WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal Homepage: www.ijrpr.com ISSN 2582-7421

KubeShield: A Kubernetes Attack Path Visualiser Using Graph-Based Security Analysis

Ramesh Manjappa Yatnalli, Rihan Shahnawaz Solkar, Tharun Gowda K, VinayKumar E, Mrs. Supriya M Kerakkanavar*

B.E. Student, Department of Computer Science and Engineering, GM Institute of Technology, Visvesvaraya Technological University (VTU), Davanagere, Karnataka, India

*****Under the Guidance of Assistant Professor, Department of Computer Science and Engineering, G M Institute of Technology, Davanagere, Karnataka, India

Email: *edigaravinaykumar@gmail.com*

ABSTRACT:

Kubernetes clusters contain many interconnected resources, including pods, service accounts, secrets, roles, role bindings and network policies, and their misconfigurations are often reported as isolated alerts. This paper presents KubeShield, a prototype Kubernetes attack path visualiser that models security-relevant relationships between cluster resources as a directed graph and applies graph algorithms to examine how individual weaknesses may combine into multi-step attack paths. A local scan agent inspects an approved Kubernetes context through the Kubernetes Python client; a graph extractor builds a JSON security graph; and a FastAPI backend validates the graph and stores it as a snapshot in MongoDB. A graph analysis engine applies Dijkstra's algorithm for weighted shortest-path analysis, Breadth-First Search (BFS) for blast-radius analysis, Depth-First Search (DFS) for directed-cycle detection, and critical-node analysis, and the results are explored through a React dashboard built on Cytoscape.js. On a mock cluster graph of 41 nodes and 48 edges, the prototype reported 18 attack paths (11 classified as critical), 4 CVE mappings and one directed cycle, and its critical-node analysis indicated that removing a single over-privileged node (web-frontend) would eliminate about 74% of the modelled paths. These outputs are structural indicators that help prioritise review; they are not proof that a path is exploitable.

Keywords: Kubernetes security; attack path analysis; attack graph; Dijkstra's algorithm; blast radius; RBAC; critical-node analysis; graph visualisation

Introduction:

Kubernetes is an open-source container orchestration platform that automates the deployment, scaling, networking and management of containerised applications. Its distributed design supports microservice architectures, but it also creates a large and interconnected attack surface. A cluster is made up of pods, nodes, namespaces, service accounts, secrets, roles, role bindings and network policies, and the security of one component can influence the security of others. Excessive privileges, insecure workload settings, exposed credentials and weak network restrictions can open routes to unauthorised access, privilege escalation and lateral movement.

In practice these weaknesses rarely occur alone. An attacker who gains a foothold in one workload may obtain a service account token, use over-permissive Role-Based Access Control (RBAC) bindings to reach further resources, and move towards more privileged components. Such a chain of connected resources and weaknesses forms an attack path. Security tools that report findings only as separate alerts make it hard to see how one weakness leads to another, and therefore hard to decide which fix removes the most risk.

KubeShield is proposed as a web-based decision-support platform that makes these relationships visible. It represents Kubernetes resources as nodes and security-relevant relationships or potential attacker capabilities as directed edges, attaches findings, risk metadata, source/sink indicators and mitigation guidance to the relevant resources, and analyses the resulting graph with Dijkstra's algorithm, BFS, DFS and critical-node analysis. Its purpose is not to replace penetration testing or cluster hardening reviews; it organises evidence and helps administrators prioritise which resources and configurations to examine first.

Contributions of this work are: (i) a local scan agent and graph-extraction workflow that converts the configuration of an approved cluster into a validated JSON security graph; (ii) a graph model in which findings, risk metadata, source/sink indicators and mitigation information are attached to resource nodes; (iii) an analysis engine for weighted shortest paths, blast radius, directed cycles and critical nodes; (iv) an interactive React dashboard for exploring the results; and (v) persistent graph snapshots and a test plan that separates functional correctness from claims about exploitability.



Fig. 1. Overview of Kubernetes architecture and a multi-step security attack

Problem Statement:

Kubernetes environments contain many resources, permissions, and connections, making it difficult to identify potential security risks. Excessive permissions and incorrect configurations may create attack paths that allow an attacker to move between resources. Existing security information can be difficult to understand when presented without a clear visual representation. Therefore, there is a need for a system that can identify, analyse, and visualise potential attack paths in a Kubernetes cluster. The proposed system, KubeShield, addresses this problem by using graph-based analysis to show resource relationships, detect risky permission paths, and help users understand possible security threats.

