Project Title: Kubernetes Cluster Security Assessment

Short Project Description:

The system generates actionable reports to improve cluster security posture.

Component	Role
Cluster Inventory Collector	Gathers information on nodes, pods, services
Role-Based Access Control (RBAC) Analyzer	Analyzes Kubernetes roles and permissions
Pod Security Evaluator	Reviews pod security settings
Network Policy Analyzer	Evaluates inter-pod network segmentation
Report Generator	Creates cluster security assessment reports

Component Details:

1. **Cluster Inventory Collector**:
    - Connects to Kubernetes API.
    - Collects data on:
        - Nodes
        - Pods
        - Namespaces
        - Etc
2. **RBAC Analyzer**:
    - Looks for:
        - Overly permissive roles
        - Missing namespace separation
        - Etc
3. **Pod Security Evaluator**:
    - Checks:
        - If pods run as root
        - Lack of security contexts
        - Etc
4. **Network Policy Analyzer**:
    - Verifies if network policies limit traffic between pods.
5. **Report Generator**:
    - Creates prioritized action items to secure the cluster.

Overall System Flow:

- Input: Access to Kubernetes API
- Output: Kubernetes cluster security audit report
- Focus on **configuration, network, and identity management auditing**

Internal Functioning of Each Module:

1. Cluster Inventory Collector

- **How it works:**
  - Authenticates with kubeconfig or in-cluster API credentials.
  - Enumerates:
    - Nodes
    - Pods
    - Services
    - Ingress controllers
    - Network policies
    - Etc
  - Pulls metadata for each object (labels, annotations, service accounts).

---

2. RBAC Analyzer

- **How it works:**
  - Parses RoleBindings, ClusterRoleBindings, Roles, and ClusterRoles.
  - Checks for:
    - Privileged roles (admin, cluster-admin)
    - Wildcard permissions ("verbs: *", "resources: *")
    - Roles granted outside intended namespaces
    - Etc
- **Flags:** Over-privileged service accounts and risky role grants.

---

3. Pod Security Evaluator

- **How it works:**
  - Parses pod specs.
  - Checks for:
    - runAsUser=0 (root pods)
    - hostPID, hostNetwork access
    - Missing securityContext or seccompProfile
    - Privileged container flags (privileged: true)
    - Etc

---

4. Network Policy Analyzer

- **How it works:**
  - Lists all NetworkPolicy objects.
  - Verifies if:

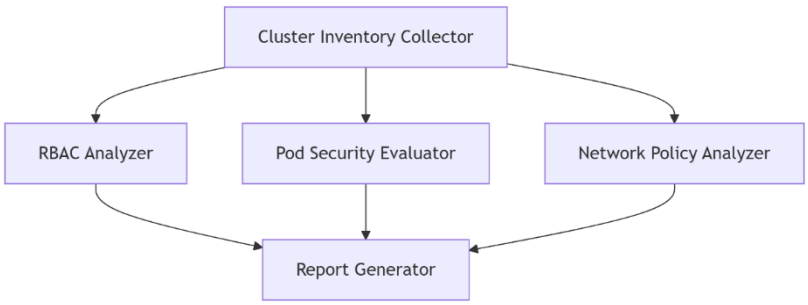
        - Segmentation exists between namespaces.
        - Traffic is restricted (deny-all first policy).
- **Flags:** Open communication between sensitive pods.

