User Self Service Guide

/ ForgeRock Access Management 5.5

Latest update: 5.5.2

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

Abstract

Guide to configuring and using ForgeRock® Access Management User Self Service features.

Table of Contents

Preface

The User Self Service Guide shows you how to configure, maintain, and troubleshoot the User Self Service feature provided by ForgeRock Access Management, which automates account registration and account name retrieval, and forgotten password reset.

This guide is written for access management designers, developers, and administrators who build, deploy, and maintain services and features for their organizations.

About ForgeRock Identity Platform™ Software

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.

Chapter 1

Introducing User Self Service

AM provides a user self service feature that enables your customers to self-register to your web site, securely reset forgotten passwords, and retrieve their usernames.

AM's user self service capabilities greatly reduces help desk costs and offers a rich online experience that strengthens customer loyalty.

Note

The Password Reset service, located in the AM console at Configure > Global Services, is deprecated.

1.1. About User Self Service

AM's user self service feature supports automated account registration for new users, forgotten password reset, and forgotten username retrieval for your existing customer base. The user self service features include the following capabilities:

- **User Self Registration**. Allows non-authenticated users to register to your site on their own. You can add additional security features like email verification, knowledge-based authentication (KBA) security questions, Google reCAPTCHA, and custom plugins to add to your User Self Registration process.
- **Knowledge-based authentication security questions**. Supports the capability to present security questions during the registration process. When enabled, the user is prompted to enter answers to pre-configured or custom security questions. Then, during the forgotten password or forgotten username process, the user is presented with the security questions and must answer them correctly to continue the process.
- **Forgotten password reset**. Allows registered users already in your system to reset their passwords. The default password policy is set in the underlying directory server and requires a minimum password length of eight characters by default. If security questions are enabled, users must also correctly answer their pre-configured security questions before resetting their passwords.
- **Forgotten username support**. Allows users to retrieve their forgotten usernames. If security questions are enabled, users must also correctly answer their pre-configured security questions before retrieving their usernames.
- **Google reCAPTCHA plugin**. Supports the ability to add a Google reCAPTCHA plugin to the registration page. This plug-in protects against any software bots that may be used against your site.

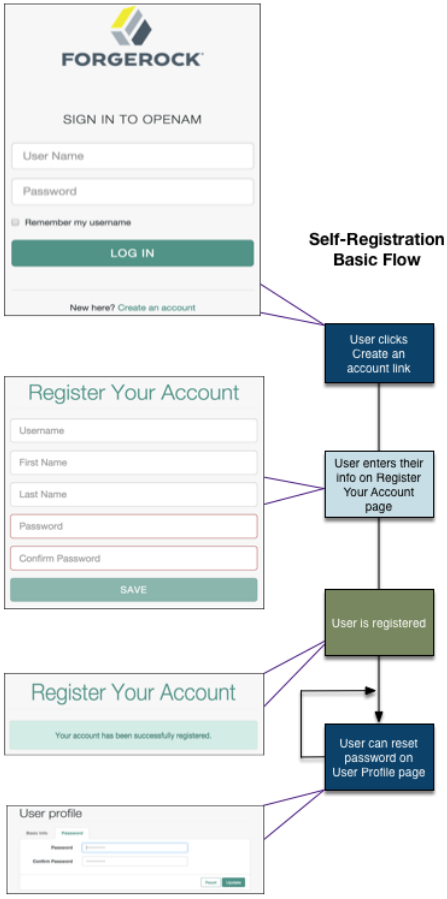
- **Configurable plugins**. Supports the ability to add plugins to customize the user services process flow. You can develop your custom code and drop the `.jar` file into your container.
- **Customizable confirmation emails**. Supports the ability to customize or localize confirmation email in plain text or HTML.
- **Password policy configuration**. Supports password policy configuration, which is enforced by the underlying DS server and manually aligned with frontend UI templates. The default password policy requires a password with a minimum length of eight characters.
- **Self registration user attribute whitelist**. Supports attribute whitelisting, which allows you to specify which attributes can be set by the user during account creation.