Objectives:

1. To Inspect Kubernetes Cluster Resources and Identify Security Risks.
2. To Construct a Graph Representation of Kubernetes Security Relationships.
3. To Develop a Kubernetes Scanning and Graph Extraction Workflow.
4. To Apply Graph Algorithms for Attack-Path and Security Analysis.
5. To Develop an Interactive Web-Based Kubernetes Security Dashboard.
6. To Provide Persistent and Reusable Security Analysis Results.
7. To Support Security Investigation and Remediation Prioritisation.

Scope of the Project:

The project focuses on identifying and analysing potential security risks in Kubernetes clusters. It collects cluster resources and represents their relationships using a directed graph. Graph algorithms are used to identify potential attack paths, analyse blast radius, detect permission cycles, and identify critical nodes. An interactive dashboard helps users visualise security findings and understand possible risks. The system also provides mitigation recommendations to help prioritise security improvements.

Literature Review

Kubernetes misconfiguration and hardening. Shamim, Bhuiyan and Rahman catalogue common misconfigurations in real deployments, including weak RBAC policies, missing access controls and insecure defaults, and condense them into ten security rules with an assessment of severity and impact. The NSA and CISA Kubernetes Hardening Guidance takes a defence-in-depth view that covers pod security, network policies, authentication, authorisation and audit logging, and stresses that each layer must be secured independently to limit lateral movement. These works define the kinds of weakness that KubeShield represents on its graph nodes, but they are written as checklists rather than as models of how weaknesses connect.

Privilege escalation and threat models. Pecka et al. show how permissions and over-privileged service accounts can be chained step by step into full cluster control, and their scenarios resemble the escalation stages that KubeShield models as edges. The Microsoft Threat Matrix for Kubernetes organises attacks into stages such as initial access, execution, persistence, privilege escalation, credential access, lateral movement and impact, following the

structure of MITRE ATT&CK; the MITRE ATT&CK for Containers matrix provides a comparable reference. These taxonomies guide how nodes and edges are named and sequenced, but a taxonomy alone does not compute which chains exist in a specific cluster.

Evidence of prevalence. Industry reports from Red Hat (State of Kubernetes Security) and Fairwinds (Kubernetes Benchmark Report) indicate that misconfigurations, vulnerable images and insufficient access controls continue to cause security incidents, and they list frequent problems such as containers running as root, missing resource limits and absent network policies. They suggest that the weaknesses KubeShield models are present in real environments and that limited visibility into how they combine remains a common difficulty. Vulnerability identifiers are drawn from the NIST National Vulnerability Database (NVD).

Graph algorithms. Dijkstra's algorithm, Breadth-First Search and Depth-First Search are standard graph techniques described in textbooks such as Cormen et al. Dijkstra's algorithm finds a minimum-weight path between two nodes, BFS explores nodes level by level and therefore supports hop-limited reachability, and DFS can detect directed cycles. Their output depends entirely on the graph that is supplied and on the weights assigned to edges: a minimum-weight path is not automatically the highest-risk path, reachability is not proof of compromise, and a cycle is a structural property rather than a confirmed vulnerability.

Research gap. The reviewed work tends to concentrate either on identifying individual misconfigurations or on describing attack stages in general terms. KubeShield explores combining live cluster inspection, graph construction, several complementary graph analyses and an interactive dashboard in one workflow. The proposed contribution is this integration and the traceable, reviewable presentation of results, not a claim that any one algorithm independently guarantees detection of every exploitable path.

Significance of the Study

KubeShield is significant because it presents the information needed to reason about Kubernetes risk in one place. Graph construction shows how resources depend on each other; shortest-path analysis gives a concrete example route between a chosen source and target; blast-radius analysis shows what lies within a chosen number of hops of a compromised resource; cycle detection highlights circular relationships in permissions that deserve review; and critical-node analysis points to resources whose removal would cut many modelled paths. Stored snapshots allow earlier scans to be re-examined or compared without scanning the cluster again, and the dashboard can serve both auditing and teaching.

The expected benefits depend on implementation quality and on how results are used. The scan agent must be run only against approved contexts with appropriately limited credentials, the backend must authenticate users and validate every submitted graph, and the weights and rules used to build the graph must be documented. The analysis is a model of potential attacker movement; it does not simulate every real-world exploit, and human reviewers remain responsible for deciding which findings are exploitable and how to remediate them.

