Authentication and Authorization Guide

/ ForgeRock Identity Management 7.1

Latest update: 7.1.6

ForgeRock AS.
201 Mission St., Suite 2900
San Francisco, CA 94105, USA
+1 415-599-1100 (US)
www.forgerock.com

Abstract

Guide to configuring authentication and authorization.

Table of Contents

Overview

This guide covers authentication, authorization, and delegated administration.

Quick Start

Authentication	Authorization & Access Control	Delegated Administration
Authenticate users securely.	Protect REST endpoints with secure authorization and access control.	Use privileges to give fine-grained administrative access to specific users.

ForgeRock Identity Platform™ serves as the basis for our simple and comprehensive Identity and Access Management solution. We help our customers deepen their relationships with their customers, and improve the productivity and connectivity of their employees and partners. For more information about ForgeRock and about the platform, see https://www.forgerock.com.

The ForgeRock Common REST API works across the platform to provide common ways to access web resources and collections of resources.

Chapter 1 Authentication

Authentication is the process of verifying who is requesting access to a resource. The user or application making the request presents credentials, making it possible to prove that the requester is who they claim to be. The goal is to authorize access to specific IDM resources, depending on the confirmed identity of the user or application making the request.

IDM supports two authentication modes:

- Using one or more of the *classic* IDM authentication modules.
- Configuring IDM as an OAuth2 Resource Server in a *platform deployment* AM as the Identity Provider.

IDM and HTTP Basic Authentication

HTTP basic authentication is a simple challenge and response mechanism whereby the client submits a user ID and password to the server. IDM understands the authorization header of the HTTP basic authentication contract. However, it deliberately does not use the full HTTP basic authentication contract and does not cause the browser built-in mechanism to prompt for username and password. It also understands utilities, such as `curl` and Postman, that can send the username and password in the Authorization header.

In general, the HTTP basic authentication mechanism does not work well with client side web applications, and applications that need to render their own login screens. Because the browser stores and sends the username and password with each request, HTTP basic authentication has significant security vulnerabilities. You can therefore send the username and password via the authorization header, and IDM returns a token for subsequent access.

Access to the IDM REST interface *requires* that the client authenticate. User self-registration requires anonymous access. For this purpose, IDM includes an `anonymous` user, with the password `anonymous`. For more information, see Internal users.

The examples in this documentation use the IDM authentication headers in all REST examples, for example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
...
```

Password Changes

Changing passwords can expose a server to potential security risks. An insecure password reset process can allow attackers to reset the passwords of other users in order to bypass authentication and gain access to user accounts.

Reauthentication forces users or clients to confirm their identity even this identity was verified previously. When passwords are changed over REST, using a PUT or PATCH request, IDM requires the `X-OpenIDM-Reauth-Password` header. If this header is absent, the server returns a `403` error.

For example, the following password change request fails:

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--header "If-Match: *" \
--request PUT \
--data '{
  "userName": "bjensen",
  "givenName": "Babs",
  "sn": "Jensen",
  "mail": "babs.jensen@example.com",
  "telephoneNumber": "555-123-1234",
  "password": "NewPassw0rd"
}' \
https://localhost:8443/openidm/managed/user/0638da14-e02e-4904-9076-b8ce8f700eb4
{
  "code": 403,
  "reason": "Forbidden",
  "message": "Access denied"
}
```

The same request, including the `X-OpenIDM-Reauth-Password` header, succeeds:

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--header "X-OpenIDM-Reauth-Password: Passw0rd" \
--header "If-Match: *" \
--request PUT \
--data '{
  "userName": "bjensen",
  "givenName": "Babs",
  "sn": "Jensen",
  "mail": "babs.jensen@example.com",
  "telephoneNumber": "555-123-1234",
  "password": "NewPassw0rd"
}' \
https://localhost:8443/openidm/managed/user/0638da14-e02e-4904-9076-b8ce8f700eb4
{
  "_id": "0638da14-e02e-4904-9076-b8ce8f700eb4",
  "_rev": "00000000fa190282",
  "userName": "bjensen",
  "givenName": "Babs",
  "sn": "Jensen",
  "mail": "babs.jensen@example.com",
  "telephoneNumber": "555-123-1234",
  ...
}
```

Character Encoding in Authentication Headers

You can use RFC 5987-encoded characters in all three IDM authentication headers (`X-OpenIDM-Username`, `X-OpenIDM-Password`, and `X-OpenIDM-Reauth-Password`). This lets you use non-ASCII characters in these header values. The RFC 5987-encoding is automatically detected and decoded when present. The following character sets are supported:

- UTF-8
- ISO 8859-1

The following command shows a request for a user (openidm-admin) whose password is `Passw£rd123`. The Unicode `£` sign (U+00A3) is encoded into the octet sequence C2 A3 using UTF-8 character encoding, then percent-encoded:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: UTF-8''Passw%C2%A3rd123" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--request GET \
"https://localhost:8443/openidm/managed/user?_queryFilter=true&_fields=_id"
```

Authenticate Users

IDM stores two types of users in its repository—internal users and managed users.

Internal users

Internal users are special user accounts that are stored separately from regular users to protect them from any reconciliation or synchronization processes. When IDM first starts up, it creates three internal users in the repository by default—`openidm-admin`, `anonymous`, and `idm-provisioning`:

openidm-admin

This user serves as the top-level administrator and has full access to all IDM resources. This account provides a fallback mechanism in the event that other users are locked out of their accounts. Do not use `openidm-admin` for regular tasks. Under normal circumstances, the `openidm-admin` account does not represent a regular user, so audit log records for this account do not represent the actions of any real person.

The default password for the `openidm-admin` user is `openidm-admin`. In production environments, you should change this password, as described in "Change the Administrator User Password". The new password is symmetrically encrypted as it is changed.

anonymous

This user enables anonymous access to IDM. It is used to interact with IDM in limited ways without further authentication, such as when a user has not yet logged in and makes a login request. The anonymous user account also allows self-registration.

The default password for the `anonymous` user is `anonymous`.

idm-provisioning

The internal user `idm-provisioning` is a service account used by AM to provision accounts in IDM. It has no password, and isn't meant to be logged in directly. If you are not planning to use AM and IDM together as a platform, you can safely remove this user.

Managed users

Regular user accounts that are stored in IDM's repository are called *managed users* because IDM effectively manages these accounts.

Both internal and managed users *must* authenticate to gain access to the server. The way in which these user types are authenticated is defined in your project's `conf/authentication.json` file.

Any request to IDM will authenticate the user and return a token. To improve tracing through logs, authenticate internal and managed users over REST by sending a POST request to the `openidm/authentication` endpoint, with `_action=login`. The following example authenticates the `openidm-admin` user:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--request POST  \
"https://localhost:8443/openidm/authentication?_action=login"
```

Attributes Used For Authentication

By default, the attribute names that are used to authenticate managed and internal users are `username` and `password`. You can change the attributes that store authentication information with the `propertyMapping` object in the `conf/authentication.json` file. The following excerpt of the `authentication.json` file shows the default authentication attributes:

```
...
"propertyMapping" : {
    "authenticationId" : "username",
    "userCredential" : "password",
    "userRoles" : "authzRoles",
    "additionalUserFields": ["adminOfOrg", "ownerOfOrg"]
},
...
```

If you change the attributes that are used for authentication, you must also change any authentication queries that use those attributes. The following authentication queries are referenced in `authentication.json`:

- `credential-internaluser-query` authenticates internal users.
- `credential-query` authenticates managed users.
- `for-username`

To change the authentication queries for a customized authentication attribute, create a `queryFilters.json` file in your project's `conf` directory. Include the authentication query IDs and the amended query filter, taking into account your changed attributes. The default authentication queries are as follows:

```
{
  "credential-query": {
    "_queryFilter": "/userName eq \"${username}\" AND /accountStatus eq \"active\""
  },
  "credential-internaluser-query": {
    "_queryFilter": "/_id eq \"${username}\""
  },
  "for-userName": {
    "_queryFilter": "/userName eq \"${uid}\""
  }
}
```

The following example `conf/queryFilters.json` file shows the authentication queries adjusted to use the `email` attribute instead of the `username` attribute:

```
{
  "credential-query": {
    "_queryFilter": "/email eq \"${email}\" AND /accountStatus eq \"active\""
  },
  "credential-internaluser-query": {
    "_queryFilter": "/_id eq \"${email}\""
  },
  "for-userName": {
    "_queryFilter": "/email eq \"${uid}\""
  }
}
```

Internal Users

Although internal users are considered to be special user accounts, you can manage them over the REST interface as you would any regular user in the repository.

To list the internal users over REST, query the `internal/user` endpoint as follows:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET  \
"http://localhost:8080/openidm/internal/user?_queryFilter=true&fields=_id"
{
  "result": [
    {
      "_id": "openidm-admin",
      "_rev": "00000000ec996921"
    },
    {
      "_id": "anonymous",
      "_rev": "00000000d95a68b1"
    },
    {
      "_id": "idm-provisioning",
      "_rev": "00000000817e3805"
    },
    {
      "_id": "connector-server-client",
      "_rev": "000000003f2a3a85"
    }
  ],
  ...
}
```

To query the details of an internal user, include the user ID in the request, for example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--request GET  \
"http://localhost:8080/openidm/internal/user/openidm-admin"
{
  "_id": "openidm-admin",
  "_rev": "00000000ec996921"
}
```

Internal users have specific authorization roles by default. These roles determine what the users can access in IDM. The `anonymous` user has only the `openidm-reg` role by default. This role grants only the resource access required to log in, register, and so forth. To identify the authorization roles for the `openidm-admin` internal user, and for information about creating and managing other administrative users, see "Administrative Users".

Change the Administrator User Password

The password of the `openidm-admin` user is `openidm-admin` by default. This password is set in the following excerpt of the `authentication.json` file:

```
{
    "name" : "STATIC_USER",
    "properties" : {
        "queryOnResource" : "internal/user",
        "username" : "openidm-admin",
        "password" : "&{openidm.admin.password}",
        "defaultUserRoles" : [
            "internal/role/openidm-authorized",
            "internal/role/openidm-admin"
        ]
    },
    "enabled" : true
}
```

The `password` property references the `openidm.admin.password` property, set in `resolver/boot.properties`:

```
openidm.admin.password=openidm-admin
```

You can change the default administrator password in a number of ways:

- Edit the `resolver/boot.properties` file before you start IDM (or restart IDM after you change this file).
- Set the value directly in the `conf/authentication.json` file.
- Update the authentication configuration over REST.

  + *Show me how*

  Get the current authentication configuration:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--request GET \
"https://localhost:8443/openidm/config/authentication"
{
  "_id": "authentication",
  "serverAuthContext": {
    ...
    "authModules": [
      ...
      {
        "name": "STATIC_USER",
        "properties": {
          "queryOnResource": "internal/user",
          "username": "openidm-admin",
          "password": "&{openidm.admin.password}",
          "defaultUserRoles": [
            "internal/role/openidm-authorized",
            "internal/role/openidm-admin"
          ]
        },
        "enabled": true
      },
      ...
    ]
  }
}
```

Change the `password` field of this `STATIC_USER` module and replace the authentication configuration:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--cacert ca-cert.pem \
--request PUT \
--data '{
  "_id": "authentication",
  "serverAuthContext": {
    "sessionModule": {
      "name": "JWT_SESSION",
      "properties": {
        "maxTokenLifeMinutes": 120,
        "tokenIdleTimeMinutes": 30,
        "sessionOnly": true,
        "isHttpOnly": true,
        "enableDynamicRoles": false
      }
    },
    "authModules": [
      {
        "name": "STATIC_USER",
        "properties": {
          "queryOnResource": "internal/user",
          "username": "anonymous",
          "password": {
            "$crypto": {
              "type": "x-simple-encryption",
              "value": {
                "cipher": "AES/CBC/PKCS5Padding",
                "stableId": "openidm-sym-default",
                "salt": "xBlTp67ze4Ca5LTocXOpoA==",
                "data": "mdibV6UabU2M+M5MK7bjFQ==",
                "keySize": 16,
                "purpose": "idm.config.encryption",
                "iv": "36D2+FumKbaUsndNQ+/+5w==",
                "mac": "ZM8GMnh0n80QwtSH6QsNmA=="
              }
            }
          },
          "defaultUserRoles": [
            "internal/role/openidm-reg"
          ]
        },
        "enabled": true
      },
      {
        "name": "STATIC_USER",
        "properties": {
          "queryOnResource": "internal/user",
          "username": "openidm-admin",
          "password": "newAdminPassword",
          "defaultUserRoles": [
            "internal/role/openidm-authorized",
            "internal/role/openidm-admin"
          ]
        },
        "enabled": true
      },
      {
        "name": "MANAGED_USER",
        "properties": {
          "augmentSecurityContext": {
            "type": "text/javascript",
            "source": "require('auth/customAuthz').setProtectedAttributes(security)"
          },
          "queryId": "credential-query",
          "queryOnResource": "managed/user",
          "propertyMapping": {
            "authenticationId": "username",
            "userCredential": "password",
            "userRoles": "authzRoles"
          },
          "defaultUserRoles": [
            "internal/role/openidm-authorized"
          ]
        },
        "enabled": true
      },
      {
        "name": "SOCIAL_PROVIDERS",
        "properties": {
          "defaultUserRoles": [
            "internal/role/openidm-authorized"
          ],
          "augmentSecurityContext": {
            "type": "text/javascript",
            "globals": {},
            "file": "auth/populateAsManagedUserFromRelationship.js"
          },
          "propertyMapping": {
            "userRoles": "authzRoles"
          }
        },
        "enabled": true
      }
    ]
  }
}' \
"https://localhost:8443/openidm/config/authentication"
{
  "_id": "authentication",
  "serverAuthContext": {
    "sessionModule": {
      "name": "JWT_SESSION",
      "properties": {
        "maxTokenLifeMinutes": 120,
        "tokenIdleTimeMinutes": 30,
        "sessionOnly": true,
        "isHttpOnly": true,
        "enableDynamicRoles": false
      }
    },
    "authModules": [
      {
        "name": "STATIC_USER",
        "properties": {
          "queryOnResource": "internal/user",
          "username": "anonymous",
          "password": {
            "$crypto": {
              "type": "x-simple-encryption",
              "value": {
                "cipher": "AES/CBC/PKCS5Padding",
                "stableId": "openidm-sym-default",
                "salt": "xBlTp67ze4Ca5LTocXOpoA==",
                "data": "mdibV6UabU2M+M5MK7bjFQ==",
                "keySize": 16,
                "purpose": "idm.config.encryption",
                "iv": "36D2+FumKbaUsndNQ+/+5w==",
                "mac": "ZM8GMnh0n80QwtSH6QsNmA=="
              }
            }
          },
          "defaultUserRoles": [
            "internal/role/openidm-reg"
          ]
        },
        "enabled": true
      },
      {
        "name": "STATIC_USER",
        "properties": {
          "queryOnResource": "internal/user",
          "username": "openidm-admin",
          "password": {
            "$crypto": {
              "type": "x-simple-encryption",
              "value": {
                "cipher": "AES/CBC/PKCS5Padding",
                "stableId": "openidm-sym-default",
                "salt": "l0trJWBzg5JKcWLzNq8QDA==",
                "data": "MKAkL9FVEq/FnWq+8a90+QcjfkEbrK7W4tIc3ORD1ck=",
                "keySize": 16,
                "purpose": "idm.config.encryption",
                "iv": "UMjU6crk332MZtEjo+wEmw==",
                "mac": "7EvTqjpmuS9PmY1aCT2s+g=="
              }
            }
          },
          "defaultUserRoles": [
            "internal/role/openidm-authorized",
            "internal/role/openidm-admin"
          ]
        },
        "enabled": true
      },
      {
        "name": "MANAGED_USER",
        "properties": {
          "augmentSecurityContext": {
            "type": "text/javascript",
            "source": "require(auth/customAuthz).setProtectedAttributes(security)"
          },
          "queryId": "credential-query",
          "queryOnResource": "managed/user",
          "propertyMapping": {
            "authenticationId": "username",
            "userCredential": "password",
            "userRoles": "authzRoles"
          },
          "defaultUserRoles": [
            "internal/role/openidm-authorized"
          ]
        },
        "enabled": true
      },
      {
        "name": "SOCIAL_PROVIDERS",
        "properties": {
          "defaultUserRoles": [
            "internal/role/openidm-authorized"
          ],
          "augmentSecurityContext": {
            "type": "text/javascript",
            "globals": {},
            "file": "auth/populateAsManagedUserFromRelationship.js"
          },
          "propertyMapping": {
            "userRoles": "authzRoles"
          }
        },
        "enabled": true
      }
    ]
  }
}
```

Authentication and Session Modules

An authentication module specifies how a user or client is authenticated. You configure authentication and session modules in your project's `conf/authentication.json` file.

IDM evaluates authentication modules in the order in which they appear in that file, and uses the first "successful" authentication module it finds. Subsequent modules are not evaluated. In a production environment, you should remove any unused authentication modules from your `authentication.json` file.

To authenticate a user or client, IDM validates the provided credentials against some resource. That resource can be either an IDM resource such as `managed/user` or `internal/user`, or it can be an external resource such as an LDAP server or social identity provider. You should prioritize the authentication modules that query IDM resources over those that query external resources. Prioritizing modules that query external resources can lead to authentication problems for internal users such as `openidm-admin`.

You can also configure authentication modules in the Admin UI. Select Configure > Authentication then select the Session or Module tab to configure the session module or authentication modules respectively. To change the order of authentication modules in the Admin UI, simply drag the modules up or down so that they appear in the order in which they should be evaluated.

Note

Modifying an authentication module in the Admin UI might affect your current session. In this case, IDM prompts you with the following message:

```
Your current session may be invalid. Click here to logout and re-authenticate.
```

When you select the *Click here* link, IDM logs you out of any current session and returns you to the login screen.

This section describes the supported authentication and session modules and how to configure them to authenticate clients securely.

IDM supports the following authentication and session modules:

- "JWT_SESSION"
- "STATIC_USER"
- "TRUSTED_ATTRIBUTE"

- "MANAGED_USER"
- "INTERNAL_USER"
- "CLIENT_CERT"
- "PASSTHROUGH"
- "SOCIAL_PROVIDERS"
- "OAUTH_CLIENT"
- "IWA"

JWT_SESSION

IDM supports one session module, the JSON Web Token (JWT) Session Module. When a client authenticates successfully, the JWT Session Module creates a JWT and sets it as a cookie on the response. On subsequent requests, the module checks for the presence of the JWT as a cookie on the request, validates the signature and decrypts it, and checks the expiration time of the JWT.

JWT sessions are entirely stateless, that is, they are not persisted in the backend. All information pertaining to the session is encrypted in the JWT.

When a request to IDM produces a JWT, that value replaces the previous one used to send that request. In this way the JWT is always updated to the latest copy. The idle timeout in the JWT is therefore continuously updated and active sessions are not abruptly killed mid-session.

By default, the JWT cookie is deleted on logout. Deleting the cookie manually ends the session. You can modify what happens to the session after a browser restart by changing the value of the `sessionOnly` property.

The default JWT Session Module configuration, in `conf/authentication.json`, is as follows:

```
"sessionModule" : {
    "name" : "JWT_SESSION",
    "properties" : {
        "maxTokenLifeMinutes" : 120,
        "tokenIdleTimeMinutes" : 30,
        "sessionOnly" : true,
        "isHttpOnly" : true,
        "enableDynamicRoles" : false
    }
}
```

Important

In a production environment, ensure that only secure cookies are used. To do so, add the following property to the session module configuration:

