SecureVault Access Control Guide

Document ID: PRD-SV-010 **Last Updated:** 2024-02-15 **Owner:** SecureVault Product Team **Classification:** Public

---

Overview

SecureVault provides fine-grained access control to manage who can access which secrets. This guide covers authentication methods, authorization policies, and best practices.

Authentication Methods

API Tokens

Token Types: | Type | Prefix | Use Case | Expiration | |——|———|———-|————| | Root | `sv_root_` | Initial setup only | Never (disable after setup) | | Admin | `sv_admin_` | Administrative operations | 90 days | | Service | `sv_svc_` | Application access | Configurable | | Read-only | `sv_read_` | Monitoring, auditing | 30 days |

Creating Tokens:

```
# Via CLI
securevault token create \
  --name "my-service" \
  --type service \
  --policies read-secrets,write-secrets \
  --ttl 30d

# Via API
curl -X POST https://vault.novatech.com/v1/auth/token/create \
  -H "Authorization: Bearer $ROOT_TOKEN" \
  -d '{"name": "my-service", "policies": ["read-secrets"], "ttl": "720h"}'
```

AppRole (Machine Authentication)

Recommended for applications and services.

Setup:

```
# 1. Create AppRole
securevault approle create my-app \
  --policies app-secrets \
  --secret-id-ttl 24h \
  --token-ttl 1h

# 2. Get Role ID (stable identifier)
securevault approle role-id my-app
# => role_id: abc123

# 3. Generate Secret ID (rotatable credential)
securevault approle secret-id my-app
# => secret_id: xyz789
```

Application Login:

```
from securevault import Client

vault = Client()
vault.auth.approle(
    role_id="abc123",
    secret_id="xyz789"
)
# Now authenticated, can access secrets
```

OIDC (User Authentication)

For human users via SSO.

Supported Providers: - Okta - Azure AD - Google Workspace - Auth0 - Generic OIDC

Configuration:

```
securevault auth enable oidc \
  --client-id "xxx" \
  --client-secret "yyy" \
  --discovery-url "https://company.okta.com/.well-known/openid-configuration" \
  --default-role "employee"
```

User Login:

```
securevault login oidc
# Opens browser for SSO authentication
```

Kubernetes Authentication

For workloads running in Kubernetes.

Setup:

```
# Enable Kubernetes auth
securevault auth enable kubernetes \
  --kubernetes-host "https://kubernetes.default.svc" \
  --token-reviewer-jwt "@/var/run/secrets/kubernetes.io/serviceaccount/token"

# Create role for namespace
securevault kubernetes role create my-app \
  --bound-service-accounts "default" \
  --bound-namespaces "production" \
  --policies "app-secrets"
```

Pod Authentication:

```
# Pod spec
spec:
  serviceAccountName: my-app
  containers:
    - name: app
      env:
        - name: VAULT_ADDR
          value: "https://vault.novatech.com"
```

Authorization Model

Policy Structure

Policies define what authenticated entities can do.

Policy Format:

```
# my-policy.hcl
path "secrets/data/myapp/*" {
  capabilities = ["read", "list"]
}

path "secrets/data/myapp/config" {
```

```
  capabilities = ["read", "update"]
}

path "secrets/metadata/myapp/*" {
  capabilities = ["list"]
}
```

Capabilities

Capability	Description
`create`	Create new secrets
`read`	Read secret values
`update`	Modify existing secrets
`delete`	Delete secrets
`list`	List secret names
`sudo`	Root-like operations
`deny`	Explicitly deny (overrides others)

Path Patterns

Pattern	Matches
`secrets/myapp`	Exact path
`secrets/myapp/*`	Direct children
`secrets/myapp/+/config`	Single segment wildcard
`secrets/*`	All under secrets/

Access Control Patterns

Environment-Based Access

```
# Development - broad access
path "secrets/data/dev/*" {
  capabilities = ["create", "read", "update", "delete", "list"]
}

# Staging - read + limited write
path "secrets/data/staging/*" {
  capabilities = ["read", "list"]
```

```
}
path "secrets/data/staging/myapp/*" {
  capabilities = ["read", "update"]
}

# Production - read-only for most
path "secrets/data/prod/*" {
  capabilities = ["read"]
}
```