Methodology

System architecture. KubeShield has five main components: a React frontend, a FastAPI backend, a MongoDB database, a local Kubernetes scan agent and a graph analysis engine (Fig. 2). The cluster is the source environment; the scan agent inspects it and the graph extractor converts what it finds into a structured graph; the backend validates and stores the graph; and the frontend lets the user investigate it. The overall sequence is: Kubernetes cluster → local scan agent → graph extraction → backend validation and storage → graph analysis → interactive visualisation.

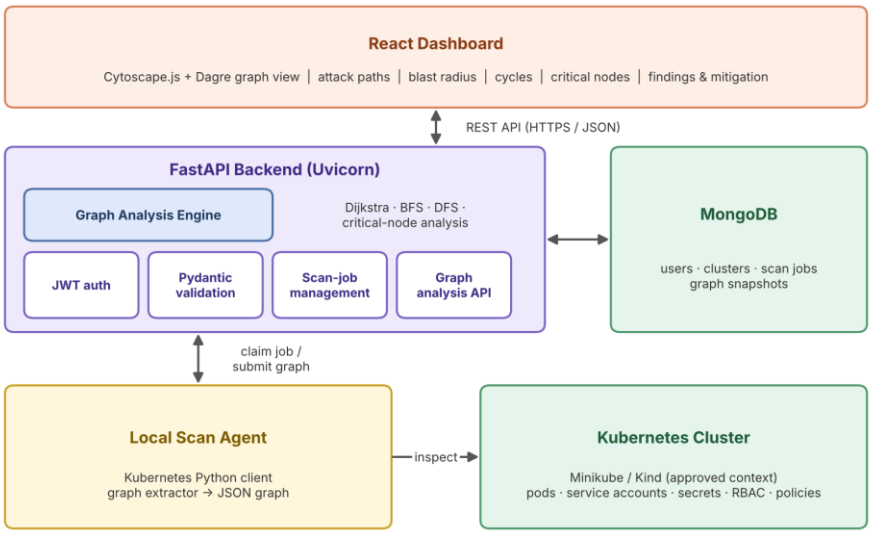


Fig. 2. KubeShield system architecture

Resource inspection and findings. The scan agent claims a scan job, validates the approved Kubernetes context and uses the Kubernetes Python client to read pods and workloads, namespaces, service accounts, secrets, roles and cluster roles, role bindings and network-policy-related configuration. It evaluates these resources for conditions such as excessive permissions, insecure workload settings, exposed credentials and weak isolation. Each condition is recorded as a finding attached to the affected resource, so that the Kubernetes resource remains the graph node and the finding supplies additional risk and investigation metadata.

Graph model and snapshots. The extracted graph is represented as JSON. Nodes are Kubernetes resources or security entities; directed edges are security-relevant relationships such as access, permission, ownership, communication or potential attacker movement; findings, risk metadata, source and sink indicators and mitigation information are attached to the relevant resources. The backend validates the payload with Pydantic models before it is accepted and then stores it as a graph snapshot in MongoDB, which allows later analyses to reuse the stored graph instead of rescanning the cluster.

Graph analysis engine. Four analyses are implemented. (a) Dijkstra's algorithm computes a minimum-weight path between a selected source and target using the weights assigned to edges. (b) BFS returns the nodes reachable from a selected node within a chosen hop depth, giving a blast-radius view grouped by distance. (c) DFS finds directed cycles, which are reported as structural features for further review. (d) Critical-node analysis removes candidate nodes and measures the change in modelled source-to-sink paths and connectivity, then ranks the nodes by effect. Graph statistics such as node and edge counts, connectivity, sources, sinks and findings are also produced for the dashboard. The interpretation limits noted in the Literature Review apply to every result: the shortest path is the lowest-weight path rather than automatically the riskiest, reachability does not confirm compromise, and a cycle or critical node indicates structure rather than proven exploitability.

Backend, agent and frontend. The backend is written in Python with FastAPI and Uvicorn, uses JWT-based token authentication with hashed passwords, and exposes endpoints for authentication, cluster registration, scan-job management, graph submission and retrieval, analysis requests, and findings and mitigation retrieval. The scan agent runs as a separate process from the API server, so that cluster access is isolated from the web application; if a scan fails, the agent reports the failure so that the job can be tracked. The frontend uses React 18, Vite and Tailwind CSS, with Cytoscape.js and Dagre layout for the interactive graph, and provides views for the graph, attack paths, blast radius, cycles, critical nodes, findings and mitigation, and scan and snapshot management.

