PingOne

These pages show you how to use PingGateway with PingOne for SSO and API security. The examples target PingOne evaluators, administrators, and architects.

Before you start

Unless otherwise stated, the examples in these pages assume you have:

- PingGateway installed.
- The sample application installed.
- A PingOne test environment as described in Create a PingOne test environment.
- A PingOne test user as described in Add a PingOne test user.

Create a PingOne test environment

In the PingOne console, create a test environment with the following values:

- **Select a solution for your Environment**: Build your own solution
- **Select solution(s) for your Environment**: `PingOne SSO`
- **ENVIRONMENT NAME**: `Test environment`
- **DESCRIPTION**: OIDC `Test environment`
- **ENVIRONMENT TYPE**: `Sandbox`

Learn more in the PingOne documentation on Adding an environment.

Add a PingOne test user

1. In the PingOne test environment under **Directory** > **Users**, add a user.

   To match a user known to the sample application, use the following values:

   - **Given Name**: `Wilhelm`
   - **Family Name**: `Wolkig`
   - **Username**: `wolkig`
   - **Email**: `wolkig@example.com`

- **Password**: Click **Generate Password** and record the generated password.

  Learn more in the PingOne documentation on Adding a user.

2. Under **Settings** > **Environment Properties** > **URLs**, copy the **Self-Service Url** for your environment.
3. In your browser's privacy or incognito mode, go to the URL you copied and sign on as the user you created.
4. When prompted, change the password so you won't have to change it again when trying an example.

   To match the credentials to those known to the sample application, set the new password to `Geh3imnis!`.

PingOne as OIDC provider

This example sets up PingOne as an OIDC provider with PingGateway as a relying party.

Prepare PingOne

After you prepare the PingOne environment and test user, follow these steps to create a PingOne OIDC web application:

1. In the test environment, create a web application with the following values:
   - **Application Name**: `oidc_client`
   - **Description**: `OIDC client`
   - **Application Type**: `OIDC Web App`
2. In the application, select the **Overview** panel and click **Protocol OpenID Connect**.
3. In the **Redirect URIs** field, add `https://ig.example.com:8443/home/id_token/callback` and save the application.
4. In the **Overview** panel, click **Resource Access**, select the `email` and `profile` scopes in addition to the default `openid` scope, and click **Save**.

   Learn more from the PingOne documentation on Editing an application - OIDC.
5. At the top-right of the page, click the slider to enable the application.
6. Go to the **Configuration** panel and make a note of the following values in the **URLs** drop-down list:
   - **OIDC Discovery Endpoint**
   - **Client ID**
   - **Client Secret**

You need the values to set up PingGateway.

Prepare PingGateway

1. Add the following basic PingGateway configuration if you have not already done so:
   a. Set up PingGateway for HTTPS, as described in Configure PingGateway for TLS (server-side).
   b. Add the following route to PingGateway to serve the sample application .css and other static resources:

      Linux Windows

      ```
      $HOME/.openig/config/routes/00-static-resources.json
      ```

      ```
      {
        "name" : "00-static-resources",
        "baseURI" : "http://app.example.com:8081",
        "condition": "${find(request.uri.path,'^/css') or
      matchesWithRegex(request.uri.path, '^/.*\\\\.ico$') or
      matchesWithRegex(request.uri.path, '^/.*\\\\.gif$')}",
        "handler": "ReverseProxyHandler"
      }
      ```

2. Base64-encode the OIDC application's **Client Secret** and set the value as an environment variable:

   ```
   $ export OIDC_SECRET_ID='<base64-encoded-client-secret>'
   ```

3. Add the following route to PingGateway, replacing the following property values with those of the OIDC application:
   - OIDC_Discovery_Endpoint: The OIDC discovery endpoint for the client application you registered.
   - Client_ID: The client ID of the application.

   Linux Windows

   ```
   $HOME/.openig/config/routes/oidc-ping.json
   ```

   ```
   {
     "name": "oidc-ping",
   ```

```
  "condition": "${find(request.uri.path,
'^/home/id_token')}",
  "properties": {
    "OIDC_Discovery_Endpoint": "OIDC Discovery endpoint of
the web app",
    "Client_ID": "Client ID of the web app"
  },
  "heap": [
    {
      "name": "SystemAndEnvSecretStore-1",
      "type": "SystemAndEnvSecretStore"
    },
    {
      "name": "AuthenticatedRegistrationHandler-1",
      "type": "Chain",
      "config": {
        "filters": [
          {
            "name":
"ClientSecretBasicAuthenticationFilter-1",
            "type":
"ClientSecretBasicAuthenticationFilter",
            "config": {
              "clientId": "&{Client_ID}",
              "clientSecretId": "oidc.secret.id",
              "secretsProvider": "SystemAndEnvSecretStore-
1"
            }
          }
        ],
        "handler": "ForgeRockClientHandler"
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "AuthorizationCodeOAuth2ClientFilter-1",
          "type": "AuthorizationCodeOAuth2ClientFilter",
          "config": {
            "clientEndpoint": "/home/id_token",
            "failureHandler": {
              "type": "StaticResponseHandler",
              "config": {
                "status": 500,
                "headers": {
                  "Content-Type": [
                    "text/html; charset=UTF-8"
                  ]
                },
                "entity": "<html><body>Error in OAuth 2.0
setup.<br> ${contexts.oauth2Failure.exception.message}
</body></html>"
              }
            },
            "registrations": [
              {
                "name": "oauth2-client",
                "type": "ClientRegistration",
                "config": {
                  "clientId": "${Client_ID}",
                  "issuer": {
                    "name": "PingOne",
                    "type": "Issuer",
                    "config": {
                      "wellKnownEndpoint": "&
{OIDC_Discovery_Endpoint}"
                    }
                  },
                  "scopes": [
                    "openid",
                    "profile",
                    "email"
                  ],
                  "authenticatedRegistrationHandler":
"AuthenticatedRegistrationHandler-1"
                }
              }
            ],
            "requireHttps": false,
            "cacheExpiration": "disabled"
          }
        }
      ],
      "handler": {
        "type": "StaticResponseHandler",
        "name": "HTMLResponse",
        "config": {
          "status": 200,
          "entity": "<!DOCTYPE html><html><head>
<title>Authentication Success</title></head><body>
<p>Welcome,
${attributes.openid.user_info.preferred_username}!</p>
<code>${attributes.openid.user_info}</code></body>
</html>",
          "headers": {
            "Content-Type": [
              "text/html"
            ]
          }
        }
      }
    }
  }
}
```