Team-Based Access

```
# Policy: team-platform
path "secrets/data/platform/*" {
  capabilities = ["create", "read", "update", "delete", "list"]
}

# Policy: team-frontend
path "secrets/data/frontend/*" {
  capabilities = ["create", "read", "update", "delete", "list"]
}
path "secrets/data/shared/api-keys" {
  capabilities = ["read"]
}
```

Application-Specific Access

```
# Policy: app-payment-service
path "secrets/data/apps/payment-service/*" {
  capabilities = ["read"]
}
path "secrets/data/shared/stripe-key" {
  capabilities = ["read"]
}
path "secrets/data/shared/database-url" {
  capabilities = ["read"]
}
```

Groups and Entities

Creating Groups

```
# Create group
securevault group create platform-team \
  --policies team-platform,shared-secrets

# Add members
securevault group add-member platform-team \
  --entity-id entity_abc123

# Add member via OIDC
securevault group add-member platform-team \
  --external-group "Platform Team" \
  --mount-accessor oidc_xxx
```

Entity Aliases

Link different auth methods to same identity:

```
# Create entity
securevault entity create "alice@novatech.com"

# Link OIDC identity
securevault entity alias create \
  --entity-id entity_alice \
  --mount-accessor oidc_xxx \
  --name "alice@novatech.com"

# Link AppRole for Alice's service
securevault entity alias create \
  --entity-id entity_alice \
  --mount-accessor approle_xxx \
  --name "alice-service"
```

Access Request Workflow

Self-Service Access Requests

1. User requests access via UI or CLI
2. Request routed to approvers (policy owners)

3. Approvers review and approve/deny
4. Approved access automatically provisioned
5. Time-limited access with auto-expiration

```
# Request access
securevault access request \
  --path "secrets/data/prod/database-creds" \
  --reason "Debugging production issue JIRA-123" \
  --duration 4h

# Approve request (approver)
securevault access approve request_xyz789
```

Emergency Access

For break-glass scenarios:

```
securevault access emergency \
  --path "secrets/data/prod/*" \
  --reason "Critical incident INC-456" \
  --duration 1h
```

Emergency access: - Bypasses normal approval - Triggers immediate alerts - Requires post-incident review - Fully audited

---

Audit and Monitoring

Access Logs

All access attempts are logged:

```
{
  "timestamp": "2024-02-15T10:30:00Z",
  "auth": {
    "method": "approle",
    "entity_id": "entity_abc",
    "policies": ["app-secrets"]
  },
  "request": {
    "operation": "read",
    "path": "secrets/data/myapp/api-key"
```

```
  },
  "response": {
    "status": "success"
  }
}
```

Access Reports

Generate access reports:

```
# Who accessed what
securevault audit report \
  --start-date 2024-02-01 \
  --end-date 2024-02-15 \
  --path "secrets/data/prod/*"

# Access by entity
securevault audit entity-access entity_abc123
```

Alerts

Configure alerts for sensitive access:

```
securevault alert create prod-access \
  --path "secrets/data/prod/*" \
  --operations "read,update,delete" \
  --notify slack:#security-alerts
```

Best Practices

Principle of Least Privilege

- Grant minimum necessary permissions
- Use specific paths, not wildcards
- Prefer read-only where possible
- Time-limit elevated access

Separation of Duties

- Different policies for different roles
- Require multiple approvers for sensitive changes
- Separate dev/staging/prod access

Token Hygiene

- Short TTLs for tokens
- Rotate tokens regularly
- Use AppRole over long-lived tokens
- Revoke unused tokens

Regular Review

- Quarterly access reviews
- Audit policy effectiveness
- Remove stale permissions
- Update policies with org changes

Troubleshooting

"Permission Denied" Errors

1. Check token is valid: `securevault token lookup`
2. Verify policies: `securevault token capabilities <path>`
3. Check path is correct (exact match required)
4. Verify no `deny` policy is in effect

Policy Not Taking Effect

1. Policies are cached briefly - wait 30 seconds
2. Verify policy is attached to token/entity
3. Check for conflicting policies
4. Use `securevault policy test` to simulate

Related Documents: Setup & Installation (PRD-SV-001), Encryption Overview (PRD-SV-005), API Reference (PRD-SV-015)