Here is the draft audit workpaper for the given control:

**Control Description (5W2H):**

The NABserv team (who) releases container images that are subjected to a vulnerability scanning process and image hardening (what). This process is integrated into the CI/CD pipeline (where) during the image build and deployment stages (when) to block images with critical vulnerabilities (why). The scanning process is automated (how) and applies to all container images used by the asset teams (how much) before deployment to production.

**Control Objective:**

The objective of the control is to ensure that only secure and hardened container images, free of critical vulnerabilities, are deployed to production. This minimizes security risks, such as unauthorized access, exploitation, or breaches, which could arise from using unscanned or vulnerable images.

**Control Frequency:**

The control is performed **continuously** as part of the CI/CD pipeline during the container image build and deployment stages.

**Control Ownership:**

* **Performed By:** NABserv team
* **Responsibility:** NABserv team owns the control and ensures scanning and hardening processes are implemented.

**Control Type:**

Preventative

**Control Classification:**

Automated

**Where is it performed:**

* **Location:** CI/CD pipeline
* **System:** CI/CD tools integrated with vulnerability scanners (e.g., Harness, AWS EKS, or related tools)

**Background Detail (if required):**

The control aims to enforce secure container practices by automatically integrating vulnerability scanning and image hardening into the CI/CD pipeline. Images that fail the predefined critical vulnerability threshold are blocked to ensure only secure builds are deployed.

**How is the control performed:**

**Inputs and Sources:**

* Container images built by the NABserv team.
* Vulnerability databases (e.g., CVE data feeds or commercial vulnerability scanner).
* CI/CD pipeline tools (e.g., AWS Harness).

**What is done to process the inputs:**

1. **Vulnerability Scanning:** Container images are scanned for known vulnerabilities during the build process.
2. **Image Hardening:** Images are checked to ensure they adhere to hardening standards (e.g., minimal base images, removal of unnecessary components).
3. **Threshold Check:** Images are evaluated against critical vulnerability thresholds (e.g., CVSS score > 9).
4. **Blocking Deployment:** Images that exceed the critical vulnerability threshold are automatically blocked and not deployed to production.

**Relevant Thresholds/Limits:**

* Images with vulnerabilities rated as **Critical** (e.g., CVSS > 9) are not allowed to proceed.
* Hardening standards align with organization-defined benchmarks.

**Outlier Tracking, Management, and Escalation:**

* Failed builds trigger automated alerts to the NABserv team.
* Issues are escalated for remediation before the image can pass through the pipeline again.

**How the control review is evidenced:**

* CI/CD logs showing successful/failed vulnerability scans.
* Automated reports from vulnerability scanning tools with details on findings.
* Audit trails in the CI/CD pipeline confirming blocked deployment attempts.

Here is the corrected and updated version of the **Control Test** template:

**Control Test:**

**Test Step:**  
Audit verified the **end-to-end process** of container image creation, hardening, and vulnerability scanning through the Jenkins pipeline and assessed whether the process can be bypassed.

* **Control Attributes:**
  1. All container images are hardened by installing the required software, CLI tools, and security certificates.
  2. Vulnerability scanning is performed on all container images before deployment.

**Test Evaluation Criteria:**

* The control evaluation was based on documented security processes, hardening baselines, and pipeline enforcement requirements.
* **Expectation:** All container images must be hardened and vulnerability scanned before deployment. The CI/CD pipeline must prevent bypassing of the hardening and scanning processes.

**Key Test Steps to Perform the Test:**

1. **What Was Looked At [File/Page/Ref]:**
   * Reviewed the **Jenkins pipeline configuration** for the container image creation and deployment process.
   * Examined the **Dockerfile** for hardening parameters (GitHub reference: docker-image-java17-redhat-ubi9/Dockerfile.amd64).
   * Analyzed deployment documentation detailing the container image process.
   * Investigated the configuration of Artifactory to understand how images are stored and accessed.
2. **How the Test Was Executed:**
   * Verified the automated steps in the Jenkins pipeline for:
     + Installation of CLI tools, security certificates, and software.
     + Execution of vulnerability scanning before deployment.
     + Enforcement of critical vulnerability thresholds (e.g., pipeline stops for critical issues).
   * Checked if the pipeline can be bypassed:
     + Evaluated whether asset teams can directly download container images from Artifactory and manually deploy them without hardening or scanning.
   * Confirmed whether there are any manual steps outside the Jenkins pipeline that allow deviations from the defined process.

