Journal of Information Systems Engineering and Management
2024, 9(2)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Closed-Loop Least Privilege for AWS Multi-Account Platforms and EKS Workloads

Naga Krishna Reddy Muppidi
Independent Researcher

ARTICLEINFO

Received: 01 Mar 2024

Revised: 18 Apr 2024

Accepted: 28 Apr 2024

ABSTRACT

Least privilege in AWS platforms is usually treated as an access-review objective, but multi-account estates and Kubernetes workload identities make it a continuous systems problem. This paper presents a closed-loop least-privilege model for AWS cloud platform teams. The model combines AWS Organizations, service control policies, IAM Access Analyzer unused-access findings and custom policy checks, CloudTrail, AWS Config, infrastructure as code, and Amazon EKS workload identity. The paper defines metrics such as unused permission ratio, entitlement half-life, role reuse risk, pod identity drift, and revocation safety score. It also presents a practical remediation workflow that converts findings into tested pull requests rather than ad hoc ticket queues.

Keywords: AWS, IAM, least privilege, Amazon EKS, platform engineering, multi-account, DevSecOps, zero trust

I. INTRODUCTION

AWS platform teams often inherit an entitlement problem before they inherit a deployment problem. A multi-account landing zone can contain hundreds of human roles, workload roles, automation roles, cross-account break-glass paths, and Kubernetes service-account mappings. Each role appears reasonable when created. Across time, however, permissions accumulate because removal is riskier than addition, teams reuse broad roles to avoid release delays, and EKS workloads blur the line between Kubernetes identity and AWS identity. This paper argues that least privilege for AWS platforms should be treated as a closed-loop platform capability rather than as a one-time security review.

The timing is specific to 2023. AWS introduced IAM Access Analyzer unused-access findings and custom policy checks at re:Invent 2023 , while Amazon EKS Pod Identity provided a new AWS-native mechanism for mapping Kubernetes service accounts to IAM roles . Those launches made it realistic to connect entitlement discovery, policy validation, infrastructure-as-code remediation, and workload identity verification in a single platform workflow. Before 2023, organizations could still do least-privilege work through CloudTrail, Access Advisor, IAM policy simulators, IRSA, and manual reviews. The difference is that unused-access analysis and policy checks can become productized signals inside a platform control plane.

The contribution of this paper is a reference model for closed-loop least privilege in AWS multi-account platforms. It focuses on account-level guardrails, EKS workload identity, role lifecycle management, and evidence generation. The goal is not to remove every unused action immediately. The goal is to create a repeatable system that reduces excess entitlement while preserving service reliability, developer velocity, and emergency access.

II. PROBLEM STATEMENT AND CONTEXT

Least privilege fails operationally when it is framed as a static target. In AWS, identity decisions are distributed across IAM roles, permission boundaries, session policies, resource policies, KMS key policies, organizations policies, and Kubernetes service accounts. A platform engineer can enforce an SCP at the organization root, but that does not prove a pod uses only the S3, DynamoDB, or KMS actions it needs. Likewise, a tight workload role does not compensate for a broad CI/CD role that can mutate production trust policies.Three forces create drift. First, delivery teams request permissions before an application is fully understood. Second, production incidents reward

temporary broadening of permissions, but rarely reward cleanup after recovery. Third, identity is often encoded in multiple systems: Terraform modules, Helm charts, GitHub or GitLab CI/CD variables, IAM trust policies, Kubernetes annotations, and ticket histories. The resulting architecture is hard to audit because no single repository contains the full entitlement story.

Lo.35Y Metric & Definition and platform interpretation

Unused Permission Ratio & Fraction of granted actions with no observed use during the review window. High values indicate role sprawl or stale policy templates.

Entitlement Half-Life & Median time to remove half of confirmed removable permissions after a finding is opened. This measures cleanup throughput, not only detection.

Role Reuse Risk & Count of unrelated workloads, teams, or environments sharing the same role. High reuse increases blast radius and blocks precise revocation.

Pod Identity Drift & Difference between the service-account-to-role mapping in Git and the mapping observed in the EKS API. Drift exposes manual changes and unsafe exceptions.

Revocation Safety Score & Confidence score combining last-used evidence, service criticality, test coverage, and rollback readiness.

III. RELATED WORK

The model stands on four bodies of work. DevOps research shows that reliable delivery depends on automation, small batches, fast feedback, and learning loops . SRE frames reliability as an engineering discipline with error budgets, toil reduction, and operational learning . Cloud-native systems research, including Borg and Kubernetes, shows that platform abstractions can reduce local complexity but also create control-plane responsibilities . Security standards define the desired controls: NIST SP 800-53 for system controls , NIST SP 800-207 for zero trust , NIST SSDF for secure software practices , and NIST container guidance for runtime boundaries .

