Configuring OpenID Connect in HashiCorp Vault

In this article

Use OpenID Connect within your workflows to authenticate with HashiCorp Vault.

Overview

OpenID Connect (OIDC) allows your GitHub Actions workflows to authenticate with a HashiCorp Vault to retrieve secrets.

This guide gives an overview of how to configure HashiCorp Vault to trust GitHub's OIDC as a federated identity, and demonstrates how to use this configuration in the hashicorp/vault-action action to retrieve secrets from HashiCorp Vault.

Prerequisites

- To learn the basic concepts of how GitHub uses OpenID Connect (OIDC), and its architecture and benefits, see "About security hardening with OpenID Connect."
- Before proceeding, you must plan your security strategy to ensure that access tokens are only allocated in a predictable way. To control how your cloud provider issues access tokens, you **must** define at least one condition, so that untrusted repositories can’t request access tokens for your cloud resources. For more information, see "About security hardening with OpenID Connect."

Adding the identity provider to HashiCorp Vault

To use OIDC with HashiCorp Vault, you will need to add a trust configuration for the GitHub OIDC provider. For more information, see the HashiCorp Vault documentation.

To configure your Vault server to accept JSON Web Tokens (JWT) for authentication:

1. Enable the JWT `auth` method, and use `write` to apply the configuration to your Vault. For `oidc_discovery_url` and `bound_issuer` parameters, use `https://token.actions.githubusercontent.com`. These parameters allow the Vault server to verify the received JSON Web Tokens (JWT) during the authentication process.

   Shell

```
vault auth enable jwt
```

Shell

```
vault write auth/jwt/config \
  bound_issuer="https://token.actions.githubusercontent.com" \
  oidc_discovery_url="https://token.actions.githubusercontent.com"
```

> **Note:** If a unique issuer URL for an enterprise was set using the REST API (as described in "About security hardening with OpenID Connect"), the values for `bound_issuer` and `oidc_discover_url` must match that unique URL. For example, for an enterprise named `octocat` that uses the unique issuer URL, `bound_issuer` and `oidc_discovery_url` must be set to `https://token.actions.githubusercontent.com/octocat`.

2. Configure a policy that only grants access to the specific paths your workflows will use to retrieve secrets. For more advanced policies, see the HashiCorp Vault Policies documentation.

Shell

```
vault policy write myproject-production - <<EOF
# Read-only permission on 'secret/data/production/*' path

path "secret/data/production/*" {
  capabilities = [ "read" ]
}
EOF
```

3. Configure roles to group different policies together. If the authentication is successful, these policies are attached to the resulting Vault access token.

Shell

```
vault write auth/jwt/role/myproject-production -<<EOF
{
  "role_type": "jwt",
  "user_claim": "actor",
  "bound_claims": {
    "repository": "user-or-org-name/repo-name"
  },
  "policies": ["myproject-production"],
  "ttl": "10m"
}
EOF
```

- `ttl` defines the validity of the resulting access token.
- Ensure that the `bound_claims` parameter is defined for your security requirements, and has at least one condition. Optionally, you can also set the `bound_subject` as well as the `bound_audiences` parameter.
- To check arbitrary claims in the received JWT payload, the `bound_claims` parameter contains a set of claims and their required values. In the above example, the role will accept any incoming authentication requests from the `repo-name` repository owned by the `user-or-org-name` account.
- To see all the available claims supported by GitHub's OIDC provider, see "About

security hardening with OpenID Connect."

For more information, see the HashiCorp Vault documentation.

Updating your GitHub Actions workflow

To update your workflows for OIDC, you will need to make two changes to your YAML:

1. Add permissions settings for the token.
2. Use the `hashicorp/vault-action` action to exchange the OIDC token (JWT) for a cloud access token.

To add OIDC integration to your workflows that allow them to access secrets in Vault, you will need to add the following code changes:

- Grant permission to fetch the token from the GitHub OIDC provider:
  - The workflow needs `permissions:` settings with the `id-token` value set to `write`. This lets you fetch the OIDC token from every job in the workflow.
- Request the JWT from the GitHub OIDC provider, and present it to HashiCorp Vault to receive an access token:
  - You can use the `hashicorp/vault-action` action to fetch the JWT and receive the access token from Vault, or you could use the Actions toolkit to fetch the tokens for your job.

This example demonstrates how to use OIDC with the official action to request a secret from HashiCorp Vault.

Adding permissions settings

The job or workflow run requires a `permissions` setting with `id-token: write`. You won't be able to request the OIDC JWT ID token if the `permissions` setting for `id-token` is set to `read` or `none`.