**Outcome:**

1. **Hardened Images:**
   * The Jenkins pipeline includes steps for installing security certificates, CLI tools, and required software during the image build process.
   * The Dockerfile specifies the hardening parameters.
2. **Vulnerability Scanning:**
   * Vulnerability scanning is integrated into the pipeline. Critical vulnerabilities (CVSS > 9) block image deployment, while medium and low vulnerabilities generate warnings.
3. **Bypassing the Process:**
   * **Issue Identified:** Asset teams can manually fetch container images from Artifactory and deploy them directly to VMs, bypassing the Jenkins pipeline.
   * **Impact:** This bypass allows deployment of non-hardened and non-scanned container images into the environment.

**Result Comparison Against Expectations:**

* **Expectation:** The process should ensure that all container images are hardened and vulnerability scanned before deployment, with no ability to bypass the pipeline.
* **Condition:** While the Jenkins pipeline enforces hardening and scanning, it can be bypassed by manually deploying images fetched from Artifactory.

**Rationale for Test Failure:**

The control is ineffective because:

* The Jenkins pipeline enforces hardening and scanning, but this process can be bypassed.
* There is no mandatory enforcement to prevent asset teams from directly deploying images stored in Artifactory without using the pipeline.

**Control Test Result:**

**Fail** – The control did not ensure mandatory hardening and vulnerability scanning of all container images. The process can be bypassed, allowing non-compliant images to be deployed.

**Exceptions:**

* **Exception 1:** Asset teams can directly download container images from Artifactory and deploy them outside the Jenkins pipeline.
* **Exception 2:** Vulnerability scanning and hardening are not enforced for images bypassing the pipeline.

**Impact:**

* Deployment of non-hardened and unscanned container images increases the risk of introducing critical vulnerabilities into the environment.
* Potential for security breaches, unauthorized access, or exploitation due to non-compliant container images.

| **Assessment Question** | **Result** | **Rationale** | **Linked Exception** |
| --- | --- | --- | --- |
| **Is this the right control to mitigate the risk?** | Fail | While the control attempts to enforce hardening and vulnerability scanning, it can be bypassed by asset teams. The absence of a minimum security baseline further weakens its effectiveness. | Exception 1: Lack of minimum security baseline.Exception 2: Inconsistent implementation. |
| **Does the control achieve its objective?** | Fail | The control ensures hardening and scanning only for NEF images via Jenkins. Images bypassing the pipeline are not hardened or scanned, leading to deployment of vulnerable images. | Exception 1: Lack of minimum security baseline.Exception 2: Inconsistent implementation. |
| **Is the control performed by the right people with requisite skills, knowledge, and experience?** | Fail | The NABserv team performs the process correctly for NEF images. However, asset teams can bypass the process, indicating inconsistent oversight and lack of enforcement. | Exception 2: Manual bypass by asset teams. |
| **Does the control have adequate segregation of duties?** | Fail | Asset teams can download images directly from Artifactory and deploy them, bypassing the centralized process. This violates segregation of duties as the same team can select and deploy images without oversight. | Exception 2: Manual bypass by asset teams. |
| **Is the control performed at the right time or in the right stage of the process?** | Fail | For NEF images, hardening and scanning occur at the build stage. For non-NEF images, the process is bypassed entirely, resulting in vulnerable images being deployed. | Exception 2: Manual bypass by asset teams. |
| **Is the control performed at the right frequency?** | Fail | Hardening and scanning occur consistently only through the Jenkins pipeline. However, manual image downloads bypass this frequency, resulting in inconsistent application of the control. | Exception 2: Manual bypass by asset teams. |
| **Is the control sustainable?** | Fail | The control is unsustainable due to its reliance on the Jenkins pipeline without enforcement mechanisms. The lack of governance and baseline configurations allows bypassing and inconsistent implementation. | Exception 1: Lack of minimum security baseline.Exception 2: Inconsistent implementation. |
| **How does it manage/escalate an issue?** | Fail | There is no centralized process to detect or escalate bypassed images. Vulnerabilities in manually deployed images go untracked, increasing the risk of unpatched container environments. | Exception 2: Inconsistent implementation. |
| **Is the control evidenced?** | Fail | While Jenkins pipeline logs provide evidence of hardening and scanning for NEF images, no evidence exists for manual deployments. Old container images running unpatched further highlight the lack of effective enforcement. | Exception 1: Lack of minimum security baseline. Exception 2: Unpatched images running. |
| **Does the control have adequate management focus?** | Fail | Management has not defined a minimum security baseline for hardening. Additionally, there is no enforcement or monitoring to ensure adherence to the container image creation process. | Exception 1: Lack of minimum security baseline.Exception 2: Inconsistent implementation. |