---

5. Report Generator

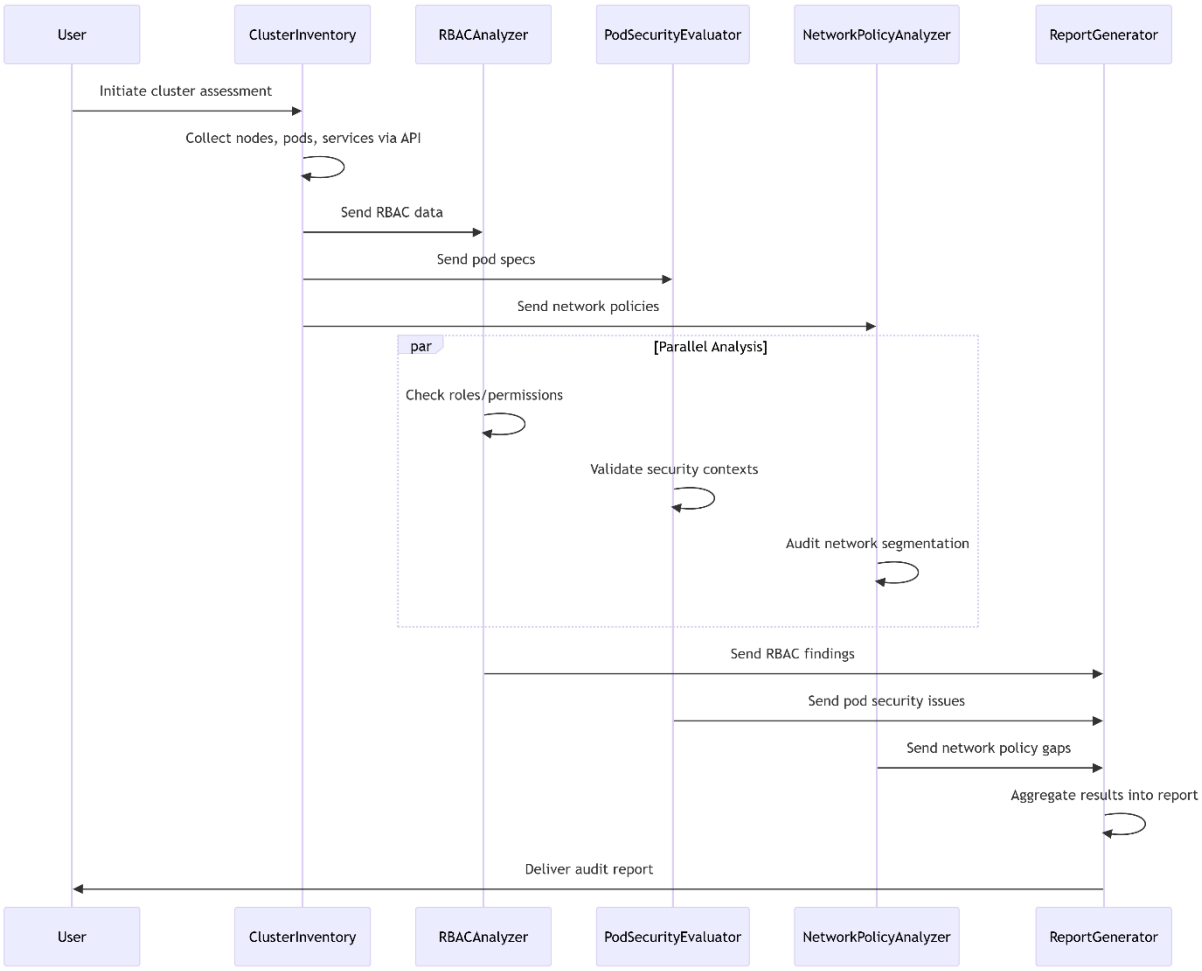
- **How it works:**
    - Aggregates results into:
        - Configuration risks
        - Identity management risks
        - Network segmentation gaps
    - Suggests best practice remediations (e.g., implement deny-all network policy).

Component Diagram



- The **Cluster Inventory Collector** gathers cluster metadata and distributes it to three analyzers: **RBAC Analyzer**, **Pod Security Evaluator**, and **Network Policy Analyzer**.
- All analyzers send their findings to the **Report Generator**, which consolidates the results into a structured report.

Sequence Diagram



- The **User** triggers the assessment, prompting the **Cluster Inventory Collector** to fetch data from the Kubernetes API.
- The inventory data is sent to the analyzers for parallel processing:
    - **RBAC Analyzer** checks role-based access controls.
    - **Pod Security Evaluator** reviews pod configurations.
    - **Network Policy Analyzer** audits network segmentation rules.
- Results are aggregated by the **Report Generator**, which produces a prioritized security report for the **User**.

Detailed Project Description: Kubernetes Cluster Security Assessment

A tool for auditing Kubernetes clusters by analyzing configurations, access controls, pod security, and network policies. The tool generates actionable reports to improve cluster security posture.

---

1. System Overview

The tool assesses Kubernetes clusters for common security risks, including:

- **RBAC Misconfigurations**: Overly permissive roles and service accounts.
- **Pod Security Gaps**: Containers running as root, privileged mode.
- **Network Segmentation Issues**: Lack of network policies or open pod-to-pod traffic.
- **Etc**.

---

2. Component Design & Implementation

2.1 Cluster Inventory Collector

Functionality:

- Gathers cluster metadata (nodes, pods, services) via the Kubernetes API.

Implementation Steps (e.g.):

1. **Authentication**:
    - Use `kubeconfig` files or in-cluster service accounts for API access.
2. **Data Collection**:
    - Use Kubernetes client libraries (`kubectl`, `client-go` for Go, `kubernetes` for Python) to fetch:
        - Nodes: `kubectl get nodes -o json`
        - Pods: `kubectl get pods --all-namespaces -o json`

        - Network Policies: `kubectl get networkpolicies -o json`
3. **Data Structuring**:
    - Store collected data in JSON format for downstream analysis.

