Vault Secure Introduction Cheat Sheet

Core Idea

- Never ship long-lived tokens.
- CI requests a wrapped token; only the short-lived wrapping token is handed to the app.
- App unwraps once, holds the real token in memory, uses/renews per TTL, never writes to disk.
- If an attacker races the unwrap, your app's unwrap fails. Alert on that signal.

Minimal Vault Setup (once)

```
# Enable auth methods you will use
vault auth enable approle
vault auth enable jwt      # only if using GitHub OIDC

# Policy: CI (issuer) can mint wrapped child tokens; scope build reads narrowly
cat > ci_issuer_policy.hcl <<'HCL'
path "auth/token/create" { capabilities = ["update"] }
# Example (optional) build-time reads:
# path "kv/data/build/*" { capabilities = ["read"] }
HCL
vault policy write ci-issuer ci_issuer_policy.hcl

# Policy: runtime token for the application (least privilege)
cat > app_runtime_policy.hcl <<'HCL'
path "kv/data/prod/app/*" { capabilities = ["read"] }
HCL
vault policy write app-runtime app_runtime_policy.hcl
```

Create an AppRole for CI

```
vault write auth/approle/role/ci-issuer \
  token_policies=ci-issuer \
  secret_id_num_uses=5 \
  secret_id_ttl=1h \
  token_ttl=15m token_max_ttl=1h

# Treat these like username/password
vault read  -field=role_id    auth/approle/role/ci-issuer/role-id   > ROLE_ID.txt
vault write -f -field=secret_id auth/approle/role/ci-issuer/secret-id > SECRET_ID.txt
```

Response Wrapping Pattern (CI handoff to app)

```
# After CI authenticates to Vault, mint a WRAPPED token for the app:
vault token create -policy=app-runtime -orphan -ttl=15m -wrap-ttl=5m -format=json \
| jq -r '.wrap_info.token' > WRAPPING_TOKEN.txt

# Deliver only WRAPPING_TOKEN to the app (artifact/env/metadata channel).
# App unwraps once at startup and keeps the real token only in memory.
```

Jenkins + Vault (AppRole + Wrapping)

Jenkins credentials

Store VAULT_ADDR, ROLE_ID, SECRET_ID as Jenkins credentials (admin scope only).

Declarative Pipeline (minimal)

```
pipeline {
  agent any
  environment {
    VAULT_ADDR = credentials('vault-addr')    // or a global env var
    ROLE_ID    = credentials('vault-role-id')
    SECRET_ID  = credentials('vault-secret-id')
  }
  stages {
    stage('Login to Vault') {
      steps {
        sh '''
          set -euo pipefail
          token=$(vault write -format=json auth/approle/login \
                    role_id="$ROLE_ID" secret_id="$SECRET_ID" \
                  | jq -r .auth.client_token)
          export VAULT_TOKEN="$token"
          wrap=$(vault token create -policy=app-runtime -orphan -ttl=15m -wrap-ttl=5m -format=json \
                 | jq -r .wrap_info.token)
          echo "$wrap" > WRAPPING_TOKEN.txt
        '''
      }
    }
    stage('Deliver to App/Deploy') {
      steps {
        sh 'echo "Deliver WRAPPING_TOKEN via your deployment mechanism (do not log it)"'
        archiveArtifacts artifacts: 'WRAPPING_TOKEN.txt', fingerprint: true
      }
    }
  }
  post { always { sh 'rm -f WRAPPING_TOKEN.txt || true' } }
}
```

Jenkins Hardening

- Short TTLs; use orphan child tokens; rotate SecretIDs regularly.
- Avoid writing secrets; if needed, use tmpfs (for example, `/dev/shm`) and wipe.
- Enable audit device; alert on unwrap failures.

GitHub Actions + Vault: Two Options

Option A: AppRole (fast lift; secrets live in GitHub)

