

SECRETS & CREDENTIAL SPRAWL.

A secret doesn't stay secret because it's hard to find — **it stays secret because someone is actively managing where it lives, how long it's valid, and when it gets rotated**, none of which happens by default.

- CENTRALIZED SECRETS VAULTING
- SHORT-LIVED DYNAMIC CREDENTIALS
- AUTOMATED SECRET SCANNING
- SCHEDULED ROTATION

01 THE MECHANISM



Hardcoded Secrets in Version Control

A credential committed to a repository doesn't disappear when the line is deleted — **it persists in git history indefinitely**, readable by anyone who can clone the repo and search its commit log, unless that history is actively rewritten.



Cloud Metadata Credential Theft

Cloud workloads fetch temporary IAM credentials from a metadata endpoint rather than storing them — **an attacker with SSRF or code execution on that workload can query the same endpoint** and walk away with valid cloud credentials.



Long-Lived, Unrotated Credentials

A static API key or password stays valid until someone manually changes it — **if rotation never happens, a credential leaked once stays exploitable indefinitely**, long after anyone remembers it was ever exposed.

02 THE FLOW

The Vaulted, Ephemeral Program

CONTAINED

1

Every secret lives in a centralized vault with scoped access control and a full audit trail, rather than scattered across code, config files, and chat messages.

2

Workloads request short-lived, dynamically issued credentials at runtime, so **a credential that leaks is only valid for a narrow window before it automatically expires**.

3

Automated secret scanning runs on every commit and CI build, so **a hardcoded credential is caught and revoked before it ever reaches production**.

The Scattered, Static Program

EXPOSED

1

Secrets live wherever it was convenient to put them at the time — hardcoded in source, pasted into config files, or committed directly to the repository.

2

Credentials are issued once, long-lived, and rarely rotated, so **a secret leaked years ago can still be valid today**, with nobody left who remembers it exists.

3

No scanning catches a hardcoded secret before it merges, so **a credential sits in git history indefinitely**, discoverable by anyone who clones the repository.

03 CORE PILLARS & COVERAGE

PILLAR	WHAT IT COVERS
Centralized Secrets Vaulting	Stores every credential in a dedicated secrets manager with scoped access control and a full audit trail, replacing scattered config files, code, and chat messages.
Short-Lived, Dynamic Credentials	Issues time-boxed, automatically expiring credentials at runtime instead of static, long-lived secrets that stay valid indefinitely once created.
Automated Secret Scanning	Scans every commit, pull request, and CI build for credential patterns before they merge, catching a hardcoded secret before it reaches production or git history.
Scheduled Rotation & Revocation	Automates credential rotation on a defined schedule and revokes orphaned credentials tied to deprecated services or departed employees.

04 DETECTION APPROACHES COMPARED

APPROACH	SUFFICIENT ALONE?	WHAT IT MISSES
Manual Secret Review Before Commit	No	Depends entirely on a human catching a hardcoded credential before it merges, which fails exactly as often as human review fails anywhere else.
Long-Lived API Keys With No Expiration	No	Leaves a leaked credential exploitable indefinitely, with no automatic window in which the exposure closes itself.
Environment Variables Without a Central Vault	No	Scatters secrets across every service's own configuration, with no unified access control, rotation, or audit trail.
Secret Scanning Without Automated Revocation	No	Detects the exposure but still depends on a human to rotate the credential before an attacker who already saw it can use it.
Centralized Vaulting + Short-Lived Credentials + Scanning + Scheduled Rotation	Yes	Covers storage, credential lifetime, detection, and revocation together, rather than any one control alone.

SECRETS & CREDENTIAL SPRAWL.

VAULT · ROTATE · SCAN

05

PROGRAM OPERATIONS

A secret removed from the latest commit isn't a secret removed. Git history retains every prior version of a file unless it's actively rewritten, and most leaked secrets are found by searching history, not the current branch.

Cloud metadata endpoints are a credential source, not just a configuration detail. Any code execution or SSRF on a workload with an attached IAM role is effectively code execution against that role's own permissions.

Rotation that requires manual effort tends not to happen. A secrets program's real rotation cadence is whatever automated tooling enforces, not whatever the policy document says it should be.

An orphaned credential is a permanent grant nobody remembers giving. A service account created once and never decommissioned keeps its access long after the reason for it is gone.

06

CONTROL MAPPING

OWASP SECRETS MANAGEMENT CHEAT SHEET

Consolidated guidance on vaulting, rotation, and eliminating hardcoded credentials from source code and configuration.

NIST SP 800-57

Key management recommendations covering cryptographic key and credential lifecycle, including generation, rotation, and retirement.

CIS CONTROLS (ACCOUNT & CREDENTIAL MANAGEMENT)

Baseline controls for inventorying, securing, and monitoring accounts and their associated credentials across an environment.

HASHICORP VAULT / CLOUD-NATIVE SECRETS MANAGERS

Industry-standard tooling for centralized secret storage, dynamic credential issuance, and automated rotation.

07

WHAT THIS DOES NOT SOLVE

Weak Credential Values

Vaulting a weak or guessable password doesn't make it strong — **storage security and credential strength are separate problems.**

Compromised Endpoints

A properly vaulted, scoped credential is still usable by **whoever has compromised the workload authorized to fetch it.**

Human Social Engineering

No amount of vaulting stops a credential from being **handed over voluntarily** to a convincing phishing attempt.

Authorization Logic Gaps

A valid, scoped credential can still **exploit an authorization flaw in the application it's granting access to.**

08

MYTHS VS. REALITY

MYTH

"Deleting a secret from the latest commit removes it from the repository."

REALITY

The credential **remains in every prior commit in git history** indefinitely, discoverable by anyone who clones the repo, unless that history is actively purged.

MYTH

"Environment variables are a secure way to store secrets."

REALITY

Environment variables are **often readable by any process on the host, logged accidentally, or exposed through debugging endpoints** — convenient, not inherently secure.

MYTH

"A cloud workload's IAM role is safe if no one has the access keys."

REALITY

An attacker with code execution or SSRF on that workload can **query the metadata service directly and obtain valid credentials** without ever needing a static key.

MYTH

"Rotating a credential is only necessary after a known leak."

REALITY

Routine rotation on a schedule **shrinks the exploitation window for leaks nobody has discovered yet**, not just ones already known about.

MYTH

"A secrets vault alone solves credential sprawl."

REALITY

A vault only helps for **secrets that are actually put into it** — scanning and enforcement are what stop new hardcoded credentials from bypassing it entirely.

09

SPEAK THE PROGRAM — KEY TERMS

SECRET SPRAWL

Credentials scattered across **code, config files, CI/CD systems, and chat messages** instead of a single, centrally managed and audited location.

DYNAMIC SECRETS

Credentials **generated on demand and automatically expiring**, issued by a secrets manager rather than created once and reused indefinitely.

CLOUD METADATA SERVICE

An internal endpoint **cloud workloads use to fetch temporary IAM credentials**, which becomes an attacker target if reachable through SSRF or code execution.

ORPHANED CREDENTIAL

A credential still valid and active for a **service or person that no longer needs it**, left ungoverned after the original purpose ended.

SECRET SCANNING

Automated detection of **credential patterns in code, commits, or logs**, used to catch hardcoded secrets before or shortly after they're exposed.

LEAST-PRIVILEGE SCOPING

Limiting a credential's permissions to **exactly what its purpose requires**, so a leak or misuse causes the smallest possible blast radius.