1109/ACCESS.2019.2911732.

]12[The Role of Artificial Intelligence Technologies in Addressing Individual Differences among Basic Education Students in Libya. (2026). *Al-Farooq Journal of Sciences*, *2*(3), 536-544. https://doi.org/10.65405/2gb8v408