AWS-native guidance adds the implementation layer. Organizations and SCPs provide hierarchical boundaries ; Control Tower and the AWS Security Reference Architecture define baseline account patterns ; CloudTrail and Config provide activity and configuration evidence . EKS identity guidance, RBAC, and policy engines describe the workload side of the problem . CISA zero-trust guidance and Kubernetes hardening guidance emphasize identity, least privilege, and continuous validation .

IV. REFERENCE ARCHITECTURE

The architecture has four loops: discovery, decision, remediation, and verification. Discovery ingests CloudTrail events, IAM Access Analyzer unused-access findings, Access Advisor last-used timestamps, IAM credential reports, EKS service accounts, namespace metadata, and infrastructure-as-code state. Decision converts those signals into a revocation plan. Remediation creates pull requests against Terraform, CDK, Helm, or GitOps repositories. Verification confirms that the deployed role, trust policy, permission boundary, service account, and workload behavior match the approved plan.

AWS closed-loop least-privilege control flow.

The entitlement graph is the key abstraction. A node can be an AWS account, OU, IAM role, IAM policy, Kubernetes namespace, service account, CI/CD runner, KMS key, S3 bucket, or database. An edge represents an ability: assume-role, pass-role, decrypt, write-object, update-deployment, attach-policy, or create-token. This graph lets the platform team ask questions that static IAM review cannot answer: Which CI/CD identities can mutate production workload roles? Which pods can obtain credentials for roles outside their namespace ownership boundary? Which roles are unused but protected by vague ownership metadata? Which resource policies bypass otherwise strict identity policies?

Journal of Information Systems Engineering and Management
2024, 9(2)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

V. EKS WORKLOAD IDENTITY

EKS complicates least privilege because Kubernetes identity and AWS identity must be reconciled. With IRSA, a Kubernetes service account is associated with an IAM role through OIDC federation and a trust condition. With EKS Pod Identity, AWS introduced a simplified association model for EKS workloads . Platform teams should treat either mechanism as a first-class identity contract, not merely as a deployment annotation.

The recommended pattern is to make workload identity a platform-owned interface. Application teams request an identity through a catalog item, specify AWS APIs and resources, and bind the identity to a namespace and service account. The platform pipeline generates the IAM role, trust policy, permission boundary, policy document, and Kubernetes association. Admission policy rejects pods that use unmanaged service accounts for AWS access. Periodic reconciliation compares the Git declaration with EKS and IAM state. Differences create findings, not tribal-knowledge tasks.

VI. REMEDIATION WORKFLOW

Not every unused-access finding should be remediated in the same way. Some permissions are unused because a disaster path has not been exercised. Some are unused because the application has moved away from a dependency. Some are unused because a role is shared across dev, test, and prod. The platform workflow should classify findings into safe-removal, verify-before-removal, split-role, and retain-with-justification categories.

A practical remediation pipeline has five gates. Gate 1 verifies ownership and environment. Gate 2 checks last-used evidence and CloudTrail coverage. Gate 3 generates a smaller policy and runs custom policy checks . Gate 4 deploys to a lower environment or a canary namespace. Gate 5 attaches post-change evidence to the finding. The rollback plan is not optional: the pull request must identify the previous policy version, the workload owner, and the expected alarm signals.

VII. EVALUATION DESIGN

A platform team can evaluate this model using a 90-day field experiment. The treatment group includes roles managed by the closed-loop workflow. The control group includes roles reviewed through traditional ticket-based access review. Primary measures are unused-permission ratio, mean remediation time, change-failure rate, and number of emergency permission restorations. Secondary measures are developer wait time, number of identity exceptions, and percent of roles with service ownership metadata.

L0.32Y Finding class & Required action

Clearly stale workload role & Remove unused actions through IaC pull request and verify with workload smoke tests.Shared platform role & Split by workload or environment before removing permissions. Emergency or break-glass role & Retain only with documented exercise evidence, approval owner, and expiration review.Unclear telemetry window & Extend observation or require synthetic exercise before removing high-risk actions.Manual console drift & Reconcile state to Git and open an exception only when ownership is documented. The evaluation measures whether closed-loop least privilege reduces unused permissions without increasing change-failure rate. The largest improvement is measured in automation roles and Kubernetes workload roles, because those roles tend to be codified and testable. Human roles may require longer review windows and stronger exception workflows.

VIII. DISCUSSION

The main risk is false confidence. CloudTrail evidence is not a complete proof of non-use, and unused-access windows can miss rare operations. Therefore, the model must treat evidence as a probability signal rather than a mathematical guarantee. Another risk is platform centralization. If every permission change requires a central team, least privilege becomes a bottleneck. The solution is to maintain golden policy modules, self-service identity requests, automated review, and transparent exception criteria.

