



Security Guide

/ ForgeRock Identity Management 7

Latest update: 7.0.4

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

Copyright © 2016-2021 ForgeRock AS.

Abstract

Guide to securing ForgeRock® Identity Management deployments.



This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License.

To view a copy of this license, visit <https://creativecommons.org/licenses/by-nc-nd/3.0/> or send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.

ForgeRock® and ForgeRock Identity Platform™ are trademarks of ForgeRock Inc. or its subsidiaries in the U.S. and in other countries. Trademarks are the property of their respective owners.

UNLESS OTHERWISE MUTUALLY AGREED BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

DejaVu Fonts

Bitstream Vera Fonts Copyright

Copyright (c) 2003 by Bitstream, Inc. All Rights Reserved. Bitstream Vera is a trademark of Bitstream, Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of the fonts accompanying this license ("Fonts") and associated documentation files (the "Font Software"), to reproduce and distribute the Font Software, including without limitation the rights to use, copy, merge, publish, distribute, and/or sell copies of the Font Software, and to permit persons to whom the Font Software is furnished to do so, subject to the following conditions:

The above copyright and trademark notices and this permission notice shall be included in all copies of one or more of the Font Software typefaces.

The Font Software may be modified, altered, or added to, and in particular the designs of glyphs or characters in the Fonts may be modified and additional glyphs or characters may be added to the Fonts, only if the fonts are renamed to names not containing either the words "Bitstream" or the word "Vera".

This License becomes null and void to the extent applicable to Fonts or Font Software that has been modified and is distributed under the "Bitstream Vera" names.

The Font Software may be sold as part of a larger software package but no copy of one or more of the Font Software typefaces may be sold by itself.

THE FONT SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK, OR OTHER RIGHT. IN NO EVENT SHALL BITSTREAM OR THE GNOME FOUNDATION BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE FONT SOFTWARE.

Except as contained in this notice, the names of Gnome, the Gnome Foundation, and Bitstream Inc., shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Font Software without prior written authorization from the Gnome Foundation or Bitstream Inc., respectively. For further information, contact: fonts at gnome dot org.

Arev Fonts Copyright

Copyright (c) 2006 by Tavmjong Bah. All Rights Reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of the fonts accompanying this license ("Fonts") and associated documentation files (the "Font Software"), to reproduce and distribute the modifications to the Bitstream Vera Font Software, including without limitation the rights to use, copy, merge, publish, distribute, and/or sell copies of the Font Software, and to permit persons to whom the Font Software is furnished to do so, subject to the following conditions:

The above copyright and trademark notices and this permission notice shall be included in all copies of one or more of the Font Software typefaces.

The Font Software may be modified, altered, or added to, and in particular the designs of glyphs or characters in the Fonts may be modified and additional glyphs or characters may be added to the Fonts, only if the fonts are renamed to names not containing either the words "Tavmjong Bah" or the word "Arev".

This License becomes null and void to the extent applicable to Fonts or Font Software that has been modified and is distributed under the "Tavmjong Bah Arev" names.

The Font Software may be sold as part of a larger software package but no copy of one or more of the Font Software typefaces may be sold by itself.

THE FONT SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF COPYRIGHT, PATENT, TRADEMARK, OR OTHER RIGHT. IN NO EVENT SHALL TAVMJONG BAH BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, INCLUDING ANY GENERAL, SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF THE USE OR INABILITY TO USE THE FONT SOFTWARE OR FROM OTHER DEALINGS IN THE FONT SOFTWARE.

Except as contained in this notice, the name of Tavmjong Bah shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Font Software without prior written authorization from Tavmjong Bah. For further information, contact: tavmjong @ free . fr.

FontAwesome Copyright

Copyright (c) 2017 by Dave Gandy, <https://fontawesome.com/>.

This Font Software is licensed under the SIL Open Font License, Version 1.1. See <https://opensource.org/licenses/OFL-1.1>.

Table of Contents







Overview	v
1. Secret Stores, Certificates and Keys	1
Configuring Secret Stores	1
Working With the Default Keystore	4
Using CA-Signed Certificates	7
Deleting Certificates	9
Removing Unused CA Certificates	9
Changing and Rotating Encryption Keys	10
Configuring IDM For a Hardware Security Module (HSM) Device	19
2. Secure Authentication	26
IDM and HTTP Basic Authentication	26
Secure Password Changes	27
Character Encoding in Authentication Headers	28
Authenticate Users	29
Authentication and Session Modules	37
Authenticating as a Different User	60
Authentication and Roles	62
3. Protect REST Endpoints With Authorization and Access Control	65
Authorization and Roles	65
Privileges and Delegation to Restrict Administrative Access	93
Administrative Users	121
Hide Unused REST Endpoints	130
4. Secure Passwords	132
Enforcing Password Policy	132
Storing Separate Passwords Per Linked Resource	135
Generating Random Passwords	135
Modifying the <code>password</code> Property	136
Rate Limiting Emails	137
5. Secure Network Connections	138
Use TLS/SSL	138
Restrict REST Access to the HTTPS Port	138
Protect Sensitive REST Interface URLs	139
Enable HTTP Strict-Transport-Security	139
Restrict the HTTP Payload Size	140
Deploy Securely Behind a Load Balancer	140
Connect to IDM Through a Proxy Server	141
6. Protect IDM Data	143
Encoding Attribute Values	143
Structure of an Encrypted Object	146
Encrypting and Decrypting Properties Over REST	147
Securing the Repository	149
Protecting Sensitive Files and Directories	150
Removing or Protecting Development and Debug Tools	150
Adjusting Log Levels	151

Disabling the API Explorer	151
Disabling Automatic Configuration Updates	151
Managing Privacy & Consent	152
Securing IDM Server Files With a Read-Only Installation	155
A. Authentication and Session Module Configuration	158

Overview

Out of the box, IDM is set up for ease of development and deployment. When you deploy IDM in production, there are specific precautions you should take to minimize security breaches. This guide describes the IDM security mechanisms and strategies you can use to reduce risk and mitigate threats to IDM security.

Quick Start

 Certificates and Keys Manage secrets, certificates and keys.	 Authentication Employ secure authentication methods.	 Authorization & Access Control Protect REST endpoints with secure authorization and access control.
 Passwords Store and manage passwords securely.	 Network Secure network connections to IDM resources.	 Data Secure IDM stored data.

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

Secret Stores, Certificates and Keys

Encryption makes it possible to protect sensitive data. IDM depends on encryption to negotiate secure network connections, and to keep sensitive data confidential. Encryption in turn depends on keys. IDM stores keys in *secret stores*. This chapter describes the supported secret stores and the features available for managing keys.

As a general precaution in production environments, avoid using self-signed certificates and certificates associated with insecure ciphers.

IDM supports the following secret store types:

- File-based keystores
- Hardware Security Modules (HSM)

Configuring Secret Stores

Secret stores are configured in your project's `conf/secrets.json` file. The `secrets.json` file has the following configuration by default:

```
{
  "stores": [
    {
      "name": "mainKeyStore",
      "class": "org.forgerock.openidm.secrets.config.FileBasedStore",
      "config": {
        "file": "&{openidm.keystore.location}&{idm.install.dir}/security/keystore.jceks",
        "storetype": "&{openidm.keystore.type|JCEKS}",
        "providerName": "&{openidm.keystore.provider|SunJCE}",
        "storePassword": "&{openidm.keystore.password|changeit}",
        "mappings": [
          {
            "secretId": "idm.default",
            "types": [ "ENCRYPT", "DECRYPT" ],
            "aliases": [ "&{openidm.config.crypto.alias|openidm-sym-default}" ]
          },
          ...
        ]
      }
    },
    {
      "name": "mainTrustStore",
      "class": "org.forgerock.openidm.secrets.config.FileBasedStore",
      "config": {
```

```

    "file": "&{openidm.truststore.location}&{idm.install.dir}/security/truststore",
    "storetype": "&{openidm.truststore.type|JKS}",
    "providerName": "&{openidm.truststore.provider|SUN}",
    "storePassword": "&{openidm.truststore.password|changeit}",
    "mappings": [
    ]
  }
},
"populateDefaults": true
}

```

The `mainKeyStore` and `mainTrustStore` properties configure the default secret stores. IDM requires these properties in order to start up. Do not change the property names because they are also provided to third-party products that need a single keystore and a single truststore.

`mainKeyStore`

The main keystore references a Java Cryptography Extension Keystore (JCEKS) located at `/path/to/openidm/security/keystore.jceks`.

`mainTrustStore`

The main truststore references a file-based truststore located at `/path/to/openidm/security/truststore`.

`populateDefaults`

When IDM first starts up, it checks the secrets configuration. If `"populateDefaults": true`, IDM writes a number of encryption keys to the keystore, required to encrypt specific data.

You can manage these keystores and truststores using the **keytool** command, included in your Java installation. For information about the **keytool** command, see <https://docs.oracle.com/en/java/javase/11/tools/keytool.html>.

Each configured store has a `name` and `class` and the following configuration properties:

`file`

For file-based secret stores, this property references the path to the store file, for example, `&{idm.install.dir}/security/keystore.jceks`. Hardware security modules do not have a `file` property.

`storetype`

The type of secret store. IDM supports a number of store types, including JCEKS, JKS, PKCS #11, and PKCS #12.

`providerName`

Sets the name of the cryptographic service provider, for example, `SunPKCS11` or `softHSM`. If no provider is specified, the JRE default is used.

storePassword

The password to the secret store. For the default IDM keystore and truststore, the password is `changeit`. You should change this password in a production deployment, as described in "Changing the Default Keystore Password".

mappings

This object enables you to map keys and certificates in the secret stores to specific encryption and decryption functionality in IDM. A secrets mapping object has the following structure:

```
{
  "secretId" : "idm.config.encryption",
  "types": [ "ENCRYPT", "DECRYPT" ],
  "aliases": [ "&{openidm.config.crypto.alias|openidm-sym-default}" ]
}
```

- `secretId` enables you to map a secret to one or more aliases and gives an indication of the secret's purpose. For example, `idm.config.encryption` indicates the aliases that are used to encrypt and decrypt sensitive configuration properties.
- `types` indicates what the keys are used for, for example, encryption and decryption of sensitive property values.
- `aliases` are the key aliases in the secret store that are used for this purpose. You can add as many aliases as necessary. The first alias in the list determines which alias is the active one. Active secrets are used for signature generation and encryption.

The aliases in the default keystore are described in "Working With the Default Keystore".

The default secret IDs and the aliases to which they are mapped are listed in "Mapping SecretIDs to Key Aliases".

Note

All these properties have a resolvable property value by default, for example `&{openidm.keystore.location}`, that enables you to use *property value substitution*. If no *configuration expression* has been set for that specific property, the value following the vertical bar is used. In the following property, the password is `changeit` unless you have set a configuration expression in one of the property resolver locations:

```
"storePassword": "&{openidm.keystore.password|changeit}"
```

For more information, see "Property Value Substitution" in the *Setup Guide*.

Mapping SecretIDs to Key Aliases

secretId	alias	Description
idm.default	openidm-sym-default	Encryption keystore for legacy JSON objects that do not contain a <code>purpose</code> value in their <code>\$crypto</code> block

secretId	alias	Description
idm.config.encryption	openidm-sym-default	Encrypts configuration information
idm.password.encryption	openidm-sym-default	Encrypts managed user passwords
idm.jwt.session.module.encryption	openidm-localhost	Encrypts JWT session tokens
idm.jwt.session.module.signing	openidm-jwtsessionhmac-key	Signs JWT session tokens using HMAC
idm.selfservice.signing	selfservice	Signs JWT session tokens using RSA
idm.selfservice.encryption	openidm-selfservice-key	Encrypts JWT self-service tokens

Working With the Default Keystore

IDM generates a number of encryption keys in a JCEKS keystore the first time the server starts up. These keys map to the secrets defined in "[Mapping SecretIDs to Key Aliases](#)". Note that the keystore, and the keys, are generated at startup and are not prepackaged. The keys are generated *only* if they do not already exist. You cannot specify custom aliases for these default keys.

To use a different keystore type, such as PKCS #12, create the keystore and generate the keys before you start IDM. This prevents IDM from generating the keys on startup. You can also convert the existing JCEKS keystore to a PKCS #12 keystore. If you use a different keystore type, you must edit the `openidm.keystore.type` property (in the `conf/secrets.json` file) to match the new type.