Testing approach. Testing covers the scan-and-extraction workflow against a local cluster (Minikube or Kind); validation and persistence of graph payloads; the analysis algorithms against graph fixtures; the API endpoints; the dashboard views; and the complete workflow from scan to visualisation. Pytest and example JSON graphs are used for controlled, repeatable checks of the algorithms.

Algorithm:

Algorithm 1: Graph-based attack path analysis of a Kubernetes cluster

Input: approved Kubernetes context, authenticated user, selected source and target resources, traversal depth.

1. Authenticate the user and register the cluster/context to be scanned; create a scan job in the backend.
2. The local scan agent claims the job and validates that the approved Kubernetes context is the intended cluster.
3. Inspect the supported resources (pods, service accounts, secrets, roles, role bindings, network-policy configuration) with the Kubernetes Python client and record security findings.
4. Build the graph: create nodes for resources, directed weighted edges for security-relevant relationships, and attach findings, risk metadata, source/sink indicators and mitigation information.
5. Submit the JSON graph to the backend; validate it against the Pydantic model and reject malformed payloads; store the accepted graph as a snapshot in MongoDB.
6. Run Dijkstra's algorithm between the selected source and target to obtain the minimum-weight path and its intermediate resources.
7. Run BFS from the selected node up to the chosen hop depth to obtain the blast-radius set grouped by distance.
8. Run DFS over the directed graph to report any directed cycles.
9. Run critical-node analysis: remove candidate nodes, recompute the modelled source-to-sink paths and rank nodes by the reduction obtained.
10. Return the results to the dashboard with the associated findings and mitigation guidance for review by an authorised user; do not treat any result as proof of exploitability.

Output: stored graph snapshot, attack-path, blast-radius, cycle and critical-node results, graph statistics, and linked findings with mitigation recommendations.

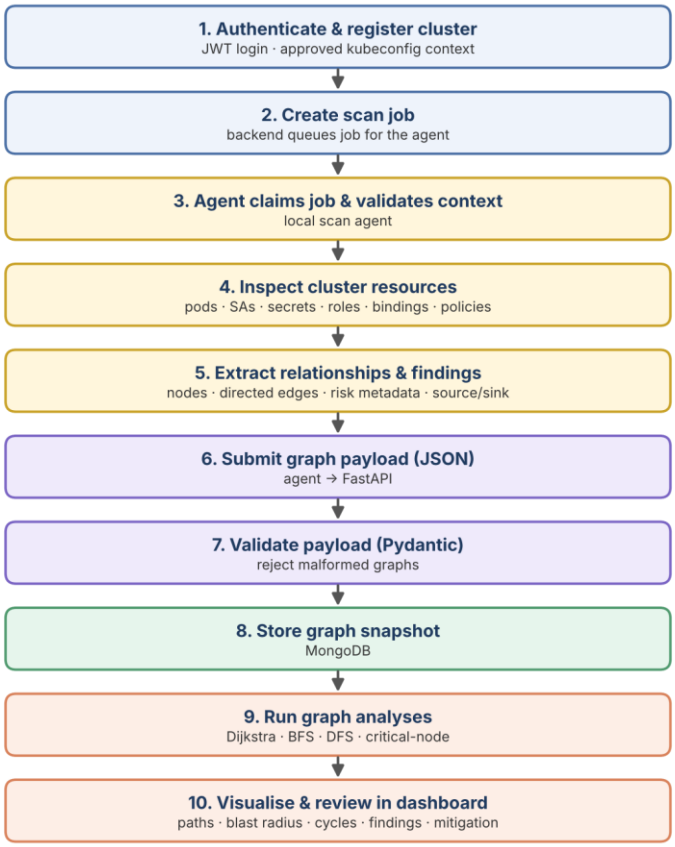


Fig. 3. End-to-end KubeShield workflow from scan job to dashboard analysis

Results and Discussions

Evaluation design. A reproducible evaluation should treat scan extraction, graph validation and persistence, algorithm correctness, API behaviour and dashboard behaviour as separate test areas, each with a documented test cluster or graph fixture. The values below are outputs of the prototype on the project's own test data. They describe what the analysis reported, and they are not measurements of detection accuracy, because no independently labelled ground truth of exploitable paths was used.