The `id-token: write` setting allows the JWT to be requested from GitHub's OIDC provider using one of these approaches:

- Using environment variables on the runner (`ACTIONS_ID_TOKEN_REQUEST_URL` and `ACTIONS_ID_TOKEN_REQUEST_TOKEN`).
- Using `getIDToken()` from the Actions toolkit.

If you need to fetch an OIDC token for a workflow, then the permission can be set at the workflow level. For example:

YAML

```yaml
permissions:
  id-token: write # This is required for requesting the JWT
  contents: read  # This is required for actions/checkout
```

If you only need to fetch an OIDC token for a single job, then this permission can be set within that job. For example:

YAML

```yaml
permissions:
  id-token: write # This is required for requesting the JWT
```

You may need to specify additional permissions here, depending on your workflow's requirements.

For reusable workflows that are owned by the same user, organization, or enterprise as the caller workflow, the OIDC token generated in the reusable workflow can be accessed from the caller's context. For reusable workflows outside your enterprise or organization, the `permissions` setting for `id-token` should be explicitly set to `write` at the caller workflow level or in the specific job that calls the reusable workflow. This ensures that the OIDC token generated in the reusable workflow is only allowed to be consumed in the caller workflows when intended.

For more information, see "Reusing workflows."

> **Note**:
>
> When the `permissions` key is used, all unspecified permissions are set to *no access*, with the exception of the metadata scope, which always gets *read* access. As a result, you may need to add other permissions, such as `contents: read`. See Automatic token authentication for more information.

Requesting the access token

The `hashicorp/vault-action` action receives a JWT from the GitHub OIDC provider, and then requests an access token from your HashiCorp Vault instance to retrieve secrets. For more information, see the HashiCorp Vault GitHub Action documentation.

This example demonstrates how to create a job that requests a secret from HashiCorp Vault.

- `<Vault URL>`: Replace this with the URL of your HashiCorp Vault.
- `<Vault Namespace>`: Replace this with the Namespace you've set in HashiCorp Vault. For example: `admin`.
- `<Role name>`: Replace this with the role you've set in the HashiCorp Vault trust relationship.
- `<Secret-Path>`: Replace this with the path to the secret you're retrieving from HashiCorp Vault. For example: `secret/data/production/ci npmToken`.

YAML

```yaml
jobs:
  retrieve-secret:
    runs-on: ubuntu-latest
    permissions:
      id-token: write
      contents: read
    steps:
      - name: Retrieve secret from Vault
        uses: hashicorp/vault-action@v2.4.0
          with:
            method: jwt
            url: <Vault URL>
            namespace: <Vault Namespace - HCP Vault and Vault Enterprise only>
            role: <Role name>
            secrets: <Secret-Path>

      - name: Use secret from Vault
        run: |
          # This step has access to the secret retrieved above; see
hashicorp/vault-action for more details.
```

Note:

- If your Vault server is not accessible from the public network, consider using a self-hosted runner with other available Vault auth methods. For more information, see "About self-hosted runners."
- `<Vault Namespace>` must be set for a Vault Enterprise (including HCP Vault) deployment. For more information, see Vault namespace.

Revoking the access token

By default, the Vault server will automatically revoke access tokens when their TTL is expired, so you don't have to manually revoke the access tokens. However, if you do want to revoke access tokens immediately after your job has completed or failed, you can manually revoke the issued token using the Vault API.

1. Set the `exportToken` option to `true` (default: `false`). This exports the issued Vault access token as an environment variable: `VAULT_TOKEN`.
2. Add a step to call the Revoke a Token (Self) Vault API to revoke the access token.

YAML

```yaml
jobs:
  retrieve-secret:
    runs-on: ubuntu-latest
    permissions:
      id-token: write
      contents: read
    steps:
      - name: Retrieve secret from Vault
        uses: hashicorp/vault-action@v2.4.0
          with:
            exportToken: true
            method: jwt
            url: <Vault URL>
            role: <Role name>
            secrets: <Secret-Path>

      - name: Use secret from Vault
        run: |
          # This step has access to the secret retrieved above; see
hashicorp/vault-action for more details.

      - name: Revoke token
        # This step always runs at the end regardless of the previous steps
result
        if: always()
        run: |
          curl -X POST -sv -H "X-Vault-Token: ${{ env.VAULT_TOKEN }}" \
            <Vault URL>/v1/auth/token/revoke-self
```

Further reading

- Using OpenID Connect with reusable workflows
- About self-hosted runners