Add VAULT_ADDR, VAULT_ROLE_ID, VAULT_SECRET_ID as repo/org secrets.

```yaml
name: ci-with-vault-approle
on: [push]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Install Vault CLI
        run: |
          sudo apt-get update -y && sudo apt-get install -y jq
          curl -fsSL https://releases.hashicorp.com/vault/1.17.0/vault_1.17.0_linux_amd64.zip -o v.zip
          unzip -o v.zip && sudo mv vault /usr/local/bin/

      - name: Login to Vault (AppRole)
        env:
          VAULT_ADDR:      ${{ secrets.VAULT_ADDR }}
          VAULT_ROLE_ID:   ${{ secrets.VAULT_ROLE_ID }}
          VAULT_SECRET_ID: ${{ secrets.VAULT_SECRET_ID }}
        run: |
          set -euo pipefail
          token=$(vault write -format=json auth/approle/login \
                    role_id="$VAULT_ROLE_ID" secret_id="$VAULT_SECRET_ID" \
                  | jq -r .auth.client_token)
          export VAULT_TOKEN="$token"
          wrap=$(vault token create -policy=app-runtime -orphan -ttl=15m -wrap-ttl=5m -format=json \
                 | jq -r .wrap_info.token)
          echo "wrapping_token=$wrap" >> $GITHUB_OUTPUT
        id: vault

      - name: Deliver WRAPPING_TOKEN
        run: echo "Deliver WRAPPING_TOKEN to app via your deploy mechanism"
```

Option B: OIDC/JWT (preferred; zero long-lived secrets)

Vault: configure JWT auth for GitHub

```bash
vault auth enable jwt

vault write auth/jwt/config \
  oidc_discovery_url="https://token.actions.githubusercontent.com" \
  default_role="gha-default"

vault write auth/jwt/role/gha-myrepo \
  role_type="jwt" \
  user_claim="sub" \
  bound_claims='{"repository":"owner/repo","ref":"refs/heads/main"}' \
  policies="ci-issuer" \
  token_ttl="15m" token_max_ttl="1h"
```

GitHub Actions: request OIDC token, login, mint wrapped token

```yaml
name: ci-with-vault-oidc
on: [push]
permissions:
  id-token: write   # required
  contents: read
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Install Vault CLI
        run: |
          sudo apt-get update -y && sudo apt-get install -y jq
          curl -fsSL https://releases.hashicorp.com/vault/1.17.0/vault_1.17.0_linux_amd64.zip -o v.zip
          unzip -o v.zip && sudo mv vault /usr/local/bin/

      - name: Fetch GitHub OIDC JWT
        id: idtoken
        run: |
          set -euo pipefail
          resp=$(curl -sSf -H "Authorization: bearer $ACTIONS_ID_TOKEN_REQUEST_TOKEN" \
                 "${ACTIONS_ID_TOKEN_REQUEST_URL}&audience=vault")
          echo "token=$(echo "$resp" | jq -r .value)" >> "$GITHUB_OUTPUT"

      - name: Login to Vault (JWT) and mint wrapped token
        env:
          VAULT_ADDR: ${{ secrets.VAULT_ADDR }}
          GHA_JWT:    ${{ steps.idtoken.outputs.token }}
        run: |
          set -euo pipefail
          vt=$(vault write -format=json auth/jwt/login role="gha-myrepo" jwt="$GHA_JWT" \
               | jq -r .auth.client_token)
          export VAULT_TOKEN="$vt"
          wrap=$(vault token create -policy=app-runtime -orphan -ttl=15m -wrap-ttl=5m -format=json \
                 | jq -r .wrap_info.token)
          echo "wrapping_token=$wrap" >> $GITHUB_OUTPUT
        id: vault

      - name: Deliver WRAPPING_TOKEN
        run: echo "Deliver WRAPPING_TOKEN to app via your deploy pipeline"
```

OIDC Hardening

- Constrain `bound_claims` (repository, ref, environment, workflow), and audiences.
- Separate roles per repo/branch; least-privilege policies.
- Short TTLs; rotate role mappings when teams change.

Application Side (common)

```
# At startup, app receives WRAPPING_TOKEN via env/secret file/metadata
VAULT_TOKEN_JSON=$(vault unwrap -format=json "$WRAPPING_TOKEN") || {
  echo "FATAL: unwrap failed"; exit 1; }
export VAULT_TOKEN=$(echo "$VAULT_TOKEN_JSON" | jq -r .auth.client_token)

# Read secrets as needed (policy constrained)
vault kv get -format=json kv/prod/app/db \
| jq -r '.data.data | @json' > /dev/shm/app-secrets.json

# Renew until TTL or re-unwrap on restart; never persist tokens.
```

Ops Guardrails (Checklist)

- Short TTLs everywhere (wrapping 2-10m; runtime 10-30m; bounded max_ttl).
- Orphan child tokens for runtime; small blast radius.
- Narrow policies (only the paths CI/app needs).
- Audit enabled; alerts on unwrap failures and unusual token use.
- Rotate AppRole SecretIDs; prefer OIDC/JWT over stored credentials when possible.
- Never log tokens; handle in tmpfs; scrub artifacts and workspaces.

Build tip: compile with `xelatex -shell-escape`. If using pdflatex, ensure `-shell-escape` and that Pygments is available to minted.