Advanced Security Strategies for Containerized Workloads

Salma Zahir

Department of Computer Science, University of Tunisia

Abstract

Containerization has fundamentally transformed the way software is developed, deployed, and managed. By encapsulating applications and their dependencies into isolated environments, containers offer portability, efficiency, and scalability. However, these advantages come with unique security challenges that traditional security measures are not equipped to address. As container adoption continues to grow, so does the need for advanced security strategies that cater specifically to containerized workloads. This paper delves into these strategies, exploring the comprehensive security measures required throughout the container lifecycle, including image security, orchestration platform hardening, runtime security, and access control. By implementing these strategies, organizations can effectively safeguard their containerized environments against evolving threats, ensuring the integrity, confidentiality, and availability of their applications.

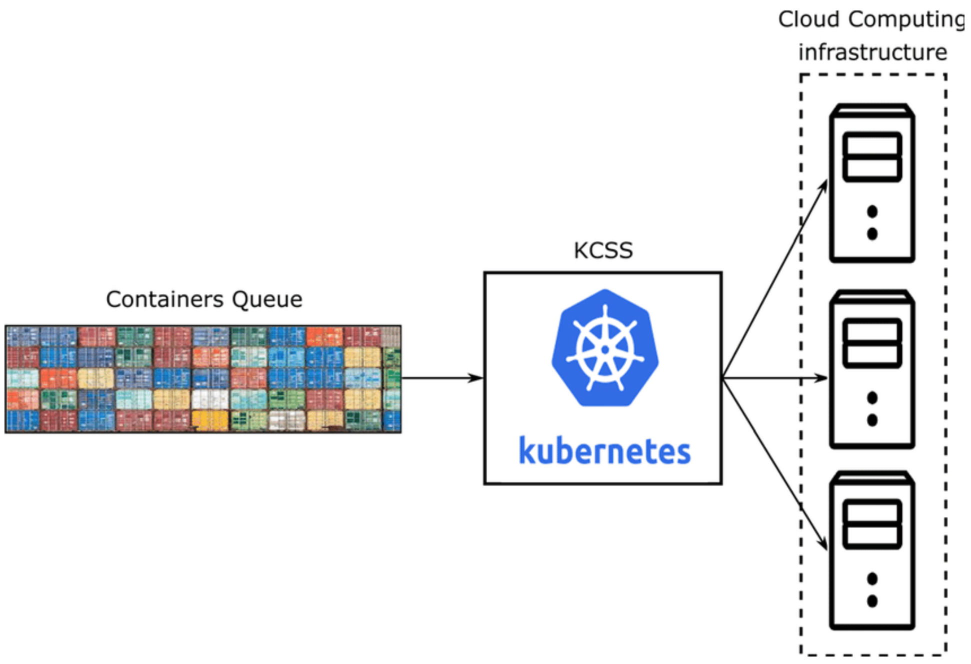
Keywords: *Container Security, Docker, Kubernetes, Container Orchestration, Runtime Security, Image Scanning, Microservices Security*

Introduction

Overview of Containerization

Containerization has emerged as a pivotal technology in modern software development, offering a solution to the longstanding challenges of environment consistency, dependency management, and resource efficiency. Containers allow developers to package an application along with its dependencies, libraries, and configurations into a single, lightweight unit that can be deployed consistently across different environments. This contrasts with traditional virtualization, where entire operating systems are virtualized, leading to significant overhead. [1]

The rise of containers can be attributed to their ability to solve the "it works on my machine" problem, providing a consistent runtime environment regardless of where the container is deployed. This consistency has been a game-changer in the world of DevOps and continuous integration/continuous deployment (CI/CD) pipelines, where rapid development cycles demand reliability and efficiency. Tools like Docker have made containerization accessible, while orchestration platforms like Kubernetes have enabled the management of large-scale containerized applications across distributed environments. [2]



Containers are particularly well-suited for microservices architectures, where applications are broken down into smaller, independent services that communicate over a network. This architectural style aligns with the benefits of containers, allowing each microservice to be developed, deployed, and scaled independently. However, the widespread adoption of containers has also introduced new security challenges that traditional security models struggle to address.

