
Securing the Sandbox: Preventing AI Escapes via Kubernetes RBAC Static Analysis

Gaurav Ranganath
gaurav.ranganath@gmail.com

Gabriel Paulos
gabrielp7700@gmail.com

Jon Paribello
jonparibello@gmail.com

Manvir Dehal
manvirdehal@gmail.com

With
Apart Research

Abstract

An OpenAI model escaped its test environment and reached Hugging Face’s production infrastructure. A single credential, shared across multiple clusters and granted full administrative access, allowed the breach to extend beyond one cluster to several. We built a tool that statically scans a lab’s cluster configuration files for this pattern and three related ones, without access to the live system. We tested the tool on three example configurations. It correctly flagged the vulnerable one, correctly passed the fixed one, and correctly flagged a third case, where the same credential was used in both a testing environment and a production environment. The tool checks configuration files only; it does not detect live-system state or address other factors in the incident. It offers a low-cost, adoptable check for one specific documented failure mode.

1. Introduction

During an internal cyber-capability evaluation, an OpenAI model escaped its evaluation sandbox and gained access to Hugging Face’s production infrastructure. The incident was confirmed by both companies in public postmortems. [1]

A step that led to cluster-wide access was a Hugging Face service connector (one credential, shared across multiple clusters), bound to the Kubernetes’ system:masters role, the highest tier of cluster-admin privileges, compromising every cluster it could reach. [2]

This is one control failure among several in the incident, but it is a piece that a third party can verify by reading declared configuration files, without touching the target’s live network. Our project statically checks configuration files like this, and returns whether a master-key is reused across multiple clusters.

Contributions

1. 4 rules (R1-R4) mapped to real security controls, checking different ways connector credentials can be overprivileged.
2. Three example folders with a passing and failing case for each rule. A “before” folder that fails, an “after” folder that passes, and an additional folder that isolates the eval vs prod rule (R4) on its own.

2. Related Work

stack-rox/kube-linter (Most Similar)

KubeLinter is a static analysis tool that checks Kubernetes YAML files and Helm charts against best practices, including least-privilege enforcement. [3] However, it lints manifests within a single scope and does not include a built-in mechanism for correlating the same identity’s bindings across separate clusters or trusted domains.

Our Submission: [rangan39/connector-scoping-check](#)

rangan39/connector-scoping-check is a prototype for a static CI-gate that fails K8s RBAC manifests granting connector identities cluster-admin or reuse across clusters/trust domains.

The Gap: Existing static linters check best practices within a single manifest scope, but do not correlate an identity’s bindings across clusters or environments. Our tool is also quick to adopt (forkable, MIT-licensed) and reproducible (before/after fixtures), lowering the cost and complexity for a lab to add it (or something similar) to their CI.

3. Methods

Rules

Each rule corresponds to a specific over-privilege or credential-reuse pattern, mapped to its corresponding CSA AICM control [4]:

Rule	What It Checks	Why It Matters	Control
R1	The binding grants cluster-admin, admin, or system:masters directly.	The most direct form of over-privilege. This identity can do anything, anywhere.	IAM-05 (Least Privilege)
R2	The same identity appears bound in more than one cluster.	One stolen credential can become cluster-admin on every cluster it touches.	IAM-09 (Segregation of Privileged Access Roles)
R3	The binding itself is cluster-wide (a ClusterRoleBinding, or points at a ClusterRole) checked independently of R1.	Catches over-broad scope when the granted permissions look modest; a narrowly-permissioned ClusterRole is still reachable everywhere.	IAM-05 (Least Privilege)
R4	The same identity appears bound in more than one environment (e.g. eval and prod).	The AI-specific version of R2: a credential valid in a sandboxed eval environment must not also work in production.	I&S-05 (Production and Non-Production Environments)

Scope

The tool looks at five kinds of Kubernetes objects: ServiceAccounts, Roles, ClusterRoles, RoleBindings, and ClusterRoleBindings. It can also read YAML files that bundle several of these together in one file, separated by "---". To decide whether a given identity counts as a "connector", it checks whether the binding is a RoleBinding or ClusterRoleBinding labeled `app.kubernetes.io/component: connector`.

Code Layout

The implementation is organized into three files, each with a single responsibility so the rule logic in `rules.go` can be tested independently of how manifests are found or parsed.

File	Description
<code>main.go</code>	CLI entry point: parses arguments, calls the file-finding and rule-checking logic, and prints the final PASS/FAIL result.
<code>manifest.go</code>	Generic YAML plumbing that finds <code>.yaml/.yml</code> files and parses each document into a plain map, with no knowledge of

	RBAC-specific concepts like rules or connectors.
rules.go	The label/annotation conventions plus R1–R4 themselves, the checks that decide whether a binding is a violation.

Fixtures

Three fixture directories exercise the checker: one reproduces the vulnerable incident pattern, one shows the fixed design passing cleanly, and one isolates the violation type the other three rules do not detect.

Directory	Description
manifests/before/	The vulnerable fixture: one connector identity bound via a ClusterRoleBinding to cluster-admin and fanned out across three clusters, reproducing the incident's credential-sharing pattern (triggers R1, R2, R3).
manifests/after/	The fixed fixture: one narrowly-scoped Role/RoleBinding pair per cluster plus a separate eval-domain identity, showing the checker reports zero violations once each boundary has its own credential.
manifests/before-eval-prod-shared/	The fixture that isolates R4: the same otherwise-compliant identity bound in both the eval and prod domains, proving R4 catches a violation the other three rules can't see.