1.2. User Self Service Process Flows

The user self service feature supports a number of different user flows depending on how you configure your security options. These options include email verification, security questions, Google reCAPTCHA, and any custom plugins that you create.

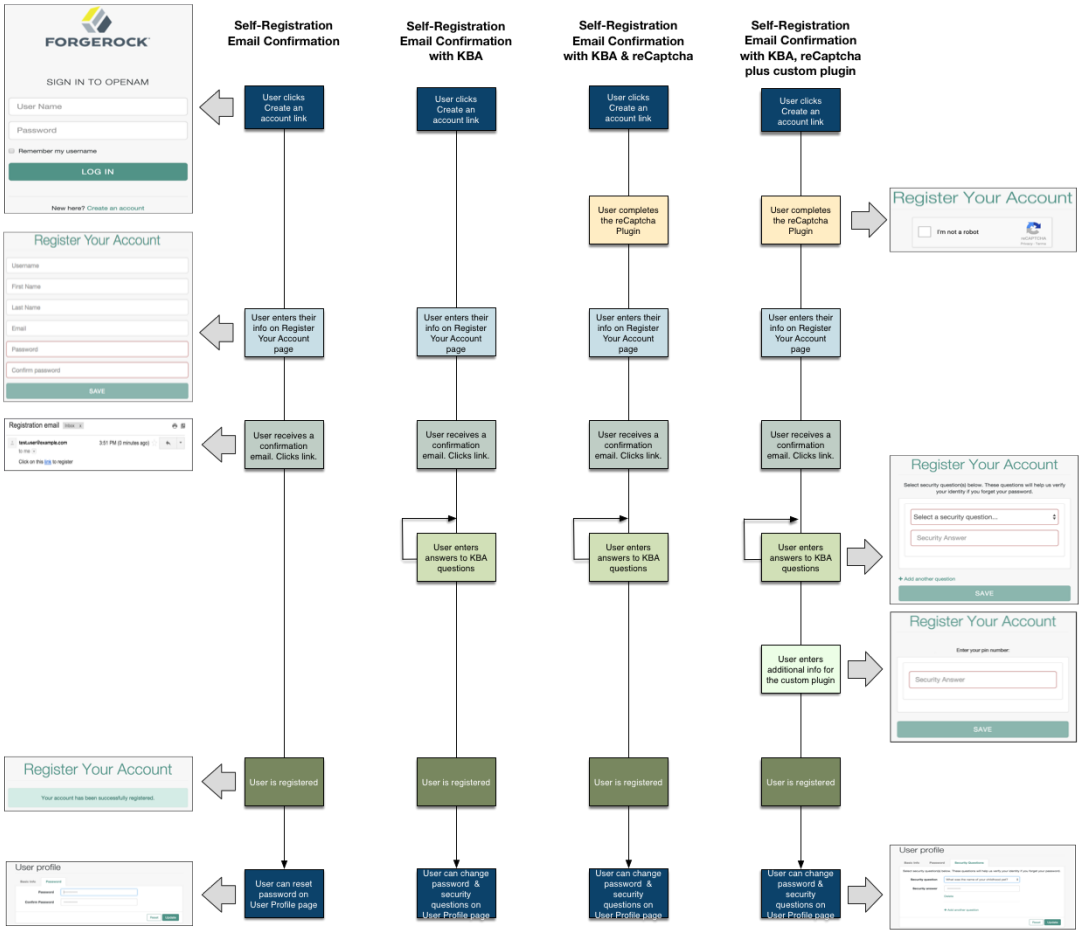
The following diagram shows the basic User Self Registration flow without the optional features:

User Self Registration Basic Flow



The following diagrams show the possible flows for User Self Registration flow with the optional features:

