SecureVault Dynamic Secrets Guide

Document ID: PRD-SV-020 **Last Updated:** 2024-02-10 **Owner:** SecureVault Product Team **Classification:** Public

---

Overview

Dynamic secrets are credentials generated on-demand for a specific client with a limited TTL. Unlike static secrets, dynamic secrets are created when requested and automatically revoked when expired, significantly improving security posture.

Benefits of Dynamic Secrets

Aspect	Static Secrets	Dynamic Secrets
Lifetime	Indefinite	Short-lived (minutes to hours)
Sharing	Often shared	Unique per client
Rotation	Manual/scheduled	Automatic
Audit	Hard to track usage	Full traceability
Compromise impact	Long-term exposure	Limited window

Supported Backends

Database Secrets

Database	Connection Types
PostgreSQL	User/password, IAM
MySQL	User/password
MongoDB	User/password, X.509
Oracle	User/password
Microsoft SQL	User/password
Redis	Password, ACL
Elasticsearch	User/password, API key

Cloud Providers

Provider	Credential Types
AWS	IAM users, STS, assumed roles
Google Cloud	Service accounts, OAuth tokens
Azure	Service principals, managed identity

Other Systems

System	Credential Types
Kubernetes	Service account tokens
SSH	Signed certificates
PKI	X.509 certificates
LDAP	Dynamic accounts
Consul	ACL tokens
Nomad	ACL tokens

Database Dynamic Secrets

PostgreSQL Configuration

Enable the Database Secrets Engine

```
securevault secrets enable database
```

Configure Connection

```
securevault write database/config/postgres-prod \
  plugin_name=postgresql-database-plugin \
  connection_url="postgresql://{{username}}:{{password}}@postgres.prod.internal:5432/mydb?ss
  allowed_roles="app-readonly,app-readwrite" \
  username="vault_admin" \
  password="initial-password"
```

Configure Rotation

```
securevault write database/config/postgres-prod \
  password_policy="db-password-policy"

# Rotate root credentials (recommended)
securevault write -force database/rotate-root/postgres-prod
```

Create Roles

```
# Read-only role
securevault write database/roles/app-readonly \
  db_name=postgres-prod \
  creation_statements="CREATE ROLE \"{{name}}\" WITH LOGIN PASSWORD '{{password}}' VALID UNT
    GRANT SELECT ON ALL TABLES IN SCHEMA public TO \"{{name}}\";" \
  revocation_statements="DROP ROLE IF EXISTS \"{{name}}\";" \
  default_ttl="1h" \
  max_ttl="24h"

# Read-write role
securevault write database/roles/app-readwrite \
  db_name=postgres-prod \
  creation_statements="CREATE ROLE \"{{name}}\" WITH LOGIN PASSWORD '{{password}}' VALID UNT
    GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO \"{{name}}\";" \
  revocation_statements="DROP ROLE IF EXISTS \"{{name}}\";" \
  default_ttl="1h" \
  max_ttl="8h"
```

Request Credentials

```
securevault read database/creds/app-readonly

# Output:
# Key                Value
# ---                -----
# lease_id           database/creds/app-readonly/abc123
# lease_duration     1h
# lease_renewable    true
# password           A1b2C3d4E5f6G7h8
# username           v-app-readonly-xyz789
```

MySQL Configuration

```
# Configure connection
securevault write database/config/mysql-prod \
```

```
  plugin_name=mysql-database-plugin \
  connection_url="{{username}}:{{password}}@tcp(mysql.prod.internal:3306)/" \
  allowed_roles="app-role" \
  username="vault_admin" \
  password="admin-password"

# Create role
securevault write database/roles/app-role \
  db_name=mysql-prod \
  creation_statements="CREATE USER '{{name}}'@'%' IDENTIFIED BY '{{password}}'; \
    GRANT SELECT, INSERT, UPDATE ON mydb.* TO '{{name}}'@'%';" \
  default_ttl="2h" \
  max_ttl="24h"
```

MongoDB Configuration

```
# Configure connection
securevault write database/config/mongodb-prod \
  plugin_name=mongodb-database-plugin \
  connection_url="mongodb://{{username}}:{{password}}@mongo.prod.internal:27017/admin" \
  allowed_roles="app-role" \
  username="vault_admin" \
  password="admin-password"

# Create role
securevault write database/roles/app-role \
  db_name=mongodb-prod \
  creation_statements='{"db": "mydb", "roles": [{"role": "readWrite"}]}' \
  default_ttl="1h" \
  max_ttl="24h"
```