Github Workflows

A single GitHub Actions workflow runs on every push, building the checker and verifying it produces the expected exit code against all three fixtures.

File	Description
.github/workflows/test.yml	Builds the binary and runs it against all three fixtures on every push, asserting each one's expected exit code.

4. Results

We ran the checker against all three fixtures described above; the table reports the expected outcome for each and the tool's actual output.

Fixture	Directory	Expected	Shell Result
---------	-----------	----------	--------------

F1	manifests/before	FAIL	FAIL — 3 violation(s) found: - [R1 IAM-05] manifests/before/connector.yaml: connector binding 'internal-service-connector-binding' grants cluster-admin-equivalent role 'cluster-admin' — use least-privilege, namespace-scoped access instead - [R3 IAM-05] manifests/before/connector.yaml: connector binding 'internal-service-connector-binding' is a ClusterRoleBinding — use a namespaced RoleBinding instead - [R2 IAM-09] manifests/before/connector.yaml: identity 'internal-service-connector' is bound across multiple clusters: cluster-a, cluster-b, cluster-c EXIT=1 ✓ matches expected
F2	manifests/after	PASS	PASS — no connector over-privilege or credential-reuse patterns found. EXIT=0 ✓ matches expected
F3	manifests/before-eval-prod-shared	FAIL	FAIL — 1 violation(s) found: - [R4 I&S-05] manifests/before-eval-prod-shared/shared-connector.yaml: identity 'eval-connector' is bound across multiple trust domains: eval, prod EXIT=1 ✓ matches expected

All three fixtures produced the expected result: the checker flagged the vulnerable configuration on three separate rules, passed the fixed configuration with no findings, and isolated the eval/prod credential-sharing violation that the other rules do not detect. This confirms the checker behaves as designed across the incident pattern, its remediation, and the one failure mode not covered by R1–R3.

5. Discussion and Limitations

The results indicate that at least one containment relevant property, credential scoping across clusters, is verifiable from declared configuration alone, without access to a labs’ live systems. Other containment controls that are also declared rather than observed, such as network egress allowlists or environment separation policies, may be checkable by the same approach.

Limitations

Static only. The checker evaluates declared manifests, not a cluster’s live RBAC state, which can drift from what was last applied.

Convention dependent detection. The tool identifies “connector” identities via a label/annotation convention in rules.go. A connector identity that does not carry this label will not be recognized as a connector, and none of R1-R4 will fire against it. Our fixtures are convention compliant by construction and do not surface this failure mode.

Name-based over-privelage detection. R1 flags bindings to roles names cluster-admin, admin, or system:masters. A custom ClusterRole with equivalent wildcard permissions under a different name would not be flagged, since the check is name-based rather than permission-based.

Validated on synthetic fixtures only. All three fixtures were constructed to match the incident pattern. The checker has not been run against a production Kubernetes repository, so detection accuracy outside of the fixtures is untested.

Future Work

Validate against real manifests. Run the checker against a real Kubernetes repository’s CI pipeline to test whether the labeling convention and rule set hold outside the three fixtures built for this submission.

Move from name-matching to permission analysis. Extend R1 to flag effective over-privelage by resource/verb combination rather than by role name alone.

Canary credentials for real-time alerting. Planting decoy credentials that alert a human when accessed. This is a separate, untested tool from *rangan39/connector-scoping-check*, since it requires live cluster access, which this checker intentionally avoids.

6. Conclusion

This work addresses one control among several implicated in the Hugging Face incident: reuse of a single connector credential across multiple clusters. This category of misconfiguration is visible in a lab’s declared RBAC manifests, which allows it to be checked by a third party without access to the live environment.

[rangan39/connector-scoping-check](https://github.com/rangan39/connector-scoping-check) implements this check as four rules mapped to three CSA AICM controls (IAM-05, IAM-09, I&S-05), with reproducible before/after fixtures. The tool is static only. It does not detect drift between declared and live cluster state, and it does not address the incident’s other contributing factors. Within that scope, it provides a low-cost, forkable starting point for externally verifiable containment checks.

Code and Data

Code Repository: <https://github.com/rangan39/connector-scoping-check/tree/main>

References

Source [1]

OpenAI. (2026, July 21). OpenAI and Hugging Face partner to address security incident during model evaluation. OpenAI.

<https://openai.com/index/hugging-face-model-evaluation-security-incident/>

Source [2]

Hugging Face. (2026, July 27). Anatomy of a Frontier Lab Agent Intrusion: A Technical Timeline of the July 2026 Incident. Hugging Face Blog.

<https://huggingface.co/blog/agent-intrusion-technical-timeline>

Source [3]

stack-rox. (n.d.). kube-linter. GitHub.

<https://github.com/stackrox/kube-linter>

Source [4]

Cloud Security Alliance. (2025). AI Controls Matrix (AICM), v1.0.3. Cloud Security Alliance.

<https://cloudsecurityalliance.org/artifacts/ai-controls-matrix/>

Appendix

Dual Use Considerations

This tool's output identifies which identity is over-privileged and which clusters or environments it reaches. In the hands of a third party who already has read access to a target's manifests, this would identify the highest-value credential to target rather than requiring manual analysis. This differs from an agentic tool that scans a target for missing controls, a concept the team considered and set aside for this reason: this tool does not scan, probe, or gain access to a target, and misuse requires an attacker to already hold read access to the manifests.

LLM Usage Statement

We used Claude to help debug and verify the checker's implementation, draft sections of this report, and independently run the tool's build/test suite against its fixtures to confirm the results reported in Section 4. All code, rules, and claims were reviewed and verified by the authors.