Table 1. Outputs reported by the KubeShield prototype

Item	Mock cluster graph (cluster-graph.json)	Dashboard scan snapshot (Fig. 4)
Graph nodes	41	68
Graph edges	48	51
Attack paths identified	18 (11 classified as critical)	Not reported
CVE mappings	4	Not reported
Security findings	Not reported	48 (3 critical)
Directed cycles (DFS)	1 (between svc-a and svc-b)	Not reported
Blast radius example (BFS)	13 nodes within 3 hops	Not reported
Critical node (removal effect)	web-frontend; about 74% of modelled paths eliminated	Not reported

Analysis of the outputs. On the mock cluster graph, the analysis reported 18 attack paths, of which 11 were classified as critical, together with 4 CVE mappings and one directed cycle. Critical-node analysis indicated that a single over-privileged node, web-frontend, accounted for most of the modelled

paths: removing it would eliminate about 74% of them. This illustrates why path-aware analysis is useful for prioritisation, because fixing one well-connected resource can matter more than fixing several isolated findings. The figure is specific to the mock graph and to the edge weights and source/sink definitions used; it should not be generalised to other clusters.

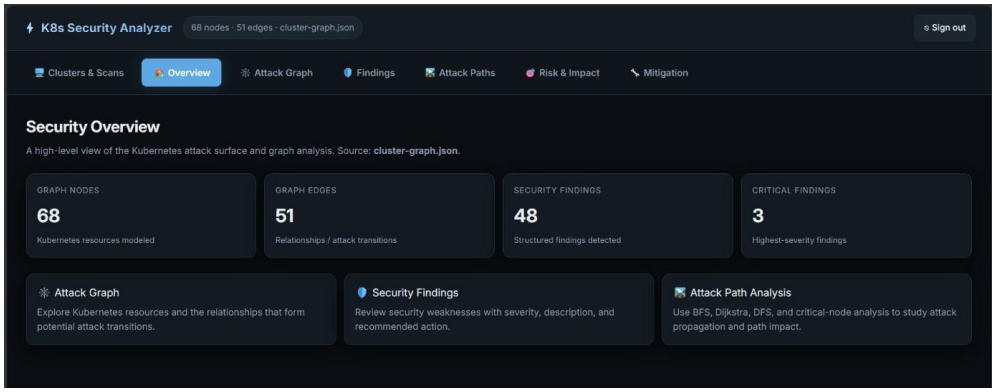


Fig. 4. Security overview: graph size, findings and critical findings for a scanned snapshot

Interactive graph and attack paths. The graph view (Fig. 5) shows resources as nodes and relationships as directed edges, with colour coding by node type and severity, and lets the user select nodes and inspect their findings. The attack-path view (Fig. 6) shows the minimum-weight route between a chosen source and target as a sequence of resources with the findings attached to each step, so that a reviewer can examine intermediate resources that may deserve attention.

