# Container Security Scanning

This document describes the container security scanning capabilities integrated into the RAS\_DASH\_CSaaS platform.

## Overview

The container scanning module provides comprehensive security analysis for: - Docker images and containers - Kubernetes clusters and workloads - Container registries - Dockerfiles and container configurations - Runtime container security

## API Endpoints

### POST /api/v1/scanner/container-scan

Execute a container security scan.

**Request Body:**

{  
 "target": "nginx:latest",  
 "checks": ["image-vulnerabilities", "dockerfile-security", "container-config"],  
 "severity": "medium",  
 "comprehensive": false,  
 "timeout": 300,  
 "customOptions": {}  
}

**Parameters:** - target (required): Container image, registry, or Kubernetes cluster to scan - checks (optional): Array of specific checks to run - image-vulnerabilities: Scan for known vulnerabilities in image layers - dockerfile-security: Analyze Dockerfile for security best practices - container-config: Check container configuration security - runtime-security: Monitor runtime container behavior - secrets-detection: Detect hardcoded secrets and credentials - compliance-checks: Verify compliance with security standards - registry-scan: Scan container registry for vulnerabilities - kubernetes-scan: Security scan of Kubernetes resources - severity (optional): Minimum severity level (low, medium, high, critical) - comprehensive (optional): Run all available checks - timeout (optional): Scan timeout in seconds (default: 300)

**Response:**

{  
 "success": true,  
 "message": "Container scan initiated successfully",  
 "data": {  
 "scanId": "scan\_123456",  
 "target": "nginx:latest",  
 "status": "running",  
 "checks": ["image-vulnerabilities"],  
 "startTime": "2025-07-30T21:30:00Z",  
 "estimatedDuration": "5-10 minutes"  
 }  
}

## Available Scan Types

### 1. Image Vulnerability Scanning

Scans container images for known security vulnerabilities using CVE databases.

**Example:**

container-scan nginx:latest --checks image-vulnerabilities --severity medium

### 2. Dockerfile Security Analysis

Analyzes Dockerfile for security best practices and potential misconfigurations.

**Example:**

container-scan ./Dockerfile --checks dockerfile-security,secrets-detection

### 3. Kubernetes Security Scanning

Comprehensive security assessment of Kubernetes cluster and workloads.

**Example:**

container-scan k8s://default --checks kubernetes-scan,compliance-checks

### 4. Container Registry Scanning

Scans container registries for vulnerabilities across multiple images.

**Example:**

container-scan registry://harbor.company.com --checks registry-scan,image-vulnerabilities

### 5. Runtime Container Security

Monitors running containers for security anomalies and policy violations.

**Example:**

container-scan container://myapp-prod --checks runtime-security,container-config

## Terminal Interface

The scanner provides preset commands in the terminal interface under the “Containers” tab:

### Available Presets:

1. **Docker Image Vulnerability Scan** - Basic image vulnerability assessment
2. **Comprehensive Container Scan** - Full security assessment with all checks
3. **Dockerfile Security Analysis** - Security analysis of Dockerfile
4. **Kubernetes Cluster Scan** - K8s cluster security assessment
5. **Container Registry Scan** - Registry-wide vulnerability scanning
6. **Runtime Container Security** - Live container monitoring

## Integration with Trivy

The container scanning functionality leverages Trivy, a comprehensive vulnerability scanner for containers:

* **Vulnerability Detection**: Scans OS packages and language-specific packages
* **Misconfiguration Detection**: Detects IaC misconfigurations
* **Secret Detection**: Finds API keys, passwords, and tokens
* **SBOM Support**: Generates Software Bill of Materials
* **Policy Enforcement**: Custom security policies

## Supported Targets

### Container Images

* Docker Hub images: nginx:latest
* Private registry images: registry.company.com/myapp:v1.0
* Local images: local/myapp:dev

### Container Registries

* Docker Hub: registry://docker.io
* Harbor: registry://harbor.company.com
* AWS ECR: registry://123456789.dkr.ecr.us-east-1.amazonaws.com

### Kubernetes

* Specific namespace: k8s://production
* All namespaces: k8s://cluster
* Specific workload: k8s://default/deployment/myapp

### Files

* Dockerfile: ./Dockerfile
* Kubernetes manifests: ./k8s/
* Docker Compose: ./docker-compose.yml

## Security Findings

Container scans generate detailed reports including:

### Vulnerability Findings

* CVE ID and severity
* Affected packages and versions
* Available fixes and patches
* CVSS scores and vectors

### Configuration Issues

* Insecure container configurations
* Privilege escalation risks
* Network security issues
* Resource limit violations

### Secrets Detection

* Hardcoded API keys
* Database credentials
* Private keys and certificates
* Authentication tokens

### Compliance Violations

* CIS Docker Benchmark
* NIST Container Security
* PCI DSS requirements
* Custom policy violations

## Best Practices

1. **Regular Scanning**: Implement automated container scanning in CI/CD pipelines
2. **Severity Thresholds**: Set appropriate severity thresholds for your environment
3. **Policy Enforcement**: Define and enforce container security policies
4. **Continuous Monitoring**: Monitor runtime containers for security events
5. **Remediation Tracking**: Track and remediate identified vulnerabilities

## Troubleshooting

### Common Issues

1. **Permission Errors**: Ensure proper registry authentication
2. **Network Timeouts**: Increase timeout values for large images
3. **Resource Limits**: Monitor system resources during scans
4. **Registry Access**: Verify network connectivity to registries

### Performance Optimization

* Use image layer caching for faster scans
* Schedule scans during off-peak hours
* Implement incremental scanning for large registries
* Use local mirrors for frequently scanned images

## API Testing

You can test the container scanning API using your laptop’s IP addresses: - **Wi-Fi**: 192.168.0.39 - **Ethernet**: 192.168.56.1 - **VPN**: 10.5.0.2

Example test command: ```powershell Invoke-RestMethod -Uri “http://192.168.0.39:3001/api/v1/scanner/container-scan” -Method POST -Headers @{“Content-Type”=“application/json”; “Authorization”=“Bearer test-token”} -Body ‘{“target”: “nginx:latest”, “checks”: [“image-vulnerabilities”], “severity”: “medium”}’