Use the **keytool** command to list the default encryption keys, as follows:

```
keytool \
-list \
-keystore /path/to/openidm/security/keystore.jceks \
-storepass changeit \
-storetype JCEKS
Keystore type: JCEKS
Keystore provider: SunJCE

Your keystore contains 5 entries

openidm-sym-default, Nov 5, 2019, SecretKeyEntry,
openidm-jwtsessionhmac-key, Nov 5, 2019, SecretKeyEntry,
selfservice, Nov 5, 2019, PrivateKeyEntry,
Certificate fingerprint (SHA-256): E9:0B:BA:FB:58:73:02:FC...:7B
openidm-selfservice-key, Nov 5, 2019, SecretKeyEntry,
openidm-localhost, Nov 5, 2019, PrivateKeyEntry,
Certificate fingerprint (SHA-256): 21:50:6C:90:C7:A7:F7:32...:1B
```

Note

If you are using IDM in a cluster, you must share these keys among all nodes in the cluster. The easiest way to do this is to generate a keystore with the appropriate keys and share the keystore in some way, for example by using a filesystem that is shared between the nodes.

Changing the Default Keystore Password

The default keystore password is `changeit`. You should change this password in a production environment.

Change the Default Keystore Password

1. Shut down the server if it is running.
2. Use the **keytool** command to change the keystore password. The following command changes the keystore password to `newPassword`:

```
keytool \  
-storepasswd \  
-keystore /path/to/openidm/security/keystore.jceks \  
-storetype jceks \  
-storepass changeit  
New keystore password: newPassword  
Re-enter new keystore password: newPassword
```

3. Change the passwords of the default encryption keys.

IDM uses a number of encryption keys, listed in "Mapping SecretIDs to Key Aliases", whose passwords are also `changeit` by default. The passwords of each of these keys must match the password of the keystore.

To get the list of keys in the keystore, run the following command:

```
keytool \  
-list \  
-keystore /path/to/openidm/security/keystore.jceks \  
-storetype jceks \  
-storepass newPassword  
Keystore type: JCEKS  
Keystore provider: SunJCE  
  
Your keystore contains 5 entries  
  
openidm-sym-default, May 4, 2021, SecretKeyEntry,  
selfservice, May 4, 2021, PrivateKeyEntry, Certificate fingerprint (SHA-256): fingerprint  
openidm-jwtsessionhmac-key, May 4, 2021, SecretKeyEntry,  
openidm-localhost, May 4, 2021, PrivateKeyEntry, Certificate fingerprint (SHA-256): fingerprint  
openidm-selfservice-key, May 4, 2021, SecretKeyEntry,
```

Change the passwords of each default encryption key as follows:

```
keytool \
-keypasswd \
-alias openidm-localhost \
-keystore /path/to/openidm/security/keystore.jceks \
-storetype jceks \
-storepass newPassword
Enter key password for <openidm-localhost> changeit
New key password for <openidm-localhost>: newPassword
Re-enter new key password for <openidm-localhost>: newPassword
```

```
keytool \
-keypasswd \
-alias openidm-sym-default \
-keystore /path/to/openidm/security/keystore.jceks \
-storetype jceks \
-storepass newPassword
Enter key password for <openidm-sym-default> changeit
New key password for <openidm-sym-default>: newPassword
Re-enter new key password for <openidm-sym-default>: newPassword
```

```
keytool \
-keypasswd \
-alias openidm-selfservice-key \
-keystore /path/to/openidm/security/keystore.jceks \
-storetype jceks \
-storepass newPassword
Enter key password for <openidm-selfservice-key> changeit
New key password for <openidm-selfservice-key>: newPassword
Re-enter new key password for <openidm-selfservice-key>: newPassword
```

```
keytool \
-keypasswd \
-alias selfservice \
-keystore /path/to/openidm/security/keystore.jceks \
-storetype jceks \
-storepass newPassword
Enter key password for <selfservice> changeit
New key password for <selfservice>: newPassword
Re-enter new key password for <selfservice>: newPassword
```

```
keytool \
-keypasswd \
-alias openidm-jwtsessionhmac-key \
-keystore /path/to/openidm/security/keystore.jceks \
-storetype jceks \
-storepass newPassword
Enter key password for <openidm-jwtsessionhmac-key> changeit
New key password for <openidm-jwtsessionhmac-key>: newPassword
Re-enter new key password for <openidm-jwtsessionhmac-key>: newPassword
```

4. Configure a new **expression resolver** file in the *Setup Guide* to store just the keystore password.
 - a. Create a new directory in **/path/to/openidm/resolver/** that will contain only the properties file for keystore passwords. For example:

```
mkdir /path/to/openidm/resolver/keystore
```

Important

Substituted properties are not encrypted by default. You *must* therefore secure access to this directory, using the appropriate permissions.

- b. Set the `IDM_ENVCONFIG_DIRS` environment variable to include the new directory:

```
export IDM_ENVCONFIG_DIRS=/path/to/openidm/resolver/,/path/to/openidm/resolver/keystore
```

- c. Create a `.json` or `.properties` file in that secure directory, that contains the new keystore password as a resolvable IDM property. For example, add one of the following files to that directory:

keystorepwd.properties

```
openidm.keystore.password=newPassword
```

keystorepwd.json

```
{
  "openidm" : {
    "keystore" : {
      "password" : "newPassword"
    }
  }
}
```

5. Restart IDM.

Important

Repeat this procedure on each node if you run multiple nodes in a cluster to ensure that the new password is present on all nodes.

Using CA-Signed Certificates

You can use existing CA-signed certificates to secure connections and data by importing the certificates into the keystore, and referencing them your `boot.properties` file. Use the **keytool** command to import an existing certificate into the keystore.

The following process imports a CA-signed certificate into the keystore, with the alias *example-com*. Replace this alias with the alias of your certificate:

1. Stop the server if it is running.
2. Back up your existing `openidm/security/keystore` and `openidm/security/truststore` files.
3. Use the **keytool** command to import your existing certificate into the keystore.

Substitute the following in this command:

- `example-cert.p12` with the name of your certificate file.
- `srcstorepass` with the password that you set to open your certificate.
- `example-com` with the existing certificate alias.
- `destination keystore password` with the password you set for the keystore.

If you have not changed the default keystore password, it is `changeit`. In a production environment, you *should* change the default keystore password. For more information, see "Changing the Default Keystore Password".

```
keytool \
-importkeystore \
-srckeystore example-cert.p12 \
-srcstoretype PKCS12 \
-srcstorepass changeit \
-srcalias example-com \
-destkeystore keystore.jceks \
-deststoretype JCEKS \
-destalias openidm-localhost
Importing keystore example-cert.p12 to keystore.jceks...
Enter destination keystore password: changeit
```

The keytool command creates a trusted certificate entry with the specified alias and associates it with the imported certificate. The certificate is imported into the keystore with the alias `openidm-localhost`. If you want to use a different alias, you must modify your `resolver/boot.properties` file to reference that alias, as shown in the following step.

Note

The certificate entry password must be the same as the IDM keystore password. If the source certificate entry password is different from the target keystore password, use the `-destkeypass` option with the same value as the `-deststorepass` option to make the certificate password match the target keystore password. If you do not make these passwords the same, no error is generated when you import the certificate (or when you read the certificate entry in the destination keystore), but IDM will fail to start with the following exception:

```
java.security.UnrecoverableKeyException: Given final block not properly padded.
```

4. If you specified an alias other than `openidm-localhost` for the new certificate, change the value of `openidm.https.keystore.cert.alias` in your `resolver/boot.properties` file to that alias. For example, if your new certificate alias is `example-com`, change the `boot.properties` file as follows:

```
openidm.https.keystore.cert.alias=example-com
```

5. Restart the server for the new certificate to be taken into account.

Deleting Certificates

If you are using CA-signed certificates for encryption, it is best practice to delete the unused default certificates from the keystore and the truststore. You can delete certificates from a keystore using the **keytool** command.

The following example deletes the **openidm-localhost** certificate from the keystore:

```
keytool \  
-delete \  
-alias openidm-localhost \  
-keystore /path/to/openidm/security/keystore.jceks \  
-storetype JCEKS \  
-storepass changeit
```

The following example deletes the **openidm-localhost** certificate from the truststore:

```
keytool \  
-delete \  
-alias openidm-localhost \  
-keystore /path/to/openidm/security/truststore \  
-storepass changeit
```

You can use similar commands to delete custom certificates from the keystore and truststore, specifying the certificate alias in the request.

Repeat these steps to delete all the default certificate aliases that you are not using in your deployment.

Removing Unused CA Certificates

The Java and IDM truststore files include a number of root CA certificates. Although the probability of a compromised root CA certificate is low, it is best practice to delete root CA certificates that are not used in your deployment.

To review the list of root CA certificates in the IDM truststore, run the following command:

```
keytool \  
-list \  
-keystore /path/to/openidm/security/truststore \  
-storepass changeit
```

On UNIX/Linux systems, you can find additional lists of root CA certificates in files named **cacerts**. These include root CA certificates associated with your Java environment, such as Oracle JDK or OpenJDK. You should be able to find that file in `${JAVA_HOME}/jre/lib/security/cacerts`.

Before changing Java environment keystore files, make sure that the Java-related **cacerts** files are up to date and verify that you have a supported Java version installed:

Supported Java Versions

Vendor	Versions
OpenJDK, including OpenJDK-based distributions: <ul style="list-style-type: none"> • AdoptOpenJDK/Eclipse Adoptium • Amazon Corretto • Azul Zulu • Red Hat OpenJDK ForgeRock tests most extensively with AdoptOpenJDK/Eclipse Adoptium.	11
Oracle Java	11

You can remove root CA certificates with the **keytool** command. For example, the following command removes the hypothetical **examplecomca2** certificate from the truststore:

```
keytool \
-delete \
-keystore /path/to/openidm/security/truststore \
-storepass changeit \
-alias examplecomca2
```

Repeat the process for all root CA certificates that are not used in your deployment.

On Windows systems, you can manage certificates with the Microsoft Management Console (MMC) snap-in tool. For more information, see [Working With Certificates](#) in the Microsoft documentation.

Changing and Rotating Encryption Keys

Most regulatory requirements mandate that the keys used to decrypt sensitive data be rotated out and replaced with new keys on a regular basis. The main purpose of rotating encryption keys is to reduce the amount of data encrypted with that key, so that the potential impact of a security breach with a specific key is reduced. You can update encryption keys in several ways, including the following:

- "Rotating Encryption Keys Manually"
- "Using Scheduled Tasks to Rotate Keys"
- "Changing the Active Alias for Managed Object Encryption"

Rotating Encryption Keys Manually

IDM evaluates keys in `secrets.json` sequentially. For example, assume that you have added a new key named `my-new-key` to the keystore, as described in "Using CA-Signed Certificates".

To use this new key to encrypt passwords, you would include `my-new-key` as the *first alias* in the `idm.password.encryption` secret, as follows:

```
{
  "secretId" : "idm.password.encryption",
  "types": [ "ENCRYPT", "DECRYPT" ],
  "aliases": [ "my-new-key", "&{openidm.config.crypto.alias|openidm-sym-default}" ]
}
```

The properties that use this key (in this case, passwords) are re-encrypted with the new key the next time the managed object is *updated*. You do not need to restart the server.

Important

If you rotate an encryption key, the *active* encryption key might not be the correct key to use for decryption of properties that have already been encrypted with a previous key.

You must therefore keep all applicable keys in `secrets.json` until every object that is encrypted with old keys have been updated with the latest key.

You can force key rotation on all managed objects by running the `triggerSyncCheck` action on the entire managed object data set. The `triggerSyncCheck` action examines the crypto blob of each object and updates the encrypted property with the correct key.