Cloud Provider Dynamic Secrets

AWS Dynamic Credentials

Configure AWS Backend

```
securevault secrets enable aws

securevault write aws/config/root \
  access_key=AKIAIOSFODNN7EXAMPLE \
  secret_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY \
  region=us-west-2
```

IAM User Role

```
securevault write aws/roles/s3-access \
  credential_type=iam_user \
  policy_document=-<<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": ["s3:GetObject", "s3:ListBucket"],
      "Resource": ["arn:aws:s3:::my-bucket/*"]
    }
  ]
}
EOF
```

STS Assumed Role

```
securevault write aws/roles/deploy-role \
  credential_type=assumed_role \
  role_arns=arn:aws:iam::123456789012:role/DeployRole \
  default_sts_ttl=1h \
  max_sts_ttl=4h
```

Request AWS Credentials

```
securevault read aws/creds/s3-access

# Output:
# Key                Value
# ---                -----
# lease_id           aws/creds/s3-access/abc123
# lease_duration     1h
# access_key         AKIAIOSFODNN7EXAMPLE
# secret_key         wJalrXUtnFEMI/K7MDENG
# security_token     (for STS)
```

Google Cloud Dynamic Credentials

```
# Enable GCP secrets engine
securevault secrets enable gcp
```

```
# Configure GCP
securevault write gcp/config \
  credentials=@gcp-credentials.json

# Create service account key role
securevault write gcp/roleset/storage-viewer \
  project="my-project" \
  secret_type="service_account_key" \
  bindings=-<<EOF
resource "//cloudresourcemanager.googleapis.com/projects/my-project" {
  roles = ["roles/storage.objectViewer"]
}
EOF

# Request credentials
securevault read gcp/key/storage-viewer
```

Azure Dynamic Credentials

```
# Enable Azure secrets engine
securevault secrets enable azure

# Configure Azure
securevault write azure/config \
  subscription_id="..." \
  tenant_id="..." \
  client_id="..." \
  client_secret="..."

# Create role
securevault write azure/roles/contributor \
  azure_roles=-<<EOF
[
  {
    "role_name": "Contributor",
    "scope": "/subscriptions/xxx/resourceGroups/my-rg"
  }
]
EOF

# Request credentials
securevault read azure/creds/contributor
```

Application Integration

SDK Usage

Python

```
import hvac

client = hvac.Client(url='https://securevault.novatech.com')
client.token = os.environ['VAULT_TOKEN']

# Get database credentials
creds = client.secrets.database.generate_credentials('app-readonly')

# Use credentials
conn = psycopg2.connect(
    host='postgres.prod.internal',
    database='mydb',
    user=creds['data']['username'],
    password=creds['data']['password']
)

# When done, revoke lease (optional - will auto-expire)
client.sys.revoke_lease(creds['lease_id'])
```

Node.js

```
const vault = require('node-vault')({
  endpoint: 'https://securevault.novatech.com',
  token: process.env.VAULT_TOKEN
});

async function getDbConnection() {
  // Get dynamic credentials
  const creds = await vault.read('database/creds/app-readonly');

  // Create connection
  const pool = new Pool({
    host: 'postgres.prod.internal',
    database: 'mydb',
    user: creds.data.username,
    password: creds.data.password
  });

  // Set up renewal
```

```
  const leaseId = creds.lease_id;
  const renewInterval = setInterval(async () => {
    try {
      await vault.write(`sys/leases/renew`, { lease_id: leaseId });
    } catch (err) {
      clearInterval(renewInterval);
      // Get new credentials
    }
  }, (creds.lease_duration / 2) * 1000);

  return pool;
}
```

Go

```
package main

import (
    "database/sql"
    vault "github.com/hashicorp/vault/api"
    _ "github.com/lib/pq"
)

func getDbCredentials(client *vault.Client) (*sql.DB, error) {
    // Read dynamic credentials
    secret, err := client.Logical().Read("database/creds/app-readonly")
    if err != nil {
        return nil, err
    }

    username := secret.Data["username"].(string)
    password := secret.Data["password"].(string)

    // Connect to database
    connStr := fmt.Sprintf(
        "host=postgres.prod.internal user=%s password=%s dbname=mydb sslmode=require",
        username, password,
    )

    db, err := sql.Open("postgres", connStr)
    if err != nil {
        return nil, err
    }

    // Start renewal goroutine
```

```
    go func() {
        renewer, _ := client.NewLifetimeWatcher(&vault.LifetimeWatcherInput{
            Secret: secret,
        })
        go renewer.Start()
        defer renewer.Stop()

        for {
            select {
            case <-renewer.DoneCh():
                // Lease expired, get new credentials
                return
            case <-renewer.RenewCh():
                // Renewed successfully
            }
        }
    }()

    return db, nil
}
```