```
"isSecure" : true
```

For more information about this module, see the Class JwtSessionModule JavaDoc.

Attempting to access IDM without the appropriate headers or session cookie results in an HTTP 401 Unauthorized, or HTTP 403 Forbidden, depending on the situation. If you authenticate using a session cookie, you must include an additional header that indicates the origin of the request.

The following example shows a successful authentication attempt and the return of a session cookie:

```
curl \
--dump-header /dev/stdout \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
"https://localhost:8443/openidm/managed/user?_queryFilter=true&_fields=_id"
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Cache-Control: no-cache
Set-Cookie: session-jwt=2l0zobpuk6st1b2m7gvhg5zas ...;Path=/
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Vary: Accept-Encoding, User-Agent
Content-Length: 82
Server: Jetty(8.y.z-SNAPSHOT)
```

The following example uses the cookie returned in the response to the previous request, and includes the `X-Requested-With` header to indicate the origin of the request. The value of the header can be any string, but should be informative for logging purposes. If you do not include the `X-Requested-With` header, IDM returns HTTP 403 Forbidden:

```
curl \
--dump-header /dev/stdout \
--header "Cookie: session-jwt=2l0zobpuk6st1b2m7gvhg5zas ..." \
--header "X-Requested-With: OpenIDM Plugin" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
"https://localhost:8443/openidm/managed/user?_queryFilter=true&_fields=_id"
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Content-Type: application/json; charset=UTF-8
Cache-Control: no-cache
Vary: Accept-Encoding, User-Agent
Content-Length: 82
Server: Jetty(8.y.z-SNAPSHOT)
```

The expiration date of the JWT cookie, January 1, 1970, corresponds to the start of UNIX time. Since that time is in the past, browsers will not store that cookie after the browser session is closed.

To request one-time authentication without a session:

```
curl \
--dump-header /dev/stdout \
--header "X-OpenIDM-NoSession: true" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--cacert ca-cert.pem \
--header "Accept-API-Version: resource=1.0" \
"https://localhost:8443/openidm/managed/user?_queryFilter=true&_fields=_id"
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Cache-Control: no-cache
Vary: Accept-Encoding, User-Agent
Content-Length: 82
Server: Jetty(8.y.z-SNAPSHOT)
```

Authentication requests are logged in the `authentication.audit.json` file. A successful authentication request is logged as follows:

```
{
  "_id": "389d15d3-bdd5-4521-ae3c-bf096d334405-915",
  "timestamp": "2019-08-02T11:53:31.110Z",
  "eventName": "SESSION",
  "transactionId": "389d15d3-bdd5-4521-ae3c-bf096d334405-912",
  "trackingIds": [
    "5f9f4941-bcbd-4cbc-97f7-e763808e4310",
    "88973bcf-0d60-41b8-9922-73718ce76e11"
  ],
  "userId": "openidm-admin",
  "principal": [
    "openidm-admin"
  ],
  "entries": [
    {
      "moduleId": "JwtSession",
      "result": "SUCCESSFUL",
      "info": {
        "org.forgerock.authentication.principal": "openidm-admin"
      }
    }
  ],
  "result": "SUCCESSFUL",
  "provider": null,
  "method": "JwtSession"
}
```

For information about querying this log, see Query the Authentication Audit Log in the *Audit Guide*.

Deterministic ECDSA signatures

By default, JWTs are signed with deterministic Elliptic Curve Digital Signature Algorithm (ECDSA). In order to use this more secure signing method, Bouncy Castle, which is included in the default IDM installation, must be installed. If Bouncy Castle is unavailable or the key is incompatible, IDM falls back to normal ECDSA.

Note

If you need to turn off the use of deterministic ECDSA, add the following line to `conf/system.properties`:

```
org.forgerock.secrets.preferDeterministicEcdsa=false
```

STATIC_USER

The `STATIC_USER` module provides an authentication mechanism that avoids database lookups by hard coding a static user. IDM includes a default `anonymous` static user, but you can create any static user for this module.

The following sample REST call uses `STATIC_USER` authentication with the `anonymous` user in the self-registration process:

```
curl \
--header "X-OpenIDM-Password: anonymous" \
--header "X-OpenIDM-Username: anonymous" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "userName": "steve",
  "givenName": "Steve",
  "sn": "Carter",
  "telephoneNumber": "0828290289",
  "mail": "scarter@example.com",
  "password": "Passw0rd"
}' \
"https://localhost:8443/openidm/managed/user/?_action=create"
```

Note

This is not the same as an anonymous request that is issued without headers.

Authenticating with the `STATIC_USER` module avoids the performance cost of reading the database for self-registration, certain UI requests, and other actions that can be performed anonymously. Authenticating the anonymous user with the `STATIC_USER` module is identical to authenticating the anonymous user with the `INTERNAL_USER` module, except that the database is not accessed. So, `STATIC_USER` authentication provides an authentication mechanism for the anonymous user that avoids the database lookups incurred when using `INTERNAL_USER`.

A sample `STATIC_USER` authentication configuration follows:

```
{
    "name" : "STATIC_USER",
    "enabled" : true,
    "properties" : {
        "queryOnResource" : "internal/user",
        "username" : "anonymous",
        "password" : "anonymous",
        "defaultUserRoles" : [
            "internal/role/openidm-reg"
        ]
    }
}
```

IDM also uses the `STATIC_USER` module to set the password and default roles of the `openidm-admin` internal user on startup. The following configuration in the `authentication.json` file sets up the `openidm-admin` user:

```
{
    "name" : "STATIC_USER",
    "properties" : {
        "queryOnResource" : "internal/user",
        "username" : "openidm-admin",
        "password" : "&{openidm.admin.password}",
        "defaultUserRoles" : [
            "internal/role/openidm-authorized",
            "internal/role/openidm-admin"
        ]
    },
    "enabled" : true
}
```

For information on changing the default `openidm-admin` password, see "Change the Administrator User Password".

TRUSTED_ATTRIBUTE

The `TRUSTED_ATTRIBUTE` authentication module lets you configure IDM to trust a specific `HttpServletRequest` attribute. To enable this module, add it to your `authentication.json` file as follows:

```
{
    "name" : "TRUSTED_ATTRIBUTE",
    "properties" : {
        "queryOnResource" : "managed/user",
        "propertyMapping" : {
            "authenticationId" : "userName",
            "userRoles" : "authzRoles"
        },
        "defaultUserRoles" : [ ],
        "authenticationIdAttribute" : "X-ForgeRock-AuthenticationId",
        "augmentSecurityContext" : {
            "type" : "text/javascript",
            "file" : "auth/populateRolesFromRelationship.js"
        }
    },
    "enabled" : true
}
```

`TRUSTED_ATTRIBUTE` authentication queries the `managed/user` resource, and allows authentication when credentials match, based on the `username` and `authzRoles` assigned to that user, specifically the `X-ForgeRock-AuthenticationId` attribute.

Tip

To use the `TRUSTED_ATTRIBUTE` module with internal `authz` roles, you must modify the `isAJAXRequest` function in `bin/defaults/script/router-authz.js` to check for the `X-Special-Trusted-User` header:

```
function isAJAXRequest() {
    var headers = context.http.headers;
    // one of these custom headers must be present for all HTTP-based requests, to prevent CSRF attacks
    if (typeof(headers["X-Requested-With"]) !== "undefined" ||
        typeof(headers["x-requested-with"]) !== "undefined" ||
        typeof(headers["Authorization"]) !== "undefined" ||
        typeof(headers["authorization"]) !== "undefined" ||
        typeof(headers["X-OpenIDM-Username"]) !== "undefined" ||
        typeof(headers["x-openidm-username"]) !== "undefined" ||
        typeof(headers["X-Special-Trusted-User"]) !== "undefined" ||
        typeof(headers["x-special-trusted-user"]) !== "undefined") {

        if ((headers["X-Requested-With"] || "").toLowerCase().startsWith("shockwaveflash")) {
            // prevent CSRF from Flash
            return false;
        }
        return true;
    }
    return false;
}
```

For a sample implementation of a custom servlet filter and the Trusted Request Attribute Authentication Module, see "*Authenticate Using a Trusted Servlet Filter*" in the *Samples Guide*.

MANAGED_USER

MANAGED_USER authentication queries the repository and allows authentication if the credentials match. Despite the module name, the query is not restricted to managed/user objects. The resource that is queried is configurable. The default configuration uses the username and password of a managed user to authenticate, as shown in the following sample configuration:

```
{
    "name" : "MANAGED_USER",
    "properties" : {
        "augmentSecurityContext": {
            "type" : "text/javascript",
            "source" : "..."
        },
        "queryId" : "credential-query",
        "queryOnResource" : "managed/user",
        "propertyMapping" : {
            "authenticationId" : "username",
            "userCredential" : "password",
            "userRoles" : "authzRoles",
            "additionalUserFields": ["adminOfOrg", "ownerOfOrg"]
        },
        "defaultUserRoles" : [
            "internal/role/openidm-authorized"
        ]
    },
    "enabled" : true
}
```

Use the augmentSecurityContext property to add custom properties to the security context of users who authenticate with this module. By default, this property adds a list of *protected properties* to the user's security context. These protected properties are defined in the managed object schema. The isProtected property is described in "Create and Modify Object Types" in the *Object Modeling Guide*.

INTERNAL_USER

INTERNAL_USER authentication queries the internal/user objects in the repository and allows authentication if the credentials match. An example configuration that uses the username and password of the internal user to authenticate follows:

```
{
    "name" : "INTERNAL_USER",
    "enabled" : true,
    "properties" : {
        "queryId" : "credential-internaluser-query",
        "queryOnResource" : "internal/user",
        "propertyMapping" : {
            "authenticationId" : "username",
            "userCredential" : "password",
            "userRoles" : "authzRoles"
        },
        "defaultUserRoles" : [ ]
    }
}
```

CLIENT_CERT

Client certificate authentication (also called *mutual SSL authentication*) occurs as part of the SSL or TLS handshake, which takes place before any data is transmitted in an SSL or TLS session. This authentication module is typically used when users have secure certificates that they install in their browsers for authentication and authorization.

The client certificate module, `CLIENT_CERT`, authenticates by validating a client certificate, transmitted through an HTTP request. IDM compares the subject DN of the request certificate with the subject DN of the truststore.

A sample `CLIENT_CERT` authentication configuration follows:

```
{
  "name" : "CLIENT_CERT",
  "properties" : {
    "augmentSecurityContext" : {
      "type" : "text/javascript",
      "globals" : { },
      "file" : "auth/mapUserFromClientCert.js"
    },
    "queryOnResource" : "managed/user",
    "defaultUserRoles" : [
      "internal/role/openidm-authorized"
    ],
    "allowedAuthenticationIdPatterns" : [
      ".*CN=localhost, O=ForgeRock.*"
    ]
  },
  "enabled" : true
}
```

When a user authenticates with a client certificate, they receive the roles listed in the `defaultUserRoles` property of the `CLIENT_CERT` module. Privileges are calculated dynamically per request when enabled in the session module.

Note

Client certificate authentication is also used when the client is a password plugin, such as those described in the Password Synchronization Plugin Guide. This process is similar to an administrative request to modify the passwords of regular users.

For password plugin clients, you must include `internal/role/openidm-cert` in the `defaultUserRoles` array (in the authentication configuration *(You can manage the authentication configuration over REST at the config/authentication endpoint, or directly in the conf/authentication.json file.)*).

Test Client Certificate Authentication

This procedure demonstrates client certificate authentication by generating a self-signed certificate, adding that certificate to the truststore, then authenticating with the certificate. At the end of this procedure, you will verify the certificate over port `8444` as defined in your project's `resolver/boot.properties` file:

```
openidm.auth.clientauthonlyports=8444
```

The example assumes an existing managed user, bjensen, with email address `bjensen@example.com`.

1. Create a self-signed certificate for user bjensen as follows:

   ```
   openssl req \
   -x509 \
   -newkey rsa:1024 \
   -keyout /path/to/key.pem \
   -out /path/to/cert.pem \
   -days 3650 \
   -nodes
   Generating a 1024 bit RSA private key
   .................++++++
   ..................++++++
   writing new private key to 'key.pem'
   -----
   You are about to be asked to enter information that will be incorporated
   into your certificate request.
   What you are about to enter is what is called a Distinguished Name or a DN.
   There are quite a few fields but you can leave some blank
   For some fields there will be a default value,
   If you enter '.', the field will be left blank.
   -----
   Country Name (2 letter code) []:US
   State or Province Name (full name) []:Washington
   Locality Name (eg, city) []:Vancouver
   Organization Name (eg, company) []:Example.com
   Organizational Unit Name (eg, section) []:
   Common Name (eg, fully qualified host name) []:localhost
   Email Address []:bjensen@example.com
   ```

   Note

   The `Email Address` is used by the `mapUserFromClientCert.js` to map the user against an existing managed user.

2. Import the client certificate into the IDM truststore:

   ```
   keytool \
   -importcert \
   -keystore /path/to/openidm/security/truststore \
   -storetype JKS \
   -storepass changeit \
   -file /path/to/cert.pem \
   -trustcacerts \
   -noprompt \
   -alias client-cert-example
   Certificate was added to keystore
   ```

   By default, users can authenticate only if their certificates have been issued by a Certificate Authority (CA) that is listed in the truststore. The default truststore includes several trusted root CA certificates, and any user certificate issued by those CAs will be trusted. Change the value of this property to restrict certificates to those issued to users in your domain, or use some other regular expression to limit who will be trusted. If you leave this property empty, no certificates will be trusted.

3. Edit your project's `conf/authentication.json` file. Add the `CLIENT_CERT` module, and add at least the email address from the certificate subject DN to the `allowedAuthenticationIdPatterns`:

```
{
  "name": "CLIENT_CERT",
  "properties": {
    "augmentSecurityContext": {
      "type": "text/javascript",
      "globals": {},
      "file": "auth/mapUserFromClientCert.js"
    },
    "queryOnResource": "managed/user",
    "defaultUserRoles": [
      "internal/role/openidm-cert",
      "internal/role/openidm-authorized"
    ],
    "allowedAuthenticationIdPatterns": [
      ".*EMAILADDRESS=bjensen@example.com.*"
    ]
  },
  "enabled": true
}
```

> **Note**
>
> The `allowedAuthenticationIdPatterns` property is unique to this authentication module. This property contains a regular expression that defines which user distinguished names (DNs) are allowed to authenticate with a certificate.

4. Send an HTTP request with your certificate file `cert.pem` to the secure port:

```
curl \
--insecure \
--cert-type PEM \
--key /path/to/key.pem \
--key-type PEM \
--cert /path/to/cert.pem \
--header "X-Requested-With: curl" \
--header "X-OpenIDM-NoSession: true" \
--request GET "https://localhost:8444/openidm/info/login"
{
  "_id": "login",
  "authenticationId": "EMAILADDRESS=bjensen@example.com, CN=localhost, O=Example.com, L=Vancouver,
 ST=Washington, C=US",
  "authorization": {
    "userRolesProperty": "authzRoles",
    "component": "managed/user",
    "authLogin": false,
    "roles": [
      "internal/role/openidm-cert",
      "internal/role/openidm-authorized"
    ],
    "ipAddress": "0:0:0:0:0:0:0:1",
    "id": "aba3e666-c0db-4669-8760-0eb21f310649",
    "moduleId": "CLIENT_CERT"
  }
}
```

Note

Because we have used a self-signed certificate in this example, you must include the `--insecure` option. You should not include this option if you are using a CA cert.

You must use the `X-Requested-With` and `X-OpenIDM-NoSession` headers for HTTP-based requests that use the CLIENT_CERT authentication module.

PASSTHROUGH

`PASSTHROUGH` authentication queries an external system, such as an LDAP server, and allows authentication if the credentials included in the REST request match those in the external system.

The following excerpt of an `authentication.json` shows a pass-through authentication configuration for an LDAP system:

```
"authModules" : [
    {
       "name" : "PASSTHROUGH",
       "enabled" : true,
       "properties" : {
          "augmentSecurityContext": {
             "type" : "text/javascript",
             "file" : "auth/populateAsManagedUser.js"
          },
          "queryOnResource" : "system/ldap/account",
          "propertyMapping" : {
             "authenticationId" : "uid",
             "groupMembership" : "ldapGroups"
          },
          "groupRoleMapping" : {
             "internal/role/openidm-admin" : ["cn=admins,ou=Groups,dc=example,dc=com"]
          },
          "managedUserLink" : "systemLdapAccounts_managedUser",
          "defaultUserRoles" : [
             "internal/role/openidm-authorized"
          ]
       },
    },
    ...
]
```

For more information on authentication module properties, see "*Authentication and Session Module Configuration*".

Many of the documented samples in the *Samples Guide* are configured for pass-through authentication. For example, the `sync-with-ldap*` samples use an external LDAP system for authentication.

SOCIAL_PROVIDERS

The `SOCIAL_PROVIDERS` module enables authentication through social identity providers that comply with OAuth 2.0 and OpenID Connect 1.0 standards.

For information about configuring this module with social identity providers such as Google, LinkedIn, and Facebook, see "Configure the Social Providers Authentication Module" in the *Self-Service Reference*.

The `/path/to/openidm/audit/authentication.audit.json` log file shows the corresponding `SOCIAL_AUTH` module, used to handle authentication for each social identity provider.

OAUTH_CLIENT

The `OAUTH_CLIENT` authentication module lets you authenticate with any OAuth 2.0-compatible client.

Unlike the social ID providers, this module is used for authentication only, not for registration. If users have registered through one of the social ID providers, and now have managed user accounts in the IDM repository, you can effectively "turn off" the social provider module, and use only this module to let your users authenticate.

The following sample excerpt of an `authentication.json` file shows the module configuration for authentication after successful registration through Google:

```
{
    "name" : "OAUTH_CLIENT",
    "properties" : {
        "propertyMapping" : {
            "authenticationId" : "uid",
            "userRoles" : "authzRoles"
        },
        "defaultUserRoles" : [
            "internal/role/openidm-authorized"
        ],
        "idpConfig" : {
            "provider" : "Google",
            "scope" : [
                "openid"
            ],
            "authenticationIdKey" : "sub",
            "clientId" : "ID",
            "clientSecret" : "secret",
            "authorizationEndpoint" : "https://accounts.google.com/o/oauth2/v2/auth",
            "tokenEndpoint" : "https://www.googleapis.com/oauth2/v4/token",
            "wellKnownEndpoint" : "https://accounts.google.com/.well-known/openid-configuration",
            "redirectUri" : "https://openidm.example.com:8443/",
            "configClass" : "org.forgerock.oauth.clients.oidc.OpenIDConnectClientConfiguration",
            "displayIcon" : "forgerock",
            "enabled" : true
        },
        "queryOnResource" : "managed/user"
    },
    "enabled" : true
}
```

For more information on authentication module properties, see "*Authentication and Session Module Configuration*".

Note

To enable a social identity provider that fully complies with OAuth 2.0 standards, use the IDM `SOCIAL_PROVIDERS` authentication module wrapper. This module is a specialized facility for sharing a social identity provider configuration with the authentication service, which you can configure as if it were a separate authentication module.

IWA

The `IWA` module lets users authenticate using Integrated Windows Authentication (IWA) with Kerberos instead of a username and password.

Windows, UNIX, and Linux systems support Kerberos v5 authentication, which can operate safely on an open, unprotected network. With Kerberos authentication, the user or client application obtains temporary credentials for a service from an *authorization server*, in the form of tickets and session keys. The *service server* handles its part of the Kerberos mutual authentication process.

To enable Kerberos authentication, IDM requires a specific Kerberos user account in Active Directory, and a keytab file that maps the service principal to this user account. The client presents IDM with a Kerberos ticket. If IDM can validate the ticket, the client is granted an encrypted session key for the IDM service. That client can then access IDM without providing a username or password, for the duration of the session.

The complete Kerberos authentication process is shown in the following diagram:

Client Authentication to IDM Using a Kerberos Ticket

This section assumes that you have an active Kerberos server acting as a Key Distribution Center (KDC). If you are running Active Directory, that service includes a Kerberos KDC by default.

The steps required to set up IWA with IDM are described in the following sections:

- "Creating a Specific Kerberos User Account"
- "Creating a Keytab File"
- "Configuring IDM for IWA"

Creating a Specific Kerberos User Account

To authenticate IDM to the Kerberos KDC you must create a specific user entry in Active Directory whose credentials will be used for this authentication. This Kerberos user account must not be used for anything else.

The Kerberos user account is used to generate the Kerberos keytab. If you change the password of this Kerberos user after you have set up IWA, you must update the keytab accordingly.

Create a new user in Active Directory as follows:

1. Select New > User and provide a login name for the user that reflects its purpose, for example, openidm@example.com.

2. Enter a password for the user. Check the *Password never expires* option and leave all other options unchecked.

   If the password of this user account expires, and is reset, you must update the keytab with the new password. It is therefore easier to create an account with a password that does not expire.

3. Click Finish to create the user.

Creating a Keytab File

A Kerberos keytab file (`krb5.keytab`) enables IDM to validate the Kerberos tickets that it receives from client browsers. You must create a Kerberos keytab file for the host on which IDM is running.

This section describes how to use the **ktpass** command, included in the Windows Server toolkit, to create the keytab file. Run the **ktpass** command on the Active Directory domain controller.

Important

- The keytab file is case-sensitive, so you must note the use of capitalization in this example.
- You must disable UAC or run the **ktpass** command as a user with administrative privileges.

The following command creates a keytab file (named `openidm.HTTP.keytab`) for the IDM service located at `openidm.example.com`.