For example, the following command forces all managed user objects to use the new key:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--header "Content-Type: application/json" \
--request POST \
"https://localhost:8443/openidm/managed/user/?_action=triggerSyncCheck"
{
  "status": "OK",
  "countTriggered": 10
}
```

In a large managed object set, the `triggerSyncCheck` action can take a long time to run on only a single node. You should therefore avoid using this action if your data set is large. An alternative to running `triggerSyncCheck` over the entire data set is to iterate over the managed data set and call `triggerSyncCheck` on each individual managed object. You can call this action manually or by using a script.

The following example shows the manual commands that must be run to launch the `triggerSyncCheck` action on all managed users. The first command uses a query filter to return all managed user IDs. The second command iterates over the returned IDs calling `triggerSyncCheck` on each ID:


```
curl \
--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"
{
  "result": [
    {
      "_id": "9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb",
      "_rev": "000000004988917b"
    },
    {
      "_id": "55ef0a75-f261-47e9-a72b-f5c61c32d339",
      "_rev": "00000000dd89d671"
    },
    {
      "_id": "998a6181-d694-466a-a373-759a05840555",
      "_rev": "000000006fea54ad"
    },
    ...
  ]
}
```

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--header "Content-Type: application/json" \
--request POST \
"https://localhost:8443/openidm/managed/user/9dce06d4-2fc1-4830-a92b-bd35c2f6bcbb?_action=triggerSyncCheck"
```

In large data sets, the most efficient way to achieve key rotation is to use the scheduler service to launch these commands. The following section shows how to use the scheduler service for this purpose.

Using Scheduled Tasks to Rotate Keys

This example uses a script to generate multiple scheduled tasks. Each scheduled task iterates over a subset of the managed object set (defined by the `pageSize`). The generated scheduled task then calls another script that launches the `triggerSyncCheck` action on each managed object in that subset.

You can set up a similar schedule as follows:

1. Create a schedule configuration named `schedule-triggerSyncCheck.json` in your project's `conf` directory. That schedule should look as follows:

```
{
  "enabled" : true,
  "persisted" : true,
  "type" : "cron",
  "schedule" : "0 * * * * ? *",
  "concurrentExecution" : false,
  "invokeService" : "script",
  "invokeContext" : {
    "waitForCompletion" : false,
    "script": {
      "type": "text/javascript",
      "name": "sync/scheduleTriggerSyncCheck.js"
    },
    "input": {
      "pageSize": 2,
      "managedObjectPath" : "managed/user",
      "quartzSchedule" : "0 * * * * ? *"
    }
  }
}
```

You can change the following parameters of this schedule configuration to suit your deployment:

pageSize

The number of objects that each generated schedule will handle. This value should be high enough not to create too many schedules. The number of schedules that is generated is equal to the number of objects in the managed object store, divided by the page size.

For example, if there are 500 managed users and a page size of 100, five schedules will be generated (500/100).

managedObjectPath

The managed object set over which the scheduler iterates. For example, `managed/user` if you want to iterate over the managed user object set.

quartzSchedule

The schedule at which these tasks should run. For example, to run the task every minute, this value would be ``0 * * * * ? *``.

- The schedule calls a `scheduleTriggerSyncCheck.js` script, located in a directory named `project-dir/script/sync`. Create the `sync` directory, and add that script as follows:

```
var managedObjectPath = object.managedObjectPath;
var pageSize = object.pageSize;
var quartzSchedule = object.quartzSchedule;

var managedObjects = openidm.query(managedObjectPath, {
  "_queryFilter": "true",
  "_fields": "_id"
});
```

```
var numberOfManagedObjects = managedObjects.result.length;

for (var i = 0; i < numberOfManagedObjects; i += pageSize) {
  var scheduleId = java.util.UUID.randomUUID().toString();
  var ids = managedObjects.result.slice(i, i + pageSize).map(function(obj) {
    return obj._id
  });
  var schedule = newSchedule(scheduleId, ids);
  openidm.create("/scheduler", scheduleId, schedule);
}

function newSchedule(scheduleId, ids) {
  var schedule = {
    "enabled": true,
    "persisted": true,
    "type": "cron",
    "schedule": quartzSchedule,
    "concurrentExecution": false,
    "invokeService": "script",
    "invokeContext": {
      "waitForCompletion": true,
      "script": {
        "type": "text/javascript",
        "name": "sync/triggerSyncCheck.js"
      },
      "input": {
        "ids": ids,
        "managedObjectPath": managedObjectPath,
        "scheduleId": scheduleId
      }
    }
  };
  return schedule;
}
```

3. Each generated scheduled task calls a script named `triggerSyncCheck.js`. Create that script in your project's `script/sync` directory. The contents of the script are as follows:

```
var ids = object.ids;
var scheduleId = object.scheduleId;
var managedObjectPath = object.managedObjectPath;

for (var i = 0; i < ids.length; i++) {
  openidm.action(managedObjectPath + "/" + ids[i], "triggerSyncCheck", {}, {});
}

openidm.delete("/scheduler/" + scheduleId, null);
```

4. When you have set up the schedule configuration and the two scripts, you can test this key rotation as follows:
 - a. Edit your project's `conf/managed.json` file to return user passwords by default by setting `"scope" : "public"`.

```
"password" : {
  ...
  "encryption" : {
    "purpose" : "idm.password.encryption"
  },
  "scope" : "public",
  ...
}
```

Because passwords are not returned by default, you will not be able to see the new encryption on the password unless you change the property's **scope**.

- b. Perform a GET request to return any managed user entry in your data set. For example:

```
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/managed/user/ccd92204-ae6-4159-879a-46eeb4362807"
{
  "_id" : "ccd92204-ae6-4159-879a-46eeb4362807",
  "_rev" : "0000000009441230",
  "preferences" : {
    "updates" : false,
    "marketing" : false
  },
  "mail" : "bjensen@example.com",
  "sn" : "Jensen",
  "givenName" : "Babs",
  "userName" : "bjensen",
  "password" : {
    "$crypto" : {
      "type" : "x-simple-encryption",
      "value" : {
        "cipher" : "AES/CBC/PKCS5Padding",
        "stableId" : "openidm-sym-default",
        "salt" : "CVrKDufzfzXfTDbCwU1Rw==",
        "data" : "1I5tWT5aRH/12hf5DgofXA==",
        "keySize" : 16,
        "purpose" : "idm.password.encryption",
        "iv" : "LGE+jnC3ZtyvrE5pfuSvtA==",
        "mac" : "BEXQ1mftxA63dXhJ06dDZQ=="
      }
    }
  },
  "accountStatus" : "active",
  "effectiveRoles" : [ ],
  "effectiveAssignments" : [ ]
}
```

Notice that the user's password is encrypted with the default encryption key (**openidm-sym-default**).

- c. Create a new encryption key in the IDM keystore:

```
keytool \
-genseckey \
-alias my-new-key \
-keyalg AES \
-keysize 128 \
-keystore /path/to/openidm/security/keystore.jceks \
-storetype JCEKS
```

- d. Shut down the server for keystore to be reloaded.
- e. Change your project's `conf/managed.json` file to change the encryption purpose for managed user passwords:

```
"password" : {
  ...
  "encryption" : {
    "purpose" : "idm.password.encryption2"
  },
  "scope" : "public",
  ...
}
```

- f. Add the corresponding `purpose` to the `secrets.json` file in the `mainKeyStore` code block:

```
"idm.password.encryption2": {
  "types": [ "ENCRYPT", "DECRYPT" ],
  "aliases": [
    {
      "alias": "my-new-key"
    }
  ]
}
```

- g. Restart the server and wait one minute for the first scheduled task to fire.
- h. Perform a GET request again to return the entry of the managed user that you returned previously:

```
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/managed/user/ccd92204-ae6-4159-879a-46eeb4362807"
{
  "_id" : "ccd92204-ae6-4159-879a-46eeb4362807",
  "_rev" : "0000000009441230",
  "preferences" : {
    "updates" : false,
    "marketing" : false
  },
  "mail" : "bjensen@example.com",
  "sn" : "Jensen",
  "givenName" : "Babs",
  "userName" : "bjensen",
```

```
"password" : {
  "$crypto" : {
    "type" : "x-simple-encryption",
    "value" : {
      "cipher" : "AES/CBC/PKCS5Padding",
      "stableId" : "my-new-key",
      "salt" : "CvRkDuzfzunXfTDbCwU1Rw==",
      "data" : "1I5tWT5aRH/12hf5DgofXA==",
      "keySize" : 16,
      "purpose" : "idm.password.encryption2",
      "iv" : "LGE+jnC3ZtyvrE5pfuSvtA==",
      "mac" : "BEXQ1mftxA63dXhJ06dDZQ=="
    }
  }
},
"accountStatus" : "active",
"effectiveRoles" : [ ],
"effectiveAssignments" : [ ]
}
```

Notice that the user password is now encrypted with `my-new-key`.

Changing the Active Alias for Managed Object Encryption

This example describes how you can configure and then change the managed object encryption key with a scheduled task. You'll create a new key, set up a managed user, add the key to `secrets.json`, restart IDM, run a `triggerSyncCheck`, and review the result.

1. Create a new key for the IDM keystore in the `security/keystore.jceks` file:

```
keytool \
-genseckey \
-alias my-new-key \
-keyalg AES \
-keysize 128 \
-keystore /path/to/openidm/security/keystore.jceks \
-storetype JCEKS
```

2. Solely for the purpose of this example, in `managed.json`, set `"scope" : "public"` to expose the applied password encryption key.
3. Create a managed 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 \
--header "Content-Type: application/json" \
--request PUT \
--data '{
  "userName": "rsutter",
  "sn": "Sutter",
  "givenName": "Rick",
  "mail": "rick@example.com",
  "telephoneNumber": "6669876987",
  "description": "Another user",
  "country": "USA",
  "password": "Passw0rd"
}' \
"https://localhost:8443/openidm/managed/user/ricksutter"
```

4. Add the newly created `my-new-key` alias to your `conf/secrets.json` file, in the `idm.password.encryption` code block:

```
"idm.password.encryption": {
  "types": [ "ENCRYPT", "DECRYPT" ],
  "aliases": [ "my-new-key", "&{openidm.config.crypto.alias|openidm-sym-default}" ]
}
```

5. To apply the new key to your configuration, shut down and restart IDM.
6. Force IDM to update the key for your users with the `triggerSyncCheck` action:

```
curl \
--header "X-OpenIDM-Username: openidm-admin" \
--header "X-OpenIDM-Password: openidm-admin" \
--header "Accept-API-Version: resource=1.0" \
--cacert ca-cert.pem \
--header "Content-Type: application/json" \
--request POST \
"https://localhost:8443/openidm/managed/user/?_action=triggerSyncCheck"
```

7. Review the result for the newly created user, `ricksutter`:

```
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/managed/user/ricksutter"
```

8. In the output, you should see the new `my-new-key` encryption key applied to that user's password:

```
...
  "password": {
    "$crypto": {
      "type": "x-simple-encryption",
      "value": {
        "cipher": "AES/CBC/PKCS5Padding",
        "stableId": "my-new-key",
        "salt": "bGyKG3PKmwH0N0fxerr1Qg==",
        "data": "6vXZiJ3ZNN/UUnsrT7dTQw==",
        "keySize": 16,
        "purpose": "idm.password.encryption",
        "iv": "doAdtxfWfFbrPIIfubGi5g==",
        "mac": "0ML6xd9qvDtD5AvMc1Tc3A=="
      }
    }
  },
  ...
}
```

Configuring IDM For a Hardware Security Module (HSM) Device

This section demonstrates how to use a PKCS #11 device, such as a hardware security module (HSM), to store the keys used to secure communications. IDM supports retrieval of secrets from HSMs either locally or over the network.

Note

On Windows systems using the 64-bit JDK, the Sun PKCS #11 provider is available *only* from JDK version 1.8b49 onwards. If you want to use a PKCS #11 device on Windows, either use the 32-bit version of the JDK, or upgrade your 64-bit JDK to version 1.8b49 or higher.

Setting Up the HSM Configuration

This section assumes that you have access to an HSM device (or a software emulation of an HSM device, such as SoftHSM) and that the HSM provider has been configured and initialized.

The command-line examples in this section use SoftHSM for testing purposes. Before you start, set the correct environment variable for the SoftHSM configuration, for example:

```
export SOFTHSM2_CONF=/usr/local/Cellar/softhsm/2.0.0/etc/softhsm2.conf
```

Also initialize slot 0 on the provider, with a command similar to the following:

```
softhsm2-util --init-token --slot 0 --label "My token 1"
```

This token initialization requests two PINs—an SO PIN and a user PIN. You can use the SO PIN to reinitialize the token. The user PIN is provided to IDM so that it can interact with the token. Remember the values of these PINs because you will use them later in this section.

The PKCS #11 standard uses a configuration file to interact with the HSM device. The following example shows a basic configuration file for SoftHSM:

```
name = softHSM
library = /usr/local/Cellar/softHSM/2.0.0/lib/softHSM/libsoftHSM2.so
slot = 1
attributes(generate, *, *) = {
    CKA_TOKEN = true
}
attributes(generate, CKO_CERTIFICATE, *) = {
    CKA_PRIVATE = false
}
attributes(generate, CKO_PUBLIC_KEY, *) = {
    CKA_PRIVATE = false
}
attributes(*, CKO_SECRET_KEY, *) = {
    CKA_PRIVATE = false
    CKA_EXTRACTABLE = true
}
```

Your HSM configuration file *must* include at least the following settings:

name

A suffix to identify the HSM provider. This example uses the `softHSM` provider.

library

The path to the PKCS #11 library.

slot

The slot number to use, specified as a string. Make sure that the slot you specify here has been initialized on the HSM device.

The `attributes` specify additional PKCS #11 attributes that are set by the HSM. For a complete list of these attributes, see the PKCS #11 Reference.

Important

If you are using the JWT Session Module, you *must* set `CKA_EXTRACTABLE = true` for secret keys in your HSM configuration file. For example:

```
attributes(*, CKO_SECRET_KEY, *) = {
    CKA_PRIVATE = false
    CKA_EXTRACTABLE = true
}
```

The HSM provider must allow secret keys to be extractable because the authentication service serializes the JWT Session Module key and passes it to the authentication framework as a base 64-encoded string.

Populating the Default Encryption Keys

When IDM first starts up, it generates a number of encryption keys required to encrypt specific data. If you are using an HSM provider, you must generate these keys manually. The secret keys must use an HMAC algorithm. The following steps set up the required encryption keys.

Note

This procedure assumes that your HSM configuration file is located at `/path/to/hsm/hsm.conf`:

1. The `openidm-sym-default` key is the default symmetric key required to encrypt the configuration. The following command generates that key in the HSM provider. The `-providerArg` must point to the HSM configuration file described in "Setting Up the HSM Configuration".

```
keytool \  
-genseckey \  
-alias openidm-sym-default \  
-keyalg HmacSHA256 \  
-keysize 256 \  
-keystore NONE \  
-storetype PKCS11 \  
-providerClass sun.security.pkcs11.SunPKCS11 \  
-providerArg /path/to/hsm/hsm.conf  
Enter keystore password:
```

Enter the password of your HSM device. If you are using SoftHSM, enter your user PIN as the keystore password. The remaining sample steps use *user PIN* as the password.

2. The `openidm-selfservice-key` is used by the Self-Service UI to encrypt managed user passwords and other sensitive data. Generate that key with a command similar to the following:

```
keytool \  
-genseckey \  
-alias openidm-selfservice-key \  
-keyalg HmacSHA256 \  
-keysize 256 \  
-keystore NONE \  
-storetype PKCS11 \  
-providerClass sun.security.pkcs11.SunPKCS11 \  
-providerArg /path/to/hsm/hsm.conf  
Enter keystore password: user PIN
```

Enter the password of your HSM device. If you are using SoftHSM, enter your user PIN as the keystore password.

3. The `openidm-jwtsessionhmac-key` is used by the JWT session module to encrypt JWT session cookies. For more information, see "JWT_SESSION". Generate the JWT session module key with a command similar to the following:

```
keytool \  
-genseckey \  
-alias openidm-jwtsessionhmac-key \  
-keyalg HmacSHA256 \  
-keysize 256 \  
-keystore NONE \  
-storetype PKCS11 \  
-providerClass sun.security.pkcs11.SunPKCS11 \  
-providerArg /path/to/hsm/hsm.conf  
Enter keystore password: user PIN
```

4. The `openidm-localhost` certificate is used to support SSL/TLS. Generate that certificate with a command similar to the following:

```
keytool \  
-genkey \  
-alias openidm-localhost \  
-keyalg RSA \  
-keysize 2048 \  
-keystore NONE \  
-storetype PKCS11 \  
-providerClass sun.security.pkcs11.SunPKCS11 \  
-providerArg /path/to/hsm/hsm.conf  
Enter keystore password: user PIN  
What is your first and last name?  
[Unknown]: localhost  
What is the name of your organizational unit?  
[Unknown]:  
What is the name of your organization?  
[Unknown]: OpenIDM Self-Signed Certificate  
What is the name of your City or Locality?  
[Unknown]:  
What is the name of your State or Province?  
[Unknown]:  
What is the two-letter country code for this unit?  
[Unknown]:  
Is CN=localhost, OU=Unknown, O=OpenIDM Self-Signed Certificate, L=Unknown, ST=Unknown, C=Unknown  
correct?  
[no]: yes
```

5. The `selfservice` certificate secures requests from the End User UI. Generate that certificate with a command similar to the following:

```
keytool \
-genkey \
-alias selfservice \
-keyalg RSA \
-keysize 2048 \
-keystore NONE \
-storetype PKCS11 \
-providerClass sun.security.pkcs11.SunPKCS11 \
-providerArg /path/to/hsm/hsm.conf
Enter keystore password: user PIN
What is your first and last name?
[Unknown]: localhost
What is the name of your organizational unit?
[Unknown]:
What is the name of your organization?
[Unknown]: OpenIDM Self Service Certificate
What is the name of your City or Locality?
[Unknown]:
What is the name of your State or Province?
[Unknown]:
What is the two-letter country code for this unit?
[Unknown]:
Is CN=localhost, O=OpenIDM Self Service Certificate, OU=None, L=None, ST=None, C=None?
[no]: yes
```

6. If you are *not* using the HSM provider for the truststore, you must add the certificates generated in the previous two steps to the default IDM truststore.

If you *are* using the HSM provider for the truststore, you can skip this step.

To add the `openidm-localhost` certificate to the IDM truststore, export the certificate from the HSM provider, then import it into the truststore, as follows:

```
keytool \
-export \
-alias openidm-localhost \
-file exportedCert \
-keystore NONE \
-storetype PKCS11 \
-providerClass sun.security.pkcs11.SunPKCS11 \
-providerArg /path/to/hsm/hsm.conf
Enter keystore password: user PIN
Certificate stored in file exportedCert

keytool \
-import \
-alias openidm-localhost \
-file exportedCert \
-keystore /path/to/openidm/security/truststore
Enter keystore password: changeit
Owner: CN=localhost, OU=Unknown, O=OpenIDM Self-Signed Certificate, L=...
Issuer: CN=localhost, OU=Unknown, O=OpenIDM Self-Signed Certificate, L=...
Serial number: 5d2554bd
Valid from: Fri Aug 19 13:11:54 SAST 2016 until: Thu Nov 17 13:11:54 SAST 2016
Certificate fingerprints:
MD5: F1:9B:72:7F:7B:79:58:29:75:85:82:EC:79:D8:F9:8D
SHA1: F0:E6:51:75:AA:CB:14:3D:C5:E2:EB:E5:7C:87:C9:15:43:19:AF:36
```

```
SHA256: 27:A5:B7:0E:94:9A:32:48:0C:22:0F:BB:7E:3C:22:2A:64:B5:45:24:14:70:...
Signature algorithm name: SHA256withRSA
Version: 3
```

Extensions:

```
#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 7B 5A 26 53 61 44 C2 5A   76 E4 38 A8 52 6F F2 89   .Z&SaD.Zv.8.Ro..
0010: 20 04 52 EE                               .R.
]
]
Trust this certificate? [no]: yes
Certificate was added to keystore
```

The default truststore password is *changeit*.

Configuring IDM to Support an HSM Provider

To enable IDM to use an HSM provider, make the following configuration changes:

In your secret store configuration (`conf/secrets.json`)

Change the `mainKeyStore` and `mainTrustStore` to reference the HSM. For example:

```
{
  "stores": [
    {
      "name": "mainKeyStore",
      "class": "org.forgerock.openidm.secrets.config.HsmBasedStore",
      "config": {
        "storetype": "&{openidm.keystore.type|PKCS11}",
        "providerName": "&{openidm.keystore.provider|SunPKCS11-softHSM}",
        "storePassword": "&{openidm.keystore.password|changeit}",
        "mappings": [
          {
            "secretId": "idm.default",
            "types": [ "ENCRYPT", "DECRYPT" ],
            "aliases": [ "&{openidm.config.crypto.alias|openidm-sym-default}" ]
          },
          {
            "secretId": "idm.config.encryption",
            "types": [ "ENCRYPT", "DECRYPT" ],
            "aliases": [ "&{openidm.config.crypto.alias|openidm-sym-default}" ]
          },
          {
            "secretId": "idm.password.encryption",
            "types": [ "ENCRYPT", "DECRYPT" ],
            "aliases": [ "&{openidm.config.crypto.alias|openidm-sym-default}" ]
          },
          {
            "secretId": "idm.jwt.session.module.encryption",
            "types": [ "ENCRYPT", "DECRYPT" ],
            "aliases": [ "&{openidm.https.keystore.cert.alias|openidm-localhost}" ]
          }
        ]
      }
    }
  ]
}
```

```

    {
      "secretId" : "idm.jwt.session.module.signing",
      "types": [ "SIGN", "VERIFY" ],
      "aliases": [ "${openidm.config.crypto.jwtsession.hmackey.alias|openidm-jwtsessionhmac-
key}" ]
    },
    {
      "secretId" : "idm.selfservice.signing",
      "types": [ "SIGN", "VERIFY" ],
      "aliases": [ "selfservice" ]
    },
    {
      "secretId" : "idm.selfservice.encryption",
      "types": [ "ENCRYPT", "DECRYPT" ],
      "aliases": [ "${openidm.config.crypto.selfservice.sharedkey.alias|openidm-selfservice-
key}" ]
    }
  ]
},
{
  "name": "mainTrustStore",
  "class": "org.forgerock.openidm.secrets.config.HsmBasedStore",
  "config": {
    "storetype": "${openidm.keystore.type|PKCS11}",
    "providerName": "${openidm.keystore.provider|SunPKCS11-softHSM}",
    "storePassword": "${openidm.keystore.password|changeit}",
    "mappings": [
    ]
  }
},
{
  "populateDefaults": false
}

```

Note

The `"populateDefaults": false` turns off the default key generation. This setting is *required* for an HSM key provider.

In the IDM Java security file (`conf/java.security`)

Specify the location of your PKCS #11 configuration file. For example:

```
security.provider.14=SunPKCS11 /path/to/pkcs11/config/pkcs11.conf
```

Templates for the `pkcs11.conf` file are included in your PKCS package.

You should now be able to start IDM with the keys in the HSM provider.

Chapter 2

Secure 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 provides a number of authentication mechanisms and uses roles to restrict access to specific REST endpoints. This chapter describes the authentication mechanisms and explains how to use authentication mechanisms securely.

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 does understand utilities such as `curl` 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" \
...
```

Secure 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 enables you to 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"
```

For more information, see [RFC 5987](#).

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. For more information about self-registration, see "*Self-Registration*" in the *Self-Service Reference*.

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"
  },
...
```

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 **repo** endpoint as follows:

```
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/internal/user?_queryFilter=true&fields=_id"
{
  "result": [
    {
      "_id": "openidm-admin",
      "_rev": "00000000ef9f6b13"
    },
    {
      "_id": "anonymous",
      "_rev": "00000000d6216802"
    },
    {
      "_id": "idm-provisioning",
      "_rev": "00000000895c385a"
    }
  ],
  "resultCount": 3,
  "pagedResultsCookie": null,
  "totalPagedResultsPolicy": "NONE",
  "totalPagedResults": -1,
  "remainingPagedResults": -1
}
```

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" \
--cacert ca-cert.pem \
--request GET \
"https://localhost:8443/openidm/internal/user/openidm-admin"
{
  "_id": "openidm-admin",
  "_rev": "00000000c7554e13"
}
```

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": "xBLTp67ze4Ca5LTocX0poA==",
          "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": "xBLTp67ze4Ca5LTocX0poA==",
                "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+QcjfkEbrK7W4tIc30RD1ck=",
            "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"
- "rsFilter"

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.

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" : "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
}
```


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 `conf/authentication.json`).