Lease Management

Understanding Leases

Every dynamic secret has an associated lease: - **lease_id**: Unique identifier for the lease - **lease_duration**: How long the credentials are valid - **lease_renewable**: Whether the lease can be extended

Renewing Leases

```
# Renew a lease
securevault lease renew database/creds/app-readonly/abc123

# Renew with specific TTL
securevault lease renew -increment=2h database/creds/app-readonly/abc123
```

Revoking Leases

```
# Revoke specific lease
securevault lease revoke database/creds/app-readonly/abc123
```

```
# Revoke all leases for a path
securevault lease revoke -prefix database/creds/app-readonly

# Revoke all database leases
securevault lease revoke -prefix database/
```

Lease Limits

```
# securevault-config.yaml
lease_limits:
  database:
    default_ttl: 1h
    max_ttl: 24h
    max_leases_per_role: 100
  aws:
    default_ttl: 1h
    max_ttl: 4h
    max_leases_per_role: 50
```

Best Practices

TTL Configuration

Use Case	Recommended TTL
CI/CD jobs	15-30 minutes
Short-running scripts	1 hour
Application services	1-8 hours
Batch processing	Duration of job + buffer

Security Recommendations

1. **Use shortest practical TTL**
   - Reduces exposure window
   - Forces regular renewal
2. **Implement credential renewal**
   - Don't wait for expiration
   - Renew at 50% of TTL

3. **Handle credential rotation gracefully**
    - Implement connection retry logic
    - Use connection pooling with refresh
4. **Audit dynamic secret usage**
    - Monitor lease creation/revocation
    - Alert on unusual patterns
5. **Revoke credentials when done**
    - Don't rely only on TTL expiration
    - Explicit revocation is cleaner

Application Patterns

```python
# Pattern: Connection pool with automatic credential refresh
class DynamicDbPool:
    def __init__(self, vault_client, role):
        self.vault = vault_client
        self.role = role
        self.pool = None
        self.lease_id = None
        self._refresh_credentials()
        self._start_renewal_thread()

    def _refresh_credentials(self):
        creds = self.vault.secrets.database.generate_credentials(self.role)
        self.lease_id = creds['lease_id']
        self.lease_duration = creds['lease_duration']

        # Create new pool with new credentials
        new_pool = create_pool(creds['data'])
        old_pool = self.pool
        self.pool = new_pool

        # Gracefully close old pool
        if old_pool:
            old_pool.close()

    def _start_renewal_thread(self):
        def renew():
            while True:
                time.sleep(self.lease_duration * 0.5)
                try:
                    self.vault.sys.renew_lease(self.lease_id)
                except:
```

```
                    self._refresh_credentials()

        thread = threading.Thread(target=renew, daemon=True)
        thread.start()

    def get_connection(self):
        return self.pool.getconn()
```

Monitoring

Metrics

Metric	Description
`secret.dynamic.create`	Dynamic secrets created
`secret.lease.renew`	Lease renewals
`secret.lease.expire`	Lease expirations
`secret.lease.revoke`	Lease revocations
`database.creds.created`	Database credentials issued

Alerts

```
alerts:
  - name: high-lease-creation
    condition: rate(secret.dynamic.create) > 100/min
    severity: warning
    notify: [security@novatech.com]

  - name: lease-renewal-failures
    condition: rate(secret.lease.renew.failure) > 10/min
    severity: critical
    notify: [ops@novatech.com]
```

Troubleshooting

Common Issues

Credentials expire during long operations: - Increase TTL for the role - Implement renewal in your application - Use longer max_ttl if needed

Too many database connections: - Reduce max_leases_per_role - Use connection pooling - Ensure proper credential revocation

Credential creation slow: - Check database admin account permissions - Verify network connectivity - Review database server load

---

Related Documents: Getting Started (PRD-SV-001), Authentication (PRD-SV-010), Rotation Policies (PRD-SV-022)