```
C:\Users\Administrator>ktpass ^
-princ HTTP/openidm.example.com@EXAMPLE.COM ^
-mapUser EXAMPLE\openidm ^
-mapOp set ^
-pass Passw0rd1 ^
-crypto ALL
-pType KRB5_NT_PRINCIPAL ^
-kvno 0 ^
-out openidm.HTTP.keytab

Targeting domain controller: host.example.com
Using legacy password setting method
Successfully mapped HTTP/openidm.example.com to openidm.
Key created.
Output keytab to openidm.HTTP.keytab:
Keytab version: 0x502
keysize 79 HTTP/openidm.example.com@EXAMPLE.COM ptype 1 (KRB5_NT_PRINCIPAL)
 vno 0 etype 0x1 (DES-CBC-CRC) keylength 8 (0x73a28fd307ad4f83)
keysize 79 HTTP/openidm.example.com@EXAMPLE.COM ptype 1 (KRB5_NT_PRINCIPAL)
 vno 0 etype 0x3 (DES-CBC-MD5) keylength 8 (0x73a28fd307ad4f83)
keysize 87 HTTP/openidm.example.com@EXAMPLE.COM ptype 1 (KRB5_NT_PRINCIPAL)
 vno 0 etype 0x17 (RC4-HMAC) keylength 16 (0xa87f3a337d73085c45f9416be5787d86)
keysize 103 HTTP/openidm.example.com@EXAMPLE.COM ptype 1 (KRB5_NT_PRINCIPAL)
 vno 0 etype 0x12 (AES256-SHA1) keylength 32 (0x6df9c282abe3be787553f23a3d1fcefc
  6fc4a29c3165a38bae36a8493e866d60)
keysize 87 HTTP/openidm.example.com@EXAMPLE.COM ptype 1 (KRB5_NT_PRINCIPAL)
 vno 0 etype 0x11 (AES128-SHA1) keylength 16 (0xf616977f071542cd8ef3ff4e2ebcc09c)
```

The **ktpass** command takes the following options:

- `-princ` specifies the service principal name in the format *`service/host-name@realm`*

  In this example (`HTTP/openidm.example.com@EXAMPLE.COM`), the client browser constructs an SPN based on the following:

  - The service name (HTTP).

The service name for SPNEGO web authentication *must* be HTTP.

- The FQDN of the host on which IDM runs (`openidm.example.com`).

  This example assumes that users will access IDM at the URL `https://openidm.example.com:8443`.

- The Kerberos realm name (`EXAMPLE.COM`).

  The realm name must be uppercase. A Kerberos realm defines the area of authority of the Kerberos authentication server.

- `-mapUser` specifies the name of the Kerberos user account to which the principal should be mapped (the account that you created in "Creating a Specific Kerberos User Account"). The username must be specified in down-level logon name format (DOMAIN\UserName). In our example, the Kerberos user name is `EXAMPLE\openidm`.

- `-mapOp` specifies how the Kerberos user account is linked. Use `set` to set the first user name to be linked. The default (`add`) adds the value of the specified local user name if a value already exists.

- `-pass` specifies a password for the principal user name. Use `*` to prompt for a password.

- `-crypto` specifies the cryptographic type of the keys that are generated in the keytab file. Use `ALL` to specify all crypto types.

  This procedure assumes a 128-bit cryptosystem, with a default RC4-HMAC-NT cryptography algorithm. You can use the **ktpass** command to view the crypto algorithm, as follows:

  ```
  C:\Users\Administrator>ktpass -in .\openidm.HTTP.keytab
  Existing keytab:
  Keytab version: 0x502
  keysize 79 HTTP/openidm.example.com@EXAMPLE.COM ptype 1 (KRB5_NT_PRINCIPAL)
   vno 0 etype 0x1 (DES-CBC-CRC) keylength 8 (0x73a28fd307ad4f83)
  keysize 79 HTTP/openidm.example.com@EXAMPLE.COM ptype 1 (KRB5_NT_PRINCIPAL)
   vno 0 etype 0x3 (DES-CBC-MD5) keylength 8 (0x73a28fd307ad4f83)
  keysize 87 HTTP/openidm.example.com@EXAMPLE.COM ptype 1 (KRB5_NT_PRINCIPAL)
   vno 0 etype 0x17 (RC4-HMAC) keylength 16 (0xa87f3a337d73085c45f9416be5787d86)
  keysize 103 HTTP/openidm.example.com@EXAMPLE.COM ptype 1 (KRB5_NT_PRINCIPAL)
   vno 0 etype 0x12 (AES256-SHA1) keylength 32 (0x6df9c282abe3be787553f23a3d1fcefc6
   fc4a29c3165a38bae36a8493e866d60)
  keysize 87 HTTP/openidm.example.com@EXAMPLE.COM ptype 1 (KRB5_NT_PRINCIPAL)
   vno 0 etype 0x11 (AES128-SHA1) keylength 16 (0xf616977f071542cd8ef3ff4e2ebcc09c)
  ```

- `-ptype` Specifies the principal type. Use `KRB5_NT_PRINCIPAL`.

- `-kvno` specifies the key version number. Set the key version number to 0.

- `-out` specifies the name of the keytab file that will be generated, for example, `openidm.HTTP.keytab`.

Note

The keys that are stored in the keytab file are similar to user passwords. You must protect the Kerberos keytab file in the same way that you would protect a file containing passwords.

For more information about the **ktpass** command, see the ktpass reference in the Windows server documentation.

Configuring IDM for IWA

To configure the IWA authentication module, add the module to your project's `conf/authentication.json` file.

This section assumes that the connection from IDM to the Active Directory Server is through an LDAP connector, and that the mapping from managed users to the users in Active Directory (in your project's `conf/sync.json` file) identifies the Active Directory target as `system/ad/account`. If you have named the target differently, modify the `"queryOnResource" : "system/ad/account"` property accordingly.

Add the IWA authentication module towards the end of your `conf/authentication.json` file. For example:

```
"authModules" : [
    ...
    {
        "name": "IWA",
        "properties": {
            "servicePrincipal": "HTTP/openidm.example.com@EXAMPLE.COM",
            "keytabFileName": "C:\\Users\\Administrator\\openidm\\security\\openidm.HTTP.keytab",
            "kerberosRealm": "EXAMPLE.COM",
            "kerberosServerName": "kdc.example.com",
            "queryOnResource": "system/ad/account",
            "maxTokenSize": 48000,
            "propertyMapping": {
                "authenticationId": "sAMAccountName",
                "groupMembership": "memberOf"
            },
            "groupRoleMapping": {
                "internal/role/openidm-admin": [ ]
            },
            "groupComparisonMethod": "ldap",
            "defaultUserRoles": [
                "internal/role/openidm-authorized"
            ],
            "augmentSecurityContext": {
                "type": "text/javascript",
                "file": "auth/populateAsManagedUser.js"
            }
        },
        "enabled": true
    },
    ...
]
```

The IWA authentication module includes the following configurable properties:

`servicePrincipal`

The Kerberos principal for authentication, in the following format:

```
HTTP/host.domain@DC-DOMAIN-NAME
```

host and *domain* correspond to the host and domain names of the IDM server. *DC-DOMAIN-NAME* is the domain name of the Windows Kerberos domain controller server. The *DC-DOMAIN-NAME* can differ from the domain name for the IDM server.

`keytabFileName`

The full path to the keytab file for the Service Principal. On Windows systems, any backslash (`\`) characters in the path must be escaped, as shown in the previous example.

`kerberosRealm`

The Kerberos Key Distribution Center realm. For the Windows Kerberos service, this is the domain controller server domain name.

`kerberosServerName`

The fully qualified domain name of the Kerberos Key Distribution Center server, such as that of the domain controller server.

`queryOnResource`

The IDM resource to check for the authenticating user; for example, `system/ad/account`.

`maxTokenSize`

During the Kerberos authentication process, the Windows server builds a token to represent the user for authorization. This property sets the maximum size of the token, to prevent DoS attacks, if the SPENGO token in the request being made is amended with extra data. The default maximum token size is `48000` bytes.

`groupRoleMapping`

Enables you to grant different roles to users who are authenticated through the `IWA` module.

You can use the `IWA` module in conjunction with the `PASSTHROUGH` authentication module. In this case, a failure in the `IWA` module lets users revert to forms-based authentication.

To add the `PASSTHROUGH` module, follow "PASSTHROUGH".

Authenticate through AM

When you use IDM and AM together in a *platform deployment*, you configure IDM to use AM bearer tokens for authentication, instead of setting up traditional authentication modules. This delegates all authentication to AM. In this configuration, IDM uses an `rsFilter` that replaces all other authentication methods.

With AM bearer tokens, all IDM endpoints that require authentication are accessed using an authorization header that contains the bearer token, instead of `X-OpenIDM-Username` and `X-OpenIDM-Password`. Endpoints that allow anonymous access can be accessed without a token.

Important

- From IDM 7.0 onwards, using AM bearer tokens for authentication is the *only supported method* of integrating IDM with AM.
- To use AM bearer tokens for authentication, your AM configuration must include at least the following configuration:
  - Two OAuth 2.0 clients: an `idm-resource-server` client to introspect the bearer token, and an `idm-provisioning` client used by AM to provision users in IDM. For information on configuring these clients, see Configure OAuth Clients in the *Platform Setup Guide*.
  - An OAuth 2 provider service.
  - An IDM provisioning service.

  Your IDM authentication configuration *(You can manage the authentication configuration over REST at the config/authentication endpoint, or directly in the conf/authentication.json file.)* must include the `rsFilter` configuration and *no other* authentication methods.

The following sample `rsFilter` configuration is also available in `/path/to/openidm/samples/example-configurations/conf/rsfilter/`:

+ *Sample rsFilter Authentication*

```
{
  "rsFilter" : {
    "clientId" : "",
    "clientSecret" : "",
    "tokenIntrospectUrl" : "http://am.example:8080/openam/oauth2/introspect",
    "scopes" : [ ],
    "cache" : {
      "maxTimeout" : "300 seconds"
    },
    "augmentSecurityContext" : {
      "type" : "text/javascript",
      "source" : "require('auth/orgPrivileges').assignPrivilegesToUser(resource, security, properties,
 subjectMapping, privileges, 'privileges', 'privilegeAssignments');"
    },
    "subjectMapping" : [
      {
        "resourceTypeMapping" : {
          "usr" : "managed/user"
        },
        "propertyMapping" : {
          "sub" : "_id"
```

```
            },
            "userRoles" : "authzRoles/*",
            "additionalUserFields" : [
              "adminOfOrg",
              "ownerOfOrg"
            ],
            "defaultRoles" : [
              "internal/role/openidm-authorized"
            ]
          }
        ],
        "staticUserMapping" : [
          {
            "subject" : "(usr!amadmin)",
            "localUser" : "internal/user/openidm-admin",
            "roles" : [
              "internal/role/openidm-authorized",
              "internal/role/openidm-admin"
            ]
          },
          {
            "subject" : "(age!idm-provisioning)",
            "localUser" : "internal/user/idm-provisioning",
            "roles" : [
              "internal/role/platform-provisioning"
            ]
          }
        ],
        "anonymousUserMapping" : {
          "localUser" : "internal/user/anonymous",
          "roles" : [
            "internal/role/openidm-reg"
          ]
        }
      }
    }
```

The `rsFilter` configuration includes the following properties:

`clientId`

The client ID of the AM OAuth 2.0 client used to introspect the bearer token (the `idm-resource-server`) client, in this example).

`clientSecret`

The client secret of the AM OAuth 2.0 client used to introspect the bearer token. IDM will encrypt this field if it isn't encrypted already.

`tokenIntrospectUrl`

The URI to reach the `oauth2/introspect` endpoint in AM, for example, `http://am.example:8080/openam/oauth2/introspect`.

`scopes`

Any scopes that are required to be present in the access token. This will vary depending on your configuration. For more information about scopes, see About Scopes in the AM *OAuth 2.0 Guide*.

`cache`

Sets the `maxTimeout`, in seconds, after which the token is removed from the cache.

`augmentSecurityContext`

Specifies a script that is executed only after a successful authentication request. The script helps to populate the expected security context. For more information, see "The `augmentSecurityContext` Trigger" in the *Scripting Guide*.

`subjectMapping`

An array of mappings that let you map AM realms to IDM managed object types. For example:

```
"subjectMapping" : [
    {
        "resourceTypeMapping" : {
            "usr" : "managed/user"
        },
        "propertyMapping" : {
            "sub" : "_id"
        },
        "userRoles" : "authzRoles/*",
        "additionalUserFields" : [
            "adminOfOrg",
            "ownerOfOrg"
        ],
        "defaultRoles" : [
            "internal/role/openidm-authorized"
        ]
    }
],
```

Each `subjectMapping` includes the following properties:

- Either a `resourceTypeMapping` or a `queryOnResource` property:
  - `resourceTypeMapping`: Maps the identity type of a subject claim in AM to a resource collection in IDM. In the access token, the subject claim is a compound identity that consists of the claim `type` and subject name, separated by a `!`.

    To use a `resourceTypeMapping`, unique Oauth2 subject claims must be enabled in AM. (From AM 7.1, these are enabled by default.) For more information about subject claims, see *About the Subject and the Subname Claims* in the section on /oauth2/userinfo.
  - `queryOnResource`: The IDM resource to check for the authenticating user; for example, `managed/user`.

Both the resourceTypeMapping and the queryOnResource properties support a dynamic handlebars template that lets a single subject mapping match multiple realms, if the managed objects are named prescriptively, and based on the realm name. For example:

```
"resourceTypeMapping" : {
    "usr" : "managed/{{substring realm 1}}"
}
```

This configuration lets an access token with the realm employee map to an IDM managed/employee, and an access token with the realm contractor map to an IDM managed/contractor. The configuration is useful if your AM and IDM deployments use a consistent realm and managed object naming.

- realm: The AM realm to which this subject mapping applies. A value of / specifies the top-level realm. If this property is absent, the mapping can apply to any realm, which is useful if the resourceTypeMapping or queryOnResource uses a dynamic handlebars template.

  You cannot have more than one mapping for the same realm, and you cannot have more than one mapping that has no realm in the configuration.

- propertyMapping: Maps fields in the AM access token to properties in IDM. This mapping is used to construct the query filter that locates the authenticating user. The default configuration maps the subject (sub) in the access token to the _id in IDM.

- userRoles: Determines the field to consult for locating the authorization roles; usually authzRoles, unless you have changed how user roles are stored. This field must be a relationship field. IDM uses the _refId from the array elements to populate the user roles in the security context.

- additionalUserFields: Determines the field to consult for locating the authorization roles; usually authzRoles, unless you have changed how user roles are stored. This field must be a relationship field. IDM uses the _refId from the array elements to populate the user roles in the security context.

- defaultRoles: The default roles that should be applied to a user who authenticates using the rsFilter.

Although you can configure an array of subject mappings, only one mapping is selected and used during the authentication process. If there is a realm attribute in the access token, that realm is used to select an appropriate mapping. If no mapping is defined for the access token's realm, or if the realm is not provided in the access token, the authentication uses a mapping that does not define a realm.

Note

If you have a remote connector server that is authenticating against AM, you must add a subject mapping specifically for the connector server. For example:

```
{
    "subject" : "RCS-OAuth-clientId",
    "localUser" : "internal/user/idm-provisioning"
}
```

The `subject` must reflect the OAuth2 client in AM that has been set up for the remote connector server. The `localUser` can be any existing user. Do not assign that user any roles to ensure that the connector server bearer token cannot be used for any other purpose.

`staticUserMapping`

Maps AM users to a matching IDM user. Can contain multiple user mappings, each with three properties: `subject`, `localUser`, and `roles`.

- `subject` of the access token (the AM user). If you have specified a `resourceTypeMapping`, the static user mapping includes the full new subject string to match service accounts or static subject mappings, for example:

  ```
  "subject" : "(usr!amadmin)"
  ```

- `localUser` is the IDM user you want to associate with the AM user identified in `subject`. For example, if `subject` is set to `(usr!amadmin)`, you might set the corresponding `localUser` to `internal/user/openidm-admin`.

- `roles` the default IDM roles that this mapped user will have after they authenticate.

Note

The `idm-provisioning` subject is a service account used by AM to provision users in IDM. You *must* include this subject in your `staticUserMapping`. For example:

```
{
    "subject": "(age!idm-provisioning)",
    "localUser": "internal/user/idm-provisioning",
    "roles" : [
        "internal/role/platform-provisioning"
    ]
}
```

`anonymousUserMapping`

The default user that will be used when no access token is included in the request. Contains two properties: `localUser` and `userRoles`.

- `localUser`: the IDM user resource referenced when no specific user is identified. For example, `internal/user/anonymous`.

- `roles`: the default roles that the anonymous user will have, usually `internal/role/openidm-reg`.

Test Authentication Through AM

1. Obtain a bearer token from AM. For example:

```
curl \
--header "X-OpenAM-Username: amAdmin" \
--header "X-OpenAM-Password: password" \
--header "Accept-API-Version: resource=2.0, protocol=1.0" \
--request POST \
--data "grant_type=client_credentials" \
--data "client_id=idm-provisioning" \
--data "client_secret=openidm" \
--data "scope=fr:idm:*" \
"http://am.example.com:8080/am/oauth2/realms/root/access_token"
{
  "access_token": "z4uKDWiv4wnxKY7OjeG04PujG8E",
  "scope": "fr:idm:*",
  "token_type": "Bearer",
  "expires_in": 3599
}
```

2. Authenticate to IDM using that bearer token:

```
curl \
--request GET \
--header "Content-Type: application/json" \
--header "Authorization: Bearer z4uKDWiv4wnxKY7OjeG04PujG8E" \
'http://localhost:8080/openidm/info/login'
{
  "_id": "login",
  "authenticationId": "idm-provisioning",
  "authorization": {
    "id": "idm-provisioning",
    "roles": [
      "internal/role/platform-provisioning"
    ],
    "component": "internal/user"
  }
}
```

See the Platform Setup Guide for complete instructions on setting up IDM to use AM bearer tokens for authentication.

Authenticate as a Different User

The `X-OpenIDM-RunAs` header lets an administrative user *masquerade* as a regular user, without needing that user's password. To support this header, you must add the `runAsProperties` object to the `properties` of your authentication module. For example:

```
"runAsProperties" : {
    "adminRoles" : [
        "internal/role/openidm-admin"
    ],
    "disallowedRunAsRoles" : [
        "internal/role/openidm-admin"
    ],
    "defaultUserRoles" : [
        "internal/role/openidm-authorized"
    ],
    "queryId" : "credential-query",
    "queryOnResource" : "managed/user",
    "propertyMapping" : {
        "authenticationId" : "username",
        "userRoles" : "authzRoles"
    },
    "augmentSecurityContext" : {
        "type" : "text/javascript",
        "source" : "require('auth/customAuthz').setProtectedAttributes(security)"
    }
}
```

This configuration lets a user who authenticates with the `openidm-admin` role masquerade as any user *except* one with the `openidm-admin` role.

If you are adding this configuration to the `STATIC_USER` module, and you are using Delegated Administration, you must add an additional `propertyMapping` to the `properties` of the authentication module. This mapping forces the privileges to be re-read into the security context when the session JWT is used on subsequent requests. For example:

```
"propertyMapping" : {
    "authenticationId" : "username"
}
```

The sample `authentication.json` file in `openidm/samples/example-configurations/conf/runas/` adds the `runAsProperties` object to the `STATIC_USER` module. Users or clients who authenticate using this module can then masquerade as other users.

In the following example, the `openidm-admin` user authenticates with the `STATIC_USER` module, and can run REST calls as user `bjensen` without that user's password:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "X-OpenIDM-RunAs: bjensen" \
--request GET \
"http://localhost:8080/openidm/info/login"
{
  "_id": "login",
  "authenticationId": "bjensen",
  "authorization": {
    "userRolesProperty": "authzRoles",
    "component": "managed/user",
    "authLogin": false,
    "adminUser": "openidm-admin",
    "roles": [
      "internal/role/openidm-authorized"
    ],
    "ipAddress": "[0:0:0:0:0:0:0:1]",
    "authenticationId": "openidm-admin",
    "protectedAttributeList": [
      "password"
    ],
    "id": "bjensen",
    "moduleId": "STATIC_USER",
    "queryId": "credential-query"
  }
}
```

The authentication output shows that the request was made as user `bjensen` but with an `adminUser` of `openidm-admin`. This information is also logged in the authentication audit log.