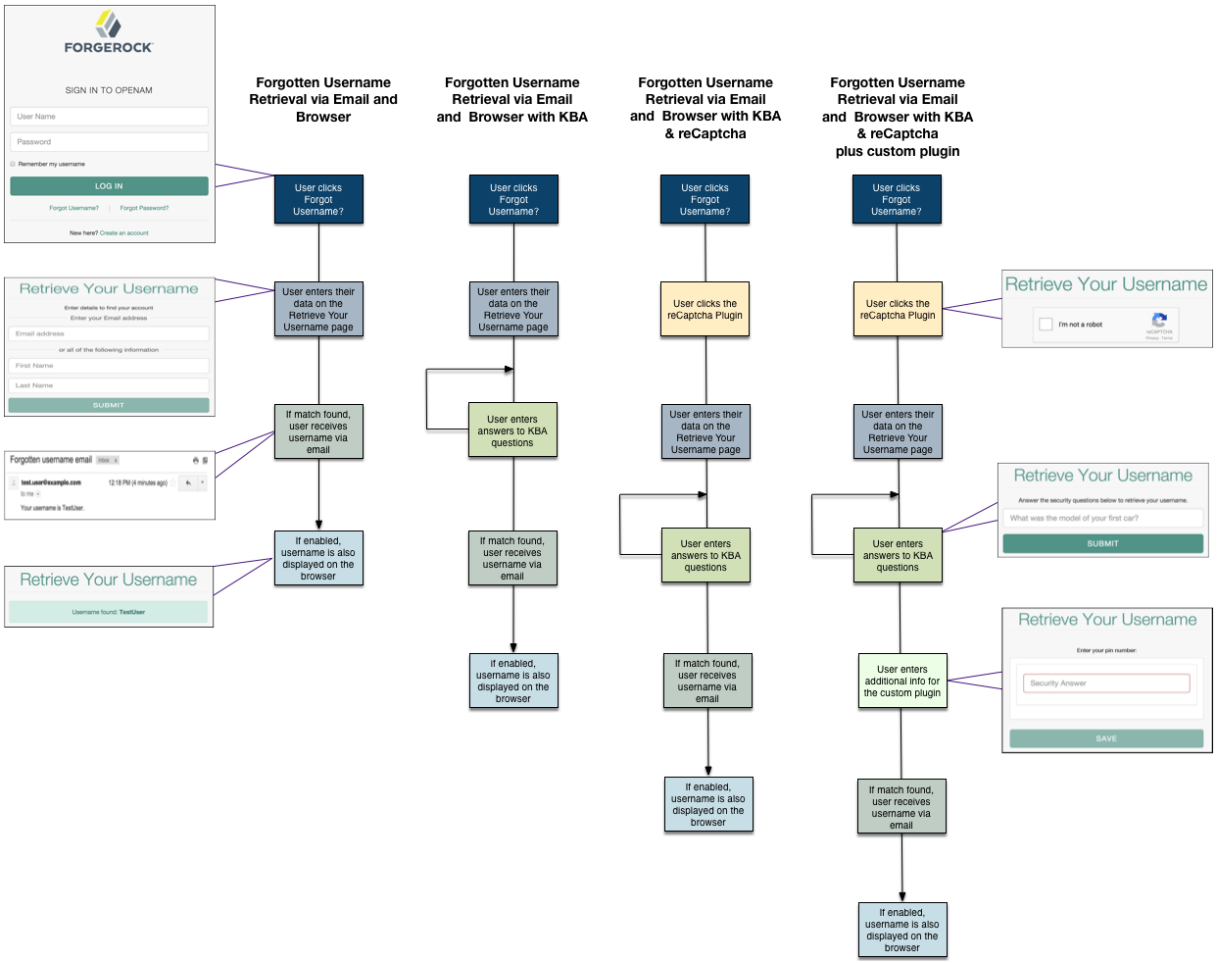
User Self Registration Flow With Options



Forgotten username retrieval and forgotten password reset support various user flows depending on how you configure your security options. If you enabled security questions and the user entered responses to each question during self-registration, the security questions are presented to the user in random order.