4. Restart PingGateway.

Validation

1. In your browser's privacy or incognito mode, go to https://ig.example.com:8443/home/id_token.

   PingOne displays the sign-on page.

2. Sign on to PingOne as the test user.

   The route displays a welcome page:

   ```
   Welcome, wolkig!

   {sub=..., preferred_username=wolkig, given_name=Wilhelm,
   updated_at=..., family_name=Wolkig, email=wolkig@example.com,
   ...}
   ```

PingOne as SAML IDP

This example shows how to use PingOne as the identity provider with unsigned/unencrypted assertions.

Before you start

1. Add the following basic PingGateway configuration if you have not already done so:
   a. Set up PingGateway for HTTPS, as described in Configure PingGateway for TLS (server-side).
   b. Add the following route to PingGateway to serve the sample application .css and other static resources:

      Linux | Windows

      ```
      $HOME/.openig/config/routes/00-static-resources.json
      ```

      ```
      {
        "name" : "00-static-resources",
        "baseURI" : "http://app.example.com:8081",
        "condition": "${find(request.uri.path,'^/css') or
      matchesWithRegex(request.uri.path, '^/.*\\\\.ico$') or
      matchesWithRegex(request.uri.path, '^/.*\\\\.gif$')}",
        "handler": "ReverseProxyHandler"
      }
      ```

2. Set up the network:

   Add `sp.example.com` to your `/etc/hosts` file:

   ```
   127.0.0.1 localhost am.example.com ig.example.com
   app.example.com sp.example.com
   ```

   Traffic to the application is proxied through PingGateway, using the host name `sp.example.com`.

3. Save the sp.xml file as the SAML service provider configuration file `$HOME/.openig/SAML/sp.xml`.

Prepare PingOne

1. Prepare the PingOne environment and test user.

   Make sure you match the test user's credentials to those known to the sample application.

2. In the PingOne test environment, create a SAML web application with the following values:
   - **Application Name**: `saml_app`
   - **Description**: `SAML application`

- **Application Type**: `SAML Application`

3. In the application, select the **Import Metadata** panel, add the SAML configuration file `sp.xml` and save the application.
4. On the **Attribute Mappings** panel, click ✎ (edit) and add the following mappings:

saml_app	PingOne
cn	Given Name
sn	Family Name

5. On the **Configuration** panel, click ✎ (edit) and set the SLO BINDING's **SUBJECT NAMEID FORMAT** to `urn:oasis:names:tc:SAML:2.0:nameid-format:transient`.
6. On the **Configuration** panel, click **Download Metadata** and save the downloaded file as the identity provider configuration file `$HOME/.openig/SAML/idp.xml`.
7. On the **Configuration** panel, record the **Initiate Single Sign-on URL**.

   You need the value to set up PingGateway.
8. At the top-right of the page, click the slider to enable the application.

Learn more from the PingOne documentation Add a SAML application.

Prepare PingGateway

1. Copy the following example SAML configuration files to `$HOME/.openig/SAML` and edit them to match your configuration:

File	Required changes
`FederationConfig.properties`	None
`fedlet.cot`	Replace idp-entityID with the value of `EntityDescriptor entityID` in `idp.xml`.
`idp-extended.xml`	Replace idp-entityID with the value of `EntityDescriptor entityID` in `idp.xml`.
`sp-extended.xml`	None

2. Make sure the PingGateway configuration at `$HOME/.openig/SAML` contains the following files:

```
$ ls -l $HOME/.openig/SAML
FederationConfig.properties
fedlet.cot
idp-extended.xml
idp.xml
sp-extended.xml
sp.xml
```

3. Add the following route to PingGateway:

Linux | Windows

```
$HOME/.openig/config/routes/saml-filter.json
```

```
{
  "name": "saml-filter",
  "baseURI": "http://app.example.com:8081",
  "condition": "${find(request.uri.path, '^/home')}",
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "SamlFilter",
          "type": "SamlFederationFilter",
          "config": {
            "assertionMapping": {
              "name": "cn",
              "surname": "sn"
            },
            "subjectMapping": "sp-subject-name",
            "redirectURI": "/home/saml-filter"
          }
        },
        {
          "name": "SetSamlHeaders",
          "type": "HeaderFilter",
          "config": {
            "messageType": "REQUEST",
            "add": {
              "x-saml-cn": [ "${toString(session.name)}" ],
              "x-saml-sn": [ "${toString(session.surname)}" ]
            }
          }
        }
      ],
      "handler": "ReverseProxyHandler"
    }
  }
}
```

4. Restart PingGateway.

Validation

Test IDP-initiated login:

1. In your browser's privacy or incognito mode, go to the URL given by the web application property **Initiate Single Sign-on URL**.