If you were to actually authenticate as user `bjensen`, without the `runAs` header, the user is authenticated with the `MANAGED_USER` authentication module. The output still shows an `authenticationId` of `bjensen` but there is no reference to an `adminUser`:

```
curl \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--header "Accept-API-Version: resource=1.0" \
--request GET \
"http://localhost:8080/openidm/info/login"
{
  "_id": "login",
  "authenticationId": "bjensen",
  "authorization": {
    "userRolesProperty": "authzRoles",
    "component": "managed/user",
    "authLogin": false,
    "authenticationIdProperty": "username",
    "roles": [
      "internal/role/openidm-authorized"
    ],
    "ipAddress": "[0:0:0:0:0:0:0:1]",
    "authenticationId": "bjensen",
    "protectedAttributeList": [
      "password"
    ],
    "id": "bjensen",
    "moduleId": "MANAGED_USER",
    "queryId": "credential-query"
  }
}
```

Authentication and Roles

Depending on how a user authenticates, the user is given a set of default *internal roles*. These roles determine how much access the user has to the server. IDM includes a number of default internal roles, on the `openidm/internal/roles` endpoint. You can configure additional internal roles to customize how you restrict access to the server.

The following internal roles are defined by default (in `conf/repo.init.json`):

openidm-admin

IDM administrator role, excluded from the reauthorization required policy definition by default.

openidm-authorized

Default role for any user who authenticates with a username and password.

openidm-cert

Default role for any user who authenticates with mutual SSL authentication.

This role applies only to mutual authentication. The shared secret (certificate) must be adequately protected. The `openidm-cert` role is excluded from the reauthorization required policy definition by default.

openidm-reg

Assigned to users who access IDM with the default anonymous account.

The `openidm-reg` role is excluded from the reauthorization required policy definition by default.

openidm-tasks-manager

Role for users who can be assigned to workflow tasks.

platform-provisioning

Role for platform provisioning access. If you are not planning to run AM and IDM together as a platform, you can safely remove this role.

When a user authenticates, IDM calculates that user's roles as follows:

- Each authentication module includes a `defaultUserRoles` property. Depending on how the user authenticates, IDM assigns the roles listed in that module's `defaultUserRoles` property to the user on authentication. The `defaultUserRoles` property is specified as an array. For most authentication modules, the user obtains the `openidm-authorized` role on authentication. For example:

```
{
    "name" : "MANAGED_USER",
    "properties" : {
        ...
        "defaultUserRoles" : [
            "internal/role/openidm-authorized"
        ]
    },
    ...
}
```

- The `userRoles` property in an authentication module maps to an attribute (or list of attributes) in a user entry that contains that user's authorization roles. This attribute is usually `authzRoles`, unless you have changed how user roles are stored.

  Any internal roles that are conditionally applied are also calculated and included in the user's `authzRoles` property at this point.

- If the authentication module includes a `groupRoleMapping`, `groupMembership`, or `groupComparison` property, IDM can assign additional roles to the user, depending on the user's group membership on an *external* system. For more information, see "Use Groups to Control Access to IDM" in the *Object Modeling Guide*.

Note

The roles calculated in sequence are cumulative. Roles with temporal restrictions are not included in that list if the current time is outside of the time assigned to the role.

Dynamic Role Calculation

By default, IDM calculates a user's roles only on authentication. You can configure IDM to recalculate a user's roles dynamically, with each request, instead of only when the user reauthenticates. To enable this feature, set `enableDynamicRoles` to `true` in the `JWT_SESSION` session module in `authentication.json`:

To enable dynamic role calculation through the Admin UI, select Configure > Authentication > Session > Enable Dynamic Roles.

Dynamic role calculation can be used independently of the *privileges* mechanism, but is required for privileges to work. For more information about privileges, see "How Privileges Restrict Administrative Access".

Roles, Authentication, and the Security Context

The Security Context (`context.security`), consists of a principal (defined by the `authenticationId` property) and an access control element (defined by the `authorization` property).

If authentication is successful, the authentication framework sets the principal. IDM stores that principal as the `authenticationId`.

The `authorization` property includes an `id`, an array of `roles`, and a `component`, that specifies the resource against which authorization is validated.

Chapter 2
Authorization and Access Control

IDM provides role-based authorization that restricts direct HTTP access to REST interface URLs. This access control applies to direct HTTP calls only. Access for internal calls (for example, calls from scripts) is not affected by this mechanism.

- "Authorization and Roles"
- "Administrative Users"

Authorization and Roles

When a user authenticates, they are given a set of default *roles*, as described in "Authentication and Roles". The authorization configuration grants access rights to users, based on these roles acquired during authentication.

You can use internal and managed roles to restrict access, with the following caveats:

- Internal roles are not meant to be provisioned or synchronized with external systems.
- Internal roles cannot be given assignments.
- Event scripts (such as `onCreate`) cannot be attached to internal roles.
- The internal role schema is not configurable.

Authorization roles are referenced in a user's `authzRoles` property by default, and are assigned when the user authenticates.

By default, managed users are assigned the `openidm-authorized` role when they authenticate. The following request shows the authorization roles for user psmith when that user logs in to the server:

```
curl \
--header "X-OpenIDM-Username: psmith" \
--header "X-OpenIDM-Password: Passw0rd" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--request GET \
"https://localhost:8443/openidm/managed/user/openidm/info/login"
{
  "_id": "login",
  "authenticationId": "psmith",
  "authorization": {
    "userRolesProperty": "authzRoles",
    "component": "managed/user",
    "authLogin": false,
    "authenticationIdProperty": "username",
    "roles": [
      "internal/role/openidm-authorized"
    ],
    "ipAddress": "0:0:0:0:0:0:0:1",
    "authenticationId": "psmith",
    "protectedAttributeList": [
      "password"
    ],
    "id": "psmith",
    "moduleId": "MANAGED_USER",
    "queryId": "credential-query"
  }
}
```

The authorization implementation is configured in two files:

- `openidm/bin/defaults/script/router-authz.js`
- *`project-dir`*`/conf/access.json`

IDM calls the `router-authz.js` script for each request, through an `onRequest` hook defined in the `router.json` file. `router-authz.js` references your project's access configuration (`access.json`) to determine the allowed HTTP requests. If access is denied, according to the configuration defined in `access.json`, the `router-authz.js` script throws an exception, and IDM denies the request.

`router.json` also defines an `onResponse` script, `relationshipFilter`. This provides additional filtering to ensure that the user has the appropriate access to see the data of the related object. You can change this behavior by extending or updating `/bin/defaults/script/relationshipFilter.js`, or by removing the `onResponse` script if you don't want additional filtering on relationships. For more information about relationships, see "*Relationships Between Objects*" in the *Object Modeling Guide*.)

Note

You can configure delegated administration to grant access that bypasses this access control.

The Router Authorization Script

The router authorization script (`router-authz.js` contains a number of functions that enforce access rules. For example, the following function controls whether users with a certain role can start a specified process:

```
function isAllowedToStartProcess() {
    var processDefinitionId = request.content._processDefinitionId;
    var key = request.content._key;
    return isProcessOnUsersList(function (process) {
        return (process._id === processDefinitionId) || (process.key === key);
    });
}
```

You can extend the default authorization mechanism by defining additional functions in `router-authz.js` and by creating new access control rules in `access.json`.

Important

Some authorization-related functions in `router-authz.js` should *not* be altered, because they affect the security of the server. Such functions are indicated in the comments in that file.

Configure Access Control in `access.json`

The `access.json` configuration includes a set of rules that govern access to specific endpoints. These rules are tested in the order in which they appear in the file. You can define more than one rule for the same endpoint. If one rule passes, the request is allowed. If all the rules fail, the request is denied.

The following rule (from a default `access.json` file) shows the access configuration structure:

```
{
    "pattern"   : "system/*",
    "roles"     : "internal/role/openidm-admin",
    "methods"   : "action",
    "actions"   : "test,testConfig,createconfiguration,liveSync,authenticate"
}
```

This rule specifies that users with the `openidm-admin` role can perform the listed actions on all `system` endpoints.

The parameters in each access rule are as follows:

`pattern`

The REST endpoint for which access is being controlled. `"*"` specifies access to all endpoints in that path. For example, `"managed/user/*"` specifies access to all managed user objects.

`roles`

A comma-separated list of the roles to which this access configuration applies.

The `roles` referenced here align with the object's security context (`security.authorization.roles`). The `authzRoles` relationship property of a managed user produces this security context value during authentication.

`methods`

A comma-separated list of the methods that can be performed with this access. Methods can include `create, read, update, delete, patch, action, query`. A value of `"*"` indicates that all methods are allowed. A value of `""` indicates that no methods are allowed.

`actions`

A comma-separated list of the allowed actions. The possible actions depend on the resource (URL) that is being exposed. Note that the `actions` in the default `access.json` file do not list all the supported actions in the *Scripting Guide* on each resource.

A value of `"*"` indicates that all actions exposed for that resource are allowed. A value of `""` indicates that no actions are allowed.

`customAuthz`

An optional parameter that lets you define a custom function for additional authorization checks. Custom functions are defined in `router-authz.js`.

`excludePatterns`

An optional parameter that lets you specify endpoints to which access should not be granted.

Change the Access Configuration Over REST

You can manage the access configuration at the endpoint `openidm/config/access`. To change an access rule, first get the current access configuration, amend it to change the access rule, then submit the updated configuration in a PUT request. This example restricts access to the `info` endpoint to users who have authenticated:

+ *Get the Current Access Configuration*

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0"  \
--request GET \
"http://localhost:8080/openidm/config/access"
{
  "_id": "access",
  "configs": [
    {
      "pattern": "info/*",
      "roles": "*",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "authentication",
      "roles": "*",
      "methods": "read,action",
      "actions": "login,logout"
    },
    {
      "pattern": "identityProviders",
      "roles": "*",
      "methods": "action",
      "actions": "getAuthRedirect,handlePostAuth,getLogoutUrl"
    },
    {
      "pattern": "identityProviders",
      "roles": "internal/role/openidm-authorized",
      "methods": "action",
      "actions": "normalizeProfile"
    },
    {
      "pattern": "identityProviders",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "config/ui/themeconfig",
      "roles": "*",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "info/uiconfig",
      "roles": "*",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "config/selfservice/kbaConfig",
      "roles": "*",
      "methods": "read",
      "actions": "*",
      "customAuthz": "checkIfAnyFeatureEnabled(['registration', 'passwordReset'])"
    },
    {
      "pattern": "config/ui/dashboard",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "info/features",
      "roles": "*",
      "methods": "query",
      "actions": "*"
    },
    {
      "pattern": "privilege",
      "roles": "*",
      "methods": "action",
      "actions": "listPrivileges"
    },
    {
      "pattern": "privilege/*",
      "roles": "*",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "selfservice/registration",
      "roles": "*",
      "methods": "read,action",
      "actions": "submitRequirements",
      "customAuthz": "checkIfAnyFeatureEnabled('registration')"
    },
    {
      "pattern": "selfservice/socialUserClaim",
      "roles": "*",
      "methods": "read,action",
      "actions": "submitRequirements",
      "customAuthz": "checkIfAnyFeatureEnabled('registration')"
    },
    {
      "pattern": "selfservice/reset",
      "roles": "*",
      "methods": "read,action",
      "actions": "submitRequirements",
      "customAuthz": "checkIfAnyFeatureEnabled('passwordReset')"
    },
    {
      "pattern": "selfservice/username",
      "roles": "*",
      "methods": "read,action",
      "actions": "submitRequirements",
      "customAuthz": "checkIfAnyFeatureEnabled('retrieveUsername')"
    },
    {
      "pattern": "selfservice/profile",
      "roles": "*",
      "methods": "read,action",
      "actions": "submitRequirements"
    },
    {
      "pattern": "selfservice/termsAndConditions",
      "roles": "*",
      "methods": "read,action",
      "actions": "submitRequirements"
    },
    {
      "pattern": "selfservice/kbaUpdate",
      "roles": "*",
      "methods": "read,action",
      "actions": "submitRequirements"
    },
    {
      "pattern": "policy/*",
      "roles": "*",
      "methods": "action",
      "actions": "validateObject",
      "customAuthz": "context.current.name === 'selfservice'"
    },
    {
      "pattern": "policy/selfservice/registration",
      "roles": "*",
      "methods": "action,read",
      "actions": "validateObject",
      "customAuthz": "checkIfAnyFeatureEnabled('registration')"
    },
    {
      "pattern": "policy/selfservice/reset",
      "roles": "*",
      "methods": "action,read",
      "actions": "validateObject",
      "customAuthz": "checkIfAnyFeatureEnabled('passwordReset')"
    },
    {
      "pattern": "selfservice/kba",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*",
      "customAuthz": "checkIfAnyFeatureEnabled('kba')"
    },
    {
      "pattern": "managed/user",
      "roles": "internal/role/openidm-reg",
      "methods": "create",
      "actions": "*",
      "customAuthz": "checkIfAnyFeatureEnabled('registration') && isSelfServiceRequest() &&
onlyEditableManagedObjectProperties('user', [])"
    },
    {
      "pattern": "managed/user",
      "roles": "*",
      "methods": "query",
      "actions": "*",
      "customAuthz": "checkIfAnyFeatureEnabled(['registration', 'retrieveUsername', 'passwordReset'])
&& isSelfServiceRequest()"
    },
    {
      "pattern": "managed/user/*",
      "roles": "*",
      "methods": "read",
      "actions": "*",
      "customAuthz": "checkIfAnyFeatureEnabled(['retrieveUsername', 'passwordReset']) &&
isSelfServiceRequest()"
    },
    {
      "pattern": "managed/user/*",
      "roles": "*",
      "methods": "patch,action",
      "actions": "patch",
      "customAuthz": "(checkIfAnyFeatureEnabled(['registration', 'passwordReset'])
|| checkIfProgressiveProfileIsEnabled()) && isSelfServiceRequest() &&
onlyEditableManagedObjectProperties('user', [])"
```

