717DEV

DEVELOPER CHEAT SHEET

Secrets triage and remediation

Find secrets, decide which are real, revoke and replace them.

THE ONE RULE

Revoke the key at the issuer, since nothing else ends it. Then prove it fails.

If a secret just leaked

Once a secret has been committed, treat it as compromised, whether or not anyone has used it.

01 **Stop.** Do not delete anything yet: not the file, the branch, the alert or a log.

02 **Write it down, and tell.** Record what leaked and start the log exports without waiting on them. Bring in the owner, security and legal as soon as an incident trigger applies.

03 **Choose.** Public, or do the issuer's logs show use by someone you cannot name? Revoke now: do step 05 first, take the failed calls, then step 04. If not, use overlap.

04 **Issue the replacement,** and move every consumer onto it.

05 **Revoke the old one** at the issuer, since nothing else ends it.

06 **Hunt.** Search the issuer's logs across the whole window of exposure.

07 **Keep the right people told,** from the owner to legal and privacy.

Never do this

- **Force push first.** A rewrite cannot reach forks, clones or pull request refs, and it destroys the SHAs that your hunt depends on.
- **Paste the secret.** A ticket, a chat and an alert comment are each one more copy. Use the key ID or a hash.
- **Close the alert before proof.** Removing the line changes nothing at the issuer, and every clone still holds a working key.
- **Test a key that is not yours.** Report it to its owner, and run TruffleHog with `--no-verification` on code you do not own.

Where secrets hide

`appsettings.*.json` `.env` `reflog` `refs/pull/` `image layers` `NEXT_PUBLIC_` `.npmrc` `~/.aws/credentials` `~/.zsh_history` `terraform.tfstate` `*.ipynb` `.mcp.json` `*.har` `CI logs` `chat`

Find it

TOOL	WHAT IT COVERS	TURN IT ON OR RUN IT
GitHub secret scanning	All history, issues, pull requests, wikis	Settings, Advanced Security, Secret Protection
Push protection	New pushes only, on the server	Same path, then Push protection, before the sweep
Validity checks	Asks the issuer. Treat `unknown` as live	Later: `secret_scanning_validity_checks=enabled`
TruffleHog	Git, organizations, files, images	Verifies by default. Pin a version, pass `--no-update`
Gitleaks, Betterleaks	History and fast local hooks	`gitleaks git --redact --report-path .gitleaks-baseline.json`
detect-secrets	The current tree only, never history	`detect-secrets scan > .secrets.baseline`
Azure DevOps Services	Pushes and history	Project settings, Repos, Repositories, Begin billing
GitLab	Free job: current state, new commits	History: one run with `SECRET_DETECTION_HISTORIC_SCAN=true`

Your first sweep

From a trusted host, with a read-only token in `GITHUB_TOKEN`. Treat the output as a secret.

```
trufflehog github --org=example-systems --issue-comments --pr-comments --gist-comments \
  --include-wikis --results=verified,unknown,unverified --no-update --json > findings.json
jq -c 'del(.Raw, .RawV2, .SecretParts)' findings.json > findings.safe.jsonl  # for the ticket
```

A local hook can be skipped, so enforce the check in CI. Plain output carries the value, so keep it out of logs.

717DEV

ASSESS

Real or not

Five questions

1. **Is it a credential at all?** Do not trust its label.
2. **Is it public by design or a documented sample?** A sample ID can sit beside a real secret.
3. **Is it live?** Unknown counts as live.
4. **Whose is it, and what can it do?** Ask the issuer, never with the leaked key.
5. **How far did it travel?** If public, treat the key as used.

Is it a credential at all?

A false positive passes three tests: name exactly what the string is, point to a source other than its author, and say why nothing of yours would accept it. These look false and are real:

+ **A placeholder that works.** Treat it as a weak password.
+ **A value in Base64.** Encoding adds no protection.
+ **A test key, or one a comment calls dead.** Ask the issuer.

Score it

WHAT IT REACHES	PUBLIC	SHARED WIDELY	PRIVATE
Production write or admin	Critical	Critical	High
Production read or customer data	Critical	High	Medium
Non-production or narrow scope	High	Medium	Low

Private means you can list everyone with access. Without that list, it is Shared widely, or Public if a public copy is possible.

Revoke now or overlap

The one test: is it public, or is someone you cannot name using it? Answer the second from a first look at the issuer's logs, never from silence.