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": "0:0:0:0:0:0:1",
    "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 *"Configuring 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. For more information, see "Configuring the Social Providers Authentication Module" in the *Self-Service Reference*.

IWA

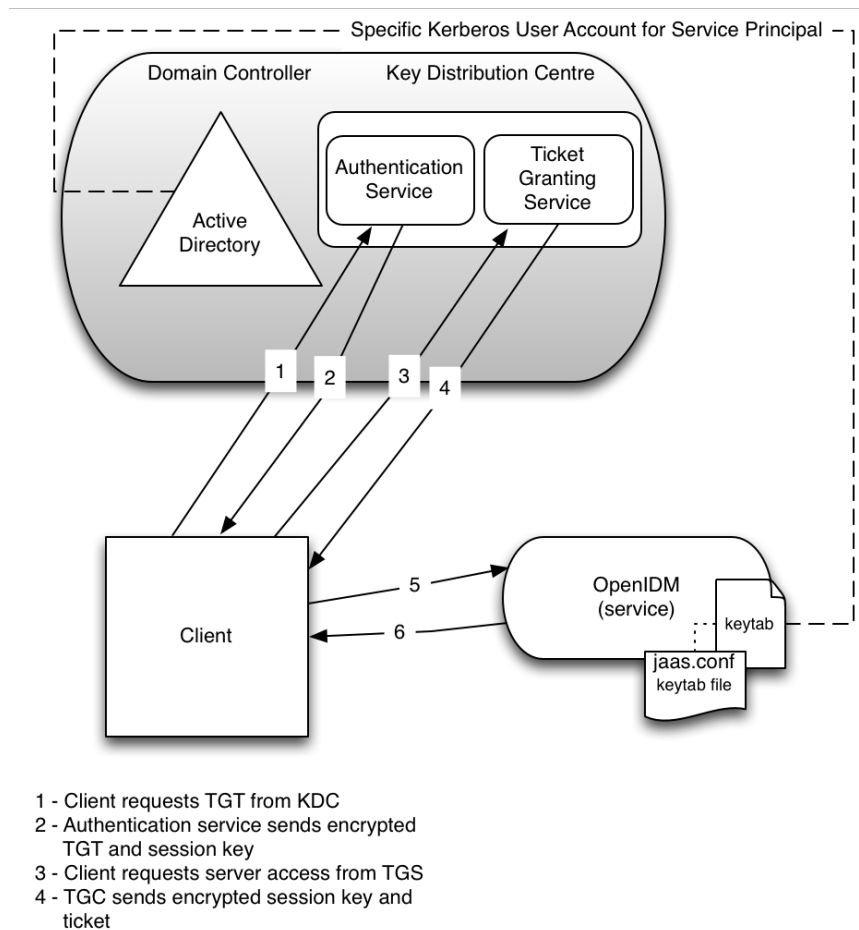
The **IWA** module lets users authenticate using Integrated Windows Authentication (IWA) with Kerberos instead of a username and password.

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".

rsFilter

IDM uses the `rsFilter` authentication module when you configure IDM to use AM bearer tokens for authentication. This module replaces all other authentication methods.

Important

From IDM 7.0 onwards, using AM bearer tokens for authentication is the only supported method of integrating IDM with AM.

When you use IDM and AM together as a platform, configure IDM to use AM bearer tokens for authentication, instead of setting up traditional authentication modules. This delegates all authentication to AM.

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.)

To use bearer tokens for authentication, your IDM `conf/authentication.json` file must include the `rsFilter` authentication module and *no other* authentication methods.

The following sample `authentication.json` file 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"
    },
    "subjectMapping" : [
      {
        "queryOnResource": "managed/user",
        "propertyMapping": {
          "sub": "_id"
        },
        "userRoles": "authzRoles/*",
        "defaultRoles" : [
          "internal/role/openidm-authorized"
        ]
      }
    ],
    "staticUserMapping": [
      {
        "subject": "amadmin",
        "localUser": "internal/user/openidm-admin",
        "roles" : [
          "internal/role/openidm-authorized",
          "internal/role/openidm-admin"
        ]
      },
      {
        "subject": "idm-provisioning",
        "localUser": "internal/user/idm-provisioning",
        "roles" : [
          "internal/role/platform-provisioning"
        ]
      }
    ],
    "anonymousUserMapping" : {
      "localUser": "internal/user/anonymous",
      "roles" : [
        "internal/role/openidm-reg"
      ]
    }
  }
}
```

The **rsFilter** module includes the following properties:

clientId

The client ID of the AM OAuth 2.0 client used to introspect the bearer token.

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.

scopes

Any scopes that are required to be present in the access token. This will vary depending on your configuration.

cache

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

subjectMapping

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

```
"subjectMapping": [
  {
    "realm": "/",
    "queryOnResource": "managed/user",
    "propertyMapping": {
      "sub": "_id"
    },
    "userRoles": "authzRoles/*"
  }
]
```

Each `subjectMapping` includes the following properties:

- **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 corresponding `queryOnResource` property uses a dynamic handlebars template (see the example below).

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.

- **queryOnResource**: The IDM resource to check for the authenticating user; for example, `managed/user`.

This field supports 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:

```
"queryOnResource": "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.

- **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.

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.

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`.
- **userRoles**: the property that determines what roles the identified user will have. Usually `authzRoles`, unless you have changed how user roles are stored.

staticUserMapping

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

- **subject** of the access token (the AM user).

- `localUser` is the IDM user you wish to associate with the AM user identified in `subject`. For example, if `subject` is set to `amAdmin`, you may wish to set `localUser` to `internal/user/openidm-admin`.
- `userRoles` refers to the property used to determine what roles and access the identified user should have. This will usually be `authzRoles`, unless you have changed how user roles are stored.

Note

The `idm-provisioning` user is a service account used by AM to provision users in IDM. Be sure to include this user in your `staticUserMapping`:

```
{
  "subject": "idm-provisioning",
  "localUser": "internal/user/idm-provisioning",
  "userRoles": "authzRoles/*"
}
```

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

Authenticating as a Different User

The `X-OpenIDM-RunAs` header enables an administrative user to *masquerade* as a regular user, without needing that user's password. To support this header, you must add a `runAsProperties` object to the required authentication module configuration.

The sample `authentication.json` file in `openidm/samples/example-configurations/conf/runas/` adds support for the header to the `INTERNAL_USER` module. This means that users or clients who authenticate using the `INTERNAL_USER` module can masquerade as other users.

The `runAsProperties` object has the following configuration:

```
"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 allows a user authenticated with the `openidm-admin` role to masquerade as any user except one with the `openidm-admin` role.

In the following example, the `openidm-admin` user authenticates with the `INTERNAL_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" \
--cacert ca-cert.pem \
--header "X-OpenIDM-RunAs: bjensen" \
--request GET \
"https://localhost:8443/openidm/info/login"
{
  "_id": "login",
  "authenticationId": "bjensen",
  "authorization": {
    "component": "managed/user",
    "authLogin": false,
    "adminUser": "openidm-admin",
    "roles": [ "internal/role/openidm-authorized" ],
    "ipAddress": "127.0.0.1",
    "protectedAttributeList": [ "password" ],
    "id": "847fe36b-115b-4769-b74a-d546f0d0ffc8",
    "moduleId": "INTERNAL_USER"
  }
}
```

The authentication output shows that the request was made as user `bjensen` but with an `adminUser` of `openidm-admin`. Note that 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" \
--cacert ca-cert.pem \
--request GET \
"https://localhost:8443/openidm/info/login"
{
  "_id": "login",
  "authenticationId": "bjensen",
  "authorization": {
    "component": "managed/user",
    "authLogin": false,
    "roles": [ "internal/role/openidm-authorized" ],
    "ipAddress": "127.0.0.1",
    "protectedAttributeList": [ "password" ],
    "id": "847fe36b-115b-4769-b74a-d546f0d0ffc8",
    "moduleId": "MANAGED_USER"
  }
}
```

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 "Privileges and Delegation to 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 3

Protect REST Endpoints With 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"
- "Privileges and Delegation to Restrict Administrative Access"
- "Administrative Users"
- "Hide Unused REST Endpoints"

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": "0:0:0:0:0:0:1",
    "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": "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(['ids','_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(['ids','_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(['ids','_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",
  "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"
  }
}
```

Privileges and Delegation to 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": "000000008fefel60",
  "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": "000000008fefel60",
      "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": "000000008fefel60",
  "_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": "000000008fefel60",
  "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": "000000000658fe17a",
  "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`.

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. Optionally, *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": "xB1Tp67ze4Ca5LTocX0poA==",
                "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"

```

- 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": "xBLTp67ze4Ca5LTocX0poA==",
                "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"

```

Hide Unused REST Endpoints

The two main use cases for IDM are data synchronization and user self-service.

If you are using IDM *only* to synchronize data sources, do not expose the server externally. In this case, all connections are initiated by IDM.

If you are using IDM *only* for user self-service, ensure that the server is placed behind a firewall or proxy, such as ForgeRock Identity Gateway. At a minimum, hide the `/admin` endpoint in the web interface via the proxy. Use the `conf/access.json` file as a guide for proxy or firewall rules.

If you are using IDM for data synchronization *and* user self-service, it is preferable to run two IDM servers or clusters, each with its own security model. Because the two use cases have very different load characteristics and security implications, running them on separate servers can help to prevent synchronization activity from impacting the performance on end-user systems.

Chapter 4

Secure Passwords

IDM provides password management features that help you enforce password policies, limit the number of passwords users must remember, and allow users to reset and change their passwords.

Enforcing Password Policy

A password policy is a set of rules defining what sequence of characters constitutes an acceptable password. Acceptable passwords generally are too complex for users or automated programs to generate or guess.

Password policies set requirements for password length, character sets that passwords must contain, dictionary words and other values that passwords must not contain. Password policies also require that users not reuse old passwords, and that users change their passwords on a regular basis.

IDM enforces password policy rules as part of the general [policy service](#). The default password policy applies the following rules to passwords as they are created and updated:

- A password property is required for any user object.
- The value of a password cannot be empty.
- The password must include at least one capital letter.
- The password must include at least one number.
- The minimum length of a password is 8 characters.
- The password cannot contain the user name, given name, or family name.

You can change these validation requirements, or include additional requirements, by configuring the policy for passwords.

Passwords are validated in several situations:

Password change and password reset

Password *change* refers to users changing their own passwords. Password *reset* refers to an administrator setting a user or account password on behalf of a user.

By default, IDM validates password values as they are provisioned.

Password recovery

Password recovery involves recovering a password or setting a new password when the password has been forgotten.

Password history

You can add validation to prevent reuse of previous password values. For more information, see ["Creating a Password History Policy"](#).

Password expiration

You can use workflows to ensure that users are able to change expiring passwords or to reset expired passwords.

Creating a Password History Policy

The sample described in *"Store Multiple Passwords For Managed Users"* in the *Samples Guide* shows how to set up a password history policy in a scenario where users have multiple different passwords across resources. You can use the scripts provided in that sample to set up a simple password history policy that prevents managed users from setting the same password that they used previously.

To create a password history policy based on the scripts in the multiple passwords sample, make the following changes to your project:

1. Copy the `pwpolicy.js` script from the multiple passwords sample to your project's `script` directory:

```
cp /path/to/openidm/samples/multiple-passwords/script/pwpolicy.js /path/to/openidm/my-project-dir/script/
```

The `pwpolicy.js` script contains an `is-new` policy definition that compares a new field value with the list of historical values for that field.

The `is-new` policy takes a `historyLength` parameter that specifies the number of historical values on which the policy should be enforced. This number must not exceed the `historySize` that you set in `conf/managed.json` to be passed to the `onCreate` and `onUpdate` scripts.

2. Copy the `onCreate-user-custom.js` and `onUpdate-user-custom.js` scripts to your project's `script` directory:

```
cp samples/multiple-passwords/script/onCreate-user-custom.js /my-project-dir/script/
cp samples/multiple-passwords/script/onUpdate-user-custom.js /my-project-dir/script/
```

These scripts validate the password history policy when a managed user is created or updated.