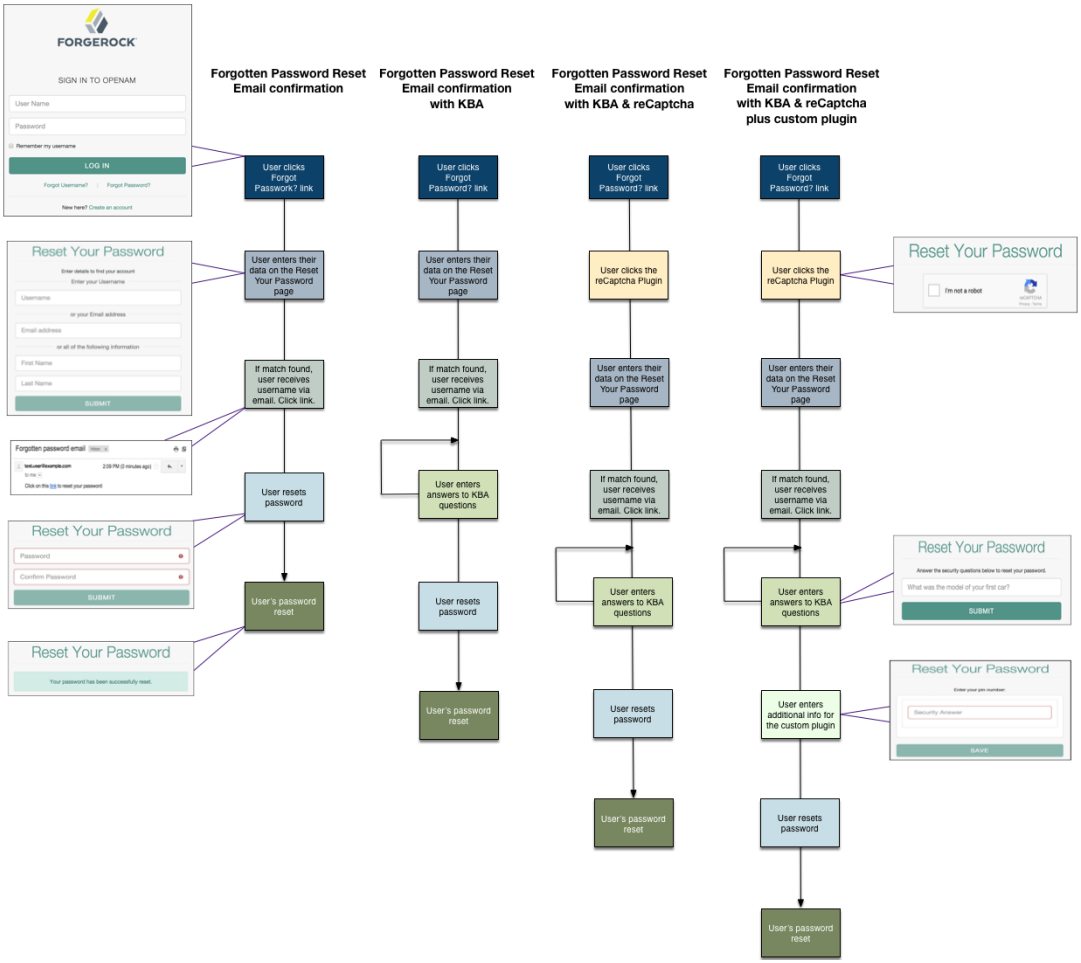
The following diagram shows the possible flows for forgotten username:

Forgotten Username Flow



The following diagram shows the possible flows for forgotten password reset:

Forgotten Password Flow



Chapter 2

Implementing User Self Service

You can configure the user self-service features to use email address verification, which sends an email containing a link for user self-registration and forgotten password reset via AM's Email Service. You can also send the forgotten username to the user by email if configured.

