Containers are configured to run without privileged access or root privileges, and privilege escalation is restricted. **(What)** This ensures that containers operate with the least privilege necessary to perform their tasks, minimizing the risk of unauthorized access, privilege escalation, or potential system compromise. **(Why)** The control is implemented through security configurations and policies that enforce restrictions on running containers as privileged users and prevent any unauthorized escalation of privileges. **(How)** By applying these controls consistently across all container environments, the security posture is strengthened, reducing the attack surface and protecting both the containerized applications and the underlying host systems from potential threats. **(How Much)** **(When)** **(Where)** **(Who)**

Here is a detailed breakdown of the control for **Privileged Access Management for Containers**, structured according to your requested categories:

**Control Objective:**

The control is performed to ensure containers operate with the least privilege necessary, preventing unauthorized access, privilege escalation, or system compromise. By restricting containers from running with privileged access or root privileges, the control helps mitigate security risks associated with container vulnerabilities, reducing the attack surface and improving the security posture of the containerized environment.

**Control Frequency:**

The control is performed **continuously** as part of the container deployment process, with periodic reviews and audits to ensure compliance. Any changes to container configurations or security policies trigger a reassessment of the control.

**Control Ownership:**

* **Performer:** The control is typically performed by the **DevOps** or **Security Operations** teams who manage container configurations and deployments.
* **Responsible Party:** The **Security Team** is responsible for ensuring the implementation and ongoing effectiveness of this control, ensuring it aligns with the organization's security policies and best practices.

**Control Type:**

* **Type:** **Preventative**  
  The control is designed to **prevent** containers from running with elevated privileges or allowing privilege escalation in the first place.

**Control Classification:**

* **Classification:** **Automated**  
  The control is largely automated through the use of security policies (e.g., OPA for Kubernetes, runtime security settings for non-Kubernetes environments) and security configuration tools.

**Where is it performed:**

* **Location/System:**  
  The control is applied across all **containerized environments**, both on-premises and in cloud environments. It applies to any system that manages containerized applications and ensures containers do not have privileged access or the ability to escalate privileges.

**Background Detail:**

Containers running with root or privileged access increase the attack surface and present a significant security risk. Without proper privilege management, an attacker exploiting a vulnerability within a container can escalate privileges and compromise the underlying host or other containers. This control is essential for enforcing secure container configurations, especially in production environments.

**How is the control performed:**

* **Key Steps:**
  1. **Configuration Setup:**  
     Containers are configured to run without root or privileged access, ensuring they operate with the least privilege.
  2. **Security Policy Enforcement:**  
     Automated security policies, such as **OPA** in Kubernetes, are enforced to restrict containers from acquiring elevated privileges.
  3. **Audit and Review:**  
     The configurations are audited to verify that no container is running with unauthorized privileges.
  4. **Incident Response:**  
     Any deviation or non-compliance is flagged for further investigation and remediation.
* **Inputs and Sources:**
  1. Container configuration files.
  2. Security policies (e.g., OPA policy files, container runtime configurations).
  3. Security audit logs and reports.
* **What is done to process the inputs:**
  1. The configuration files are checked for the presence of root or privileged access.
  2. Security policies are applied to ensure containers cannot escalate privileges.
  3. Audit tools and logs are reviewed to ensure compliance with security requirements.
* **Relevant thresholds/limits:**
  1. Containers running with root privileges or privilege escalation rights are considered non-compliant.
  2. Any deviation from the security policies, such as containers running in privileged mode, triggers an alert.
* **How outliers are tracked, managed, escalated:**
  1. **Outliers** such as containers running with privileged access are tracked through automated alerts.
  2. These outliers are managed by a remediation process, where configurations are updated to meet security standards.
  3. **Escalation:** If the issue persists or is part of a larger security breach, it is escalated to senior security personnel for investigation.
* **How the control review is evidenced:**
  1. Evidence of the control’s effectiveness can be found in security audit logs, policy enforcement logs, and configuration management reports. Additionally, periodic control reviews are documented in compliance check reports, which are stored in version-controlled repositories (e.g., GitHub or internal documentation platforms).

This provides a comprehensive view of the control, its operation, and the necessary steps for ensuring privileged access management is effectively enforced across container environments.

**Control Test: Privileged Access Management for Containers**

**Test Objective:**  
To verify that containers do not run with privileged access or root privileges, and privilege escalation is restricted. The test checks compliance across the entire container population, including both Kubernetes and non-Kubernetes environments.