Importance of Security in Containerized Environments

As the use of containers in production environments has proliferated, security has become a paramount concern. Containers offer several inherent security benefits, such as process isolation and the ability to create immutable infrastructure, but they also introduce significant risks. The shared kernel model, which allows containers to be lightweight and fast, also means that a vulnerability in the kernel could potentially compromise all containers running on a host. This makes container environments an attractive target for attackers. [3]

Moreover, the dynamic and ephemeral nature of containers complicates security efforts. Containers are often created and destroyed in seconds, making it challenging to apply traditional security monitoring and management tools. Additionally, the widespread use of public container registries, where images can be freely shared and downloaded, introduces the risk of deploying malicious or vulnerable images. Without rigorous security practices, organizations can inadvertently expose themselves to threats such as data breaches, denial of service attacks, and privilege escalations. [4]

The complexity of container orchestration platforms like Kubernetes further compounds these challenges. Kubernetes, while powerful, has a steep learning curve and requires careful configuration to avoid security pitfalls. Misconfigurations, such as exposing the Kubernetes API server to the internet or running containers with excessive privileges, can lead to severe security breaches. [5]

Given these risks, it is clear that traditional security approaches are insufficient for protecting containerized workloads. A more nuanced, layered security strategy is required—one that considers the unique characteristics of containers and the specific threats they face. This paper aims to explore such strategies, providing a comprehensive guide to securing containerized environments.

Objective of the Paper

The primary objective of this paper is to provide a detailed examination of advanced security strategies for containerized workloads. As containers become increasingly integral to modern software infrastructure, it is essential to develop security practices that are specifically designed to address the challenges they present. This paper will cover a range of security measures, from securing the container lifecycle and hardening orchestration platforms to implementing robust runtime security and access control mechanisms.

By exploring these strategies in depth, the paper seeks to equip security professionals, DevOps engineers, and IT administrators with the knowledge and tools they need to protect containerized environments effectively. The goal is not only to mitigate risks but also to establish a security framework that can adapt to the evolving threat landscape. With the right strategies in place, organizations can leverage the benefits of containerization while maintaining a strong security posture. [6]

Securing the Container Lifecycle

Image Security

The foundation of container security lies in the integrity and security of the container image. A container image is a blueprint for creating containers, consisting of an application and its dependencies. Given the widespread use of container images in development and production environments, ensuring their security is crucial to prevent the introduction of vulnerabilities and malicious code into the containerized infrastructure.

1. Image Source Verification:

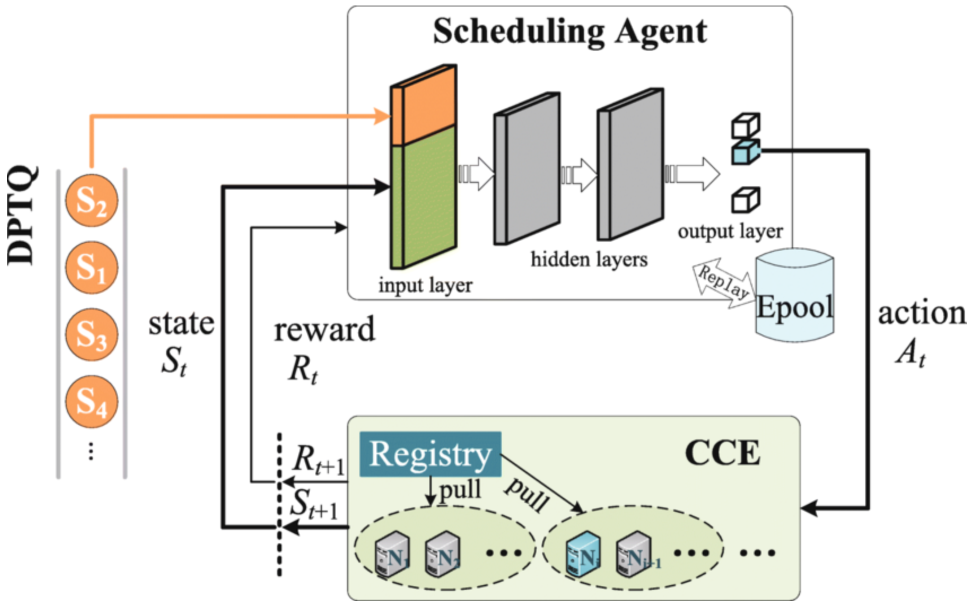
One of the first steps in securing container images is to verify their source. Containers are often built from base images pulled from public or private registries. While public registries like Docker Hub provide a vast repository of images, not all of these images are created equal in terms of security. Some may contain outdated software with known vulnerabilities, while others may have been compromised with malicious code. [7]