Follow the sections and procedures on this chapter to implement the user self-service features using the AM console. For information on how to use the RESTful API functionality, see "RESTful User Self Service".

Creating a User Self-Service Service Instance

1. Log in to the AM console as an administrator, for example, `amadmin`.
2. Navigate to Realms > *Realm Name* > Services and select Add a Service.
3. Select User Self-Service from the list of possible services.
4. Populate the values of the Encryption Key Pair Alias and the Signing Secret Key Alias properties with the names of the key pair aliases in your JCEKS keystore. Note that the name of the demo keys shows with a gray color; that does not mean the fields are filled in.

   For example, if you are using the demo keys in the default `keystore.jceks` file, set the properties as follows:

   - Encryption Key Pair Alias to `selfserviceenctest`.
   - Signing Secret Key Alias to `selfservicesigntest`.

   Note

   The demo key aliases are for test or evaluation purposes. Do not use them in production environments. To create new key aliases, see "Changing User Self-Service Key Aliases" in the *Setup and Maintenance Guide*.

5. (Optional) Enable the user self-service features.
6. Select Create.

To configure the different user self-service features, follow the steps in the sections below:

- "Configuring the Email Service"

- "Configuring the Google reCAPTCHA Plugin"
- "Configuring Knowledge-Based Security Questions"
- "Configuring User Registration"
- "Configuring the Forgotten Password Reset Feature"
- "Configuring the Forgotten Username Feature"

2.1. Configuring the Email Service

The user self-service feature lets you send confirmation emails via AM's SMTP Email Service to users who are registering at your site or resetting forgotten passwords. If you choose to send confirmation emails, you can configure the Email Service by realm or globally.

Important

Each individual user must have a unique email address to use the email features of user self-service.

To Configure the Email Service

Perform the following steps to configure the Email Service:

1. Log in to the AM console as the administrator.
2. Navigate to Realms > *Realm Name* > Services.
3. Select Add a Service and choose Email Service from the list of available services.
4. Configure the Email Service:
    a. In the Mail Server Host Name field, enter the hostname of the mail server. If you are using the Google SMTP server, you must also configure the Google Mail settings to enable access for less secure applications.
    b. In the Mail Server Authentication Username field, enter the username to authenticate to the mail server. If you are testing on a Google account, you can enter a known Gmail address.
    c. In the Mail Server Authentication Password field, enter the password corresponding to the username used to authenticate to the mail server.
    d. In the Email From Address field, enter the email address from which to send the email notifications. For example, `no-reply@example.com`.
    e. Select Create.
    f. (Optional) Configure additional properties in the Email Service as needed.

For more information about the different configuration properties, see "Email Service" in the *Reference*.

2.2. Configuring the Google reCAPTCHA Plugin

The user self-service feature supports the Google reCAPTCHA plugin, which can be placed on the Register Your Account, Reset Your Password, and Retrieve Your Username pages. The Google reCAPTCHA plugin protects your user self-service implementation from software bots.