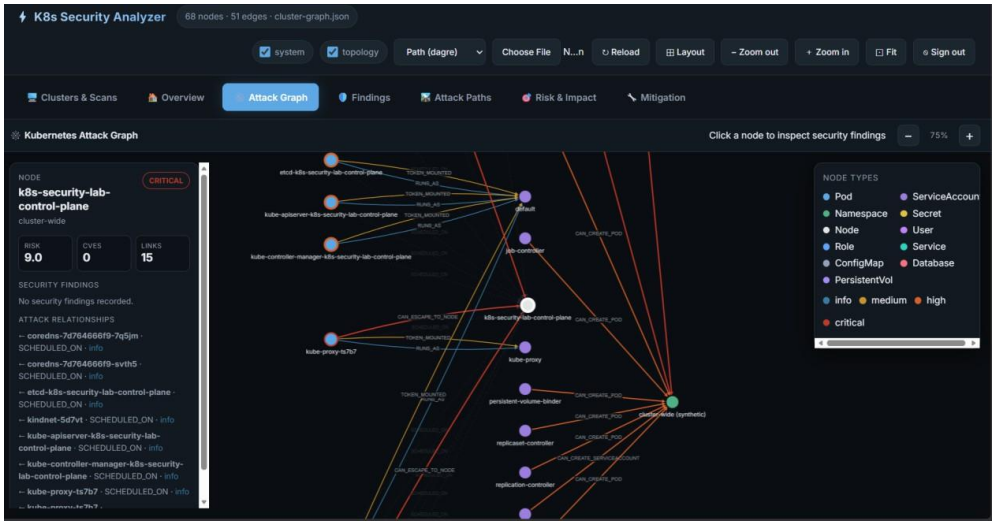


Fig. 5. Interactive attack graph view with node details and legend

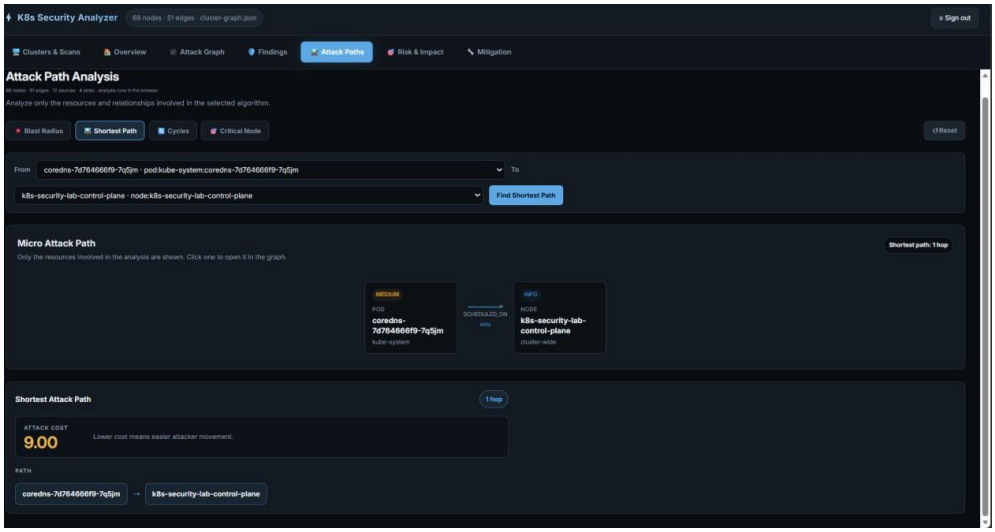


Fig. 6. Attack path analysis (weighted shortest path) between selected resources

Blast radius and critical nodes. The blast-radius view (Fig. 7) lists resources reachable from a selected node within the requested hop depth, grouped by distance, which separates directly connected resources from those reachable only through several relationships. The critical-node view (Fig. 8) ranks resources by the effect of their removal on modelled attack paths and connectivity. Both views report graph reachability and structure; neither confirms that every reachable or highly ranked resource can actually be compromised.