   PingOne displays the sign-on page.

2. Sign on to PingOne as the test user.

   PingGateway displays the sample application home page.

Test SP-initiated login:

1. In your browser's privacy or incognito mode, go to https://sp.example.com:8443/home.
2. Sign on as the test user.

   The request is redirected to the sample application.

   PingGateway displays the sample application home page.

TIP

If a request returns an HTTP 414 URI Too Long error, read URI Too Long error.

PingOne Authorize integration

Use PingOne Authorize with PingGateway to protect APIs and web applications. When you add a PingOneApiAccessManagementFilter to a route, PingGateway requests an authorization decision from PingOne Authorize based on the request and associated access token.

This example uses PingOne as an OIDC provider issuing an ID token with an access token. PingGateway enforces authorization decisions from PingOne Authorize. In this example, the decision to authorize access to a protected application depends on an access token scope.

Before you begin

If you haven't already done so:

- Install PingGateway.
- Install the sample application.
- Prepare a PingOne test environment and a test user.

Learn more in the overview page for these examples.

Add an OIDC client

Follow these steps to create a PingOne OIDC web application:

1. In the test environment, create a web application with the following values:
   - **Application Name**: `oidc_client`
   - **Description**: `OIDC client`
   - **Application Type**: **OIDC Web App**
2. On the **Overview** tab of the details panel, click **Protocol OpenID Connect**.
3. In the **Redirect URIs** field, enter `https://ig.example.com:8443/home/sso/callback` . Click **Save**.
4. At the top-right of the application's profile, click the toggle to enable the application.
5. On the **Configuration** tab, make a note of the following values in the **URLs** list:
   - **OIDC Discovery Endpoint**
   - **Client ID**
   - **Client Secret**

   You need the values to set up PingGateway.

Enable PingOne Authorize

Add PingOne Authorize to the test environment.

Add an API gateway

PingGateway uses a PingOne API gateway account to authenticate to PingOne Authorize:

1. In the test environment, go to **Authorization** > **API Gateways** and click + .

2. Enter PingGateway as the **Name**. Click **Save**.
3. Next to **Credentials**, click + and copy the API gateway credential.

   You need the credential value to set up PingGateway.

Add an API service

In this example, PingOne Authorize uses a PingOne API service definition to reach authorization decisions.

The API service holds references to the following:

- The base URL for PingGateway, which is the base URL for what a user sees in their browser.
- The base URLs for the protected sample application, which is the object of an authorization decision.
- The resource to protect, which is the PingGateway URL proxying for the sample application.
- The scope required for access in this example.
    1. In the test environment, go to **Authorization** > **API Services** and click + .
    2. On the **New API Service** panel, add the following settings, then click **Save**. Accept the default values for the settings not listed here:

Setting	Value
Name	`PingGateway`
Base URLs	`https://app.example.com:8444` (for the sample application) `https://ig.example.com:8443` (for PingGateway)
Enable Custom Policies	Select this.

    3. Click **Deploy** so you can use the new API service.

Update the automatically created resource

When you added the PingGateway API service, PingOne automatically created a corresponding resource by default. Update the resource so its audience (the `aud` in access tokens) reflects the PingGateway URL and so that it requires a `read` scope to get it:

1. In the test environment, go to **Applications** > **Resources** and click the PingGateway resource.
2. On the **Overview** tab, click the **Pencil** icon, and, in the **Audience** field, enter `https://ig.example.com:8443`. Click **Save**.
3. On the **Scopes** tab, click **Add Scope**, and in the **Scope Name** field, enter `read`. Click **Save**.

   This adds the `read` scope to the PingGateway service.
4. Go to **Applications** > **Applications** and click **oidc_client**.
5. On the **Resources** tab, edit **Allowed Scopes**, and select the **read** scope checkbox. Click **Save**.

Configure PingGateway

1. Prepare the secrets PingGateway needs to act as an OIDC RP and an API gateway when making requests to PingOne.

   Base64-encode the API gateway's credential and the OIDC application's client secret and set both values as environment variables:

   ```
   $ export GATEWAY_SECRET_ID='<base64-encoded-api-gateway-
   credential>'
   $ export OIDC_SECRET_ID='<base64-encoded-client-secret>'
   ```
2. Restart PingGateway to read the environment variables.
3. Add the following route to PingGateway, replacing the property values to match those from the test environment:

   Linux Windows

   ```
   $HOME/.openig/config/routes/oidc-ping.json
   ```

   ```
   {
     "name": "pingone-aam",
     "condition": "${find(request.uri.path, '^/home/sso')}",
     "baseURI": "https://app.example.com:8444",
     "properties": {
       "gatewayServiceUrl": "https://http-access-
   api.pingone.eu/v1/environments/test-environment-id",
       "oidcClientId": "oidc-client-id",
       "oidcWellKnownEndpoint": "https://auth.pingone.eu/test-
   environment-id/as/.well-known/openid-configuration"
   ```

```
  },
  "heap": [
    {
      "name": "BlindTrustReverseProxyHandler",
      "type": "ReverseProxyHandler",
      "comment": "For evaluation and testing only",
      "config": {
        "tls": {
          "type": "ClientTlsOptions",
          "config": {
            "trustManager": {
              "type": "TrustAllManager"
            },
            "hostnameVerifier": "ALLOW_ALL"
          }
        }
      }
    },
    {
      "name": "SystemAndEnvSecretStore-1",
      "type": "SystemAndEnvSecretStore"
    },
    {
      "name": "AuthenticatedRegistrationHandler-1",
      "type": "Chain",
      "config": {
        "filters": [
          {
            "name": "ClientSecretBasicAuthenticationFilter-1",
            "type": "ClientSecretBasicAuthenticationFilter",
            "config": {
              "clientId": "&{oidcClientId}",
              "clientSecretId": "oidc.secret.id",
              "secretsProvider": "SystemAndEnvSecretStore-1"
            }
          }
        ],
        "handler": "ForgeRockClientHandler"
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "type": "AuthorizationCodeOAuth2ClientFilter",
          "config": {
            "clientEndpoint": "/home/sso",
            "failureHandler": {
              "type": "StaticResponseHandler",
              "config": {
                "status": 500,
                "headers": {
                  "Content-Type": [
                    "text/plain"
                  ]
                },
                "entity": "Error:
${contexts.oauth2Failure.error}\nDescription:
${contexts.oauth2Failure.description}"
              }
            },
            "registrations": [
              {
                "type": "ClientRegistration",
                "config": {
                  "clientId": "&{oidcClientId}",
                  "issuer": {
                    "type": "Issuer",
                    "config": {
                      "wellKnownEndpoint": "&
{oidcWellKnownEndpoint}"
                    }
                  },
                  "scopes": [
                    "openid",
                    "read"
                  ],
                  "authenticatedRegistrationHandler":
"AuthenticatedRegistrationHandler-1"
                }
              }
            ]
          }
        },
        {
          "type": "PingOneApiAccessManagementFilter",
          "config": {
            "gatewayServiceUri": "&{gatewayServiceUrl}",
            "secretsProvider": "SystemAndEnvSecretStore-1",
            "gatewayCredentialSecretId": "gateway.secret.id",
            "accessToken":
"${attributes.openid.access_token}",
            "_sidebandHandler": {
              "_comment": "s/_sidebandHandler/sidebandHandler/
to troubleshoot AAM decisions",
              "type": "ClientHandler",
              "capture": "all"
            }
          }
        }
      ],
      "handler": "BlindTrustReverseProxyHandler"
    }
  }
}
```

Notice the following features of the route:

- The route properties use the settings you collected from the PingOne environment.
- The heap has:
  - A handler to trust the self-signed sample application certificate for HTTPS blindly.

    NOTE

    In production deployments, use certificates you don't have to trust blindly.

  - A `SystemAndEnvSecretStore` to read get the base64-encoded secrets from the environment variables.
  - A handler to get an ID token as the OIDC client.
- The `AuthorizationCodeOAuth2ClientFilter` gets the ID token with the `read` scope.
- The `PingOneApiAccessManagementFilter` makes the authorization decision request to PingOne Authorize. It sends the access token from the OIDC ID token.
- On success, the `BlindTrustReverseProxyHandler` gets the resource from the sample application.

