Engineer

Operational secret

a credential a program needs, like a key or a password

What it is

An operational secret is a value that lets software, or the person who runs it, act on a system, and that gives the same power to anyone else who reads it. A database connection password, an API token, the private half of an SSH key pair and the passphrase that encrypts a backup are all operational secrets. In everyday use the word shrinks to secrets, and the work of creating, storing, rotating and destroying them is called secrets management.

Two things set them apart from the code around them. The first is the reader. A user's password sits in someone's memory and is typed at a prompt, while an operational secret is read by a program, a deploy script or a scheduled job, and nobody has to remember it. The OWASP Secrets Management Cheat Sheet therefore gives it a lifecycle of its own: generated with the least privilege it needs, rotated on a schedule, revoked when it may have leaked, and set to expire where the platform allows, with lifetimes from minutes to years. User passwords sit outside that regular rotation. The second is the line between a secret and the configuration that points at it. A hostname or a backup repository path says where something is. The secret is the part that lets the holder in, and the part whose copies have to be counted.

Secrets also nest. The store that holds them needs a secret of its own, and the cheat sheet notes that this primary secret, such as a cloud provider's root credentials, usually has to be kept somewhere else.

Use the concept in a decision

Put "What is an operational secret?" to work in a real decision. We can help relate the concept to the system you are building or buying. Accounts, passwords and permissions arranged so people hold exactly the access their job needs, and leavers lose it that day.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Wrote 1,000 RBAC rules to secure Linux servers — 1,000 access rules written to close a security gap

Wrote 1,000 RBAC rules to secure Linux servers Get in touch →

Why it matters

A secret works for whoever presents it. A token carries no record of who was meant to hold it, so each copy widens the set of people and programs that can use it. A source repository is the worst place for one to land: every clone carries the full history, so a value committed once travels to every laptop, build runner and mirror that pulls, and it stays in that history after the line holding it is deleted.

The standards turn this into requirements. The OWASP Application Security Verification Standard asks for a secrets management solution such as a key vault to create, store, control access to and destroy backend secrets, and states that secrets must not be included in application source code or in build artefacts. It asks for least privilege on access to them and, at its highest level, for them to expire and rotate. NIST SP 800-53 adds control enhancement IA-5(7): unencrypted static authenticators must not be embedded in applications or other static storage. Its guidance closes a loophole too, since a value used exactly as it is stored counts as unencrypted, whatever file it sits in.

The trade-offs are real. A central store is one more system that has to be up when it is needed most, and the cheat sheet points out that in an incident people expect new credentials quickly. Sharing one secret among several services saves work and costs attribution, because after a leak there is no telling which holder lost it. Manual handling grows with every secret, which is why the cheat sheet favours automated rotation, and it warns that restarting an application leaves a stolen credential working. Only revoking it ends that.

What it looks like in practice

This site states its rule on the page about how the company works securely: secrets stay out of the repository. Credentials are held in a secret store or in the deployment layer, kept out of source and out of anything a build produces, and an automated scan runs on every commit and stops the commit when it finds one.

The engineering notes for the repository fill in the how. The scanner is gitleaks, run over the working tree on every commit and again in the full quality check, and a hit is a hard stop that nobody is meant to allowlist away. The dependency folder and the generated output trees are excluded, because they are copies rebuilt from scratch. The notes also record a quieter failure: when a file disappears mid-scan, gitleaks prints an error line and still exits successfully, so the task that runs it fails on any error line. The site itself is static and holds no application secrets, and it deploys over SSH with keys the owner keeps.

Keeping secrets out of the repository moves them somewhere its checks cannot reach, and a preflight play for secrets deals with that gap. The secrets had just been split per host and the code was right, but the operator's encrypted vault, kept on the operator's own machine and out of sight of the quality gate, still held the old single-host value. The first converge reached the live host, ran eight tasks and refused on the ninth, after it had already written swap settings to production. The answer was a play that contacts no host, gathers no facts and runs the same two task files as the real converge against the vault, in about a second, before provisioning does anything else. Reusing the code rather than restating the rule keeps the check and the converge from drifting apart. A bad secret now costs a two-second refusal on a laptop.

The split itself is the move to one value per host for every operational secret. The platform had one host and was about to have two, and several secrets had been written as single values, among them the backup repository, its passphrase and an alert switch. Each became a map keyed by inventory hostname, and two reasons are recorded beside the configuration. A dead-man's switch shared by two hosts stays green while either one still checks in, so the second host hides the death of the first. Separate restic repositories can have distinct master keys even when they use the same passphrase. But someone who obtains that passphrase and can read both repositories' encrypted data and key files can unlock both, so sharing it couples their exposure rather than isolating the hosts. The existing host appears in all four maps with its repository and both switches left empty, which says the wiring exists and the value is still owed by the operator.

Where it goes wrong

A secret lands in a commit. A scanner flags a value in a diff, or someone spots a token in a pushed file, because it sat in a configuration file or a test fixture when the commit was made. The OWASP cheat sheet gives the order of repair: revoke the exposed value first, then rotate it, and only then remove it from the code or

the logs. Rewriting history to drop the commit changes every later commit and breaks links to them, so it needs planning. Detection belongs before the commit, in a pre-commit hook or the editor, where a hit costs nothing.

The scan passes over files it never read. A run ends green while part of the tree went unexamined: a file vanished mid-walk, an exclusion was drawn too wide, or the secret has no recognisable shape. Pattern rules catch tokens with a fixed format and miss a password a person made up, which is why the cheat sheet suggests more than one tool. Treat any scanner error as a failure, and keep exclusions to copies a build recreates.

One secret serves many holders. Two hosts, two services or a whole team use the same value. When it leaks, the logs cannot say which copy went, and replacing it means touching every holder at once. SP 800-53 asks for the authenticators of group or role accounts to change whenever membership changes, and a secret per holder makes that one replacement instead of a coordinated one.

The secret ships with the build or reaches a log. A container image, a bundled script or a debug line carries the value out of the place it was meant to stay. ASVS names build artefacts next to source code, and the cheat sheet asks that secrets stay out of logs, masked where a line has to refer to one. Scan what the build produces as well as the repository.

Further reading

- OWASP Secrets Management Cheat Sheet — the lifecycle of a secret, detection before the commit and the order of repair after a leak.
- OWASP Application Security Verification Standard — the secret management requirements in group 13.3: a vault, least privilege and rotation.
- NIST SP 800-53 Rev. 5 — authenticator management in IA-5, with IA-5(7) against embedded static authenticators.
- Gitleaks — the scanner this site runs over its working tree on every commit.



Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is an operational secret?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/secret/