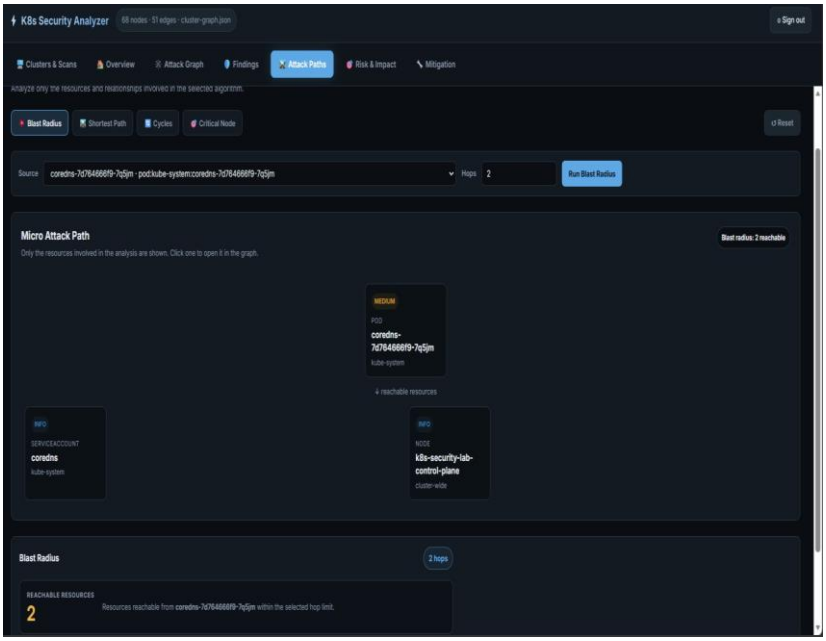


Fig. 7. Blast-radius analysis (BFS) from a selected node

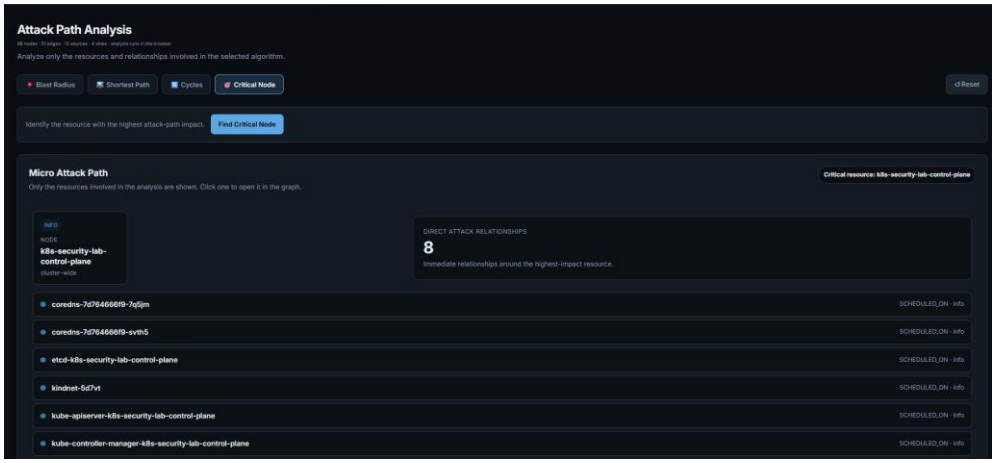


Fig. 8. Critical-node identification

Findings and mitigation. The findings view (Fig. 9) lists detected issues with severity, the affected resource and a description, for example excessive permissions, privileged workloads and exposed secrets, and the mitigation view (Fig. 10) pairs findings with recommended corrective actions such as reducing excessive permissions, tightening workload security settings, restricting unnecessary exposure and strengthening access control. Linking findings to graph elements lets the user move from a graph-level observation to the specific configuration that needs attention.