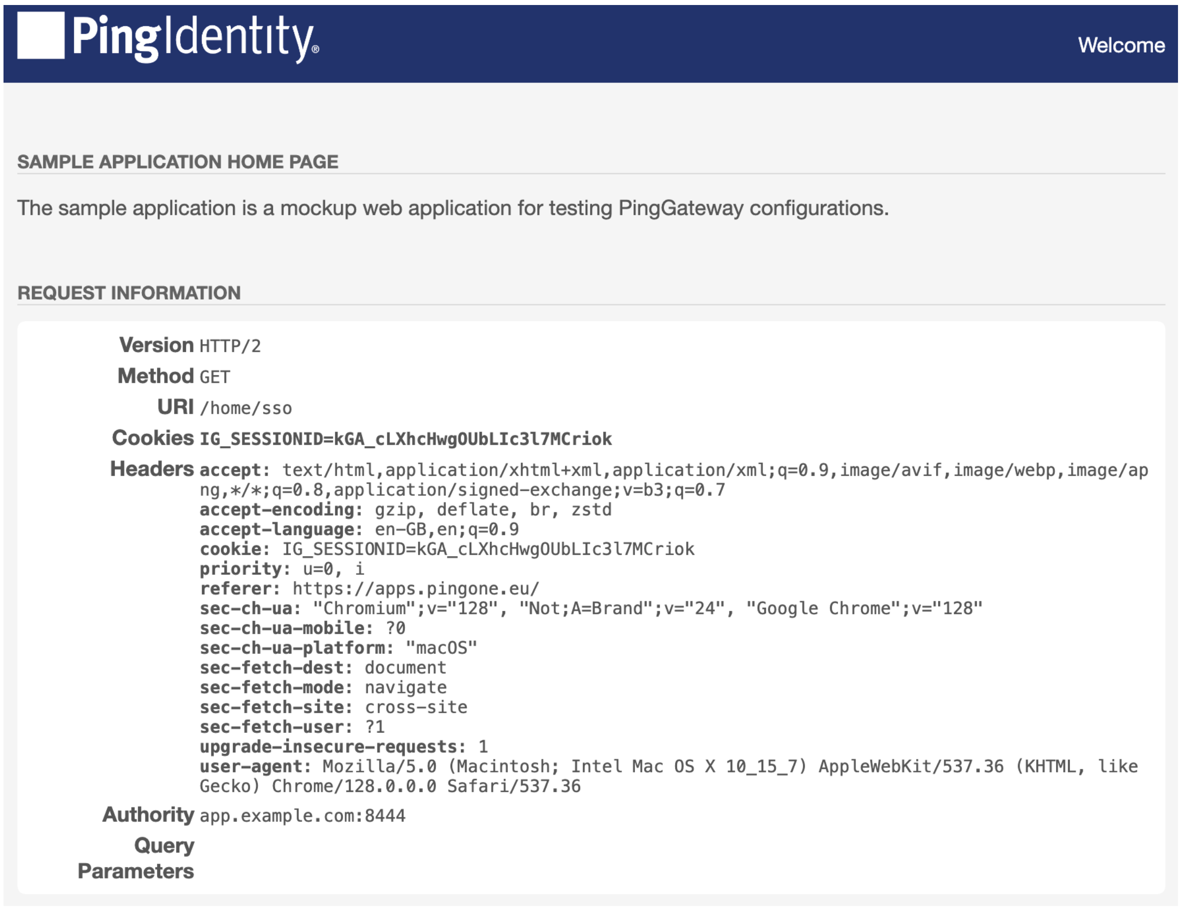
Validation

1. In your browser's privacy or incognito mode, go to https://ig.example.com:8443/home/sso.

   PingOne displays the sign-on page.

2. Sign on to PingOne as the test user.

   PingGateway displays the sample application home page:



SAMPLE APPLICATION HOME PAGE

The sample application is a mockup web application for testing PingGateway configurations.

REQUEST INFORMATION

```
   Version HTTP/2
    Method GET
       URI /home/sso
   Cookies IG_SESSIONID=kGA_cLXhcHwgOUbLIc3l7MCriok
   Headers accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/ap
           ng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
           accept-encoding: gzip, deflate, br, zstd
           accept-language: en-GB,en;q=0.9
           cookie: IG_SESSIONID=kGA_cLXhcHwgOUbLIc3l7MCriok
           priority: u=0, i
           referer: https://apps.pingone.eu/
           sec-ch-ua: "Chromium";v="128", "Not;A=Brand";v="24", "Google Chrome";v="128"
           sec-ch-ua-mobile: ?0
           sec-ch-ua-platform: "macOS"
           sec-fetch-dest: document
           sec-fetch-mode: navigate
           sec-fetch-site: cross-site
           sec-fetch-user: ?1
           upgrade-insecure-requests: 1
           user-agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like
           Gecko) Chrome/128.0.0.0 Safari/537.36
 Authority app.example.com:8444
     Query
Parameters
```

Troubleshooting

If you get unexpected errors, try these debugging options:

- Update the `PingOneApiAccessManagementFilter` configuration to enable the capture decorator.

  Change `"_sidebandHandler"` to `"sidebandHandler"` and save your work to let PingGateway reload the route.

- In the test environment, go to **Authorization** > **Recent Decisions**, select the PingGateway decision endpoint, and select a decision to visualize.

  PingOne shows you how PingOne Authorize arrived at the authorization decision.

  On the **Request** and **Response** tabs, you find detailed traces of the parameters for each.

User experience

When PingOne Authorize denies access or the example route fails to get an access token, the browser displays text, which isn’t user-friendly.

Consider using a StaticResponseHandler to show a more friendly page to the user.

PingOne Protect integration

Use PingOne Protect risk evaluations with web applications protected by PingGateway. PingGateway routes requests based on the level of risk PingOne Protect associates with them.

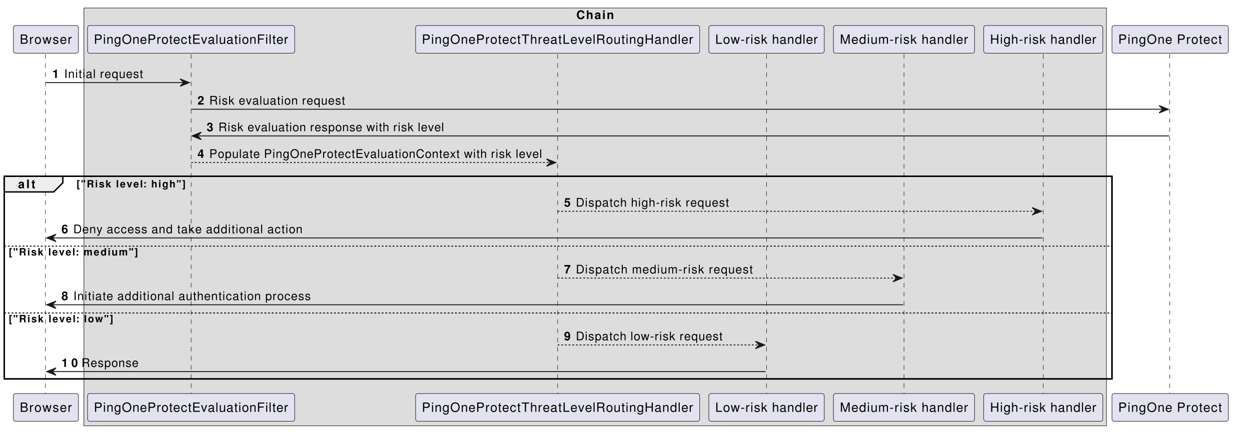
Risk management

PingOne Protect monitors end-user requests and generates a risk score of low, medium, or high based on the user’s activity and device context. You configure and fine-tune risk policies and train risk models prior to using PingGateway with PingOne Protect. Learn more in Threat Protection using PingOne Protect.

You configure PingGateway routes to react to risk scores from PingOne Protect dynamically. For example, if the risk score is medium, PingGateway can direct the user to complete additional verification. If the risk score is high, PingGateway can deny access to the resource instead.