3. Update your policy configuration (`conf/policy.json`) to reference the new policy definition by adding the policy script to the `additionalFiles` array:

```
{
  "type" : "text/javascript",
  "file" : "policy.js",
  "additionalFiles": [ "script/pwpolicy.js" ],
  ...
}
```

4. Update your project's `conf/managed.json` file as follows:

- Add a `fieldHistory` property to the managed user object:

```
"fieldHistory" : {
  "title" : "Field History",
  "type" : "object",
  "viewable" : false,
  "searchable" : false,
  "userEditable" : false,
  "scope" : "private"
}
```

The value of this field is a map of field names to a list of historical values for that field. These lists of values are used by the `is-new` policy to determine if a new value has already been used.

- Update the managed user object to call the scripts when a user is created, updated:

```
"name" : "user",
"onCreate" : {
  "type" : "text/javascript",
  "file" : "script/onCreate-user-custom.js",
  "historyFields" : [
    "password"
  ],
  "historySize" : 4
},
"onUpdate" : {
  "type" : "text/javascript",
  "file" : "script/onUpdate-user-custom.js",
  "historyFields" : [
    "password"
  ],
  "historySize" : 4
},
...
```

Important

If you have any other script logic that is executed on these events, you must update the scripts to include that logic, or add the password history logic to your current scripts.

- Add the `is-new` policy to the list of policies enforced on the `password` property of a managed user. Specify the number of historical values that the policy should check in `historyLength` property:

```
"password" : {
  ...
  "policies" : [
    {
      "policyId" : "at-least-X-capitals",
      "params" : {
        "numCaps" : 1
      }
    },
    ...
    {
      "policyId" : "is-new",
      "params" : {
        "historyLength" : 4
      }
    },
    ...
  ]
}
```

You should now be able to test the password history policy by creating a new managed user, and having that user update their password. If the user specifies the same password used within the previous four passwords, the update request is denied with a policy error.

Storing Separate Passwords Per Linked Resource

You can store multiple passwords in a single managed user entry to enable synchronization of different passwords on different external resources.

To store multiple passwords, extend the managed user schema to include additional properties for each target resource. You can set separate policies on each of these new properties, to ensure that the stored passwords adhere to the password policies of the specific external resources.

To use this custom managed object property and its policies to update passwords on an external resource, you must make the corresponding configuration and script changes in your deployment. For a detailed sample that implements multiple passwords, see ["Store Multiple Passwords For Managed Users"](#) in the *Samples Guide*. That sample can also help you set up password history policies.

Generating Random Passwords

In certain situations, you might want to generate a random password when users are created.

You can customize your user creation logic to include a randomly generated password that complies with the default password policy. This functionality is included in the default crypto script, `bin/defaults/script/crypto.js`, but is not invoked by default. For an example of how this functionality might be used, see the `openidm/bin/defaults/script/onCreateUser.js` script. The following section of that file

(commented out by default) means that users created through the Admin UI, or directly over the REST interface, will have a randomly generated password added to their entry:

```
if (!object.password) {
    // generate random password that aligns with policy requirements
    object.password = require("crypto").generateRandomString([
        { "rule": "UPPERCASE", "minimum": 1 },
        { "rule": "LOWERCASE", "minimum": 1 },
        { "rule": "INTEGERS", "minimum": 1 },
        { "rule": "SPECIAL", "minimum": 1 }
    ], 16);
}
```

Note that changes made to scripts take effect after the time set in the `recompile.minimumInterval`, described in *"Script Configuration"* in the *Scripting Guide*.

The generated password can be encrypted or hashed, in accordance with the managed user schema, defined in `conf/managed.json`. For more information, see "Encoding Attribute Values". Note that synchronizing hashed passwords is not supported.

You can use this random string generation in a number of situations. Any script handler that is implemented in JavaScript can call the `generateRandomString` function.

Modifying the `password` Property

To use a property other than the default `password` property to store passwords, you must change the following files:

`policy.json`

If you want to enforce password validation rules on a different property, change the `password` property in this file.

`managed.json`

Modify the `password` object in this file, which also includes password complexity policies.

`sync.json`

If you change the `password` property, make sure that you limit the change to the appropriate system, designated as `source` or `target`.

`selfservice-reset.json`

If you are setting up self-service password reset, as described in *"Password Reset"* in the *Self-Service Reference*, change the value of `identityPasswordField` from `password` to the desired new property.

Every UI file that includes `password` as a property name

Whenever there's a way for a user to enter a password, the associated `HTML` page will include a password entry. For example, the `LoginTemplate.html` file includes the `password` property. A full list of default files with the `password` property include:

- `_passwordFields.html`
- `_resetPassword.html`
- `ConfirmPasswordDialogTemplate.html`
- `EditPasswordPageView.html`
- `LoginTemplate.html`
- `MandatoryPasswordChangeDialogTemplate.html`
- `resetStage-initial.html`
- `UserPasswordTab.html`

This list does not include any custom UI files that you might have created.

Rate Limiting Emails

No rate limiting is applied to password reset emails, or any emails sent by the IDM server. This means that an attacker can potentially spam a known user account with an infinite number of emails, filling that user's inbox. In the case of password reset, the spam attack can obscure an actual password reset attempt.

In a production environment, you must configure email rate limiting through the network infrastructure in which IDM runs. Configure the network infrastructure to detect and prevent frequent repeated requests to publicly accessible web pages, such as the password reset page. You can also handle rate limiting within your email server.

Chapter 5

Secure Network Connections

This chapter explains how to secure incoming connections and ports. As a general precaution in production environments, avoid communication over insecure HTTP.

- "Use TLS/SSL"
- "Restrict REST Access to the HTTPS Port"
- "Protect Sensitive REST Interface URLs"
- "Enable HTTP Strict-Transport-Security"
- "Restrict the HTTP Payload Size"
- "Deploy Securely Behind a Load Balancer"
- "Connect to IDM Through a Proxy Server"

Use TLS/SSL

Use TLS/SSL to access IDM, ideally with mutual authentication so that only trusted systems can invoke each other. TLS/SSL protects data on the network. Mutual authentication with strong certificates, imported into the truststore and keystore of each application, provides a level of confidence for trusting application access.

Restrict REST Access to the HTTPS Port

In a production environment, you should restrict REST access to a secure port. To do so, make the following changes to a default installation:

- Edit your project's `conf/jetty.xml` configuration file.

Comment out or delete the `<Call name="addConnector">` code block that includes the `openidm.port.http` property. Keep the `<Call name="addConnector">` code blocks that contain the `openidm.port.https` and `openidm.port.mutualauth` properties.

Set the `openidm.port.https` and `openidm.port.mutualauth` port numbers in the `resolver/boot.properties` file.

- Edit your project's `conf/config.properties` file.

Set the `org.osgi.service.http.enabled` property to false, as shown in the following excerpt:

```
# Enable pax web http/https services to enable jetty
org.osgi.service.http.enabled=false
org.osgi.service.http.secure.enabled=true
```

Use a certificate to secure REST access, over HTTPS. You can use self-signed certificates in a test environment. In production, all certificates should be signed by a certificate authority. The examples in this guide assume a CA-signed certificate named `ca-cert.pem`.

Protect Sensitive REST Interface URLs

Anything attached to the router is accessible with the default policy, including the repository. If you do not need such access, deny it in the authorization policy to reduce the attack surface.

In addition, you can deny direct HTTP access to system objects in production, particularly access to `action`. As a rule of thumb, do not expose anything that is not used in production.

For an example that shows how to protect sensitive URLs, see ["Configure Access Control in access.json"](#).

Enable HTTP Strict-Transport-Security

HTTP Strict-Transport-Security (HSTS) is a web security policy that forces browsers to make secure HTTPS connections to specified web applications. HSTS can protect websites against passive eavesdropper and active man-in-the-middle attacks.

IDM provides an HSTS configuration but it is disabled by default. To enable HSTS, locate the following excerpt in your `conf/jetty.xml` file:

```
<New id="tlsHttpConfig" class="org.eclipse.jetty.server.HttpConfiguration">
  ...
  <Call name="addCustomizer">
    <Arg>
      <New class="org.eclipse.jetty.server.SecureRequestCustomizer">
        <!-- Enable SNI Host Check when true -->
        <Arg name="sniHostCheck" type="boolean">true</Arg>
        <!-- Enable Strict-Transport-Security header and define max-age when >= 0 seconds -->
        <Arg name="stsMaxAgeSeconds" type="long">-1</Arg>
        <!-- If enabled, add includeSubDomains to Strict-Transport-Security header when true -->
        <Arg name="stsIncludeSubdomains" type="boolean">false</Arg>
      </New>
    </Arg>
  </Call>
  ...
```

Set the following arguments:

stsMaxAgeSeconds

This parameter sets the length of time, in seconds, that the browser should remember that a site can only be accessed using HTTPS.

For example, the following setting applies the HSTS policy and remains in effect for an hour:

```
<Arg name="stsMaxAgeSeconds" type="long">3600</Arg>
```

stsMaxAgeSeconds

If this parameter is `true`, the HSTS policy is applied to the domain of the issuing host as well as its subdomains:

```
<Arg name="stsIncludeSubdomains" type="boolean">true</Arg>
```

For more information about HSTS, read [this article](#).

Restrict the HTTP Payload Size

Restricting the size of HTTP payloads can protect the server against large payload HTTP DDoS attacks. IDM includes a servlet filter that limits the size of an incoming HTTP request payload, and returns a `413 Request Entity Too Large` response when the maximum payload size is exceeded.

By default, the maximum payload size is 5MB. You can configure the maximum size in your project's `conf/servletfilter-payload.json` file. That file has the following structure by default:

```
{
  "classPathURLs" : [ ],
  "systemProperties" : { },
  "requestAttributes" : { },
  "scriptExtensions" : { },
  "initParams" : {
    "maxRequestSizeInMegabytes" : 5
  },
  "urlPatterns" : [
    "/"
  ],
  "filterClass" : "org.forgerock.openidm.jetty.LargePayloadServletFilter"
}
```

Change the value of the `maxRequestSizeInMegabytes` property to set a different maximum HTTP payload size. The remaining properties in this file are described in "Register Additional Servlet Filters" in the *Installation Guide*.

Deploy Securely Behind a Load Balancer

IDM prevents URL-hijacking, with the following code block in the `conf/jetty.xml` file:

```
<Call name="addCustomizer">
  <Arg>
    <New class="org.eclipse.jetty.server.SecureRequestCustomizer">
      <!-- Enable SNI Host Check when true -->
      <Arg name="sniHostCheck" type="boolean">true</Arg>
      <!-- Enable Strict-Transport-Security header and define max-age when >= 0 seconds -->
      <Arg name="stsMaxAgeSeconds" type="long">-1</Arg>
      <!-- If enabled, add includeSubDomains to Strict-Transport-Security header when true -->
      <Arg name="stsIncludeSubdomains" type="boolean">>false</Arg>
    </New>
  </Arg>
</Call>
```

If you are deploying IDM behind a system such as a load balancer, firewall, or a reverse proxy, you must uncomment the next section in `jetty.xml`, so that Jetty honors `X-Forwarded-Host` headers:

```
<Call name="addCustomizer">
  <Arg>
    <New class="org.eclipse.jetty.server.ForwardedRequestCustomizer">
      <Set name="forcedHost">
        <Call class="org.forgerock.openidm.jetty.Param" name="getProperty">
          <Arg>openidm.host</Arg>
        </Call>:<Call class="org.forgerock.openidm.jetty.Param" name="getProperty">
          <Arg>openidm.port.https</Arg>
        </Call>
      </Set>
    </New>
  </Arg>
</Call>
```

Connect to IDM Through a Proxy Server

To configure IDM to communicate through a proxy server:

1. Add the following JVM parameters to the value of `OPENIDM_OPTS` in your startup script (`startup.sh` or `startup.bat`):

-Dhttps.proxyHost

Hostname or IP address of the proxy server; for example, `proxy.example.com` or `192.168.0.1`.

-Dhttps.proxyPort

Port number used by IDM; for example, `8443` or `9443`.