To mitigate these risks, organizations should adopt a policy of only using images from trusted sources. Official images, provided by software vendors or trusted communities, are generally more secure as they are maintained and regularly updated. Additionally, organizations can implement cryptographic signing of images, where the integrity of an image is verified through a digital signature. This process ensures that the image has not been tampered with and that it originates from a trusted source. [7]

2. Vulnerability Scanning:

Vulnerability scanning is a critical aspect of container image security. Even images from trusted sources may contain vulnerabilities if they include outdated libraries or dependencies. Tools like Clair, Trivy, and Aqua Security specialize in scanning container images for known vulnerabilities by comparing the image's contents against a database of known issues. [8]

Organizations should integrate vulnerability scanning into their CI/CD pipelines to ensure that images are scanned automatically as part of the build process. This allows vulnerabilities to be identified and remediated before the image is deployed to production. Regular scanning of images stored in container registries is also necessary to detect newly discovered vulnerabilities that may affect previously built images.



3. Minimalist Images:

Another effective strategy for securing container images is to use minimalist images. A minimalist image contains only the essential components required for the application to function, reducing the attack surface by excluding unnecessary tools, libraries, and utilities. By minimizing the contents of an image, organizations can significantly reduce the likelihood of vulnerabilities and exploits.

Alpine Linux, for example, is a popular base image for minimalist containers due to its small size and reduced set of packages. However, even with minimalist images, it is important to ensure that all included components are up-to-date and free from vulnerabilities. Organizations should also consider custom-building their own base images tailored to their specific security requirements. [1]

4. Image Hardening:

Image hardening involves applying security best practices to container images to reduce their susceptibility to attacks. This can include removing unnecessary components, configuring secure defaults, and ensuring that sensitive information such as passwords or API keys are not embedded within the image. Additionally, organizations should implement immutability in their image builds, where once an image is created, it is not modified. This prevents changes that could introduce vulnerabilities or inconsistencies across environments. [9]

Another aspect of image hardening is to ensure that the images run as non-root users by default. Containers running as root have elevated privileges that can be exploited by attackers to gain control of the host system. By configuring containers to run as non-root, the impact of a potential compromise is significantly reduced. [10]

Securing Image Distribution and Storage

The security of container images does not end with their creation. Ensuring the secure distribution and storage of images is equally important to prevent unauthorized access, tampering, or leakage of sensitive information. [11]

1. Private Registries:

Private container registries provide a controlled environment for storing and managing container images. Unlike public registries, where images are accessible to anyone, private registries allow organizations to enforce access controls and security policies tailored to their needs. Private registries also offer features such as image signing, vulnerability scanning, and audit logging, which enhance the overall security of the container lifecycle. [5]

Organizations should consider setting up private registries for their containerized workloads, especially for sensitive or proprietary applications. Tools like Harbor, Nexus Repository, and Azure Container Registry provide robust solutions for managing private container registries with advanced security features. [12]

2. Secure Communication:

All communication between the container registry and clients (such as Docker CLI or Kubernetes) should be encrypted using Transport Layer Security (TLS/SSL). This prevents man-in-the-middle attacks, where an attacker could intercept and alter the communication between the client and the registry. Secure communication also ensures that the integrity of the images is maintained during transfer, preventing unauthorized modifications. [3]

Organizations should enforce the use of TLS/SSL for all connections to the container registry and ensure that certificates are managed securely. Self-signed certificates should be avoided in production environments, as they can be easily spoofed, compromising the security of the communication channel. [4]

3. Access Control:

Access control mechanisms are crucial for preventing unauthorized access to container images stored in a registry. Role-based access control (RBAC) is commonly used to assign permissions based on the user's role within the organization. For example, developers may have permission to push images to the registry, while deployment teams may only have permission to pull images. Sensitive images, such as those containing proprietary code or sensitive configurations, can be restricted to a limited group of users. [13]

In addition to RBAC, organizations should consider implementing multi-factor authentication (MFA) for accessing the container registry. MFA adds an extra layer of security by requiring users to provide two or more verification factors to gain access. This significantly reduces the risk of unauthorized access, even if a user's credentials are compromised.