The following sequence diagram shows the PingGateway configuration objects involved in a risk management flow:

- The PingOneProtectEvaluationFilter calls PingOne Protect.
- It populates the PingOneProtectEvaluationContext based on the response.
- The PingOneProtectThreatLevelRoutingHandler dispatches the request based on its risk level.
- After prompting the user to complete additional actions following a risk evaluation, the PingOneProtectFeedbackSuccessFilter or PingOneProtectFeedbackFailureFilter can send feedback to PingOne Protect and override the risk level in the session context.


Chain
Browser
PingOneProtectEvaluationFilter
PingOneProtectThreatLevelRoutingHandler
Low-risk handler
Medium-risk handler
High-risk handler
PingOne Protect
1 Initial request
2 Risk evaluation request
3 Risk evaluation response with risk level
4 Populate PingOneProtectEvaluationContext with risk level
alt
["Risk level: high"]
5 Dispatch high-risk request
6 Deny access and take additional action
["Risk level: medium"]
7 Dispatch medium-risk request
8 Initiate additional authentication process
["Risk level: low"]
9 Dispatch low-risk request
10 Response

1. The initial request enters a Chain configured for risk evaluation with PingOne Protect.
2. The PingOneProtectEvaluationFilter includes data about the request in an API call to PingOne Protect.
3. PingOne Protect provides a risk evaluation response.
4. The PingOneProtectEvaluationFilter populates the PingOneProtectEvaluationContext with the risk level based on the evaluation response and passes control to the PingOneProtectThreatLevelRoutingHandler.
5. -10. The PingOneProtectThreatLevelRoutingHandler dispatches the request based on the risk level. The downstream handler returns a response to the initial request that is appropriate for the risk level. The downstream handlers can prompt additional actions to verify the user’s identity.

Although not shown in the sequence diagram, when a `medium` risk level leads to additional authentication steps and the user successfully completes the steps, a PingOneProtectFeedbackSuccessFilter included in the process updates PingOne Protect to indicate the successful outcome.

The sequence diagram also omits device profile data collection. When the PingOneProtectEvaluationFilter includes `"deviceProfile"` settings, PingGateway gathers profile data from the user-agent by sending it a self-submitting form page that uses JavaScript to retrieve the profile information. (If JavaScript is disabled in the browser, PingGateway can’t get the device profile data.) PingGateway includes the device profile in PingOne Protect risk evaluation requests.

PingGateway runs at the outer edge of your systems, the place where all inbound traffic first arrives and all outbound traffic leaves, just inside the network infrastructure.

PingGateway is well-placed to capture signals about the traffic and its risk profile.

Example: protect against session degradation

Session degradation arises when a valid user session gets used in unexpected ways, increasing the risk the session has been hijacked or otherwise compromised. Session degradation is why we can't let users stay signed in forever unless we can authenticate them again.

Together, PingGateway and PingOne Protect help you automate fine-grained risk evaluation, avoiding the distraction of additional authentication steps unless they're required for a risky request. Users stay signed in with their current session, which they experience as a **keep me signed in** feature. You nevertheless protect their assets from hijacking without needing constantly to verify their identity.

This example demonstrates risk management where the principal already has a valid session. The example doesn't demonstrate how to configure PingOne Protect.

When PingOne Protect returns a risk evaluation, PingGateway responds based on the risk level. This example demonstrates the following responses:

High risk

Deny access to the requested resource.

Although not shown in this brief example, you could route the request to a honeypot.

Medium risk

Prompt the user to reauthenticate to verify their identity.

The user can perform step up or transactional authentication at this point.

Low risk

Let the request pass through unchanged.

Before you begin

- Configure PingOne Protect. Learn more in the PingOne Protect documentation.
- Implement CDSSO with PingOne Advanced Identity Cloud or CDSSO for self-managed AM. This example opts for CDSSO with PingOne Advanced Identity Cloud. When using self-managed AM, also read Self-managed AM.
- For additional protection with medium-risk requests, implement Step up authorization for a transaction.
- Verify you can successfully authenticate with CDSSO to the sample application.

Configure PingGateway

What follows extends the CDSSO route to protect against session degradation:

1. Sign on to the PingOne environment with PingOne Protect as an administrator and add an application for PingGateway.

   Use the following hints regarding non-default configuration settings:

Setting	Value
Application Name	PingGateway
Application Type	**Worker**
Application profile > **Roles**	**Environment Admin** **Identity Data Admin**

2. In the PingOne environment, find and record the values of the following settings:

Property	Description
Environment UUID	This is `<env-uuid>` in other property examples.
PingGateway client ID	PingGateway credentials to access PingOne as the application you registered.
PingGateway client secret	
Policy set UUID	The PingOne Protect policy for risk evaluation requests.
Service endpoint	For risk evaluation and feedback service requests. Example: `https://api.pingone.eu/v1/environments/<env-uuid>`
Token endpoint	For PingGateway to get an access token. Example: `https://auth.pingone.eu/<env-uuid>/as/token`

3. In the PingGateway AdminHttpApplication (`admin.json`) file, make sure the maximum total headers size is large enough to accommodate request headers for the device profile cookies.

   After PingGateway collects device profile data, it stores the data in cookies on the user-agent. The user-agent returns these to PingGateway in the `Cookie` request header. Set the `maxTotalHeadersSize` for the PingGateway server ports large enough to avoid HTTP 431 Request Header Fields Too Large errors; for example:

```
"connectors": [
  {
    "port": 8080,
    "maxTotalHeadersSize": 32768
  },
  {
    "port": 8443,
    "maxTotalHeadersSize": 32768,
    "tls": "TlsConf"
  }
],
```

You can skip this step if you don't use `"deviceProfile"` settings in the PingOneProtectEvaluationFilter.