For example:

```
# Only set OPENIDM_OPTS if not already set
[ -z "$OPENIDM_OPTS" ] && OPENIDM_OPTS="-Xmx1024m -Xms1024m -Dhttps.proxyHost=localhost -Dhttps.proxyPort=8443"
```

2. Enable the `ForwardedRequestCustomizer` class so that Jetty honors `X-Forwarded-` headers.

To enable the class, uncomment the following excerpt in your `conf/jetty.xml` file:

```
<Call name="addCustomizer">
  <Arg>
    <New class="org.eclipse.jetty.server.ForwardedRequestCustomizer">
      <Set name="forcedHost">
        <Call class="org.forgerock.openidm.jetty.Param" name="getProperty">
          <Arg>openidm.host</Arg>
        </Call>:
        <Call class="org.forgerock.openidm.jetty.Param" name="getProperty">
          <Arg>openidm.port.https</Arg>
        </Call>
      </Set>
    </New>
  </Arg>
</Call>
```

For more information on this class, see the [Jetty documentation](#).

Chapter 6

Protect IDM Data

Beyond relying on end-to-end availability of TLS/SSL to protect data, IDM also supports explicit encryption of data that goes on the network. This can be important if the TLS/SSL termination happens prior to the final endpoint.

IDM also supports encryption of data stored in the repository, using the symmetric keys specified in `conf/secrets.json`. This protects against some attacks on the data store. Explicit table mapping is supported for encrypted string values.

IDM automatically encrypts sensitive data (such as passwords) in configuration files, and replaces clear text values when the system first reads the configuration file. Take care with configuration files that contain clear text values that IDM has not yet read and encrypted.

Encoding Attribute Values

There are two ways to encode attribute values for managed objects—reversible encryption and salted hashing algorithms. Attribute values that might be encoded include passwords, authentication questions, credit card numbers, and social security numbers. If passwords are already encoded on the external resource, they are generally excluded from the synchronization process. For more information, see "*Secure Passwords*".

You configure attribute value encoding, per schema property, in the managed object configuration (in your project's `conf/managed.json` file). The following sections show how to use reversible encryption and salted hash algorithms to encode attribute values.

Encoding Attribute Values With Reversible Encryption

The following excerpt of a `managed.json` file shows a managed object configuration that encrypts and decrypts the `password` attribute using the default symmetric key:

```
{
  "objects" : [
    {
      "name" : "user",
      ...
      "schema" : {
        ...
        "properties" : {
          ...
          "password" : {
            "title" : "Password",
            ...
            "encryption" : {
              "purpose" : "idm.password.encryption"
            },
            "scope" : "private",
          }
        }
      }
    }
  ]
}
```

To encrypt attribute values from the command-line, see "**encrypt**" in the *Setup Guide*.

+ To Configure Encryption Through the UI

1. Select Configure > Managed Objects, and select the object type whose property values you want to encrypt (for example User).
2. On the Properties tab, select the property whose value should be encrypted and select the Encrypt checkbox.

Encoding Attribute Values by Using Salted Hash Algorithms

To encode attribute values with salted hash algorithms, add the **secureHash** property to the attribute definition and define the hashing configuration. The configuration depends on the algorithm that you choose.

If you do not specify an algorithm, **SHA-256** is used by default. MD5 and SHA-1 are supported for legacy reasons but you should use a more secure algorithm in production environments.

The following list shows the supported hash algorithms and their configurations:

SHA-256

```
"secureHash" : {
  "algorithm" : "SHA-256",
  "saltLength" : 16
}
```

SHA-384

```
"secureHash" : {  
  "algorithm" : "SHA-384",  
  "saltLength" : 16  
}
```

SHA-512

```
"secureHash" : {  
  "algorithm" : "SHA-512",  
  "saltLength" : 16  
}
```

Bcrypt

```
"secureHash" : {  
  "algorithm" : "BCRYPT",  
  "cost" : 16  
}
```

Scrypt

```
"secureHash" : {  
  "algorithm" : "SCRYPT",  
  "hashLength" : 16,  
  "saltLength" : 16,  
  "n" : 32768,  
  "r" : 8,  
  "p" : 1  
}
```

Password-Based Key Derivation Function 2 (PBKDF2)

```
"secureHash" : {  
  "algorithm" : "PBKDF2",  
  "hashLength" : 16,  
  "saltLength" : 16,  
  "iterations" : 10,  
  "hmac" : "SHA-256"  
}
```

Warning

Some one-way hash functions are designed to be computationally *expensive*. Functions such as PBKDF2, Bcrypt, and Scrypt are designed to be relatively slow even on modern hardware. This makes them generally less susceptible to brute force attacks. *However*, computationally expensive functions can dramatically increase response times. If you use these functions, be aware of the performance impact and perform extensive testing before deploying your service in production. Do not use functions like PBKDF2 and Bcrypt for any accounts that are used for frequent, short-lived connections.

Hashing is a one-way operation, such that the original value cannot be recovered. Therefore, if you hash the value of any property, you cannot synchronize that property value to an external resource. For managed object

properties with hashed values, you must either exclude those properties from the mapping or set a random default value if the external resource requires the property.

The following excerpt of a managed object configuration shows that values of the `password` attribute are hashed using the `SHA-256` algorithm:

```
{
  "objects" : [
    {
      "name" : "user",
      ...
      "schema" : {
        ...
        "properties" : {
          ...
          "password" : {
            "title" : "Password",
            ...
            "secureHash" : {
              "algorithm" : "SHA-256"
            },
            "scope" : "private",
          }
        }
      }
    }
  ]
}
```

To hash attribute values from the command-line, see "`secureHash`" in the *Setup Guide*.

+ To Configure Hashing Through the UI

You can configure hashing of properties through the Admin UI, but the functionality is limited to setting the hash algorithm. Not all algorithms are supported in the UI, and none of the enhanced configuration options are supported. To configure attribute hashing in the UI:

1. Select **Configure > Managed Objects**, and select the object type whose property values you want to hash (for example, **User**).
2. On the **Properties** tab, select the property whose value must be hashed, select **Privacy & Encryption**, then select the **Hashed** checkbox.
3. Select the algorithm that should be used to hash the property value.

Structure of an Encrypted Object

Encrypted objects and properties, such as passwords, include a `$crypto` object, that has the following structure:

```
"password": {
  "$crypto": {
    "type": "x-simple-encryption",
    "value": {
      "cipher": "AES/CBC/PKCS5Padding",
      "stableId": "openidm-sym-default",
      "salt": "Gwi+AGrn+VB0Tmyq+TTuw==",
      "data": "+9i7XAXpwZBXYTVE0BkM+w==",
      "keySize": 16,
      "purpose": "idm.password.encryption",
      "iv": "4xtI88eFu5tgfm8ooq+yqQ==",
      "mac": "N1zsYo71M/b/G6iL0hNohA=="
    }
  }
}
```

Most of the properties in the encrypted object `value` are self-explanatory and indicate how the property was encrypted. Specific IDM properties include the following:

- The `stableId` indicates the key alias that was used to encrypt the property value.
- The `purpose` refers to the secret ID used to encrypt the property value. For more information about secret IDs, see ["Configuring Secret Stores"](#).

Encrypting and Decrypting Properties Over REST

The `openidm.encrypt` and `openidm.decrypt` functions of the Resource API enable you to encrypt and decrypt property values. To use these functions over the REST interface, run the `?_action=eval` action on the `script` endpoint.

The following example uses the `openidm.encrypt` function to encrypt a password value:


```
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 '{
  "type": "text/javascript",
  "globals": {
    "val": {
      "myKey": "myPassword"
    }
  },
  "source": "openidm.encrypt(val,null,\"idm.password.encryption\");"
}' \
"https://localhost:8443/openidm/script?action=eval"
{
  "$crypto": {
    "type": "x-simple-encryption",
    "value": {
      "cipher": "AES/CBC/PKCS5Padding",
      "stableId": "openidm-sym-default",
      "salt": "qAS/eG7zdnFyK5H8lXvqTA==",
      "data": "zewf6hRlyjp34EFJqUGpdnzzFCPJ52IaX4V97jdQlSI=",
      "keySize": 16,
      "purpose": "idm.password.encryption",
      "iv": "A4pIiY6kG6t0uLyLmJAoWQ==",
      "mac": "sFDJqg0Mmp0Ftl+1q1Bjzw=="
    }
  }
}
```

The following example uses the `openidm.decrypt` function to decrypt the password value:

```
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 '{
  "type": "text/javascript",
  "globals": {
    "val": {
      "$crypto": {
        "type": "x-simple-encryption",
        "value": {
          "cipher": "AES/CBC/PKCS5Padding",
          "stableId": "openidm-sym-default",
          "salt": "qAS/eG7zdnFyK5H8lXvqTA==",
          "data": "zewf6hR1yjp34EFJqUGpdnzzFCPJs2IaX4V97jdQlSI=",
          "keySize": 16,
          "purpose": "idm.password.encryption",
          "iv": "A4pIiY6kG6t0uLyLmJAoWQ==",
          "mac": "sFDJqg0Mmp0Ftl+1q1Bjzw=="
        }
      }
    }
  },
  "source": "openidm.decrypt(val);",
}' \
https://localhost:8443/openidm/script?_action=eval"
{
  "myKey": "myPassword"
}
```

For more information about the `openidm.encrypt` and `openidm.decrypt` functions, see `openidm.encrypt(value, cipher, alias)` and `openidm.decrypt(value)` in the *Scripting Guide*.

Securing the Repository

Configuration data and, in most deployments, user data, are stored in the IDM repository. In production deployments, you must secure access to the repository, and encrypt sensitive stored data.

For JDBC repositories, use a strong password for the connection to the repository and change at least the password of the database user (`openidm` by default). When you change the database username and/or password, update your database connection configuration file (`conf/datasource.jdbc-default.json`).

For a DS repository, change the `bindDN` and `bindPassword` for the directory server user in the `ldapConnectionFactory` property in the `repo.ds.json` file.

In both cases, the password is encrypted on server startup, using the key specified in the `idm.password.encryption` secret ID in `conf/secrets.json`.

Protecting Sensitive Files and Directories

Protect IDM files from access by unauthorized users. In particular, prevent other users from reading files in at least the `openidm/resolver/` and `openidm/security/` directories.

The objective is to limit access to the user that is running the service. Depending on the operating system and configuration, that user might be `root`, `Administrator`, `openidm`, or something similar.

Protecting Sensitive Files in Unix

1. Make sure that user and group ownership of the installation and project directories is limited to the user running the IDM service.
2. Disable access of any sort for `other` users. One simple command for that purpose, from the `/path/to/openidm` directory, is:

```
chmod -R o-rwx .
```

Protecting Sensitive Files in Windows

1. The IDM process in Windows is normally run by the `Local System` service account.
2. If you are concerned about the security of this account, you can set up a service account that only has permissions for IDM-related directories, then remove User access to the directories noted above. You should also configure the service account to deny local and remote login. For more information, see the [User Rights Assignment](#) article in Microsoft's documentation.

Removing or Protecting Development and Debug Tools

Before you deploy IDM in production, remove or protect development and debug tools, including the Felix web console that is exposed under `/system/console`. Authentication for this console is not integrated with authentication for IDM.

- To remove the Felix web console, remove the web console bundle and all of the plugin bundles related to the web console, as follows:

```
rm /path/to/openidm/bundle/org.apache.felix.webconsole*.jar
rm /path/to/openidm/bundle/openidm-felix-webconsole-7.0.4.jar
```

Also remove the `felix.webconsole.json` configuration file from your project's `conf` directory.

```
rm /path/to/project-dir/conf/felix.webconsole.json
```

- Alternatively, protect access to the Felix web console by changing the credentials in your project's `conf/felix.webconsole.json` file. This file contains the username and password to access the console, by default:

```
{
  "username" : "admin",
  "password" : "admin"
}
```

Adjusting Log Levels

In production, set log levels to **INFO** to ensure that you capture enough information to help diagnose issues, but do not expose unnecessary information. For more information, see *"Configure Server Logs"* in the *Monitoring Guide*.

At start up and shut down, **INFO** can produce many messages. During stable operation, **INFO** generally results in log messages only when coarse-grain operations such as scheduled reconciliation start or stop.

Important

The default IDM log formatter encodes all control characters (such as newline characters) using URL-encoding, to protect against log forgery. For more information, see *"Configure Server Logs"* in the *Monitoring Guide*.