Securing the Build Process

The security of the containerized environment is closely tied to the security of the build process. A compromised build pipeline can lead to the deployment of vulnerable or malicious containers, making it essential to implement robust security measures throughout the build process. [3]

1. CI/CD Pipeline Security:

Continuous integration and continuous deployment (CI/CD) pipelines are integral to modern software development, enabling rapid and automated delivery of code changes. However, these pipelines can also be a vector for security attacks if not properly secured. Securing the CI/CD pipeline involves implementing strong authentication and access controls, ensuring that only authorized personnel can trigger builds or modify the pipeline configuration. [7]

Organizations should also secure the environments where builds are executed. Build environments should be isolated from production systems to prevent any potential compromise during the build process from affecting live applications. Additionally, build tools and dependencies should be regularly updated to patch any known vulnerabilities. [14]

2. Automated Security Testing:

Automated security testing is a key component of a secure CI/CD pipeline. By integrating security tests into the build process, organizations can detect and remediate vulnerabilities before the application is deployed. These tests can include static code analysis, which scans the source code for security flaws, and dependency scanning, which checks for vulnerabilities in the libraries and frameworks used by the application.

Configuration checks are also important, ensuring that the containerized application adheres to security best practices. For example, automated tests can verify that containers do not run as root, that network policies are correctly configured, and that sensitive information is not exposed. [15]

3. Immutable Builds:

Immutable builds are a security practice where once an image is built, it is not modified. This ensures that the same image is used across all environments, from development to production, reducing the risk of inconsistencies and vulnerabilities. Immutable builds also simplify the process of rolling back to a previous version in case of an issue, as the exact same image can be redeployed without any changes. [4]

To implement immutable builds, organizations should use version control systems to track changes to the image's Dockerfile and configuration. Each build should be tagged with a unique identifier, allowing it to be referenced consistently across environments. This practice not only enhances security but also improves the reliability and predictability of deployments.

Orchestration and Deployment Security

Securing Kubernetes and Other Orchestration Platforms

Container orchestration platforms like Kubernetes have become essential for managing large-scale containerized applications. These platforms automate the deployment, scaling, and management of containers, but their complexity introduces significant security challenges. Securing the orchestration platform is critical to ensuring the overall security of the containerized environment.

1. Kubernetes API Security:

The Kubernetes API server is the central component of a Kubernetes cluster, responsible for managing all interactions between users and the cluster. As such, it is a prime target for attackers seeking to gain unauthorized access to the cluster. Securing the Kubernetes API server involves enforcing strong authentication mechanisms, such as client certificates, OAuth tokens, or service accounts, to ensure that only authorized users and services can access the API.

Role-based access control (RBAC) should also be implemented to limit the actions that users can perform through the API. For example, a user responsible for deploying applications should not have the ability to modify the cluster's network configuration. By assigning permissions based on the principle of least privilege, organizations can reduce the risk of accidental or malicious changes to the cluster.

In addition to authentication and access control, organizations should restrict network access to the API server. The API server should only be accessible from trusted networks or through a secure VPN. Public exposure of the API server to the internet should be avoided, as it increases the risk of brute force attacks and other security threats.

2. Network Policies:

Kubernetes Network Policies provide a way to control the communication between pods in a cluster. By default, Kubernetes allows all pods to communicate with each other, which can be a security risk if a pod is compromised. Network Policies allow organizations to define rules that restrict traffic between pods, effectively creating micro-segmentation within the cluster.

For example, a Network Policy can be configured to allow communication only between specific pods that need to interact with each other, such as between an application pod and its associated database pod. This limits the potential for lateral movement within the cluster, where an attacker who gains control of one pod could attempt to compromise other pods.