```
    },
    {
      "pattern": "external/email",
      "roles": "*",
      "methods": "action",
      "actions": "send",
      "customAuthz": "checkIfAnyFeatureEnabled(['registration', 'retrieveUsername', 'passwordReset'])
 && isSelfServiceRequest()"
    },
    {
      "pattern": "schema/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "consent",
      "roles": "internal/role/openidm-authorized",
      "methods": "action,query",
      "actions": "*"
    },
    {
      "pattern": "*",
      "roles": "internal/role/openidm-admin",
      "methods": "*",
      "actions": "*",
      "excludePatterns": "repo,repo/*"
    },
    {
      "pattern": "system/*",
      "roles": "internal/role/openidm-admin",
      "methods": "create,read,update,delete,patch,query",
      "actions": ""
    },
    {
      "pattern": "system/*",
      "roles": "internal/role/openidm-admin",
      "methods": "script",
      "actions": "*"
    },
    {
      "pattern": "system/*",
      "roles": "internal/role/openidm-admin",
      "methods": "action",
      "actions": "test,testConfig,createconfiguration,liveSync,authenticate"
    },
    {
      "pattern": "repo",
      "roles": "internal/role/openidm-admin",
      "methods": "*",
      "actions": "*",
      "customAuthz": "disallowCommandAction()"
    },
    {
      "pattern": "repo/*",
      "roles": "internal/role/openidm-admin",
      "methods": "*",
      "actions": "*",
      "customAuthz": "disallowCommandAction()"
    },
    {
      "pattern": "repo/link",
      "roles": "internal/role/openidm-admin",
      "methods": "action",
      "actions": "command",
      "customAuthz": "request.additionalParameters.commandId === 'delete-mapping-links'"
    },
    {
      "pattern": "managed/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "create,read,query,patch"
    },
    {
      "pattern": "internal/role/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "read,query"
    },
    {
      "pattern": "profile/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "create,read,action,update",
      "actions": "*"
    },
    {
      "pattern": "policy/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "read,action",
      "actions": "*"
    },
    {
      "pattern": "schema/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "read"
    },
    {
      "pattern": "consent",
      "roles": "internal/role/platform-provisioning",
      "methods": "action,query",
      "actions": "*"
    },
    {
      "pattern": "selfservice/kba",
      "roles": "internal/role/platform-provisioning",
      "methods": "read"
    },
    {
      "pattern": "selfservice/terms",
      "roles": "internal/role/platform-provisioning",
      "methods": "read"
    },
    {
      "pattern": "identityProviders",
      "roles": "internal/role/platform-provisioning",
      "methods": "read"
    },
    {
      "pattern": "external/email",
      "roles": "internal/role/platform-provisioning",
      "methods": "action",
      "actions": "sendTemplate"
    },
    {
      "pattern": "policy/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,action",
      "actions": "*"
    },
    {
      "pattern": "config/ui/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "authentication",
      "roles": "internal/role/openidm-authorized",
      "methods": "action",
      "actions": "reauthenticate"
    },
    {
      "pattern": "*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,action,delete",
      "actions": "bind,unbind",
      "customAuthz": "ownDataOnly()"
    },
    {
      "pattern": "*",
      "roles": "internal/role/openidm-authorized",
      "methods": "update,patch,action",
      "actions": "patch",
      "customAuthz": "ownDataOnly() && onlyEditableManagedObjectProperties('user', []) &&
reauthIfProtectedAttributeChange()"
    },
    {
      "pattern": "selfservice/user/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "patch,action",
      "actions": "patch",
      "customAuthz": "(request.resourcePath === 'selfservice/user/' +
context.security.authorization.id) && onlyEditableManagedObjectProperties('user', [])"
    },
    {
      "pattern": "endpoint/getprocessesforuser",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "endpoint/gettasksview",
      "roles": "internal/role/openidm-authorized",
      "methods": "query",
      "actions": "*"
    },
    {
      "pattern": "workflow/taskinstance/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "action",
      "actions": "complete",
      "customAuthz": "isMyTask()"
    },
    {
      "pattern": "workflow/taskinstance/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,update",
      "actions": "*",
      "customAuthz": "canUpdateTask()"
    },
    {
      "pattern": "workflow/processinstance",
      "roles": "internal/role/openidm-authorized",
      "methods": "create",
      "actions": "*",
      "customAuthz": "isAllowedToStartProcess()"
    },
    {
      "pattern": "workflow/processdefinition/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "*",
      "actions": "read",
      "customAuthz": "isOneOfMyWorkflows()"
    },
    {
      "pattern": "managed/user",
      "roles": "internal/role/openidm-cert",
      "methods": "patch,action",
      "actions": "patch",
      "customAuthz": "isQueryOneOf({'managed/user': ['for-userName']}) &&
 restrictPatchToFields(['password'])"
    },
    {
      "pattern": "internal/usermeta/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*",
      "customAuthz": "ownRelationship()"
    },
    {
      "pattern": "internal/notification/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,delete",
      "actions": "*",
      "customAuthz": "ownRelationship()"
    },
    {
      "pattern": "managed/user/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,query",
      "actions": "*",
      "customAuthz": "ownRelationshipCollection(['idps','_meta','_notifications'])"
    },
    {
      "pattern": "notification",
      "roles": "internal/role/openidm-authorized",
      "methods": "action",
      "actions": "deleteNotificationsForTarget",
      "customAuthz": "request.additionalParameters.target ===
 (context.security.authorization.component + '/' + context.security.authorization.id)"
    },
    {
      "pattern": "managed/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*",
      "customAuthz": "ownIDP()"
    }
  ]
}
```

+ *Replace the Access Configuration*

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-type: application/json" \
--header "Accept-API-Version: resource=1.0"  \
--request PUT \
--data '{
  "_id": "access",
  "configs": [
    {
      "pattern": "info/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "authentication",
      "roles": "*",
      "methods": "read,action",
      "actions": "login,logout"
    },
    {
      "pattern": "identityProviders",
      "roles": "*",
      "methods": "action",
      "actions": "getAuthRedirect,handlePostAuth,getLogoutUrl"
    },
    {
      "pattern": "identityProviders",
      "roles": "internal/role/openidm-authorized",
      "methods": "action",
      "actions": "normalizeProfile"
    },
    {
      "pattern": "identityProviders",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "config/ui/themeconfig",
      "roles": "*",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "info/uiconfig",
      "roles": "*",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "config/selfservice/kbaConfig",
      "roles": "*",
      "methods": "read",
      "actions": "*",
      "customAuthz": "checkIfAnyFeatureEnabled(['registration', 'passwordReset'])"
    },
    {
      "pattern": "config/ui/dashboard",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "info/features",
      "roles": "*",
      "methods": "query",
      "actions": "*"
    },
    {
      "pattern": "privilege",
      "roles": "*",
      "methods": "action",
      "actions": "listPrivileges"
    },
    {
      "pattern": "privilege/*",
      "roles": "*",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "selfservice/registration",
      "roles": "*",
      "methods": "read,action",
      "actions": "submitRequirements",
      "customAuthz": "checkIfAnyFeatureEnabled('registration')"
    },
    {
      "pattern": "selfservice/socialUserClaim",
      "roles": "*",
      "methods": "read,action",
      "actions": "submitRequirements",
      "customAuthz": "checkIfAnyFeatureEnabled('registration')"
    },
    {
      "pattern": "selfservice/reset",
      "roles": "*",
      "methods": "read,action",
      "actions": "submitRequirements",
      "customAuthz": "checkIfAnyFeatureEnabled('passwordReset')"
    },
    {
      "pattern": "selfservice/username",
      "roles": "*",
      "methods": "read,action",
      "actions": "submitRequirements",
      "customAuthz": "checkIfAnyFeatureEnabled('retrieveUsername')"
    },
    {
      "pattern": "selfservice/profile",
      "roles": "*",
      "methods": "read,action",
      "actions": "submitRequirements"
    },
    {
      "pattern": "selfservice/termsAndConditions",
      "roles": "*",
      "methods": "read,action",
      "actions": "submitRequirements"
    },
    {
      "pattern": "selfservice/kbaUpdate",
      "roles": "*",
      "methods": "read,action",
      "actions": "submitRequirements"
    },
    {
      "pattern": "policy/*",
      "roles": "*",
      "methods": "action",
      "actions": "validateObject",
      "customAuthz": "context.current.name === 'selfservice'"
    },
    {
      "pattern": "policy/selfservice/registration",
      "roles": "*",
      "methods": "action,read",
      "actions": "validateObject",
      "customAuthz": "checkIfAnyFeatureEnabled('registration')"
    },
    {
      "pattern": "policy/selfservice/reset",
      "roles": "*",
      "methods": "action,read",
      "actions": "validateObject",
      "customAuthz": "checkIfAnyFeatureEnabled('passwordReset')"
    },
    {
      "pattern": "selfservice/kba",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*",
      "customAuthz": "checkIfAnyFeatureEnabled('kba')"
    },
    {
      "pattern": "managed/user",
      "roles": "internal/role/openidm-reg",
      "methods": "create",
      "actions": "*",
      "customAuthz": "checkIfAnyFeatureEnabled('registration') && isSelfServiceRequest() &&
onlyEditableManagedObjectProperties('user', [])"
    },
    {
      "pattern": "managed/user",
      "roles": "*",
      "methods": "query",
      "actions": "*",
      "customAuthz": "checkIfAnyFeatureEnabled(['registration', 'retrieveUsername', 'passwordReset'])
&& isSelfServiceRequest()"
    },
    {
      "pattern": "managed/user/*",
      "roles": "*",
      "methods": "read",
      "actions": "*",
      "customAuthz": "checkIfAnyFeatureEnabled(['retrieveUsername', 'passwordReset']) &&
isSelfServiceRequest()"
    },
    {
      "pattern": "managed/user/*",
      "roles": "*",
      "methods": "patch,action",
      "actions": "patch",
      "customAuthz": "(checkIfAnyFeatureEnabled(['registration', 'passwordReset'])
|| checkIfProgressiveProfileIsEnabled()) && isSelfServiceRequest() &&
onlyEditableManagedObjectProperties('user', [])"
    },
    {
      "pattern": "external/email",
      "roles": "*",
      "methods": "action",
      "actions": "send",
      "customAuthz": "checkIfAnyFeatureEnabled(['registration', 'retrieveUsername', 'passwordReset'])
&& isSelfServiceRequest()"
    },
    {
      "pattern": "schema/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "consent",
      "roles": "internal/role/openidm-authorized",
      "methods": "action,query",
      "actions": "*"
    },
    {
      "pattern": "*",
      "roles": "internal/role/openidm-admin",
      "methods": "*",
      "actions": "*",
      "excludePatterns": "repo,repo/*"
    },
    {
      "pattern": "system/*",
      "roles": "internal/role/openidm-admin",
      "methods": "create,read,update,delete,patch,query",
      "actions": ""
    },
    {
      "pattern": "system/*",
      "roles": "internal/role/openidm-admin",
      "methods": "script",
      "actions": "*"
    },
    {
      "pattern": "system/*",
      "roles": "internal/role/openidm-admin",
      "methods": "action",
      "actions": "test,testConfig,createconfiguration,liveSync,authenticate"
    },
    {
      "pattern": "repo",
      "roles": "internal/role/openidm-admin",
      "methods": "*",
      "actions": "*",
      "customAuthz": "disallowCommandAction()"
    },
    {
      "pattern": "repo/*",
      "roles": "internal/role/openidm-admin",
      "methods": "*",
      "actions": "*",
      "customAuthz": "disallowCommandAction()"
    },
    {
      "pattern": "repo/link",
      "roles": "internal/role/openidm-admin",
      "methods": "action",
      "actions": "command",
      "customAuthz": "request.additionalParameters.commandId === 'delete-mapping-links'"
    },
    {
      "pattern": "managed/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "create,read,query,patch"
    },
    {
      "pattern": "internal/role/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "read,query"
    },
    {
      "pattern": "profile/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "create,read,action,update",
      "actions": "*"
    },
    {
      "pattern": "policy/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "read,action",
      "actions": "*"
    },
    {
      "pattern": "schema/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "read"
    },
    {
      "pattern": "consent",
      "roles": "internal/role/platform-provisioning",
      "methods": "action,query",
      "actions": "*"
    },
    {
      "pattern": "selfservice/kba",
      "roles": "internal/role/platform-provisioning",
      "methods": "read"
    },
    {
      "pattern": "selfservice/terms",
      "roles": "internal/role/platform-provisioning",
      "methods": "read"
    },
    {
      "pattern": "identityProviders",
      "roles": "internal/role/platform-provisioning",
      "methods": "read"
    },
    {
      "pattern": "external/email",
      "roles": "internal/role/platform-provisioning",
      "methods": "action",
      "actions": "sendTemplate"
    },
    {
      "pattern": "policy/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,action",
      "actions": "*"
    },
    {
      "pattern": "config/ui/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "authentication",
      "roles": "internal/role/openidm-authorized",
      "methods": "action",
      "actions": "reauthenticate"
    },
    {
      "pattern": "*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,action,delete",
      "actions": "bind,unbind",
      "customAuthz": "ownDataOnly()"
    },
    {
      "pattern": "*",
      "roles": "internal/role/openidm-authorized",
      "methods": "update,patch,action",
      "actions": "patch",
      "customAuthz": "ownDataOnly() && onlyEditableManagedObjectProperties('user', []) &&
reauthIfProtectedAttributeChange()"
    },
    {
      "pattern": "selfservice/user/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "patch,action",
      "actions": "patch",
      "customAuthz": "(request.resourcePath === 'selfservice/user/' +
context.security.authorization.id) && onlyEditableManagedObjectProperties('user', [])"
    },
    {
      "pattern": "endpoint/getprocessesforuser",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "endpoint/gettasksview",
      "roles": "internal/role/openidm-authorized",
      "methods": "query",
      "actions": "*"
    },
    {
      "pattern": "workflow/taskinstance/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "action",
      "actions": "complete",
      "customAuthz": "isMyTask()"
    },
    {
      "pattern": "workflow/taskinstance/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,update",
      "actions": "*",
      "customAuthz": "canUpdateTask()"
    },
    {
      "pattern": "workflow/processinstance",
      "roles": "internal/role/openidm-authorized",
      "methods": "create",
      "actions": "*",
      "customAuthz": "isAllowedToStartProcess()"
    },
    {
      "pattern": "workflow/processdefinition/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "*",
      "actions": "read",
      "customAuthz": "isOneOfMyWorkflows()"
    },
    {
      "pattern": "managed/user",
      "roles": "internal/role/openidm-cert",
      "methods": "patch,action",
      "actions": "patch",
      "customAuthz": "isQueryOneOf({'managed/user': ['for-userName']}) &&
 restrictPatchToFields(['password'])"
    },
    {
      "pattern": "internal/usermeta/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*",
      "customAuthz": "ownRelationship()"
    },
    {
      "pattern": "internal/notification/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,delete",
      "actions": "*",
      "customAuthz": "ownRelationship()"
    },
    {
      "pattern": "managed/user/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,query",
      "actions": "*",
      "customAuthz": "ownRelationshipCollection(['idps','_meta','_notifications'])"
    },
    {
      "pattern": "notification",
      "roles": "internal/role/openidm-authorized",
      "methods": "action",
      "actions": "deleteNotificationsForTarget",
      "customAuthz": "request.additionalParameters.target ===
 (context.security.authorization.component + '/' + context.security.authorization.id)"
    },
    {
      "pattern": "managed/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*",
      "customAuthz": "ownIDP()"
    }
  ]
}' \
"http://localhost:8080/openidm/config/access"
{
  "_id": "access",
  "configs": [
    {
      "pattern": "info/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "authentication",
      "roles": "*",
      "methods": "read,action",
      "actions": "login,logout"
    },
    {
      "pattern": "identityProviders",
      "roles": "*",
      "methods": "action",
      "actions": "getAuthRedirect,handlePostAuth,getLogoutUrl"
    },
    {
      "pattern": "identityProviders",
      "roles": "internal/role/openidm-authorized",
      "methods": "action",
      "actions": "normalizeProfile"
    },
    {
      "pattern": "identityProviders",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "config/ui/themeconfig",
      "roles": "*",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "info/uiconfig",
      "roles": "*",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "config/selfservice/kbaConfig",
      "roles": "*",
      "methods": "read",
      "actions": "*",
      "customAuthz": "checkIfAnyFeatureEnabled(['registration', 'passwordReset'])"
    },
    {
      "pattern": "config/ui/dashboard",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "info/features",
      "roles": "*",
      "methods": "query",
      "actions": "*"
    },
    {
      "pattern": "privilege",
      "roles": "*",
      "methods": "action",
      "actions": "listPrivileges"
    },
    {
      "pattern": "privilege/*",
      "roles": "*",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "selfservice/registration",
      "roles": "*",
      "methods": "read,action",
      "actions": "submitRequirements",
      "customAuthz": "checkIfAnyFeatureEnabled('registration')"
    },
    {
      "pattern": "selfservice/socialUserClaim",
      "roles": "*",
      "methods": "read,action",
      "actions": "submitRequirements",
      "customAuthz": "checkIfAnyFeatureEnabled('registration')"
    },
    {
      "pattern": "selfservice/reset",
      "roles": "*",
      "methods": "read,action",
      "actions": "submitRequirements",
      "customAuthz": "checkIfAnyFeatureEnabled('passwordReset')"
    },
    {
      "pattern": "selfservice/username",
      "roles": "*",
      "methods": "read,action",
      "actions": "submitRequirements",
      "customAuthz": "checkIfAnyFeatureEnabled('retrieveUsername')"
    },
    {
      "pattern": "selfservice/profile",
      "roles": "*",
      "methods": "read,action",
      "actions": "submitRequirements"
    },
    {
      "pattern": "selfservice/termsAndConditions",
      "roles": "*",
      "methods": "read,action",
      "actions": "submitRequirements"
    },
    {
      "pattern": "selfservice/kbaUpdate",
      "roles": "*",
      "methods": "read,action",
      "actions": "submitRequirements"
    },
    {
      "pattern": "policy/*",
      "roles": "*",
      "methods": "action",
      "actions": "validateObject",
      "customAuthz": "context.current.name === 'selfservice'"
    },
    {
      "pattern": "policy/selfservice/registration",
      "roles": "*",
      "methods": "action,read",
      "actions": "validateObject",
      "customAuthz": "checkIfAnyFeatureEnabled('registration')"
```

```
   },
   {
     "pattern": "policy/selfservice/reset",
     "roles": "*",
     "methods": "action,read",
     "actions": "validateObject",
     "customAuthz": "checkIfAnyFeatureEnabled('passwordReset')"
   },
   {
     "pattern": "selfservice/kba",
     "roles": "internal/role/openidm-authorized",
     "methods": "read",
     "actions": "*",
     "customAuthz": "checkIfAnyFeatureEnabled('kba')"
   },
   {
     "pattern": "managed/user",
     "roles": "internal/role/openidm-reg",
     "methods": "create",
     "actions": "*",
     "customAuthz": "checkIfAnyFeatureEnabled('registration') && isSelfServiceRequest() &&
onlyEditableManagedObjectProperties('user', [])"
   },
   {
     "pattern": "managed/user",
     "roles": "*",
     "methods": "query",
     "actions": "*",
     "customAuthz": "checkIfAnyFeatureEnabled(['registration', 'retrieveUsername', 'passwordReset'])
&& isSelfServiceRequest()"
   },
   {
     "pattern": "managed/user/*",
     "roles": "*",
     "methods": "read",
     "actions": "*",
     "customAuthz": "checkIfAnyFeatureEnabled(['retrieveUsername', 'passwordReset']) &&
isSelfServiceRequest()"
   },
   {
     "pattern": "managed/user/*",
     "roles": "*",
     "methods": "patch,action",
     "actions": "patch",
     "customAuthz": "(checkIfAnyFeatureEnabled(['registration', 'passwordReset'])
|| checkIfProgressiveProfileIsEnabled()) && isSelfServiceRequest() &&
onlyEditableManagedObjectProperties('user', [])"
   },
   {
     "pattern": "external/email",
     "roles": "*",
     "methods": "action",
     "actions": "send",
     "customAuthz": "checkIfAnyFeatureEnabled(['registration', 'retrieveUsername', 'passwordReset'])
&& isSelfServiceRequest()"
   },
   {
     "pattern": "schema/*",
     "roles": "internal/role/openidm-authorized",
     "methods": "read",
     "actions": "*"
   },
   {
     "pattern": "consent",
     "roles": "internal/role/openidm-authorized",
     "methods": "action,query",
     "actions": "*"
   },
   {
     "pattern": "*",
     "roles": "internal/role/openidm-admin",
     "methods": "*",
     "actions": "*",
     "excludePatterns": "repo,repo/*"
   },
   {
     "pattern": "system/*",
     "roles": "internal/role/openidm-admin",
     "methods": "create,read,update,delete,patch,query",
     "actions": ""
   },
   {
     "pattern": "system/*",
     "roles": "internal/role/openidm-admin",
     "methods": "script",
     "actions": "*"
   },
   {
     "pattern": "system/*",
     "roles": "internal/role/openidm-admin",
     "methods": "action",
     "actions": "test,testConfig,createconfiguration,liveSync,authenticate"
   },
   {
     "pattern": "repo",
     "roles": "internal/role/openidm-admin",
     "methods": "*",
     "actions": "*",
     "customAuthz": "disallowCommandAction()"
   },
   {
     "pattern": "repo/*",
     "roles": "internal/role/openidm-admin",
     "methods": "*",
     "actions": "*",
     "customAuthz": "disallowCommandAction()"
   },
   {
     "pattern": "repo/link",
     "roles": "internal/role/openidm-admin",
     "methods": "action",
     "actions": "command",
     "customAuthz": "request.additionalParameters.commandId === 'delete-mapping-links'"
   },
   {
     "pattern": "managed/*",
     "roles": "internal/role/platform-provisioning",
     "methods": "create,read,query,patch"
```

```
    },
    {
      "pattern": "internal/role/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "read,query"
    },
    {
      "pattern": "profile/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "create,read,action,update",
      "actions": "*"
    },
    {
      "pattern": "policy/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "read,action",
      "actions": "*"
    },
    {
      "pattern": "schema/*",
      "roles": "internal/role/platform-provisioning",
      "methods": "read"
    },
    {
      "pattern": "consent",
      "roles": "internal/role/platform-provisioning",
      "methods": "action,query",
      "actions": "*"
    },
    {
      "pattern": "selfservice/kba",
      "roles": "internal/role/platform-provisioning",
      "methods": "read"
    },
    {
      "pattern": "selfservice/terms",
      "roles": "internal/role/platform-provisioning",
      "methods": "read"
    },
    {
      "pattern": "identityProviders",
      "roles": "internal/role/platform-provisioning",
      "methods": "read"
    },
    {
      "pattern": "external/email",
      "roles": "internal/role/platform-provisioning",
      "methods": "action",
      "actions": "sendTemplate"
    },
    {
      "pattern": "policy/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,action",
      "actions": "*"
    },
    {
      "pattern": "config/ui/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "authentication",
      "roles": "internal/role/openidm-authorized",
      "methods": "action",
      "actions": "reauthenticate"
    },
    {
      "pattern": "*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,action,delete",
      "actions": "bind,unbind",
      "customAuthz": "ownDataOnly()"
    },
    {
      "pattern": "*",
      "roles": "internal/role/openidm-authorized",
      "methods": "update,patch,action",
      "actions": "patch",
      "customAuthz": "ownDataOnly() && onlyEditableManagedObjectProperties('user', []) &&
 reauthIfProtectedAttributeChange()"
    },
    {
      "pattern": "selfservice/user/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "patch,action",
      "actions": "patch",
      "customAuthz": "(request.resourcePath === 'selfservice/user/' +
 context.security.authorization.id) && onlyEditableManagedObjectProperties('user', [])"
    },
    {
      "pattern": "endpoint/getprocessesforuser",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*"
    },
    {
      "pattern": "endpoint/gettasksview",
      "roles": "internal/role/openidm-authorized",
      "methods": "query",
      "actions": "*"
    },
    {
      "pattern": "workflow/taskinstance/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "action",
      "actions": "complete",
      "customAuthz": "isMyTask()"
    },
    {
      "pattern": "workflow/taskinstance/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,update",
      "actions": "*",
      "customAuthz": "canUpdateTask()"
    },
    {
      "pattern": "workflow/processinstance",
      "roles": "internal/role/openidm-authorized",
      "methods": "create",
      "actions": "*",
      "customAuthz": "isAllowedToStartProcess()"
    },
    {
      "pattern": "workflow/processdefinition/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "*",
      "actions": "read",
      "customAuthz": "isOneOfMyWorkflows()"
    },
    {
      "pattern": "managed/user",
      "roles": "internal/role/openidm-cert",
      "methods": "patch,action",
      "actions": "patch",
      "customAuthz": "isQueryOneOf({'managed/user': ['for-userName']}) &&
 restrictPatchToFields(['password'])"
    },
    {
      "pattern": "internal/usermeta/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*",
      "customAuthz": "ownRelationship()"
    },
    {
      "pattern": "internal/notification/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,delete",
      "actions": "*",
      "customAuthz": "ownRelationship()"
    },
    {
      "pattern": "managed/user/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read,query",
      "actions": "*",
      "customAuthz": "ownRelationshipCollection(['idps','_meta','_notifications'])"
    },
    {
      "pattern": "notification",
      "roles": "internal/role/openidm-authorized",
      "methods": "action",
      "actions": "deleteNotificationsForTarget",
      "customAuthz": "request.additionalParameters.target ===
 (context.security.authorization.component + '/' + context.security.authorization.id)"
    },
    {
      "pattern": "managed/*",
      "roles": "internal/role/openidm-authorized",
      "methods": "read",
      "actions": "*",
      "customAuthz": "ownIDP()"
    }
  ]
```

```
}
```