Disabling the API Explorer

The *"REST API Explorer"* serves up REST API documentation. The API Explorer can help you identify endpoints, and run REST calls against those endpoints. To protect production servers from unwanted API descriptor requests, set the following property in your **resolver/boot.properties** file:

```
openidm.apidescriptor.enabled=false
```

Disabling Automatic Configuration Updates

By default, IDM polls the JSON files in the **conf** directory periodically for any changes to the configuration. In a production system, it is recommended that you disable automatic polling for updates to prevent untested configuration changes from disrupting your identity service.

To disable automatic polling for configuration changes, edit the **conf/system.properties** file for your project, and uncomment the following line:

```
# openidm.fileinstall.enabled=false
```

This setting also disables the file-based configuration view, which means that IDM reads its configuration only from the repository.

Important

Before you disable automatic polling, you must have started the server at least once to ensure that the configuration has been loaded into the repository. Be aware, if automatic polling is enabled, IDM immediately uses changes to scripts called from a JSON configuration file.

When your configuration is complete, you can disable writes to configuration files. To do so, add the following line to the `conf/config.properties` file for your project:

```
felix.fileinstall.enableConfigSave=false
```

Managing Privacy & Consent

As an end user, you might want to control what happens to your personal data. For IDM, that means control of how your data is shared with external systems. The example in "Marketo Connector" in the *Connectors Guide* shows how you can generate a marketing leads database, only for those users who have selected a specific preference. Also read "Configure Privacy and Consent" in the *Self-Service Reference*.

IDM allows you to regulate access to two different kinds of personal data:

- *User information*: while marketers want user information such as addresses and telephone numbers, IDM allows you to let individual users decide whether to share that data. For more information, see "Regulating HTTP Access to Personal Data".
- *Account information*: by default, IDM prevents REST-based access to passwords with the `private` scope, as defined in the `managed.json` file. You can extend this protection to other properties. For more information, see "Restricting HTTP Access to Sensitive Data".

To configure Privacy & Consent in the End User UI, see "Configure Privacy and Consent" in the *Self-Service Reference*.

Regulating HTTP Access to Personal Data

In some cases, you might want to allow users to choose whether to share their personal data. "Configure User Preferences" in the *Self-Service Reference* describes how to allow users to select basic preferences for updates and marketing. They can select these preferences when they register and in the End User UI.

Examine the `managed.json` file for your project. Every relevant property should include two settings that determine whether a user can choose to share or not share that property:

- `isPersonal`: When set to `true`, specifies personally identifying information. By default, the `isPersonal` option for `userName` and `postalAddress` is set to `true`.

`usageDescription`: Includes additional information that can help users understand the sensitivity of a specific property such as `telephoneNumber`.

The `consentedMappings` property in a managed user object enables the user to specify an array of mappings (target systems) with which they consent to sharing their identifying information. The following sample excerpt of the default managed user object schema shows the `consentedMappings` property definition:

```
"consentedMappings": {
  "title": "Consented Mappings",
  "description": "Consented Mappings",
  "type": "array",
  "viewable": false,
  "searchable": false,
  "userEditable": true,
  "usageDescription": "",
  "isPersonal": false,
  "items": {
    "type": "object",
    "title": "Consented Mapping",
    "properties": {
      "mapping": {
        "title": "Mapping",
        "description": "Mapping",
        "type": "string",
        "viewable": true,
        "searchable": true,
        "userEditable": true
      },
      "consentDate": {
        "title": "Consent Date",
        "description": "Consent Date",
        "type": "string",
        "viewable": true,
        "searchable": true,
        "userEditable": true
      }
    },
    "order": [
      "mapping",
      "consentDate"
    ],
    "required": [
      "mapping",
      "consentDate"
    ]
  },
  "returnByDefault": false,
  "isVirtual": false
}
```

Restricting HTTP Access to Sensitive Data

You can protect specific sensitive managed data by marking the corresponding properties as `private`. Private data, whether it is encrypted or not, is not accessible over the REST interface. Properties that are marked as private are removed from an object when that object is retrieved over REST.

To mark a property as private, set its `scope` to `private` in the `conf/managed.json` file.

The following extract of the `managed.json` file shows how HTTP access is prevented on the `password` property:

```
{
  "objects": [
    {
      "name": "user",
      "schema": {
        "id": "http://jsonschema.net",
        "title": "User",
        ...
        "properties": {
          ...
          "password": {
            "title": "Password",
            ...
            "encryption": {
              "purpose": "idm.password.encryption"
            },
            "scope": "private",
            ...
          }
        }
      }
    }
  ]
}
```

Tip

To configure private properties by using the Admin UI:

1. Select Configure > Managed Objects, and select the object type whose property values you want to make private (for example User).
2. On the Properties tab, select the property that must be private and select the Private checkbox.

A potential caveat relates to private properties. If you use an HTTP `GET` request, you won't even see private properties. Even if you know all relevant private properties, a `PUT` request would replace the entire object in the repository. In addition, that request would effectively remove all private properties from the object. To work around this limitation, use a `POST` request to update only those properties that require change.

For example, to update the `givenName` of user `jdoe`, you could run the following command:

```
$ 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 '[
{
  "operation": "replace",
  "field": "/givenName",
  "value": "Jon"
}
]' \
"https://localhost:8443/openidm/managed/user?_action=patch&_queryId=for-username&uid=jdoe"
```

Note

The filtering of private data applies only to direct HTTP read and query calls on managed objects. No automatic filtering is done for internal callers, and the data that these callers choose to expose.

Securing IDM Server Files With a Read-Only Installation

One method of locking down the server is to install IDM on a read-only file system.

This section assumes that you have prepared the read-only volume appropriate for your Linux/UNIX installation environment and that you have set up a regular Linux user named `idm` and a dedicated volume for the `/idm` directory.

Configure the dedicated volume device, `/dev/volume` in the `/etc/fstab` file, as follows:

```
/dev/volume /idm ext4 ro,defaults 1,2
```

When you run the **mount -a** command, the `/dev/volume` volume device is mounted on the `/idm` directory.

You can switch between read-write and read-only mode for the `/idm` volume with the following commands:

```
sudo mount -o remount,rw /idm
sudo mount -o remount,ro /idm
```

Confirm the result with the **mount** command, which should show that the `/idm` volume is mounted in read-only mode:

```
/dev/volume on /idm type ext4 (ro)
```

Set up the `/idm` volume in read-write mode:

```
sudo mount -o remount,rw /idm
```

With the following commands, you can unpack the IDM binary in the `/idm` directory, and give user `idm` ownership of all files in that directory:

```
sudo unzip /idm/IDM-7.0.4.zip
sudo chown -R idm.idm /idm
```

When you have installed IDM on a read-only file system, redirect audit and logging data to writable volumes. This procedure assumes a user `idm` with Linux administrative (superuser) privileges.

1. Create an external directory where IDM can send logging, auditing, and internal repository information:

```
sudo mkdir -p /var/log/openidm/audit
sudo mkdir /var/log/openidm/logs
sudo mkdir -p /var/cache/openidm/felix-cache
sudo mkdir /var/run/openidm
```


Alternatively, route audit data to a remote data store. For an example of how to send audit data to a MySQL repository, see *"Direct Audit Information To MySQL"* in the *Samples Guide*.

2. Give the `idm` user ownership of the newly created directories:

```
sudo chown -R idm.idm /var/log/openidm
sudo chown -R idm.idm /var/cache/openidm
sudo chown -R idm.idm /var/run/openidm
```

3. Modify the following configuration files:

conf/audit.json

Make sure the `handlerForQueries` is the JSON audit event handler and change the `logDirectory` property to the `/var/log/openidm/audit` subdirectory:

```
{
  "eventHandlers" : [
    {
      "class" : "org.forgerock.audit.handlers.json.JsonAuditEventHandler",
      "config" : {
        "name" : "json",
        "logDirectory" : "/var/log/openidm/audit",
        ...
      },
      ...
    }
  ]
}
```

conf/logging.properties

Change the `java.util.logging.FileHandler.pattern` property as follows:

```
java.util.logging.FileHandler.pattern = /var/log/openidm/logs/openidm%u.log
```

conf/config.properties

Activate and redirect the `org.osgi.framework.storage` property as follows:

```
# If this value is not absolute, then the felix.cache.rootdir controls
# how the absolute location is calculated. (See buildNext property)
org.osgi.framework.storage={felix.cache.rootdir|{user.dir}}/felix-cache

# The following property is used to convert a relative bundle cache
# location into an absolute one by specifying the root to prepend to
# the relative cache path. The default for this property is the
# current working directory.
felix.cache.rootdir=/var/cache/openidm
```

Note

You might want to set up additional redirection for the following:

- Connectors. Depending on the connector, and the read-only volume, consider configuring connectors to direct output to writable volumes.

- Scripts. If you are using Groovy, examine the `conf/script.json` file for your project. Make sure that output such as to the `groovy.target.directory` is directed to an appropriate location, such as `idm.data.dir`.

Adjust the value of the `OPENIDM_PID_FILE` in the `startup.sh` and `shutdown.sh` scripts.

For RHEL 6 and Ubuntu 14.04 systems, the default shell is bash. You can set the value of `OPENIDM_PID_FILE` for user `idm` by adding the following line to `/home/idm/.bashrc`:

```
export OPENIDM_PID_FILE=/var/run/openidm/openidm.pid
```

If you have set up a different command line shell, adjust your changes accordingly.

When you log in again as user `idm`, your `OPENIDM_PID_FILE` variable should redirect the process identifier file, `openidm.pid` to the `/var/run/openidm` directory, ready for access by the `shutdown.sh` script.

While the volume is still mounted in read-write mode, start IDM normally:

```
./startup.sh -p project-dir
```

The first startup of IDM either processes the signed certificate that you added, or generates a self-signed certificate, and encrypts any passwords in the various configuration files.

Stop IDM if it is running.

You can now mount the `/idm` directory in read-only mode. The configuration in `/etc/fstab` ensures that Linux mounts the `/idm` directory in read-only mode the next time that system is booted.

```
sudo mount -o remount,ro /idm
```

You can now start IDM, configured on a secure read-only volume.

```
./startup.sh -p project-dir
```

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

Static User Module

Authentication Property	Property as Listed in the Admin UI	Description
<code>enabled</code>	Module Enabled	Does IDM use the module?
<code>queryOnResource</code>	Query on Resource	Endpoint hard coded to user <code>anonymous</code>
<code>username</code>	Static User Name	Default for the static user, <code>anonymous</code>
<code>password</code>	Static User Password	Default for the static user, <code>anonymous</code>
<code>defaultUserRoles</code>	Static User Role	Normally set to <code>openidm-reg</code> 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
<code>enabled</code>	Module Enabled	Does IDM use the module?
<code>queryOnResource</code>	Query on Resource	Endpoint to query
<code>queryId</code>	Use Query ID	A defined <code>queryId</code> searches against the <code>queryOnResource</code> endpoint. An undefined <code>queryId</code> against <code>queryOnResource</code> with <code>action=reauthenticate</code>
<code>defaultUserRoles</code>	Default User Roles	Normally blank for managed users
<code>authenticationId</code>	Authentication ID	Defines how account credentials are derived from a <code>queryOnResource</code> endpoint
<code>userCredential</code>	User Credential	Defines how account credentials are derived from a <code>queryOnResource</code> endpoint; if required, typically <code>password</code> or <code>userPassword</code>
<code>userRoles</code>	User Roles	Defines how account roles are derived from a <code>queryOnResource</code> endpoint

Authentication Property	Property as Listed in the Admin UI	Description
<code>groupMembership</code>	Group Membership	Provides more information for calculated roles
<code>groupRoleMapping</code>	Group Role Mapping	Provides more information for calculated roles
<code>groupComparisonMethod</code>	Group Comparison Method	Provides more information for calculated roles
<code>managedUserLink</code>	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 <code>managedUserLink</code> might be <code>systemLdapAccounts_managedUser</code>
<code>augmentSecurityContext</code>	Augment Security Context	Includes a script that is executed only after a successful authentication request. For more information on this property, see "Authenticating as a Different User".
<code>servicePrincipal</code>	Kerberos Service Principal	(IWA only) For more information, see "IWA"
<code>keytabFileName</code>	Keytab File Name	(IWA only) For more information, see "IWA"
<code>kerberosRealm</code>	Kerberos Realm	(IWA only) For more information, see "IWA"
<code>kerberosServerName</code>	Kerberos Server Name	(IWA only) For more information, see "IWA"