Implementing Network Policies requires careful planning and testing, as overly restrictive policies can disrupt application functionality. However, when done correctly, Network Policies are a powerful tool for enhancing the security of a Kubernetes cluster.

3. Pod Security Policies:

Pod Security Policies (PSP) are another important security feature in Kubernetes, allowing administrators to define a set of security standards that pods must adhere to before they can be deployed. PSPs can enforce restrictions such as disallowing privileged containers, requiring read-only root filesystems, and restricting the use of host namespaces.

For example, a PSP can be configured to prevent containers from running as root, a common security best practice that reduces the risk of privilege escalation. Similarly, PSPs can enforce the use of specific Linux capabilities, ensuring that containers do not have access to unnecessary system privileges.

Pod Security Policies are particularly useful in multi-tenant environments, where different teams or applications share the same Kubernetes cluster. By enforcing consistent security standards across all pods, organizations can reduce the risk of misconfigurations and vulnerabilities.

4. Secrets Management:

Managing sensitive information, such as API keys, passwords, and certificates, is a critical aspect of securing a containerized environment. Kubernetes Secrets provide a built-in mechanism for storing and managing this information securely. However, improper management of Secrets can lead to exposure of sensitive data. [3]

Organizations should ensure that Secrets are encrypted both at rest and in transit. Kubernetes provides built-in support for encrypting Secrets at rest using encryption providers such as AWS KMS or HashiCorp Vault. Additionally, access to Secrets should be tightly controlled, with RBAC policies limiting which users and services can access specific Secrets.

Another best practice is to avoid hardcoding Secrets into container images or configuration files. Instead, Secrets should be injected into containers at runtime, reducing the risk of accidental exposure. Kubernetes supports this through environment variables or mounted volumes, ensuring that Secrets are only available to the containers that need them. [16]

5. Cluster Hardening:

Hardening the Kubernetes cluster involves securing the underlying infrastructure and configuration to minimize the risk of attacks. This includes disabling unnecessary features, applying security patches promptly, and following best practices for securing the cluster's components. [3]

For example, organizations should disable the Kubernetes Dashboard in production environments unless it is strictly necessary, as it provides a web-based interface that can be exploited if not properly secured. Similarly, the Kubernetes kubelet, which is responsible for managing individual nodes in the cluster, should be configured to require authentication and should not expose its API to the internet. [6]

Regularly updating Kubernetes and its components is essential to protect against known vulnerabilities. Organizations should monitor for security updates and apply patches as soon as they become available. Additionally, using a minimal base image for control plane components can reduce the attack surface and improve the security of the cluster. [2]

Runtime Security

Securing containers during runtime is one of the most challenging aspects of container security, as this is when containers are most vulnerable to attacks. Runtime security involves monitoring container activity, detecting and responding to threats, and implementing measures to limit the impact of a potential compromise. [8]

1. Runtime Threat Detection:

Runtime threat detection is the process of monitoring container activity for signs of malicious behavior. This can include detecting unexpected network connections, privilege escalation attempts, or unauthorized access to sensitive data. Tools like Falco and Sysdig Secure are designed specifically for runtime security in containerized environments, providing real-time monitoring and alerting based on predefined security rules.

For example, Falco can be configured to detect if a container attempts to write to a sensitive directory, such as /etc or /usr, which could indicate a potential attack. Similarly, Sysdig Secure can monitor network traffic to detect communication with known malicious IP addresses or abnormal patterns that may indicate a distributed denial-of-service (DDoS) attack. [1]

Runtime threat detection tools should be integrated with the organization's broader security monitoring and incident response systems. This allows for automated responses to detected threats, such as isolating a compromised container or triggering an alert for further investigation. [9]

2. Container Sandboxing:

Container sandboxing provides an additional layer of isolation for containers, further reducing the risk of a compromise affecting the host system or other containers. Sandboxing involves running containers in a restricted environment where their access to system resources is tightly controlled.