The model also changes platform-team incentives. Success is not the number of findings opened. Success is a decreasing entitlement half-life, fewer reused roles, lower blast radius, and stable production behavior. This makes least privilege measurable in the same way that reliability and delivery performance are measurable.

Journal of Information Systems Engineering and Management
2024, 9(2)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

IX. CONCLUSION

In 2023, AWS platform teams gained better primitives for treating least privilege as a continuous control loop rather than a periodic audit. IAM Access Analyzer unused-access findings and custom policy checks supply decision signals, while EKS Pod Identity improves the workload identity interface. This architecture combines those signals with AWS Organizations, Control Tower, CloudTrail, Config, IaC pull requests, and EKS reconciliation. For senior platform engineers, the lesson is practical: least privilege should be productized as a platform capability with metrics, ownership, rollback, and evidence.

The views and opinions expressed in this article are solely those of the author and do not represent or reflect the views, positions, policies, or opinions of the author's employer or any affiliated organization. The content is provided for informational purposes only. The employer makes no representations or warranties regarding the accuracy or completeness of the content and does not endorse or accept responsibility for any statements, conclusions, or materials contained herein.

REFERENCES

[1] J. Barr, "IAM Access Analyzer updates: Find unused access, check policies before deployment," AWS News Blog, Nov. 26, 2023.

[2] AWS Containers Blog, "Amazon EKS Pod Identity: a new way for applications on EKS to obtain IAM credentials," Dec. 28, 2023.

[3] Amazon Web Services, "IAM roles for service accounts," Amazon EKS User Guide, accessed 2023.

[4] Amazon Web Services, "AWS Organizations user guide," AWS Documentation, accessed 2023.

[5] Amazon Web Services, "Service control policies," AWS Organizations User Guide, accessed 2023.

[6] Amazon Web Services, "AWS Control Tower user guide," AWS Documentation, accessed 2023.

[7] Amazon Web Services, "AWS Config developer guide," AWS Documentation, accessed 2023.

[8] Amazon Web Services, "AWS CloudTrail user guide," AWS Documentation, accessed 2023.

[9] Amazon Web Services, "AWS Well-Architected Framework," AWS Documentation, accessed 2023.

[10] Amazon Web Services, "AWS Security Reference Architecture," AWS Prescriptive Guidance, accessed 2023.

[11] Joint Task Force, "Security and privacy controls for information systems and organizations," NIST SP 800-53 Rev. 5, 2020.

[12] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, "Zero Trust Architecture," NIST SP 800-207, 2020.

[13] M. Souppaya, K. Scarfone, and D. Dodson, "Secure Software Development Framework (SSDF) Version 1.1," NIST SP 800-218, 2022.

[14] R. Chandramouli, "Implementation of DevSecOps for a microservices-based application with service mesh," NIST SP 800-204C, 2022.

[15] M. Souppaya, J. Morello, and K. Scarfone, "Application container security guide," NIST SP 800-190, 2017.

[16] Cybersecurity and Infrastructure Security Agency, "Zero Trust Maturity Model Version 2.0," 2023.

[17] National Security Agency and CISA, "Kubernetes Hardening Guidance," v1.2, 2022.

[18] Center for Internet Security, "CIS Amazon Web Services Foundations Benchmark," v1.4.0, 2021.

[19] OWASP Foundation, "OWASP Kubernetes Top 10," 2022.

[20] Open Source Security Foundation, "SLSA v1.0: Supply-chain Levels for Software Artifacts," 2023.

[21] L. Tsai, S. Devadas, J. Borkmann, and the Sigstore community, "Sigstore: Software signing for everybody," Linux Foundation, 2022.

[22] Open Policy Agent Authors, "Open Policy Agent documentation," accessed 2023.

[23] Kubernetes Authors, "Using RBAC authorization," Kubernetes Documentation, accessed 2023.

[24] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles, "A survey of DevOps concepts and challenges," ACM Computing Surveys, vol. 52, no. 6, 2019.

[25] N. Forsgren, J. Humble, and G. Kim, *Accelerate: The Science of Lean Software and DevOps*. IT Revolution Press, 2018.

[26] J. Humble and D. Farley, *Continuous Delivery*. Addison-Wesley, 2010.

[27] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, Eds., *Site Reliability Engineering*. OReilly Media, 2016.

[28] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes," ACM Queue, vol. 14, no. 1, pp. 70–93, 2016.

[29] Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in EuroSys, 2015.

[30] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architects Perspective*. Addison-Wesley, 2015.