4. Set an environment variable for the PingGateway client secret.

   PingGateway uses a SystemAndEnvSecretStore to retrieve the client secret, so you must base64-encode the value you found in the PingOne application profile:

```
$ export CLIENT_SECRET_ID='<base-64-encoded-client-secret>'
```

5. Restart PingGateway to load the updated `admin.json` settings and the environment variable.
6. Update the CDSSO route to add risk management:

```
{
  "name": "risk",
  "baseURI": "https://app.example.com:8444/login",
  "condition": "${find(request.uri.path, '^/home/cdsso')}",
  "properties": {
    "amInstanceUrl": "https://myTenant.forgeblocks.com/am",
    "clientId": "my-application-client-id",
    "policySetId": "my-policy-set-id",
    "serviceEndpoint":
"https://api.pingone.eu/v1/environments/my-environment-id",
    "tokenEndpoint": "https://auth.pingone.eu/my-environment-
id/as/token"
  },
  "heap": [
    {
      "name": "BlindTrustReverseProxyHandler",
      "type": "ReverseProxyHandler",
      "comment": "For evaluation and testing only",
      "config": {
        "tls": {
          "type": "ClientTlsOptions",
          "config": {
            "trustManager": {
              "type": "TrustAllManager"
            },
            "hostnameVerifier": "ALLOW_ALL"
          }
        }
      }
    },
    {
      "name": "SystemAndEnvSecretStore-1",
      "type": "SystemAndEnvSecretStore"
    },
    {
      "name": "AmService-1",
      "type": "AmService",
      "config": {
        "url": "&{amInstanceUrl}",
        "realm": "/alpha",
        "agent": {
          "username": "ig_agent",
          "passwordSecretId": "agent.secret.id"
        },
        "secretsProvider": "SystemAndEnvSecretStore-1",
        "sessionCache": {
          "enabled": false
        }
      }
    },
    {
      "name": "ClientCredentialsOAuth2ClientFilter-1",
      "type": "ClientCredentialsOAuth2ClientFilter",
      "config": {
        "tokenEndpoint": "&{tokenEndpoint}",
        "scopes": [
          "openid",
          "profile",
          "email",
          "p1"
        ],
        "endpointHandler": {
          "name": "AccessTokenHandler",
          "type": "Chain",
          "config": {
            "filters": [
              {
                "type":
"ClientSecretBasicAuthenticationFilter",
                "config": {
                  "clientId": "&{clientId}",
                  "clientSecretId": "client.secret.id",
                  "secretsProvider": "SystemAndEnvSecretStore-
1"
                }
              }
            ],
            "handler": "ForgeRockClientHandler"
          }
        }
      }
    },
    {
      "name": "RiskEndpointHandler",
      "type": "Chain",
      "config": {
        "filters": [
          "ClientCredentialsOAuth2ClientFilter-1"
        ],
        "handler": "ForgeRockClientHandler"
      }
    },
    {
      "name": "PingOneService-1",
      "type": "PingOneService",
      "config": {
        "serviceEndpoint": "&{serviceEndpoint}",
        "endpointHandler": "RiskEndpointHandler"
      }
    },
    {
      "name": "StepUpHandler",
      "type": "Chain",
      "config": {
        "filters": [
          {
            "name": "PolicyEnforcementFilter-1",
            "type": "PolicyEnforcementFilter",
            "config": {
              "application": "PEP-CDSSO",
              "ssoTokenSubject": "${contexts.cdsso.token}",
              "amService": "AmService-1"
            }
          },
          {
            "name": "SuccessFeedbackFilter",
            "type": "PingOneProtectFeedbackSuccessFilter",
            "config": {
              "pingOneService": "PingOneService-1",
              "postEvaluationAssumedRiskLevel": "low"
            }
          }
        ],
        "handler": "ReverseProxyHandler"
      }
    },
    {
      "name": "FailureHandler",
      "type": "Chain",
      "config": {
        "filters": [
          {
            "name": "FailureFeedbackFilter",
            "type": "PingOneProtectFeedbackFailureFilter",
            "config": {
              "pingOneService": "PingOneService-1"
            }
          }
        ],
        "handler": {
          "type": "StaticResponseHandler",
          "config": {
            "status": 403,
            "headers": {
              "Content-Type": [
                "text/plain; charset=UTF-8"
              ]
            },
            "entity": "HTTP 403 Forbidden"
          }
        }
      }
    }
  ],
  "handler": {
    "type": "Chain",
    "config": {
      "filters": [
        {
          "name": "CrossDomainSingleSignOnFilter-1",
          "type": "CrossDomainSingleSignOnFilter",
          "config": {
            "redirectEndpoint": "/home/cdsso/redirect",
            "authCookie": {
              "path": "/home",
              "name": "ig-token-cookie"
            },
            "amService": "AmService-1"
          }
        },
        {
          "name": "PingOneProtectEvaluationFilter-1",
          "type": "PingOneProtectEvaluationFilter",
          "config": {
            "pingOneService": "PingOneService-1",
            "policySet": "&{policySetId}",
            "userId":
"${contexts.cdsso.claimsSet.getClaim('subname')}",
            "nonEvaluatedUrls": "${find(request.uri.path,
'/home/cdsso/redirect')}",
            "deviceProfile": {
              "callbackEndpoint":
"/home/cdsso/profilecallback"
            }
          }
        }
      ],
      "handler": {
        "name": "PingOneProtectThreatLevelRoutingHandler-1",
        "type": "PingOneProtectThreatLevelRoutingHandler",
        "config": {
          "levels": {
            "low": "BlindTrustReverseProxyHandler",
            "medium": "StepUpHandler",
            "high": "FailureHandler"
          }
        }
      }
    }
    }
}
```

Notice the following features of the updated route:

- The route properties use the settings you collected from the PingOne environment.