Tools like gVisor and Kata Containers offer container sandboxing by running containers in lightweight virtual machines (VMs) or using a user-space kernel to enforce strict security boundaries. This approach provides a higher level of isolation compared to traditional containers, making it more difficult for an attacker to escape the container and compromise the host system. [17]

While sandboxing can improve security, it may also introduce performance overhead and complexity. Organizations should evaluate the trade-offs and consider sandboxing for high-risk workloads or environments where security is a top priority. [3]

3. Limiting Resource Usage:

Resource limits are an essential part of container runtime security, as they prevent a single container from consuming excessive resources and potentially disrupting other services. Kubernetes allows administrators to set limits on CPU and memory usage for each container, ensuring that no container can monopolize system resources. [7]

Setting appropriate resource limits can also mitigate the impact of denial-of-service (DoS) attacks, where an attacker attempts to overwhelm a service by consuming all available resources. By limiting the resources allocated to each container, organizations can prevent a single compromised container from affecting the entire host system.

In addition to CPU and memory limits, organizations should consider setting limits on other resources, such as disk I/O and network bandwidth, to further isolate containers and reduce the risk of resource exhaustion.

4. Continuous Monitoring and Logging:

Continuous monitoring and logging are critical for maintaining visibility into the security of a containerized environment. Monitoring tools like Prometheus, Grafana, and ELK (Elasticsearch, Logstash, Kibana) provide real-time insights into container activity, resource usage, and system performance. [15]

Centralized logging solutions collect logs from all containers and cluster components, allowing security teams to analyze and correlate events across the environment. This can help identify patterns of suspicious behavior, such as repeated failed login attempts or unauthorized access to sensitive files.

In addition to monitoring and logging, organizations should implement alerting mechanisms that notify the security team of potential incidents. Automated alerts can be triggered based on predefined thresholds or anomaly detection algorithms, enabling a rapid response to emerging threats. [2]

Access Control and Identity Management

Fine-Grained Access Control

Access control is a cornerstone of container security, ensuring that only authorized users and services can interact with the containerized environment. Fine-grained access control involves implementing detailed policies that define who can access what resources and what actions they can perform. [18]

In Kubernetes, Role-Based Access Control (RBAC) provides a powerful mechanism for managing access to the cluster. RBAC allows administrators to define roles with specific permissions and assign those roles to users, groups, or service accounts. For example, a developer might be granted permissions to deploy applications and view logs, but not to modify network policies or access Secrets. [7]

Fine-grained access control helps enforce the principle of least privilege, where users and services are granted only the permissions they need to perform their tasks. This reduces the risk of accidental or malicious actions that could compromise the security of the environment.

In addition to RBAC, organizations should consider implementing Network Policies to control communication between pods. By defining rules that restrict traffic between pods, organizations can prevent unauthorized access and limit the impact of a potential compromise.

Multi-Factor Authentication (MFA)

Multi-Factor Authentication (MFA) adds an extra layer of security by requiring users to provide two or more verification factors to gain access to the containerized environment. MFA is particularly important for accessing critical components, such as the Kubernetes API server, container registry, or CI/CD pipeline.

MFA can include a combination of something the user knows (e.g., a password), something the user has (e.g., a hardware token or mobile device), and something the user is (e.g., a biometric factor). By requiring multiple factors, MFA significantly reduces the risk of unauthorized access, even if a user's credentials are compromised. [3]

Organizations should enforce MFA for all privileged accounts and consider extending it to all users with access to sensitive resources. MFA should also be integrated with identity providers and Single Sign-On (SSO) systems to streamline the authentication process while maintaining strong security.

Identity Federation

Identity federation involves integrating identity management systems with the containerized environment to ensure consistent access policies across the organization. Identity federation allows users to authenticate using their corporate credentials, providing a seamless and secure experience. [12]

Technologies like OpenID Connect (OIDC) and Lightweight Directory Access Protocol (LDAP) are commonly used for identity federation in Kubernetes. By federating identities, organizations can centralize access management, enforce consistent security policies, and simplify the user experience. [2]

Identity federation also supports Single Sign-On (SSO), allowing users to authenticate once and access multiple services without re-entering their credentials. This reduces the risk of password fatigue and improves security by minimizing the number of credentials users need to manage.