+ **Revoke now** if either is yes.
+ **Overlap, minutes to hours, with an owner,** if production, private, no sign of use in the logs, and revoking would stop customers.
+ **Planned slot, inside the response clock,** if test, development or inactive, in a private repository.
+ **Short planned outage** if the issuer cannot hold two valid keys at once.

Is it live?

+ **In a public commit,** skip the test and revoke it now.
+ **Validate only what belongs to your organization,** with the smallest call that answers, and stop at yes or no.
+ **Only its own issuer should see the value.** Keep it off the command line, out of URLs and out of online decoders.

Record the decision

Record the ID, decision, reason, evidence, who and when. Keep the secret out of `resolution_comment`, which cannot be deleted.

RESOLVE AS	ONLY
`false_positive`	After the three false positive tests
`used_in_tests`	When the value reaches nothing real
`revoked`	After the issuer has killed it
`wont_fix`	With an owner and a review date

Response clock

SCORE	REVOKE WITHIN	WHAT THAT MEANS
Critical	1 hour	Page the owner at any hour
High	24 hours	Pull the owner in today
Medium	7 days	Schedule it this week
Low	30 days	Backlog, with an owner and a date

Our recommendation, counted from the alert to revocation. A key confirmed dead at the issuer drops to Low.

Incident triggers

We recommend six triggers. At the first, bring in your security lead and whoever owns legal and privacy, without waiting for proof of abuse.

+ Use that you cannot account for
+ Access to personal, payment, health or customer data
+ Reach into production or a signing key
+ Missing or expired logs
+ A public secret with unknown exposure
+ A key owned by a customer or partner

717DEV

FIX

Revoke it, then prove it fails

Zero-downtime pattern

01 **Issue the replacement** into the issuer's free slot. A second key from the same config file is not a clean slot.

02 **Move every consumer.** Deploy the new value everywhere, and confirm that nothing presents the old one.

03 **Revoke the old one** at the issuer, first if the one test says so. The rotation is complete only when the old key is revoked.

04 **Prove it fails.** Repeat one cheap read with the old value, once. Read the error code: a 403 can mean a live key.

Rewrite history last

+ **A rewrite does not undo a leak.** It comes last, and usually not at all if public.
+ **Afterward, everyone re-clones,** since one pull and push from an old clone brings the secret back.

In a fresh clone, with `git-filter-repo` 2.47 or later:

```
git-filter-repo --sensitive-data-removal \
  --replace-text ../passwords.txt \
  --replace-message ../passwords.txt
```

Revoke and prove, by credential

AWS access key	REVOKE	`aws iam update-access-key --user-name ci-deploy --access-key-id KEY_ID --status Inactive` and delete it after the hunt. End its sessions with a deny on `aws:TokenIssueTime` on the user and every role it could assume.
	PROOF	`aws sts get-caller-identity` with only the old key, no profile or `AWS_SESSION_TOKEN`, returns 403. No error code is documented for an Inactive key, so save the exact text.
Azure Storage key	REVOKE	`az storage account keys renew -g rg-billing-prod -n stexamplebilling --key key1 -o none`
	PROOF	The old key gets 403 `AuthenticationFailed`. Only that counts, since `AuthorizationFailure` is the firewall.
Azure SAS	REVOKE	Renew the key that signed it, delete its stored access policy and never reuse the name, or, for a user delegation SAS, `az storage account revoke-delegation-keys`
	PROOF	The URL returns 403. Allow 30 seconds for a stored access policy, a delay for a delegation key.
Entra client secret	REVOKE	`az ad app credential delete --id APP_ID --key-id KEY_ID`. Never `reset` without `--append`, and send its password to a file.
	PROOF	A token request returns `AADSTS7000215`. Tokens already issued last 60 to 90 minutes, and up to 24 hours under continuous access evaluation.
GitHub token	REVOKE	Delete it in Developer settings. If it is not yours, `POST /credentials/revoke`, unauthenticated.
	PROOF	`GET /user` returns 401. Run it once, since repeated failures bring a temporary 403.
Deploy or SSH key	REVOKE	`gh api --method DELETE /repos/example-systems/billing-api/keys/KEY_ID` and, on a host, its line in `authorized_keys` and `authorized_keys2`.
	PROOF	`ssh -i old_key -o IdentitiesOnly=yes -T git@github.com` gets `Permission denied (publickey).`
Database password	REVOKE	PostgreSQL: `\password app_user`, then `pg_terminate_backend`. SQL Server: `ALTER LOGIN`, `KILL`, and `DBCC FLUSHAUTHCACHE` on Azure SQL Database. MySQL: `ALTER USER`, `DISCARD OLD PASSWORD`, `KILL CONNECTION`.
	PROOF	`password authentication failed for user`, error 18456 state 8 in the log, or `ERROR 1045 (28000)`.
JWT or session signing key	REVOKE	Publish a new key and `kid`. Take the old key out of the JWKS, verifier caches and fallback lists now.
	PROOF	Your verifier refuses a token signed with the old key, once its JWKS cache has refreshed.
Stripe key	REVOKE	Dashboard, API keys, overflow menu, Rotate key, with Expiration set to Now.
	PROOF	`GET https://api.stripe.com/v1/balance` returns 401. A 403 is a live restricted key.
Slack token	REVOKE	`auth.revoke`. Slack does not say it ends webhooks, so remove a leaked one in the app's settings.
	PROOF	`auth.test` returns `"ok": false`. Never test a webhook: it sends a real message.

