DOI 10.70286/ISU-29.10.2025.005

EVALUATING SECRET-MANAGEMENT VIA SIX PLATFORM-AGNOSTIC LEAKAGE SCENARIOS

Molnar Vitalii
Postgraduate student
Department of Information Security
Lviv Polytechnic National University, Ukraine

Modern cloud-native systems handle secrets—tokens, passwords, API keys, encryption materials, and configuration credentials—across build, deploy, and runtime, where leakage risk accumulates at each handoff [1].

We frame the comparison around rotation and revocation windows consistent with established key-management guidance and evaluate auditability and operational burden as first-class outcomes [2].

A concise comparison of the five strategies and their key evaluation axes is provided in Table 1.

Table 1. Comparison of secret-management strategies across residual risk, audit visibility, incident control (rotate/revoke), operational burden, and best-fit context.

Approach	Residual Risk	Audit Visibility	Incident Control (Rotate / Revoke)	Operational Burden	Best-fit Context
Env-only	High	Low	Medium / High	Low	Prototypes, short-lived tokens
Parameter Store	Medium	Medium	Medium / Medium–High	Medium	Serverless, small services
Secret Vault	Low–Medium	High	Low–Medium / Medium	High	Regulated domains, complex CI/CD
KMS-envelope + Store	Low–Medium	High	Medium / Medium	Medium	Audit-centric, CI-intensive setups
Sealed Secrets + Runtime Store	Medium	Medium	Medium / Medium	Medium	Kubernetes/GitOps, declarative pipelines

The study compares five widely used secret-management patterns to ensure breadth without redundancy: environment variables (“env-only”), parameter stores, dedicated secret vaults, KMS-backed envelope encryption, and Kubernetes sealed secrets aligned with GitOps workflows [3].

KMS-envelope delegates cryptographic operations to a managed key service so that only ciphertext and metadata persist, and decrypt events are centrally auditable and governed by key policy [2].

Dedicated secret vaults elevate secrets to first-class objects with versioning, dynamic credentials, rotation hooks, and detailed access logs that strengthen revocation and forensics in production pipelines [4].

Sealed secrets protect declarative manifests by encrypting them under a cluster-resident controller key, which makes repository snapshots unintelligible off-cluster and fits naturally with GitOps workflows [5].

Managed secret stores used at runtime complement these approaches by providing retrieval APIs, per-secret policy boundaries, and integration patterns for refresh in long-running services [6].

The threat model reflects common delivery-chain and runtime realities in cloud-native systems—environment/process exposure and unsafe logging, CI pipeline replication, node/pod compromise and service-account escalation, control-plane or KMS misuse, and GitOps-specific repository risks [3].

These routes define the evaluation lenses: expected blast radius under each vector, leakage observability through audit and telemetry, and the practical ability to rotate and revoke without breaching performance or delivery objectives [3].

We evaluate six reproducible leakage scenarios that cover the above vectors while keeping platform assumptions minimal so results transfer across stacks:

- Environment / process exposure. Simulate env dumps and inspect memory/diagnostic artifacts for clear-text secret residues [1].
- Logging leakage. Induce controlled faults and review logs/error pages to detect verbatim or reversible secret prints [1].
- CI pipeline replication. Compromise a CI job and trace how credentials propagate across variables, artifacts, and logs under pipeline scoping [3].
- Node/Pod compromise & SA escalation. Gain shell on a node/pod to probe mounted secrets, local agents, and metadata services, attempting service-account escalation [3].
- Control-plane / KMS misuse. Emulate privileged access to a secret store or the KMS to assess feasibility of bulk reads and the quality of decrypt-audit signals under centralized key policy [2].
- GitOps repo exfiltration (Sealed Secrets). Exfiltrate repositories and encrypted manifests to verify ciphertext-only exposure off-cluster and assess controller-key governance in-cluster [5].

Incident handling is measured along two windows aligned with key-lifecycle practice: time-to-rotate (issue and propagate a new value) and time-to-revoke (when the old value becomes effectively useless in running systems) [2].

To capture exposure during transition, we compute a simple area-over-threshold measure of time under risk, and we record the risk of false revocation that can cause avoidable downtime or failed releases in real pipelines [2].

Results are reported as profiles rather than a single composite score: blast radius per scenario, leakage observability via audit/telemetry, rotation and revocation dynamics, developer friction and release impact, operational cost, characteristic failure points, and fit for Kubernetes/Git workflows [3]. This profile supports a Pareto view of residual risk versus operational burden, enabling selection of points that match

organizational tolerance and delivery constraints without imposing one-size-fits-all rankings [3].

In qualitative pilots, env-only offers minimal adoption friction yet shows the largest blast radius under environment-dump and CI-leak vectors, whereas vaults strengthen revocation discipline and forensics through versioned objects and access logs [4]. Sealed secrets reliably prevent repository-level disclosure but shift residual risk to controller-key governance in the cluster and must be paired with runtime retrieval for dynamic tokens [5]. Envelope encryption centralizes cryptographic control and audit but still requires a distribution mechanism to refresh values in live services; in practice, teams pair KMS-envelope with a store or a vault to close that gap [2][6].

Guidance is accordingly context-specific: serverless workloads benefit from a managed store hardened with KMS-envelope so that no long-lived cleartext persists in code, images, or artifact logs [6].

Kubernetes platforms practicing declarative GitOps can use sealed secrets for manifests and a runtime store for refreshable credentials via sidecars or init containers without service restarts [5].

Overall, the six-scenario procedure provides a stable way to profile residual risk and operability and can be rerun as systems evolve without changing platform assumptions.

References

1. OWASP. Secrets Management Cheat Sheet. OWASP Cheat Sheet Series, 2024. URL: https://cheatsheetseries.owasp.org/cheatsheets/Secrets_Management_Cheat_Sheet.html
2. Barker E. Recommendation for Key Management—Part 1: General (NIST SP 800-57pt1r5). Gaithersburg, MD: National Institute of Standards and Technology, 2020. DOI: https://doi.org/10.6028/NIST.SP.800-57pt1r5
3. Cloud Native Computing Foundation (CNCF). Cloud Native Security Whitepaper v1.0. 2022. URL: https://github.com/cncf/tag-security/blob/main/security-whitepaper/cloud-native-security-whitepaper.md
4. HashiCorp. Vault — Product Documentation. 2025. URL: https://developer.hashicorp.com/vault/docs
5. Bitnami Labs. Sealed Secrets: A Kubernetes Controller and Tool for One-Way Encrypted Secrets. 2025. URL: https://github.com/bitnami-labs/sealed-secrets
6. Amazon Web Services. AWS Secrets Manager — Documentation. 2025. URL: https://docs.aws.amazon.com/secretsmanager/