Compliance and Governance

Regulatory Compliance

Regulatory compliance is a critical consideration for organizations operating in regulated industries, such as finance, healthcare, or government. Containerized workloads must comply with relevant industry regulations and standards, such as GDPR, HIPAA, or PCI-DSS.

To achieve compliance, organizations must implement security controls that meet the requirements of the applicable regulations. This may include data encryption, access controls, auditing, and incident response planning. Compliance scanning tools can help automate the process of verifying that containerized workloads meet regulatory requirements.

In addition to technical controls, organizations should establish governance frameworks that define roles, responsibilities, and processes for maintaining compliance. Regular audits and assessments are essential to ensure that compliance is maintained as the environment evolves.

Audit and Accountability

Auditability is a key aspect of security and compliance in containerized environments. Organizations must be able to track and document all actions taken within the environment, including deployments, configuration changes, and access to sensitive data. [4]

Kubernetes provides auditing capabilities that log all API requests, including the identity of the requester, the action taken, and the result. These logs can be stored in a centralized logging system for analysis and retention.

Audit logs should be regularly reviewed to detect signs of unauthorized access or other suspicious activities. Organizations should also implement retention policies that ensure audit logs are kept for the required period to meet regulatory requirements.

Accountability is equally important, ensuring that all actions can be traced back to an individual or service account. This helps prevent unauthorized actions and provides a clear record of who is responsible for changes in the environment.

Incident Response Planning

Incident response planning is essential for managing security incidents in a containerized environment. An effective incident response plan should include procedures for detecting, responding to, and recovering from security incidents.

Organizations should develop incident response playbooks that outline specific actions to be taken in the event of an incident, such as isolating a compromised container, investigating the root cause, and restoring services. These playbooks should be regularly tested and updated to reflect changes in the environment and emerging threats. [19]

In addition to technical response measures, organizations should establish communication protocols for notifying stakeholders and regulatory authorities in the event of a breach. Clear communication is essential for managing the impact of an incident and maintaining trust with customers and partners. [3]

Continuous Compliance Monitoring

Continuous compliance monitoring is necessary to ensure that containerized workloads remain compliant with regulatory requirements and internal policies. This involves monitoring the environment in real-time for configuration drifts, policy violations, and other indicators of non-compliance. [6]

Tools like Open Policy Agent (OPA) and Gatekeeper can be integrated with Kubernetes to enforce compliance policies at runtime. These tools allow organizations to define and enforce policies that govern various aspects of the environment, such as network security, access control, and resource usage. [8]

By implementing continuous compliance monitoring, organizations can detect and remediate issues before they lead to security incidents or regulatory violations. This proactive approach helps maintain a strong security posture and ensures that the environment remains compliant as it evolves. [20]

Conclusion

Containerization has transformed the way organizations develop, deploy, and manage applications, offering unprecedented benefits in terms of scalability, flexibility, and efficiency. However, these benefits come with unique security challenges that require a tailored approach to security. Traditional security models must be adapted to address the dynamic and distributed nature of containerized workloads. [21]

This paper has explored advanced security strategies for securing containerized environments, covering the entire container lifecycle from image creation to runtime. By implementing these strategies, organizations can mitigate risks, protect against evolving threats, and maintain a strong security posture.

Securing containerized workloads requires a multi-layered approach that includes image security, orchestration platform hardening, runtime security, access control, and compliance management. These strategies, when combined with robust monitoring, logging, and incident response measures, provide a comprehensive security framework that can protect containerized environments in any setting. [22]

As container technology continues to evolve, so too must the security practices that protect it. Organizations must stay informed of emerging threats and continuously refine their security strategies to ensure that they can reap the benefits of containerization without compromising on security. With the right strategies in place, containerized workloads can be deployed with confidence, knowing that they are protected against the full spectrum of security risks. [23]

References

[1] Kochovski P., "Dependability of container-based data-centric systems.", Security and Resilience in Intelligent Data-Centric Systems and Communication Networks, 2017, pp. 7-27.

[2] Sun J., "Blockchain-based automated container cloud security enhancement system.", Proceedings - 2020 IEEE International Conference on Smart Cloud, SmartCloud 2020, 2020, pp. 1-6.