**Test Steps:**

1. **Kubernetes Environment (OPA Policies Enforcement):**
   * **Test Evaluation Criteria:**  
     The control requires that containers in Kubernetes are not allowed to run with privileged roles (e.g., root access), and privilege escalation must be restricted through **Open Policy Agent (OPA)** policies. The OPA policies should be securely managed and protected in a centralized GitHub repository.
   * **Test Execution:**
     + The GitHub repository storing the OPA policies was checked to verify that the policies restrict privileged access and prevent privilege escalation.
     + The **secure settings** on the GitHub repository were reviewed to ensure that no unauthorized modifications could be made to the policies.
     + The repository was confirmed to have **secure repository settings** enabled, restricting unauthorized changes and ensuring the integrity of the policies.
   * **Outcome:**  
     No changes were detected in the OPA policies, and the repository settings were confirmed to be secure, ensuring that the policies governing privileged access were intact.
2. **Non-Kubernetes Environment (Lack of Control):**
   * **Test Evaluation Criteria:**  
     Containers in non-Kubernetes environments should not run with root or privileged access. Privilege escalation must be restricted by configuring the container runtime or using other container security mechanisms.
   * **Test Execution:**
     + It was tested that containers can run as privileged containers (root access) in the non-Kubernetes environment.
     + No restrictions were found in place to prevent privilege escalation for the containers.
   * **Outcome:**  
     The test confirmed that containers can indeed run as privileged containers in the non-Kubernetes environment, and there were no implemented controls to restrict privilege escalation.

**Overall Control Test Result:**

**Failure** – The control failed across the entire container population. While Kubernetes environments are controlled through OPA policies, non-Kubernetes containers were found to have the ability to run as privileged users (root access) and there were no controls in place to prevent privilege escalation.

**Impact of Control Failure:**

The failure of this control poses significant security risks across the containerized environment:

* **Privilege Escalation Risk:** Containers running with root privileges or without restrictions on privilege escalation increase the potential for attackers to exploit container vulnerabilities. An attacker could escalate privileges within a compromised container, gaining control over the underlying host system and other containers.
* **Attack Surface Expansion:** Privileged access increases the attack surface, as containers with root privileges can access sensitive system resources and data, further compounding the potential for security breaches.
* **Compliance and Security Posture:** Failure to enforce least-privilege access in non-Kubernetes environments undermines the security framework of the containerized systems. This control's failure exposes the organization to non-compliance with internal security standards and industry regulations, putting the organization at risk of breaches, data leaks, or other critical security incidents.
* **Operational Impact:** Unauthorized privilege escalation could lead to unintentional misconfigurations, unauthorized changes to the system, or the propagation of malicious code, all of which can disrupt operations and damage the integrity of the environment.

**Test Outcome Evaluation:**

* **The result compared against expectations:**
  + The expectation was that containers would not run as privileged users and would be restricted from privilege escalation, but containers in the non-Kubernetes environment were found to be running as privileged users (root access), which is in violation of the control's requirements. This is considered a failure, as the control was not functioning as expected.
* **Rationale for test failure:**
  + The test failed because, in the non-Kubernetes environment, containers were able to run as privileged users, and there were no controls in place to prevent privilege escalation. This presents a critical security risk, as containers should adhere to the principle of least privilege to minimize potential attack vectors. Since the control was not implemented effectively in this environment, it failed to prevent privileged access as intended.

**Exceptions:**

* **Exception 1 Raised:** In the non-Kubernetes environment, containers were able to run as privileged containers (root access), and no privilege escalation restrictions were in place. This represents a critical security gap requiring immediate remediation.  
    
  Here is the completed **Control Assessment** table based on the control test results:

|  |  |  |
| --- | --- | --- |
| * **Design Effectiveness Assessment Questions** | * **Result** | * **Rationale** |
| * **Is this the right control to mitigate the risk?** | * Fail | * The control is designed to restrict privileged access and prevent privilege escalation; however, in the non-Kubernetes environment, containers were found to be running as privileged users. |
| * **Does the control achieve its objective?** | * Fail | * The control failed to achieve its objective in the non-Kubernetes environment where containers were able to run as privileged users, violating the intended restriction on privileged access. |
| * **Is the control performed by the right people with requisite skills, knowledge, and experience?** | * N/A | * This question is not applicable as the failure was due to a lack of proper control enforcement rather than the execution of the control itself. |
| * **Does the control have adequate segregation of duties?** | * Pass | * The control itself appears to be designed with segregation of duties, especially for Kubernetes environments where centralized control over policies is implemented. |
| * **Is the control performed at the right time or in the right stage of the process?** | * Pass | * The control is applied at the deployment stage to prevent privileged access during container startup, ensuring that access restrictions are applied when containers are initialized. |
| * **Is the control performed at the right frequency?** | * Pass | * The control is evaluated continuously through the use of centralized policy management tools (e.g., GitHub for OPA policies), ensuring ongoing enforcement of privileged access restrictions. |
| * **Is the control sustainable?** | * Fail | * The control is not sustainable in non-Kubernetes environments, as containers were found to be running with root access and without restrictions on privilege escalation. |
| * **How does it manage/escalate an issue?** | * Fail | * The control does not adequately manage or escalate issues in non-Kubernetes environments, as there were no mechanisms to prevent or alert when containers are run as privileged users. |
| * **Is the control evidenced?** | * Pass | * Evidence for the Kubernetes environment is available via GitHub repository checks and policy validations, confirming that the control is implemented and functioning as intended. |
| * **Does the control have adequate management focus?** | * Fail | * The control's lack of enforcement in the non-Kubernetes environment highlights an insufficient focus on comprehensive container security management across all environments. |

* **Exceptions Raised:**
* **Exception 1:** In the non-Kubernetes environment, containers were able to run as privileged containers (root access), and no privilege escalation restrictions were in place. This exposes the system to security vulnerabilities and requires immediate remediation.