717DEV

FIX AND PROVE

Hunt, replace, prove

Hunt for abuse

The window opens at the first commit or earlier, not at the alert, and closes when the last thing the credential spawned has died.

+ **AWS:** loop `aws cloudtrail lookup-events` on its `AccessKeyId` over every Region, then on each `ASIA` session key it created.
+ **Azure:** `AADServicePrincipalSignInLogs`, `AzureActivity` and `StorageBlobLogs` exist only if sent to Log Analytics before the incident. If not, use the Entra admin center and Activity log.
+ **GitHub:** on Enterprise Cloud, organization owners can search the audit log by token hash. Git events last 7 days.
+ **Abuse looks like** `GetCallerIdentity` then `CreateUser` from an unknown address, or `listKeys` or `git.clone` by a stranger.

Missing logs mean the answer is unknown.

The replacement ladder

Worst to best. Stand on the highest rung that the other side of the connection supports.

+ **Hard-coded in source:** never.
+ **Platform store, narrowly scoped:** CI secret, app setting.
+ **Encrypted in the repository:** SOPS.
+ **Vault-held, read at run time, rotated:** Key Vault, Secrets Manager.
+ **Short-lived, issued on demand:** Vault dynamic credentials, STS, user delegation SAS.
+ **No secret at all:** managed identity, OIDC federation.

Prove the fix

A finding closes on evidence: what the key created has ended, and the window was hunted.

+ **The old credential fails at the issuer,** with the output saved.
+ **A second person has repeated that test,** or watched it run, and read the record.

Keep it fixed

+ **Every guardrail needs a check that someone reads:** push protection, CI scan, Azure Policy, log retention.
+ **An inventory with owners:** one row per credential, not per copy, with names and metadata, never a value.
+ **A rotation drill once a quarter,** by the runbook as written.

Twelve false assumptions

- "It is in the CI/CD secret store, so it is safe."	Anyone who can edit a pipeline can read the secrets it uses.
- "It is masked in the logs, so nobody can see it."	Masking applies only to log text.
- "The repository is private, so the key is private."	Everyone with access to the repository can read the key.
- "We deleted it in the next commit."	The old commit holds it, and the key still works.
- "We rotated it, so we are done."	Sessions and forged tokens outlive the old key.
- "`.gitignore` keeps it out of Git."	A rule does nothing for a file Git already tracks.
- "It is encoded, so it is protected."	Base64 is a format, and reversing it needs no key.
- "Environment variables are the secure way."	Better than source, and still readable at run time.
- "The scanner came back clean."	It means only that no rule matched what it could read.
- "It is only a test key."	Its permissions decide what it can reach, whatever its name.
- "It is in the vault, so the problem is solved."	Then the question is who can read the vault.
- "Terraform marks it sensitive."	That redacts output. State keeps the plain value.

CI/CD stores, in three tiers. Unencrypted, readable with no decrypt step: CodeBuild `PLAINTEXT`, Kubernetes Secrets in default etcd. Encrypted, with the key beside it, so whoever holds the server holds the secret: Jenkins, self-managed GitLab, TeamCity by default. Elsewhere, anyone who can edit a pipeline can still read it: GitHub Actions, GitLab.com, Azure Pipelines, CircleCI, Bitbucket.

© 2026 717 DEV LLC