  Replace the placeholders in the route properties with the settings you collected.
- The heap has an evaluation endpoint handler to get an access token for risk evaluation requests.
- The PingOneProtectEvaluationFilter uses the evaluation endpoint handler to make the risk evaluation request and populate the PingOneProtectEvaluationContext (implicit in the configuration).
- The PingOneProtectThreatLevelRoutingHandler uses the context to route the request based on the risk level:
  - Low-risk requests pass through unchanged.
  - Medium-risk requests use step up authorization for the request.
  - High-risk requests and requests where the PingOneProtectEvaluationFilter failed to update the context get denied.

Validation

1. In your browser's privacy or incognito mode, go to https://ig.ext.com:8443/home/cdsso.

   PingOne Advanced Identity Cloud displays the login page.
2. Log in to PingOne Advanced Identity Cloud as the test user.

   After authentication, PingGateway gets an access token and uses it to make a risk evaluation request. The PingOneProtectThreatLevelRoutingHandler routes the result:
   - Low-risk requests go directly to the sample app.
   - Medium-risk requests prompt the user for the authorization passcode `7890`.
   - PingGateway denies access to other requests.

You've successfully demonstrated risk management to prevent session degradation. Adapt the route to the specifics of your use case.

Self-managed AM

When you use transactional authorization with self-managed AM in this example, you must configure PingGateway to use *its* URI rather than the URI of the protected application when confirming the transaction.

You can do this by using a PolicyEnforcementFilter `"resourceUriProvider"` to rebase the protected URI on the PingGateway URI.

1. Add the following ScriptableResourceUri.groovy script to PingGateway:

   Linux | Windows

   ```
   $HOME/.openig/scripts/groovy/ScriptableResourceUri.groovy
   ```

   ▼ *ScriptableResourceUri.groovy*

   ```
   package scripts.groovy

   import org.forgerock.http.routing.UriRouterContext
   import org.forgerock.http.util.Uris

   /**
    * Sample scriptable {@code RequestResourceUriProvider} (for
   use with a {@code PolicyEnforcementFilter}).
    *
    * This sample script "rebases" the in-flight request URI on
   top of the original URI. The term "in-flight" represents
    * the request's URI as it's manipulated as it progresses
   through the route. This is to ensure route components have
   had
    * the opportunity to act on the request prior to policy
   evaluation - e.g. to manipulate query parameters they may be
    * responsible for managing.
    *
    * <p>Example - indicates in-flight request URI's path,
   query and fragment retained and rebased on to original URI:
    * <ul>
    *     <li><b>original request URI</b>:
   https://ig.ext.com:8443/home?param1=1&param2</li>
    *     <li><b>in-flight request URI</b>:
   http://app.example.com:8080/home?param2</li>
    *     <li><b>result (rebased) URI</b>:
   https://ig.ext.com:8443/home?param2</li>
    * </ul>
    *
    * <p>Example config:
   ```

```
 * <pre>
 * {@code {
 *   "name" : "RebasingResourceUriProvider",
 *   "type" : "ScriptableResourceUriProvider",
 *   "config" : {
 *     "file" : "ScriptableResourceUri.groovy",
 *     "type": "application/x-groovy"
 *   }
 * }
 * }
 * </pre>
 *
 * <p>Example {@code PolicyEnforcementFilter} usage:
 * <pre>
 * {@code {
 *   "type" : "PolicyEnforcementFilter",
 *   "config" : {
 *     "amService" : "...",
 *     "application" : "...",
 *     "resourceUriProvider": "RebasingResourceUriProvider",
 *     ...
 *   }
 * }
 * }
 * </pre>
 */
def uriRouterContext =
context.asContext(UriRouterContext.class)

URI originalUri = uriRouterContext.getOriginalUri();
URI requestUri = request.getUri().asURI();
URI pathAndParams = new URI(null,
                            null,
                            requestUri.getPath(),
                            requestUri.getQuery(),
                            requestUri.getFragment());
logger.trace("ScriptableResourceUri rebasing {} with request
parameters {}", originalUri, pathAndParams);
String rebasedUri = Uris.rebase(pathAndParams,
originalUri).toASCIIString();
logger.trace("ScriptableResourceUri rebased to URI '{}'",
rebasedUri);
return rebasedUri;
```

2. Add a reference to the script in the route heap:

```
{
    "name": "RebasingResourceUriProvider",
    "type": "ScriptableResourceUriProvider",
    "config": {
        "file": "ScriptableResourceUri.groovy",
        "type": "application/x-groovy"
    }
}
```

3. In the PolicyEnforcementFilter for the route, use the reference to the script in the `"resourceUriProvider"` setting:

```
{
    "name": "PolicyEnforcementFilter-1",
    "type": "PolicyEnforcementFilter",
    "config": {
        "application": "PEP-CDSSO",
        "ssoTokenSubject": "${contexts.cdsso.token}",
        "amService": "AmService-1",
        "resourceUriProvider": "RebasingResourceUriProvider"
    }
}
```

4. Make sure the route's `"amInstanceUrl"` reflects your self-managed AM.

5. Save your changes to the route.

   After PingGateway reloads the route, you can validate the scenario with your self-managed AM deployment.

Was this helpful?