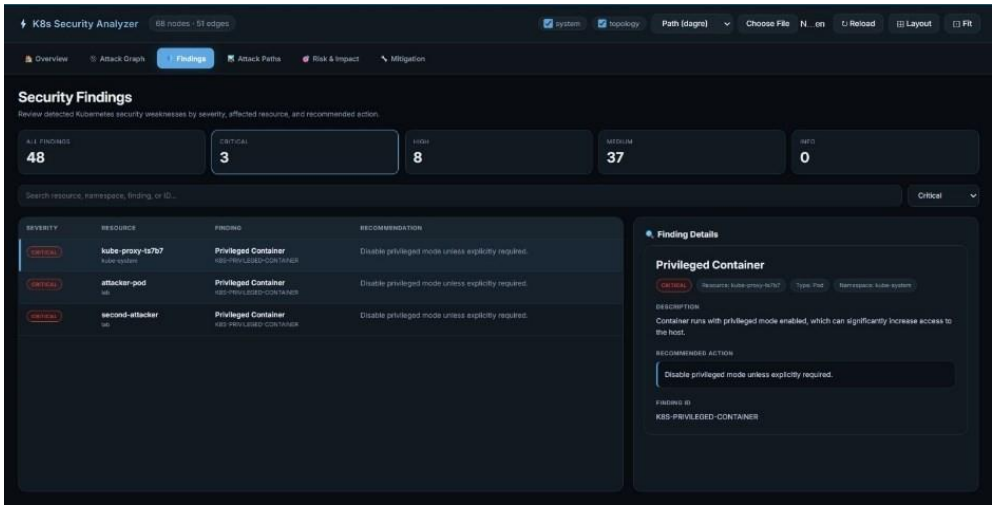


Fig. 9. Security findings with severity and details

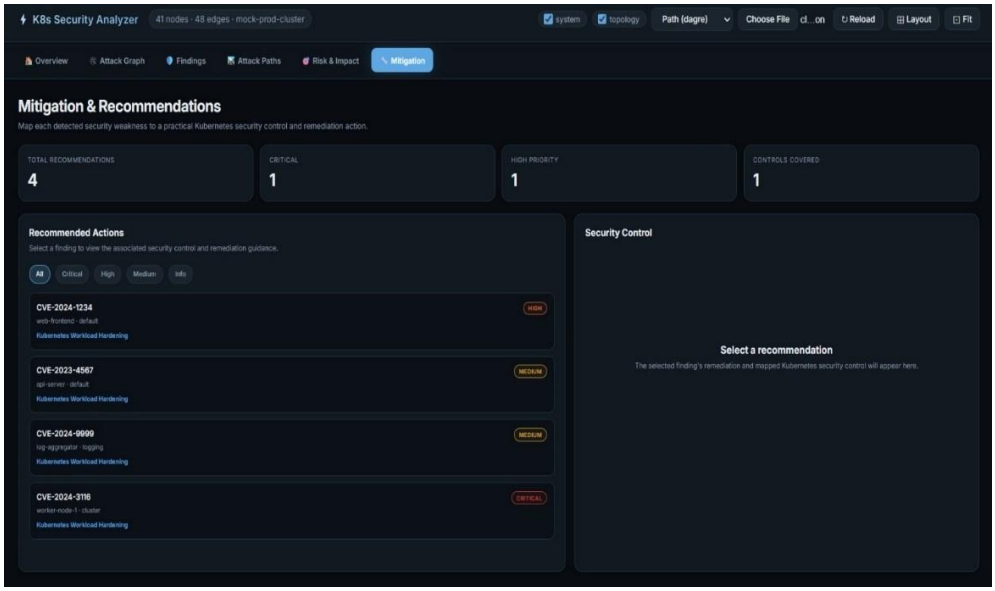


Fig. 10. Mitigation and recommendations view

Discussion and limitations. The combined design improves visibility of how weaknesses connect, but its components have different limitations. The graph reflects only the resources and relationships the scan agent can extract and the weights the project assigns, so missing relationships or poorly chosen weights change the paths that are found. A minimum-weight path is not necessarily the most dangerous path; blast radius shows reachability rather than confirmed impact; a cycle is not automatically an exploitable escalation loop; and critical-node rankings measure structural importance rather than vulnerability severity. These limitations should remain visible in the dashboard and documentation.

Conclusion

This paper presented KubeShield, a prototype that converts the configuration of a Kubernetes cluster into a directed security graph and analyses it with Dijkstra's algorithm, Breadth-First Search, Depth-First Search and critical-node analysis. A local scan agent, a FastAPI backend with MongoDB snapshot storage and a React dashboard make the results available for interactive investigation of attack paths, blast radius, cycles, critical resources, findings and mitigation guidance. The main value of the design lies in showing how individual weaknesses combine and in helping administrators decide which resources to examine first. The results are indicators for human review rather than proof of exploitability, and a defensible evaluation requires repeatable tests on documented clusters, independent ground truth where available, and transparent reporting of limitations.

Applications and Future Work

Applications include security auditing of Kubernetes clusters, detection of multi-step attack chains in cloud-native environments, security training and education, integration into DevSecOps pipelines so that analysis runs when configurations change, and risk reporting for compliance reviews. Future work includes continuous monitoring and scheduled re-scanning with comparison of snapshots over time; contextual risk scoring that may use machine-

learning models once suitable labelled data exists; mapping each attack step to the MITRE ATT&CK for Containers framework; multi-cluster and multi-cloud analysis across services such as Amazon EKS, Google GKE and Azure AKS; real-time alerts through email, Slack or PagerDuty when new high-risk paths appear; and independent evaluation on larger, labelled clusters together with usability and security testing with representative users.

References:

Research Papers:

1. I. S. Shamim, F. A. Bhuiyan, and A. Rahman, "Ten Security Commandments for Kubernetes," in *IEEE Secure Development Conference*, 2020.
2. National Security Agency (NSA) and Cybersecurity and Infrastructure Security Agency (CISA), "Kubernetes Hardening Guidance," Technical Report, 2022.
3. J. Pecka et al., "Privilege Escalation in Kubernetes," *Central European Cybersecurity Conference (CECC)*, 2022.
4. Microsoft, "Threat Matrix for Kubernetes," *Microsoft Security Blog*, Redmond, WA, USA, Apr. 2021.
5. Red Hat Inc., "State of Kubernetes Security Report," Technical Report, Red Hat, Raleigh, NC, USA, 2023.
6. Fairwinds Inc., "Kubernetes Benchmark Report," Technical Report, Fairwinds, Boston, MA, USA, 2024.
7. National Institute of Standards and Technology (NIST), "National Vulnerability Database (NVD)," Gaithersburg, MD, USA, 2023. [Online]. Available: https://nvd.nist.gov
8. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA, USA: MIT Press, 2009.
9. MITRE Corporation, "MITRE ATT&CK for Containers Framework," McLean, VA, USA, 2022. [Online]. Available: https://attack.mitre.org/matrices/enterprise/containers
10. The Kubernetes Authors, "Kubernetes Documentation," The Kubernetes Community, ongoing.