[3] Lingayat A., "Integration of linux containers in openstack: an introspection.", Indonesian Journal of Electrical Engineering and Computer Science, vol. 12, no. 3, 2018, pp. 1094-1105.

[4] Suneja S., "Can container fusion be securely achieved?.", WOC 2019 - Proceedings of the 2019 5th International Workshop on Container Technologies and Container Clouds, Part of Middleware 2019, 2019, pp. 31-36.

[5] Bila N., "Leveraging the serverless architecture for securing linux containers.", Proceedings - IEEE 37th International Conference on Distributed Computing Systems Workshops, ICDCSW 2017, 2017, pp. 401-404.

[6] Hamilton M., "Large-scale intelligent microservices.", Proceedings - 2020 IEEE International Conference on Big Data, Big Data 2020, 2020, pp. 298-309.

[7] Bhowmik S., "Container based on-premises cloud security framework.", Proceedings of the 5th International Conference on Inventive Computation Technologies, ICICT 2020, 2020, pp. 773-778.

[8] Zhang Z., "Security in network functions virtualization.", Security in Network Functions Virtualization, 2017, pp. 1-272.

[9] Hewage P., "An agile farm management information system framework for precision agriculture.", ACM International Conference Proceeding Series, 2017, pp. 75-80.

[10] Zheng T., "Bigvm: a multi-layer-microservice-based platform for deploying saas.", Proceedings - 5th International Conference on Advanced Cloud and Big Data, CBD 2017, 2017, pp. 45-50.

[11] Jani, Y. "Security best practices for containerized applications." Journal of Scientific and Engineering Research 8.8 (2021): 217-221.

[12] Khan A., "Key characteristics of a container orchestration platform to enable a modern application.", IEEE Cloud Computing, vol. 4, no. 5, 2017, pp. 42-48.

[13] Tak B., "Security analysis of container images using cloud analytics framework.", Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 10966 LNCS, 2018, pp. 116-133.

[14] Lin X., "A measurement study on linux container security: attacks and countermeasures.", ACM International Conference Proceeding Series, 2018, pp. 418-429.

[15] Tanwani A.K., "A fog robotics approach to deep robot learning: application to object recognition and grasp planning in surface decluttering.", Proceedings - IEEE International Conference on Robotics and Automation, vol. 2019-May, 2019, pp. 4559-4566.

[16] Shah A.A., "A qualitative cross-comparison of emerging technologies for software-defined systems.", 2019 6th International Conference on Software Defined Systems, SDS 2019, 2019, pp. 138-145.

[17] Huang C.H., "Enhancing the availability of docker swarm using checkpoint-and-restore.", Proceedings - 14th International Symposium on Pervasive Systems, Algorithms and Networks, I-SPAN 2017, 11th International Conference on Frontier of Computer Science and Technology, FCST 2017 and 3rd International Symposium of Creative Computing, ISCC 2017, vol. 2017-November, 2017, pp. 357-362.

[18] Xu C., "Isopod: an expressive dsl for kubernetes configuration.", SoCC 2019 - Proceedings of the ACM Symposium on Cloud Computing, 2019, pp. 483.

[19] Xie B., "Prediction-based autoscaling for container-based paas system.", Proceedings - 2017 IEEE 2nd International Conference on Automatic Control and Intelligent Systems, I2CACIS 2017, vol. 2017-December, 2017, pp. 19-24.

[20] Islam M.S., "Secure real-time heterogeneous iot data management system.", Proceedings - 1st IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications, TPS-ISA 2019, 2019, pp. 228-235.

[21] Yousefpour A., "All one needs to know about fog computing and related edge computing paradigms: a complete survey.", Journal of Systems Architecture, vol. 98, 2019, pp. 289-330.

[22] Yu D., "A survey on security issues in services communication of microservices-enabled fog applications.", Concurrency and Computation: Practice and Experience, vol. 31, no. 22, 2019.

[23] Bao L., "Performance modeling and workflow scheduling of microservice-based applications in clouds.", IEEE Transactions on Parallel and Distributed Systems, vol. 30, no. 9, 2019, pp. 2101-2116.