Grant Internal Authorization Roles Manually

Apart from the default roles that users get when they authenticate, you can grant internal authorization roles manually, over REST or through the Admin UI. This mechanism works in the same way as the granting of managed roles. For information about granting managed roles, see Grant Roles to a User in the *Object Modeling Guide*. To grant an internal role manually through the Admin UI:

1. Select Manage > User and click the user to whom you want to grant the role.
2. Select the Authorization Roles tab and click Add Authorization Roles.
3. Select Internal Role as the Type, click in the Authorization Roles field to select from the list of defined Internal Roles, then click Add.

To manually grant an internal role over REST, add a reference to the internal role to the user's `authzRoles` property. The following command adds the `openidm-admin` role to user bjensen (with ID `9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb`):

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--cacert ca-cert.pem \
--request PATCH \
--data '[
  {
    "operation": "add",
    "field": "/authzRoles/-",
    "value": {"_ref" : "internal/role/openidm-admin"}
  }
]' \
"https://localhost:8443/openidm/managed/user/9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb"
{
  "_id": "9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb",
  "_rev": "0000000050c62938",
  "mail": "bjensen@example.com",
  "givenName": "Barbara",
  "sn": "Jensen",
  "description": "Created By CSV",
  "userName": "bjensen",
  "telephoneNumber": "1234567",
  "accountStatus": "active",
  "memberOfOrgIDs": [],
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

You can also grant internal roles dynamically using conditional role grants in the *Object Modeling Guide*.

Note

Because internal roles are not managed objects, you cannot manipulate them in the same way as managed roles. Therefore you cannot add a user to an internal role, as you would to a managed role.

To add users directly to an internal role, add the users as values of the role's `authzMembers` property. For example:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--cacert ca-cert.pem \
--request POST \
--data '{"_ref":"managed/user/bjensen"}' \
"https://localhost:8443/openidm/internal/role/3042798d-37fd-49aa-bae3-52598d2c8dc4/authzMembers?
_action=create"
```

Secure Access to Workflows

The End User UI is integrated with the embedded Flowable workflow engine, enabling users to interact with workflows. Available workflows are displayed under the Processes item on the Dashboard. In order for a workflow to be displayed here, the workflow definition file must be present in the `openidm/workflow` directory.

A sample workflow integration with the End User UI is provided in `openidm/samples/provisioning-with-workflow`, and documented in "*Provision Users With Workflow*" in the *Samples Guide*. Follow the steps in that sample for an understanding of how the workflow integration works.

General access to workflow-related endpoints is based on the access rules defined in the `conf/access.json` file. The configuration defined in `conf/process-access.json` specifies who can invoke workflows. By default, all users with the role `openidm-authorized` or `openidm-admin` can invoke any available workflow. The default `process-access.json` file is as follows:

```
{
    "workflowAccess" : [
        {
            "propertiesCheck" : {
                "property" : "_id",
                "matches" : ".*",
                "requiresRole" : "internal/role/openidm-authorized"
            }
        },
        {
            "propertiesCheck" : {
                "property" : "_id",
                "matches" : ".*",
                "requiresRole" : "internal/role/openidm-admin"
            }
        }
    ]
}
```

`property`

Specifies the property used to identify the process definition. By default, process definitions are identified by their `_id`.

`matches`

A regular expression match is performed on the process definitions, according to the specified property. The default (`"matches" : ".*"`) implies that all process definition IDs match.

`requiresRole`

Specifies the authorization role that is required for users to have access to the matched process definition IDs. In the default file, users with the role `openidm-authorized` or `openidm-admin` have access.

To extend the process action definition file, identify the processes to which users should have access, and specify the qualifying authorization roles. For example, if you want to allow access to users with a role of `ldap`, add the following code block to the `process-access.json` file:

```
{
    "propertiesCheck" : {
        "property" : "_id",
        "matches" : ".*",
        "requiresRole" : "ldap"
    }
}
```

To configure multiple roles with access to the same workflow process, simply add additional `propertiesCheck` objects. The following example grants access to users with a role of doctor and nurse to the same workflows:

```
{
     "propertiesCheck" : {
          "property" : "_id",
          "matches" : ".*",
          "requiresRole" : "doctor"
     }
},
{
     "propertiesCheck" : {
          "property" : "_id",
          "matches" : ".*",
          "requiresRole" : "nurse"
     }
}
```

Administrative Users

The default IDM administrative user is `openidm-admin`. In a production environment, you might want to replace this user with a managed or internal user with the same roles, specifically the `openidm-admin` and `openidm-authorized` roles.

You can create either an internal or managed user with the same roles as the default `openidm-admin` user. To add these roles to an existing managed user, see "Grant Internal Authorization Roles Manually". The following procedure creates a new administrative internal user (`admin`):

1. Create an internal user:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--cacert ca-cert.pem \
--request PUT \
--data '{
  "password": "Passw0rd"
}' \
"https://localhost:8443/openidm/internal/user/admin"
{
  "_id": "admin",
  "_rev": "00000000210f6746"
}
```

2. Add a `STATIC_USER` authentication module to the authentication configuration:

+ *Using the Filesystem*

Edit the `conf/authentication.json` file, and add the following:

```
{
  "name" : "STATIC_USER",
  "properties" : {
    "queryOnResource" : "internal/user",
    "username" : "admin",
    "password" : "Passw0rd",
    "defaultUserRoles" : [
      "internal/role/openidm-authorized",
      "internal/role/openidm-admin"
    ]
  },
  "enabled" : true
}
```

+ *Using REST*

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--request PATCH \
--data '[
  {
    "operation": "add",
    "field": "/serverAuthContext/authModules/-",
    "value": {
      "name" : "STATIC_USER",
      "properties" : {
        "queryOnResource" : "internal/user",
        "username" : "admin",
        "password" : "Passw0rd",
        "defaultUserRoles" : [
          "internal/role/openidm-authorized",
          "internal/role/openidm-admin"
        ]
      },
      "enabled" : true
    }
  }
]' \
"https://localhost:8443/openidm/config/authentication"
{
  "_id": "authentication",
  "serverAuthContext": {
    ...
    "authModules": [
      ...
      {
        "name": "STATIC_USER",
        "properties": {
          "queryOnResource": "internal/user",
          "username": "admin",
          "password": "{encrypted password}",
          "defaultUserRoles": [
            "internal/role/openidm-authorized",
            "internal/role/openidm-admin"
          ]
        },
        "enabled": true
      },
      ...
    ]
  }
}
```

3. To verify the changes, perform a REST call or log in to the Admin UI as the new admin user. For example, query the list of internal users:

```
curl \
--header "X-OpenIDM-Username: admin" \
--header "X-OpenIDM-Password: Passw0rd" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--request GET \
"https://localhost:8443/openidm/internal/user?_queryFilter=true"
{
  "result": [
    {
      "_id": "admin",
      "_rev": "00000000f8e1665a"
    }
  ],
  ...
}
```

4. (Optional) *After* you have verified the new admin user, you can delete or disable the `openidm-admin` user:

   + *Delete 'openidm-admin' User*

   1. Delete the `openidm-admin` object:

      ```
      curl \
      --header "X-OpenIDM-Username: admin" \
      --header "X-OpenIDM-Password: Passw0rd" \
      --header "Accept-API-Version: resource=1.0" \
      --cacert ca-cert.pem \
      --request DELETE \
      "https://localhost:8443/openidm/internal/user/openidm-admin"
      {
        "_id": "openidm-admin",
        "_rev": "00000000210f6746"
      }
      ```

   2. Delete the authentication module for `"username" : "openidm-admin"`:

      + *Using the Filesystem*

      Edit the `conf/authentication.json` file, and delete:

```
{
  "name" : "STATIC_USER",
  "properties" : {
    "queryOnResource" : "internal/user",
    "username" : "openidm-admin",
    "password" : "&{openidm.admin.password}",
    "defaultUserRoles" : [
      "internal/role/openidm-authorized",
      "internal/role/openidm-admin"
    ]
  },
  "enabled" : true
}
```

+ *Using REST*

a. Get the current authentication configuration:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--request GET \
"https://localhost:8443/openidm/config/authentication"
{
  "_id": "authentication",
  "serverAuthContext": {
    ...
    "authModules": [
      ...
      {
        "name": "STATIC_USER",
        "properties": {
          "queryOnResource": "internal/user",
          "username": "openidm-admin",
          "password": "&{openidm.admin.password}",
          "defaultUserRoles": [
            "internal/role/openidm-authorized",
            "internal/role/openidm-admin"
          ]
        },
        "enabled": true
      },
      ...
    ]
  }
}
```

b. Remove the authentication module for `"username" : "openidm-admin"`, and replace the authentication configuration:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--cacert ca-cert.pem \
--request PUT \
--data '{
  "_id": "authentication",
  "serverAuthContext": {
    "sessionModule": {
      "name": "JWT_SESSION",
      "properties": {
        "maxTokenLifeMinutes": 120,
        "tokenIdleTimeMinutes": 30,
        "sessionOnly": true,
        "isHttpOnly": true,
        "enableDynamicRoles": false
      }
    },
    "authModules": [
      {
        "name": "STATIC_USER",
        "properties": {
          "queryOnResource": "internal/user",
          "username": "anonymous",
          "password": {
            "$crypto": {
              "type": "x-simple-encryption",
              "value": {
                "cipher": "AES/CBC/PKCS5Padding",
                "stableId": "openidm-sym-default",
                "salt": "xBlTp67ze4Ca5LTocXOpoA==",
                "data": "mdibV6UabU2M+M5MK7bjFQ==",
                "keySize": 16,
                "purpose": "idm.config.encryption",
                "iv": "36D2+FumKbaUsndNQ+/+5w==",
                "mac": "ZM8GMnh0n80QwtSH6QsNmA=="
              }
            }
          },
          "defaultUserRoles": [
            "internal/role/openidm-reg"
          ]
        },
        "enabled": true
      },
      {
        "name": "STATIC_USER",
        "properties": {
          "queryOnResource": "internal/user",
          "username": "admin",
          "password": "{encrypted password}",
          "defaultUserRoles": [
            "internal/role/openidm-authorized",
            "internal/role/openidm-admin"
          ]
        },
        "enabled": true
      },
      {
        "name": "MANAGED_USER",
        "properties": {
          "augmentSecurityContext": {
            "type": "text/javascript",
            "source": "require('auth/customAuthz').setProtectedAttributes(security)"
          },
          "queryId": "credential-query",
          "queryOnResource": "managed/user",
          "propertyMapping": {
            "authenticationId": "username",
            "userCredential": "password",
            "userRoles": "authzRoles"
          },
          "defaultUserRoles": [
            "internal/role/openidm-authorized"
          ]
        },
        "enabled": true
      },
      {
        "name": "SOCIAL_PROVIDERS",
        "properties": {
          "defaultUserRoles": [
            "internal/role/openidm-authorized"
          ],
          "augmentSecurityContext": {
            "type": "text/javascript",
            "globals": {},
            "file": "auth/populateAsManagedUserFromRelationship.js"
          },
          "propertyMapping": {
            "userRoles": "authzRoles"
          }
        },
        "enabled": true
      }
    ]
  }
}' \
"https://localhost:8443/openidm/config/authentication"
```

3. Prevent the `openidm-admin` user from being recreated on startup.

   Delete the following lines from the `internal/user` array in `conf/repo.init.json`:

```
{
    "id" : "openidm-admin",
    "password" : "&{openidm.admin.password}"
}
```

+ *Disable 'openidm-admin' User*

Change the `enabled` state of the authentication module for `"username" : "openidm-admin"`:

+ *Using the Filesystem*

Edit the `conf/authentication.json` file:

```
{
  "name" : "STATIC_USER",
  "properties" : {
    "queryOnResource" : "internal/user",
    "username" : "openidm-admin",
    "password" : "&{openidm.admin.password}",
    "defaultUserRoles" : [
      "internal/role/openidm-authorized",
      "internal/role/openidm-admin"
    ]
  },
  "enabled" : false
}
```

+ *Using REST*

1. Get the current authentication configuration:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--request GET \
"https://localhost:8443/openidm/config/authentication"
{
  "_id": "authentication",
  "serverAuthContext": {
    ...
    "authModules": [
      ...
      {
        "name": "STATIC_USER",
        "properties": {
          "queryOnResource": "internal/user",
          "username": "openidm-admin",
          "password": "&{openidm.admin.password}",
          "defaultUserRoles": [
            "internal/role/openidm-authorized",
            "internal/role/openidm-admin"
          ]
        },
        "enabled": true
      },
      ...
    ]
  }
}
```

2. Change the enabled state of the authentication module for `"username" : "openidm-admin"`, and replace the authentication configuration:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--cacert ca-cert.pem \
--request PUT \
--data '{
  "_id": "authentication",
  "serverAuthContext": {
    "sessionModule": {
      "name": "JWT_SESSION",
      "properties": {
        "maxTokenLifeMinutes": 120,
        "tokenIdleTimeMinutes": 30,
        "sessionOnly": true,
        "isHttpOnly": true,
        "enableDynamicRoles": false
      }
    },
    "authModules": [
      {
        "name": "STATIC_USER",
        "properties": {
          "queryOnResource": "internal/user",
          "username": "anonymous",
          "password": {
            "$crypto": {
              "type": "x-simple-encryption",
              "value": {
                "cipher": "AES/CBC/PKCS5Padding",
                "stableId": "openidm-sym-default",
                "salt": "xBlTp67ze4Ca5LTocXOpoA==",
                "data": "mdibV6UabU2M+M5MK7bjFQ==",
                "keySize": 16,
                "purpose": "idm.config.encryption",
                "iv": "36D2+FumKbaUsndNQ+/+5w==",
                "mac": "ZM8GMnh0n80QwtSH6QsNmA=="
              }
            }
          },
          "defaultUserRoles": [
            "internal/role/openidm-reg"
          ]
        },
        "enabled": true
      },
      {
        "name": "STATIC_USER",
        "properties": {
          "queryOnResource": "internal/user",
          "username": "openidm-admin",
          "password": "&{openidm.admin.password}",
          "defaultUserRoles": [
            "internal/role/openidm-authorized",
            "internal/role/openidm-admin"
          ]
        },
        "enabled": false
      },
      {
        "name": "MANAGED_USER",
        "properties": {
          "augmentSecurityContext": {
            "type": "text/javascript",
            "source": "require('auth/customAuthz').setProtectedAttributes(security)"
          },
          "queryId": "credential-query",
          "queryOnResource": "managed/user",
          "propertyMapping": {
            "authenticationId": "username",
            "userCredential": "password",
            "userRoles": "authzRoles"
          },
          "defaultUserRoles": [
            "internal/role/openidm-authorized"
          ]
        },
        "enabled": true
      },
      {
        "name": "SOCIAL_PROVIDERS",
        "properties": {
          "defaultUserRoles": [
            "internal/role/openidm-authorized"
          ],
          "augmentSecurityContext": {
            "type": "text/javascript",
            "globals": {},
            "file": "auth/populateAsManagedUserFromRelationship.js"
          },
          "propertyMapping": {
            "userRoles": "authzRoles"
          }
        },
        "enabled": true
      }
    ]
  }
}' \
"https://localhost:8443/openidm/config/authentication"
```

Chapter 3
Delegated Administration

Delegated administration lets you give fine-grained administrative access to specific users, based on a *privilege* mechanism.

How Privileges Restrict Administrative Access

Privileges enable you to grant administrative access to specific endpoints and objects, without needing to grant full administrative access to the server. For example, you might want to allow users with a help desk or support role to update the information of another user, without allowing them to delete user accounts or change the IDM system configuration.

You can use privileges to delegate specific administrative capabilities to non-administrative users, without exposing the Admin UI to those users. If a user has been granted a privilege that allows them to see a list of users and user information, for example, they can access this list directly through the End User UI.

Note

A delegated administrator does not have access to the same methods over REST as a regular administrator. IDM does not allow delegated administrator requests such as POST or DELETE. To add or remove relationships, use PATCH. For examples, see "Managed Roles" in the *Object Modeling Guide*.

The privilege mechanism requires dynamic role calculation, which is disabled by default. To enable it, set the `enableDynamicRoles` property to `true` in your `conf/authentication.json` file, or select Configure > Authentication > Session > Enable Dynamic Roles in the Admin UI. For more information about dynamic role calculation, see "Dynamic Role Calculation".

For more information on managing privileges over REST, see "Privileges" in the *REST API Reference*.

Determine Access Privileges

IDM determines what access a user has as follows:

1. IDM checks the `onRequest` script specified in `router.json`. By default, this script calls `router-authz.js`.
2. If access requirements are not satisfied, IDM then checks for any privileges associated with the user's roles.

`onResponse` and `onFailure` scripts are supported when using privileges. `onFailure` scripts are called only if both the `onRequest` script *and* the privilege filter fail. `onRequest`, `onResponse`, and `onFailure` scripts are not required for the privilege mechanism.

Create Privileges

Privileges are assigned to internal roles. A privilege specifies the following information:

- The service path to which users with that internal role have access.
- The methods and actions allowed on that service path.
- The specific attributes of the objects at that service path, to which access is allowed.

You can use a query filter within a privilege so that the privilege applies to only a subset of managed objects.

The `privileges` property is an array, and can contain multiple privileges. Each privilege can contain:

accessFlags

A list of attributes within a managed object that you wish to give access to. Each attribute has two fields:

- `attribute`—the name of the property you are granting access to.
- `readOnly` (boolean)—determines what level of access is allowed.

Attributes marked as `"readOnly": true` can be viewed but not edited. Attributes marked as `"readOnly": false` can be both viewed and edited. Attributes that are not listed in the `accessFlags` array cannot be viewed or edited.

Note

- Privileges are not automatically aware of changes to the managed object schema. If new properties are added, removed, or made mandatory, you must update any existing privileges to account for these changes. When a new property is added, it has a default permission level of `NONE` in existing privileges, including when the privilege is set to access all attributes.
- IDM applies policy validation when creating or updating a privilege, to ensure that all required properties are writable when the `CREATE` permission is assigned. This validation does not run when

schema changes are made, however, so you must verify that any existing privileges adhere to defined policies.

actions

A list of the specific actions allowed if the `ACTION` permission has been specified. Allowed actions must be explicitly listed.

description (optional)

A description of the privilege.

filter (optional)

This property lets you apply a static or dynamic query filter to the privilege, which can be used to limit the scope of what the privilege allows the user to access.

Static Filter Example

To allow a delegated administrator to access information only about users for the `stateProvince` Washington, include a static filter such as:

```
filter : "stateProvince eq \"Washington\""
```

Dynamic Filter Example

Dynamic filters insert values from the authenticated resource. To allow a delegated administrator to access information only about users in their own `stateProvince`, include a dynamic filter by wrapping the parameter in curly braces:

```
filter : "stateProvince eq \"{{stateProvince}}\""
```

Users with query filter privileges cannot edit the properties specified in the filter in ways that would cause the privilege to lose access to the object. For example, if a user with either of the preceding example privileges attempted to edit another user's `stateProvince` field to anything not matching the query filter, the request would return a `403 Forbidden` error.

Note

Fields must be *searchable* by IDM to be used in a privilege filter. Make sure that the field you are filtering on has `"searchable" : true` set in `repo.jdbc.json`. This is not necessary if you are using a DS or a PostgreSQL repository.

Privilege filters are another layer of filter *in addition to* any other query filters you create. This means any output must satisfy all filters to be included.

name

The name of the privilege being created.

path

The path to the service you want to allow members of this privilege to access. For example, `managed/user`.

permissions

A list of permissions this privilege allows for the given path. The following permissions are available:

- `VIEW`—allows reading and querying the path, such as viewing and querying managed users.
- `CREATE`—allows creation at the path, such as creating new managed users.
- `UPDATE`—allows updating or patching existing information, such as editing managed user details.
- `DELETE`—allows deletion, such as deleting users from `managed/user`.
- `ACTION`—allows users to perform actions at the given path, such as custom scripted actions.

Note

Actions that require additional filtering on the results of the action are not currently supported.

Adding Privileges Using the Admin UI

The easiest way to modify privileges is using the Admin UI.

1. From the navigation bar, click Manage > Role.
2. From the Roles page, click the Internal tab, and then click an existing role (or create a new role).
3. From the *Role Name* page, click the Privileges tab.

   IDM displays the current privileges for the role.
4. To add privileges, click Add Privileges.
   - In the Add a privilege window, enter information, as necessary, and click Add.

+ *Adding Privileges Using REST*

The following example creates a new `support` role with privileges that let members view, create, and update information about users, but not delete users:

```
curl \
--header "X-OpenIDM-UserName: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--header "Content-Type: application/json" \
--cacert ca-cert.pem \
--request PUT \
--data '{
  "name": "support",
  "description": "Support Role",
  "privileges": [ {
    "name": "support",
    "description": "Support access to user information.",
    "path": "managed/user",
    "permissions": [
      "VIEW", "UPDATE", "CREATE"
    ],
    "actions": [],
    "filter": null,
    "accessFlags": [
      {
        "attribute" : "userName",
        "readOnly" : false
      },
      {
        "attribute" : "mail",
        "readOnly" : false
      },
      {
        "attribute" : "givenName",
        "readOnly" : false
      },
      {
        "attribute" : "sn",
        "readOnly" : false
      },
      {
        "attribute" : "accountStatus",
        "readOnly" : true
      }
    ]
  } ]
}' \
"https://localhost:8443/openidm/internal/role/support"
{
  "_id": "support",
  "_rev": "00000000bfbac2ed",
  "name": "support",
  "description": "Support Role",
  "temporalConstraints": [],
  "condition": null,
  "privileges": [
    {
      "name": "support",
      "description": "Support access to user information.",
      "path": "managed/user",
      "permissions": [
        "VIEW",
        "UPDATE",
        "CREATE"
      ],
      "actions": [],
      "filter": null,
      "accessFlags": [
        {
          "attribute": "userName",
          "readOnly": false
        },
        {
          "attribute": "mail",
          "readOnly": false
        },
        {
          "attribute": "givenName",
          "readOnly": false
        },
        {
          "attribute": "sn",
          "readOnly": false
        },
        {
          "attribute": "accountStatus",
          "readOnly": true
        }
      ]
    }
  ]
}
```

Policies Related to Privileges

When creating privileges, IDM runs policies found in `policy.json` and `policy.js`, including the five policies used for validating privileges:

`valid-accessFlags-object`

Verifies that `accessFlag` objects are correctly formatted. Only two fields are permitted in an `accessFlag` object: `readOnly`, which must be a boolean; and `attribute`, which must be a string.

`valid-array-items`

Verifies that each item in an array contains the properties specified in `policy.json`, and that each of those properties satisfies any specific policies applied to it. By default, this is used to verify that each privilege contains `name`, `path`, `accessFlags`, `actions`, and `permissions` properties, and that the `filter` property is valid if included.

`valid-permissions`

Verifies that the permissions set on the privilege are all valid and can be achieved with the `accessFlags` that have been set. It checks:

- `CREATE` permissions must have write access to all properties required to create a new object.
- `CREATE` and `UPDATE` permissions must have write access to at least one property.
- `ACTION` permissions must include a list of allowed actions, with at least one action included.

- If any attributes have write access, then the privilege must also have either `CREATE` or `UPDATE` permission.
- All permissions listed must be valid types of permission: `VIEW`, `CREATE`, `UPDATE`, `ACTION`, or `DELETE`. Also, no permissions are repeated.

`valid-privilege-path`

Verifies that the `path` specified in the privilege is a valid object with a schema for IDM to reference. Only objects with a schema (such as `managed/user`) can have privileges applied.

`valid-query-filter`

Verifies that the query filter used to filter privileges is a valid query.

For more information about policies and creating custom policies, see "*Use Policies to Validate Data*" in the *Object Modeling Guide*.

Use Privileges to Create a Delegated Administrator

You can use the IDM REST API to create an `internal/role` with privileges that have object, array, and relationship type attribute access. You can then use that role as a delegated administrator to perform operations on those attributes.

Use the following example to create a delegated administrator:

Note

If you want to experiment with delegated administrators in Postman, download and import this Postman collection.

+ *Step 1. Create a Managed Role*

To ensure a role object exists when roles are requested, you must create a managed role.

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "If-None-Match: *" \
--request PUT \
--data '{
  "name": "testManagedRole",
  "description": "a managed role for test"
}' \
"http://localhost:8080/openidm/managed/role/testManagedRole"
{
  "_id": "testManagedRole",
  "_rev": "00000000e0945865",
  "name": "testManagedRole",
  "description": "a managed role for test"
}
```

+ *Step 2. Create a "Manager" User*

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "If-None-Match: *" \
--request PUT \
--data '{
  "userName": "psmith",
  "sn": "Smith",
  "givenName": "Patricia",
  "mail": "psmith@example.com",
  "telephoneNumber": "082082082",
  "password": "Passw0rd"
}' \
"http://localhost:8080/openidm/managed/user/psmith"
{
  "_id": "psmith",
  "_rev": "000000008fefe160",
  "userName": "psmith",
  "sn": "Smith",
  "givenName": "Patricia",
  "mail": "psmith@example.com",
  "telephoneNumber": "082082082",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

+ *Step 3. Create Additional Users*

In this step, you'll create two users with the following attributes:

- `preferences`

- `manager`
- `roles`

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "If-None-Match: *" \
--request PUT \
--data '{
  "userName": "scarter",
  "sn": "Carter",
  "givenName": "Steven",
  "mail": "scarter@example.com",
  "telephoneNumber": "082082082",
  "password": "Passw0rd",
  "preferences": {
    "updates": true,
    "marketing": false
  },
  "manager": {"_ref" : "managed/user/psmith"},
  "roles": [{"_ref" : "managed/role/testManagedRole"}]
}' \
"http://localhost:8080/openidm/managed/user/scarter"
{
  "_id": "scarter",
  "_rev": "00000000a8d501f8",
  "userName": "scarter",
  "sn": "Carter",
  "givenName": "Steven",
  "mail": "scarter@example.com",
  "telephoneNumber": "082082082",
  "preferences": {
    "updates": true,
    "marketing": false
  },
  "accountStatus": "active",
  "effectiveRoles": [
    {
      "_ref": "managed/role/testManagedRole"
    }
  ],
  "effectiveAssignments": []
}
```

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "If-None-Match: *" \
--request PUT \
--data '{
  "userName": "jdoe",
  "sn": "Doe",
  "givenName": "John",
  "mail": "jdoe@example.com",
  "telephoneNumber": "082082082",
  "password": "Passw0rd",
  "preferences": {
    "updates": true,
    "marketing": false
  },
  "manager": {"_ref" : "managed/user/psmith"},
  "roles": [{"_ref" : "managed/role/testManagedRole"}]
}' \
"http://localhost:8080/openidm/managed/user/jdoe"
{
  "_id": "jdoe",
  "_rev": "00000000b174fbd4",
  "userName": "jdoe",
  "sn": "Doe",
  "givenName": "John",
  "mail": "jdoe@example.com",
  "telephoneNumber": "082082082",
  "preferences": {
    "updates": true,
    "marketing": false
  },
  "accountStatus": "active",
  "effectiveRoles": [
    {
      "_ref": "managed/role/testManagedRole"
    }
  ],
  "effectiveAssignments": []
}
```

+ *Step 4. Create Another User*

You will delegate an internal/role with privileges to this user in the next step:

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "If-None-Match: *" \
--request PUT \
--data '{
  "userName": "bjensen",
  "sn": "Jensen",
  "givenName": "Barbara",
  "mail": "bjensen@example.com",
  "telephoneNumber": "082082082",
  "password": "Passw0rd"
}' \
"http://localhost:8080/openidm/managed/user/bjensen"
{
  "_id": "bjensen",
  "_rev": "0000000022fae330",
  "userName": "bjensen",
  "sn": "Jensen",
  "givenName": "Barbara",
  "mail": "bjensen@example.com",
  "telephoneNumber": "082082082",
  "accountStatus": "active",
  "effectiveRoles": [],
  "effectiveAssignments": []
}
```

+ *Step 5. Create an "internal/role"*

This role will have the following privileges:

- A `managed/user` privilege with accessFlags attributes that are of types: "String", "boolean", and "number"; but also for:
  - An object type that is not a relationship (`preferences`).
  - An object type that is a relationship (`manager`).
  - Array types that are relationships (`roles`, `authzRoles`, `reports`).
- A `managed/role` privilege for viewing details of the "roles" property of a managed user.
- An `internal/role` privilege for viewing the details of the "authzRoles" property of a managed user.

Note

You can populate the privilege `filter` field to apply a finer level of permissions for what a delegated administrator can see or do with certain objects. The `filter` field is omitted in this example to allow all.

For properties that are *not* relationships, such as `preferences`, you can't specify finer-grained permissions. For example, you can't set permissions on `preferences/marketing`.

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "If-None-Match: *" \
--request PUT \
--data '{
  "name": "internal_role_with_object_array_and_relationship_privileges",
  "description": "an internal role that has privileges for object & array types and relationships",
  "privileges": [
    {
      "name": "managed_user_privilege",
      "path": "managed/user",
      "permissions": [
        "VIEW",
        "CREATE",
        "UPDATE",
        "DELETE"
      ],
      "actions": [],
      "accessFlags": [
        {
          "attribute": "userName",
          "readOnly": false
        },
        {
          "attribute": "password",
          "readOnly": false
        },
        {
          "attribute": "givenName",
          "readOnly": false
        },
        {
          "attribute": "sn",
          "readOnly": false
        },
        {
          "attribute": "mail",
          "readOnly": false
        },
        {
          "attribute": "description",
          "readOnly": false
        },
        {
          "attribute": "accountStatus",
          "readOnly": false
        },
        {
          "attribute": "telephoneNumber",
          "readOnly": false
        },
        {
          "attribute": "postalAddress",
          "readOnly": false
        },
        {
          "attribute": "city",
          "readOnly": false
        },
        {
          "attribute": "postalCode",
          "readOnly": false
        },
        {
          "attribute": "country",
          "readOnly": false
        },
        {
          "attribute": "stateProvince",
          "readOnly": false
        },
        {
          "attribute": "preferences",
          "readOnly": false
        },
        {
          "attribute": "roles",
          "readOnly": false
        },
        {
          "attribute": "manager",
          "readOnly": false
        },
        {
          "attribute": "authzRoles",
          "readOnly": false
        },
        {
          "attribute": "reports",
          "readOnly": false
        }
      ]
    },
    {
      "name": "managed_role_privilege",
      "path": "managed/role",
      "permissions": [
        "VIEW"
      ],
      "actions": [],
      "accessFlags": [
        {
          "attribute": "name",
          "readOnly": true
        },
        {
          "attribute": "description",
          "readOnly": true
        }
      ]
    },
    {
      "name": "internal_role_privilege",
      "path": "internal/role",
      "permissions": [
        "VIEW"
      ],
      "actions": [],
      "accessFlags": [
        {
          "attribute": "name",
          "readOnly": true
        },
        {
          "attribute": "description",
          "readOnly": true
        },
        {
          "attribute": "authzMembers",
          "readOnly": true
        }
      ]
    }
  ]
}' \
"http://localhost:8080/openidm/internal/role/testInternalRole"
{
  "_id": "testInternalRole",
  "_rev": "0000000079775d19",
  "name": "internal_role_with_object_array_and_relationship_privileges",
  "description": "an internal role that has privileges for object & array types and relationships",
  "temporalConstraints": null,
  "condition": null,
  "privileges": [
    {
      "name": "managed_user_privilege",
      "path": "managed/user",
      "permissions": [
        "VIEW",
        "CREATE",
        "UPDATE",
        "DELETE"
      ],
      "actions": [],
      "accessFlags": [
        {
          "attribute": "userName",
          "readOnly": false
        },
        {
          "attribute": "password",
          "readOnly": false
        },
        {
          "attribute": "givenName",
          "readOnly": false
        },
        {
          "attribute": "sn",
          "readOnly": false
        },
        {
          "attribute": "mail",
          "readOnly": false
        },
        {
          "attribute": "description",
          "readOnly": false
        },
        {
          "attribute": "accountStatus",
          "readOnly": false
        },
        {
          "attribute": "telephoneNumber",
          "readOnly": false
        },
        {
          "attribute": "postalAddress",
          "readOnly": false
        },
        {
          "attribute": "city",
          "readOnly": false
        },
        {
          "attribute": "postalCode",
          "readOnly": false
        },
        {
          "attribute": "country",
          "readOnly": false
        },
        {
          "attribute": "stateProvince",
          "readOnly": false
        },
        {
          "attribute": "preferences",
          "readOnly": false
        },
        {
          "attribute": "roles",
          "readOnly": false
        },
        {
          "attribute": "manager",
          "readOnly": false
        },
        {
          "attribute": "authzRoles",
          "readOnly": false
        },
        {
          "attribute": "reports",
          "readOnly": false
        }
      ]
    },
    {
      "name": "managed_role_privilege",
      "path": "managed/role",
      "permissions": [
        "VIEW"
      ],
      "actions": [],
      "accessFlags": [
        {
          "attribute": "name",
          "readOnly": true
        },
        {
          "attribute": "description",
          "readOnly": true
        }
      ]
    },
    {
      "name": "internal_role_privilege",
      "path": "internal/role",
      "permissions": [
        "VIEW"
      ],
      "actions": [],
      "accessFlags": [
        {
          "attribute": "name",
          "readOnly": true
        },
        {
          "attribute": "description",
          "readOnly": true
        },
        {
          "attribute": "authzMembers",
          "readOnly": true
        }
      ]
    }
  ]
}
```

+ *Step 6. Create the Relationship Between User and "internal/role"*

In this step, assign the internal/role from step 5 to the user created in step 4 by creating a relationship:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Content-Type: application/json" \
--request POST \
--data '{
  "_ref": "managed/user/bjensen",
  "_refProperties": {}
}' \
"http://localhost:8080/openidm/internal/role/testInternalRole/authzMembers?_action=create"
{
  "_id": "732d3ab1-4319-41de-801b-80f4f4c97ef2",
  "_rev": "00000000e6dd99e0",
  "_ref": "managed/user/bjensen",
  "_refResourceCollection": "managed/user",
  "_refResourceId": "bjensen",
  "_refProperties": {
    "_id": "732d3ab1-4319-41de-801b-80f4f4c97ef2",
    "_rev": "00000000e6dd99e0"
  }
}
```

+ *Step 7. Perform Operations as a Delegated Administrator*

You can now perform operations as a delegated administrator, such as:

+ *Query All Users*

The query results display all users' properties that are allowed by the privileges:

```
curl \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--request GET \
"http://localhost:8080/openidm/managed/user?_queryFilter=true&_pageSize=100&_fields=*,*_ref/*"
{
  "result": [
    {
      "_id": "psmith",
      "_rev": "000000008fefe160",
      "userName": "psmith",
      "sn": "Smith",
      "givenName": "Patricia",
      "mail": "psmith@example.com",
      "telephoneNumber": "082082082",
      "accountStatus": "active",
      "reports": [
        {
          "_ref": "managed/user/scarter",
          "_refResourceCollection": "managed/user",
          "_refResourceId": "scarter",
          "_refProperties": {
            "_id": "c4e296ba-b0bb-44b8-a3e5-8d7c1656cef2",
            "_rev": "00000000e6f694a4"
          },
          "userName": "scarter",
          "sn": "Carter",
          "givenName": "Steven",
          "mail": "scarter@example.com",
          "telephoneNumber": "082082082",
          "preferences": {
            "updates": true,
            "marketing": false
          },
          "accountStatus": "active",
          "_rev": "00000000a8d501f8",
          "_id": "scarter"
        },
        {
          "_ref": "managed/user/jdoe",
          "_refResourceCollection": "managed/user",
          "_refResourceId": "jdoe",
          "_refProperties": {
            "_id": "1e3dd17d-a540-4652-984a-60bd60e546d5",
            "_rev": "0000000066ee928d"
          },
          "userName": "jdoe",
          "sn": "Doe",
          "givenName": "John",
          "mail": "jdoe@example.com",
          "telephoneNumber": "082082082",
          "preferences": {
            "updates": true,
            "marketing": false
          },
          "accountStatus": "active",
          "_rev": "00000000b174fbd4",
          "_id": "jdoe"
        }
      ],
      "manager": null,
      "roles": [],
      "authzRoles": [],
      "_notifications": [],
      "_meta": {
        "_ref": "internal/usermeta/0c15f08b-cf2e-4408-b302-4f46a40bf943",
        "_refResourceCollection": "internal/usermeta",
        "_refResourceId": "0c15f08b-cf2e-4408-b302-4f46a40bf943",
        "_refProperties": {
          "_id": "da3e2429-ae6f-4ea6-b5db-d3112f7c9d6a",
          "_rev": "00000000fd019b55"
        },
        "_rev": "000000003d8f5ca1",
        "_id": "0c15f08b-cf2e-4408-b302-4f46a40bf943"
      }
    },
    {
      "_id": "scarter",
      "_rev": "00000000a8d501f8",
      "userName": "scarter",
      "sn": "Carter",
      "givenName": "Steven",
      "mail": "scarter@example.com",
      "telephoneNumber": "082082082",
      "preferences": {
        "updates": true,
        "marketing": false
      },
      "accountStatus": "active",
      "reports": [],
      "manager": {
        "_ref": "managed/user/psmith",
        "_refResourceCollection": "managed/user",
        "_refResourceId": "psmith",
        "_refProperties": {
          "_id": "c4e296ba-b0bb-44b8-a3e5-8d7c1656cef2",
          "_rev": "00000000e6f694a4"
        },
        "userName": "psmith",
        "sn": "Smith",
        "givenName": "Patricia",
        "mail": "psmith@example.com",
        "telephoneNumber": "082082082",
        "accountStatus": "active",
        "_rev": "000000008fefe160",
        "_id": "psmith"
      },
      "roles": [
        {
          "_ref": "managed/role/testManagedRole",
          "_refResourceCollection": "managed/role",
          "_refResourceId": "testManagedRole",
          "_refProperties": {
            "_id": "352d7864-3143-4c56-ae11-8f75c96e980a",
            "_rev": "00000000b9ef9689"
          },
          "name": "testManagedRole",
          "description": "a managed role for test",
          "_rev": "00000000e0945865",
          "_id": "testManagedRole"
        }
      ],
      "authzRoles": [],
      "_notifications": [],
      "_meta": {
        "_ref": "internal/usermeta/6677aad2-def9-4507-9ea0-edd95da8da43",
        "_refResourceCollection": "internal/usermeta",
        "_refResourceId": "6677aad2-def9-4507-9ea0-edd95da8da43",
        "_refProperties": {
          "_id": "cc32ab82-084a-455c-bf97-3f2f2a71f848",
          "_rev": "00000000f4819bb6"
        },
        "_rev": "0000000090ae5c88",
        "_id": "6677aad2-def9-4507-9ea0-edd95da8da43"
      }
    },
    {
      "_id": "jdoe",
      "_rev": "00000000b174fbd4",
      "userName": "jdoe",
      "sn": "Doe",
      "givenName": "John",
      "mail": "jdoe@example.com",
      "telephoneNumber": "082082082",
      "preferences": {
        "updates": true,
        "marketing": false
      },
      "accountStatus": "active",
      "reports": [],
      "manager": {
        "_ref": "managed/user/psmith",
        "_refResourceCollection": "managed/user",
        "_refResourceId": "psmith",
        "_refProperties": {
          "_id": "1e3dd17d-a540-4652-984a-60bd60e546d5",
          "_rev": "0000000066ee928d"
        },
        "userName": "psmith",
        "sn": "Smith",
        "givenName": "Patricia",
        "mail": "psmith@example.com",
        "telephoneNumber": "082082082",
        "accountStatus": "active",
        "_rev": "000000008fefe160",
        "_id": "psmith"
      },
      "roles": [
        {
          "_ref": "managed/role/testManagedRole",
          "_refResourceCollection": "managed/role",
          "_refResourceId": "testManagedRole",
          "_refProperties": {
            "_id": "a3f6be90-3009-4e87-af46-257306617bd9",
            "_rev": "00000000b8f69498"
          },
          "name": "testManagedRole",
          "description": "a managed role for test",
          "_rev": "00000000e0945865",
          "_id": "testManagedRole"
        }
      ],
      "authzRoles": [],
      "_notifications": [],
      "_meta": {
        "_ref": "internal/usermeta/5b844d7e-c200-4b67-9fad-fa346740c79d",
        "_refResourceCollection": "internal/usermeta",
        "_refResourceId": "5b844d7e-c200-4b67-9fad-fa346740c79d",
        "_refProperties": {
          "_id": "42aa7cf0-6726-461b-92f9-1a22dab0b3c3",
          "_rev": "000000003aa1993e"
        },
        "_rev": "000000003e4f5bba",
        "_id": "5b844d7e-c200-4b67-9fad-fa346740c79d"
      }
    },
    {
      "_id": "bjensen",
      "_rev": "0000000022fae330",
      "userName": "bjensen",
      "sn": "Jensen",
      "givenName": "Barbara",
      "mail": "bjensen@example.com",
      "telephoneNumber": "082082082",
      "accountStatus": "active",
      "reports": [],
      "manager": null,
      "roles": [],
      "authzRoles": [
        {
          "_ref": "internal/role/testInternalRole",
          "_refResourceCollection": "internal/role",
          "_refResourceId": "testInternalRole",
          "_refProperties": {
            "_id": "732d3ab1-4319-41de-801b-80f4f4c97ef2",
            "_rev": "00000000e6dd99e0"
          },
          "_id": "testInternalRole",
          "name": "internal_role_with_object_array_and_relationship_privileges",
          "description": "an internal role that has privileges for object & array types and
 relationships",
          "_rev": "0000000079775d19"
        }
      ],
      "_notifications": [],
      "_meta": {
        "_ref": "internal/usermeta/0fbeb220-5e95-42b4-9bdd-0464e23194d4",
        "_refResourceCollection": "internal/usermeta",
        "_refResourceId": "0fbeb220-5e95-42b4-9bdd-0464e23194d4",
        "_refProperties": {
          "_id": "cbdb3794-1629-424d-8d7a-9e9b0c93287f",
          "_rev": "000000002b5199f1"
        },
        "_rev": "000000002fbc5b92",
        "_id": "0fbeb220-5e95-42b4-9bdd-0464e23194d4"
      }
    }
  ],
  "resultCount": 4,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```

+ *Read a Specified User's Preferences Object*

```
curl \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--request GET \
"http://localhost:8080/openidm/managed/user/jdoe?_fields=preferences"
{
  "_id": "jdoe",
  "_rev": "00000000b174fbd4",
  "preferences": {
    "updates": true,
    "marketing": false
  }
}
```

+ *Query a Specified User's Roles*

```
curl \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--request GET \
"http://localhost:8080/openidm/managed/user/scarter/roles?_queryFilter=true&_fields=*"
{
  "result": [
    {
      "_id": "352d7864-3143-4c56-ae11-8f75c96e980a",
      "_rev": "00000000b9ef9689",
      "_refResourceCollection": "managed/role",
      "_refResourceId": "testManagedRole",
      "_refResourceRev": "00000000e0945865",
      "name": "testManagedRole",
      "description": "a managed role for test",
      "_ref": "managed/role/testManagedRole",
      "_refProperties": {
        "_id": "352d7864-3143-4c56-ae11-8f75c96e980a",
        "_rev": "00000000b9ef9689"
      }
    }
  ],
  "resultCount": 1,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```

+ *Read a Specified User's Manager*

```
curl \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--request GET \
"http://localhost:8080/openidm/managed/user/scarter/manager?_fields=*"
{
  "_id": "c4e296ba-b0bb-44b8-a3e5-8d7c1656cef2",
  "_rev": "00000000e6f694a4",
  "_refResourceCollection": "managed/user",
  "_refResourceId": "psmith",
  "_refResourceRev": "000000008fefe160",
  "userName": "psmith",
  "sn": "Smith",
  "givenName": "Patricia",
  "mail": "psmith@example.com",
  "telephoneNumber": "082082082",
  "accountStatus": "active",
  "_ref": "managed/user/psmith",
  "_refProperties": {
    "_id": "c4e296ba-b0bb-44b8-a3e5-8d7c1656cef2",
    "_rev": "00000000e6f694a4"
  }
}
```

+ *Update a Specified User's Reports*

```
curl \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[ {
   "operation" : "replace",
   "field" : "reports",
   "value" : [{"_ref" : "managed/user/scarter"}]
} ]' \
"http://localhost:8080/openidm/managed/user/psmith"
{
  "_id": "psmith",
  "_rev": "000000008fefe160",
  "userName": "psmith",
  "sn": "Smith",
  "givenName": "Patricia",
  "mail": "psmith@example.com",
  "telephoneNumber": "082082082",
  "accountStatus": "active"
}
```

+ *Assign a Specified User's Manager*

```
curl \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[
  {
    "operation": "add",
    "field": "manager",
    "value": {"_ref" : "managed/user/psmith"}
  }
]' \
http://localhost:8080/openidm/managed/user/jdoe
{
  "_id": "jdoe",
  "_rev": "00000000b174fbd4",
  "userName": "jdoe",
  "sn": "Doe",
  "givenName": "John",
  "mail": "jdoe@example.com",
  "telephoneNumber": "082082082",
  "preferences": {
    "updates": true,
    "marketing": false
  },
  "accountStatus": "active"
}
```

+ *Remove a Specified User's Manager*

```
curl \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[
  {
     "operation": "remove",
     "field": "manager"
  }
]' \
http://localhost:8080/openidm/managed/user/jdoe
{
  "_id": "jdoe",
  "_rev": "00000000b174fbd4",
  "userName": "jdoe",
  "sn": "Doe",
  "givenName": "John",
  "mail": "jdoe@example.com",
  "telephoneNumber": "082082082",
  "preferences": {
    "updates": true,
    "marketing": false
  },
  "accountStatus": "active"
}
```

+ *Update a Specified User's Manager*

```
curl \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--header "Content-Type: application/json" \
--request PATCH \
--data '[
  {
    "operation": "replace",
    "field": "manager",
    "value": {"_ref" : "managed/user/jdoe"}
  }
]' \
"http://localhost:8080/openidm/managed/user/scarter"
{
  "_id": "scarter",
  "_rev": "00000000a8d501f8",
  "userName": "scarter",
  "sn": "Carter",
  "givenName": "Steven",
  "mail": "scarter@example.com",
  "telephoneNumber": "082082082",
  "preferences": {
    "updates": true,
    "marketing": false
  },
  "accountStatus": "active"
}
```

+ *Delete a Specified User*

```
curl \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--header "Content-Type: application/json" \
--request DELETE \
"http://localhost:8080/openidm/managed/user/psmith"
{
  "_id": "psmith",
  "_rev": "000000008fefe160",
  "userName": "psmith",
  "sn": "Smith",
  "givenName": "Patricia",
  "mail": "psmith@example.com",
  "telephoneNumber": "082082082",
  "accountStatus": "active"
}
```

+ *Create a User*

- Using POST:

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--request POST \
--data '{
  "userName": "psmith",
  "sn": "Smith",
  "givenName": "Patricia",
  "mail": "psmith@example.com",
  "telephoneNumber": "082082082",
  "password": "Passw0rd"
}' \
"http://localhost:8080/openidm/managed/user"
{
  "_id": "e5f6a856-9f3c-49fd-904c-c5f87004b682",
  "_rev": "000000004bbde938",
  "userName": "psmith",
  "sn": "Smith",
  "givenName": "Patricia",
  "mail": "psmith@example.com",
  "telephoneNumber": "082082082",
  "accountStatus": "active"
}
```

- Using PUT:

```
curl \
--header "Content-Type: application/json" \
--header "X-OpenIDM-Username: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--header "If-None-Match: *" \
--request PUT \
--data '{
  "userName": "psmith",
  "sn": "Smith",
  "givenName": "Patricia",
  "mail": "psmith@example.com",
  "telephoneNumber": "082082082",
  "password": "Passw0rd"
}' \
"http://localhost:8080/openidm/managed/user/psmith"
{
  "_id": "psmith",
  "_rev": "00000000658fe17a",
  "userName": "psmith",
  "sn": "Smith",
  "givenName": "Patricia",
  "mail": "psmith@example.com",
  "telephoneNumber": "082082082",
  "accountStatus": "active"
}
```

Note

For more examples, including working with filters, see the Postman collection.

Note

All patches are done with a PATCH request. Delegated administrator operations do not currently support using POST actions for patch requests (POST `_action=patch` will not work).

Get Privileges on a Resource

To determine which privileges a user has on a service, you can query the privilege endpoint for a given resource path or object, based on the user you are currently logged in as. For example, if bjensen is a member of the support role mentioned in the previous example, checking their privileges for the `managed/user` resource would look like this:

```
curl \
--header "X-OpenIDM-UserName: bjensen" \
--header "X-OpenIDM-Password: Passw0rd" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--request GET \
"https://localhost:8443/openidm/privilege/managed/user"
```

```
{
  "VIEW": {
    "allowed": true,
    "properties": [
      "userName",
      "givenName",
      "sn",
      "mail",
      "accountStatus"
    ]
  },
  "CREATE": {
    "allowed": true,
    "properties": [
      "userName",
      "givenName",
      "sn",
      "mail"
    ]
  },
  "UPDATE": {
    "allowed": true,
    "properties": [
      "userName",
      "givenName",
      "sn",
      "mail"
    ]
  },
  "DELETE": {
    "allowed": false
  },
  "ACTION": {
    "allowed": false,
    "actions": []
  }
}
```

In the above example, `accountStatus` is listed as a property for `VIEW`, but not for `CREATE` or `UPDATE`, because the privilege sets this property to be read only. Since both `CREATE` and `UPDATE` need the ability to write to a property, setting `readOnly` to false applies to both permissions. If you need more granular control, split these permissions into two privileges.

In addition to checking privileges for a resource, it is also possible to check privileges for specific objects within a resource, such as `managed/user/scarter`.

Appendix A. Authentication and Session Module Configuration

This appendix includes configuration details for the authentication modules described in "Authentication and Session Modules".

Authentication modules, as configured in the `authentication.json` file, include a number of properties.

Session Module

Authentication Property	Property as Listed in the Admin UI	Description
`keyAlias`	(not shown)	Used by the Jetty Web server to service SSL requests.
`maxTokenLifeMinutes`	Max Token Life (in seconds)	Maximum time before a session is cancelled. Note the different units for the property and the UI.
`tokenIdleTimeMinutes`	Token Idle Time (in seconds)	Maximum time before an idle session is cancelled. Note the different units for the property and the UI.
`sessionOnly`	Session Only	Whether the session continues after browser restarts.

Static User Module

Authentication Property	Property as Listed in the Admin UI	Description
`enabled`	Module Enabled	Does IDM use the module?
`queryOnResource`	Query on Resource	Endpoint hard coded to user `anonymous`
`username`	Static User Name	Default for the static user, `anonymous`
`password`	Static User Password	Default for the static user, `anonymous`
`defaultUserRoles`	Static User Role	Normally set to `openidm-reg` for self-registration

The following table applies to several authentication modules:

- Managed User
- Internal User
- Client Cert
- Passthrough
- IWA

The IWA module includes several Kerberos-related properties listed at the end of the table.

Common Module Properties

Authentication Property	Property as Listed in the Admin UI	Description
`enabled`	Module Enabled	Does IDM use the module?
`queryOnResource`	Query on Resource	Endpoint to query
`queryId`	Use Query ID	A defined `queryId` searches against the `queryOnResource` endpoint. An undefined `queryId` against `queryOnResource` with `action=reauthenticate`
`defaultUserRoles`	Default User Roles	Normally blank for managed users
`authenticationId`	Authentication ID	Defines how account credentials are derived from a `queryOnResource` endpoint
`userCredential`	User Credential	Defines how account credentials are derived from a `queryOnResource` endpoint; if required, typically `password` or `userPassword`
`userRoles`	User Roles	Defines how account roles are derived from a `queryOnResource` endpoint

Authentication Property	Property as Listed in the Admin UI	Description
`groupMembership`	Group Membership	Provides more information for calculated roles
`groupRoleMapping`	Group Role Mapping	Provides more information for calculated roles
`groupComparisonMethod`	Group Comparison Method	Provides more information for calculated roles
`managedUserLink`	Managed User Link	For pass-through authentication, this property specifies the mapping from the system resource to the IDM managed user. For example, if the user authenticates using their account in an LDAP directory, the `managedUserLink` might be `systemLdapAccounts_managedUser`
`augmentSecurityContext`	Augment Security Context	Includes a script that is executed only after a successful authentication request. For more information on this property, see "Authenticate as a Different User".
`servicePrincipal`	Kerberos Service Principal	(IWA only) For more information, see "IWA"
`keytabFileName`	Keytab File Name	(IWA only) For more information, see "IWA"
`kerberosRealm`	Kerberos Realm	(IWA only) For more information, see "IWA"
`kerberosServerName`	Kerberos Server Name	(IWA only) For more information, see "IWA"

IDM Glossary

correlation query	A correlation query specifies an expression that matches existing entries in a source repository to one or more entries in a target repository. A correlation query might be built with a script, but it is not the same as a correlation script. For more information, see "*Correlating Source Objects With Existing Target Objects*" in the *Synchronization Guide*.
correlation script	A correlation script matches existing entries in a source repository, and returns the IDs of one or more matching entries on a target repository. While it skips the intermediate step associated with a `correlation query`, a correlation script can be relatively complex, based on the operations of the script.
entitlement	An entitlement is a collection of attributes that can be added to a user entry via roles. As such, it is a specialized type of `assignment`. A user or device with an entitlement gets access rights to specified resources. An entitlement is a property of a managed object.
JCE	Java Cryptographic Extension, which is part of the Java Cryptography Architecture, provides a framework for encryption, key generation, and digital signatures.
JSON	JavaScript Object Notation, a lightweight data interchange format based on a subset of JavaScript syntax. For more information, see the JSON site.
JSON Pointer	A JSON Pointer defines a string syntax for identifying a specific value within a JSON document. For information about JSON Pointer syntax, see the JSON Pointer RFC.

JWT	JSON Web Token. As noted in the JSON Web Token draft IETF Memo, "JSON Web Token (JWT) is a compact URL-safe means of representing claims to be transferred between two parties." For IDM, the JWT is associated with the `JWT_SESSION` authentication module.
managed object	An object that represents the identity-related data managed by IDM. Managed objects are configurable, JSON-based data structures that IDM stores in its pluggable repository. The default configuration of a managed object is that of a user, but you can define any kind of managed object, for example, groups or roles.
mapping	A policy that is defined between a source object and a target object during reconciliation or synchronization. A mapping can also define a trigger for validation, customization, filtering, and transformation of source and target objects.
OSGi	A module system and service platform for the Java programming language that implements a complete and dynamic component model. For more information, see What is OSGi? Currently, only the Apache Felix container is supported.
reconciliation	During reconciliation, comparisons are made between managed objects and objects on source or target systems. Reconciliation can result in one or more specified actions, including, but not limited to, synchronization.
resource	An external system, database, directory server, or other source of identity data to be managed and audited by the identity management system.
REST	Representational State Transfer. A software architecture style for exposing resources, using the technologies and protocols of the World Wide Web. REST describes how distributed data objects, or resources, can be defined and addressed.
role	IDM distinguishes between two distinct role types - provisioning roles and authorization roles. For more information, see "Managed Roles" in the *Object Modeling Guide*.
source object	In the context of reconciliation, a source object is a data object on the source system, that IDM scans before attempting to find a corresponding object on the target system. Depending on the defined mapping, IDM then adjusts the object on the target system (target object).
synchronization	The synchronization process creates, updates, or deletes objects on a target system, based on the defined mappings from the source system. Synchronization can be scheduled or on demand.

system object	A pluggable representation of an object on an external system. For example, a user entry that is stored in an external LDAP directory is represented as a system object in IDM for the period during which IDM requires access to that entry. System objects follow the same RESTful resource-based design principles as managed objects.
target object	In the context of reconciliation, a target object is a data object on the target system, that IDM scans after locating its corresponding object on the source system. Depending on the defined mapping, IDM then adjusts the target object to match the corresponding source object.