Output:

- Structured inventory of cluster resources.

2.2 RBAC Analyzer

Functionality:

- Identifies overly permissive roles and role bindings.

Implementation Steps (e.g.):

1. **Data Parsing**:
    - Extract Roles, ClusterRoles, RoleBindings, and ServiceAccounts from inventory.
2. **Rule Engine**:
    - Define YAML rules for risky configurations
    - Example
      ```yaml
      rules:
        - id: "WILDCARD_VERBS"
          description: "Role allows wildcard verbs (*)"
          condition: "verbs: '*'"
          severity: "High"
      ```
    - Check for:
        - Wildcards (`resources: "*"`, `verbs: "*"`),
        - Cluster-admin roles assigned to non-system accounts,
        - Roles granted across namespaces.
3. **Tools**:
    - Reference open-source tools like **kubectl-who-can** or **rbac-lookup**.

Output:

- List of RBAC violations with severity levels.

2.3 Pod Security Evaluator

Functionality:

- Audits pods for insecure configurations.

Implementation Steps (e.g.):

1. **Pod Spec Analysis**:
    - Parse `securityContext` and `pod.spec` fields.
2. **Security Checks**:
    - Flag pods with:
        - `runAsUser: 0` (root execution),
        - `privileged: true`,
        - Missing `securityContext` or `seccompProfile`,
        - Use of `hostNetwork` or `hostPID`.
3. **Benchmarking**:
    - Align checks with **Kubernetes Pod Security Standards** (Baseline/Restricted).

Output:

- List of non-compliant pods and recommended fixes.

2.4 Network Policy Analyzer

Functionality:

- Evaluates network segmentation and policy enforcement.

Implementation Steps (e.g.):

1. **Policy Extraction**:

    - Fetch existing NetworkPolicy objects.
2. **Rule Checks**:
    - Verify if namespaces have a default "deny-all" policy.
    - Identify pods exposed to all traffic (`ingress: {}`).
    - Check for overly permissive rules (e.g., `cidr: 0.0.0.0/0`).
3. **Tools**:
    - Use **Cilium** or **Calico** for policy visualization.

Output:

- List of network exposure risks and policy gaps.

2.5 Report Generator

Functionality:

- Consolidates findings into prioritized reports.

Implementation Steps (e.g.):

1. **Data Aggregation**:
    - Merge results from RBAC, Pod Security, and Network Policy analyzers.
2. **Report Templates**:
    - **HTML**: Use Jinja2 for interactive tables and filters.
    - **PDF**: Generate via `WeasyPrint` or `pandoc`.
    - **SARIF**: For CI/CD integration (e.g., GitHub Advanced Security).
    - **Etc**.
3. **Prioritization**:
    - Rank findings by severity (Critical > High > Medium > Low).
    - Include remediation steps (e.g., "Restrict role verbs to specific actions").

Output:

- Final report with:

    - RBAC violations,
    - Pod security misconfigurations,
    - Network policy gaps,
    - Step-by-step remediation guidance.

---

3. Technology Stack (e.g.)

- **Kubernetes Interaction**: `kubectl`, `client-go`, `kubernetes` Python library, etc.
- **RBAC Analysis**: Custom rule engine, `rbac-police, etc`.
- **Pod Security**: Open Policy Agent (OPA) for policy checks, etc.
- **Reporting**: Jinja2, WeasyPrint, SARIF SDK, etc.

---

4. Evaluation & Validation

1. **Test Clusters**:
    - Deploy clusters with intentional misconfigurations (e.g., privileged pods, wildcard roles, etc).
2. **Accuracy**:
    - Compare results against manual audits or tools like **kube-bench**.
3. **Performance**:
    - Measure scan time for clusters with 1000+ pods.

---

5. Development Roadmap

1. **Phase 1**: Build Cluster Inventory Collector and RBAC Analyzer.
2. **Phase 2**: Implement Pod Security and Network Policy analyzers.
3. **Phase 3**: Develop reporting engine with SARIF/HTML support.
4. **Phase 4 (optional)**: Validate on real-world clusters (e.g., EKS, GKE).

---

6. Challenges & Mitigations (optional)

- **Scale**: Optimize API calls with pagination and caching.
- **Complex Policies**: Use OPA or Kyverno for advanced policy validation.
- **False Positives**: Refine rules using community benchmarks like CIS Kubernetes.

---

7. Glossary

- **RBAC**: Role-Based Access Control
- **CIDR**: Classless Inter-Domain Routing
- **SARIF**: Static Analysis Results Interchange Format
- **OPA**: Open Policy Agent