Note

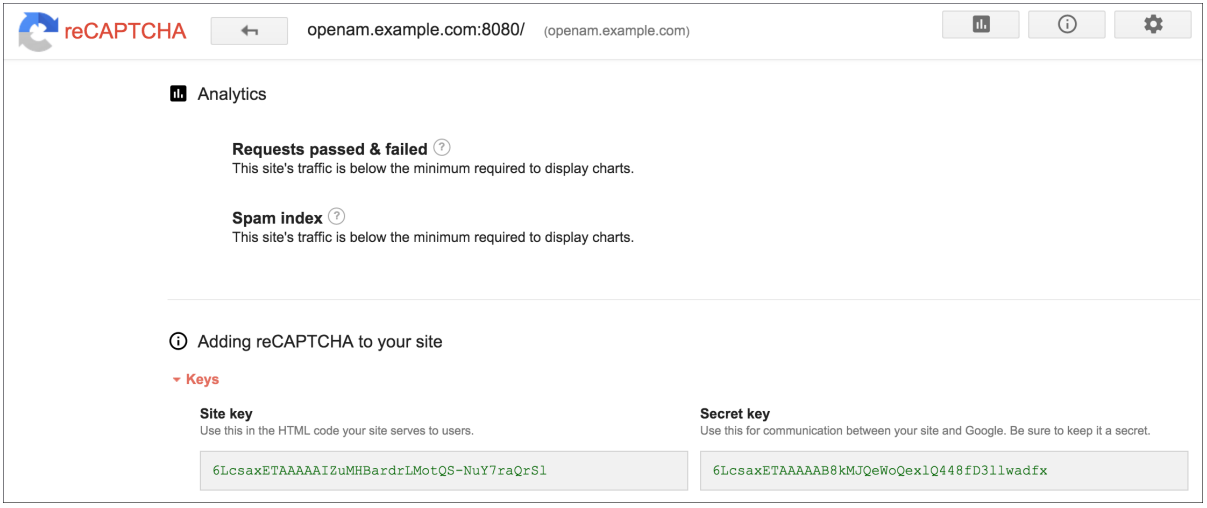
Google reCAPTCHA is the only supported plugin for user self-service. Any other Captcha service will require a custom plugin.

To Configure the Google reCAPTCHA Plugin

1. Register your web site at a Captcha provider, such as Google reCAPTCHA, to get your site and secret key.

   When you register your site for Google reCAPTCHA, you only need to obtain the site and secret key, which you enter in the User Self Service configuration page in the AM console. You do not have to do anything with client-side integration and server-side integration. The Google reCAPTCHA plugin appears automatically on the Register Your Account, Reset Your Password, and Retrieve Your Username pages after you configure it in the AM console.

Google reCAPTCHA Page



2. Log in to the AM console as an administrator.
3. Navigate to Realms > *Realm Name* > Services and select the User Self-Service service.
4. Select the General Configuration tab.
5. In the Google reCAPTCHA Site Key field, enter the site key that you obtained from the Google reCAPTCHA site.
6. In the Google reCAPTCHA Secret Key field, enter the secret key that you obtained from the Google reCAPTCHA site.
7. In the Google reCAPCHA Verification URL field, leave the URL by default.
8. Save your changes.

2.3. Configuring Knowledge-Based Security Questions

Knowledge-based authentication (KBA) is an authentication mechanism in which the user must correctly answer a number of pre-configured security questions that are set during the initial registration setup. If successful, the user is granted the privilege to carry out an action, such as registering an account, resetting a password, or retrieving a username. The security questions are presented in a random order to the user during the User Self Registration, forgotten password reset, and forgotten username processes.

AM provides a default set of security questions and easily allows AM administrators and users to add their own custom questions.

To Configure Security Questions

1. Log in to the AM console as the administrator.
2. Navigate to Realms > *Realm Name* > Services and select the User Self-Service service.
3. Select the General Configuration tab.
4. In the Security Questions field, several questions are available by default. Enter your own questions as required. The syntax is `OrderNum|ISO-3166-2 Country Code|Security Question`. For example, `5|en|What is your dog's name?`. Make sure that order numbers are unique.

   Warning

   You should never remove any security questions as a user may have reference to a given question.

5. In the Minimum Answers to Define field, enter the number of security questions that will be presented to the user during the registration process.

6. In the Minimum Answers to Verify field, enter the number of security questions that must be answered during the Forgotten Password and Forgotten Username services.
7. Save your changes.

2.4. Configuring User Registration

AM provides self-registration features that allow users to create an account.

You can configure AM to perform user registration, or you can delegate user registration to IDM, depending on your requirements.

This section covers configuring AM to:

- **Perform user registration itself.** See "To Configure User Self Registration by AM".
- **Delegate user registration to IDM.** See "To Delegate User Self Registration to IDM".

To Configure User Self Registration by AM

Although you can configure user self registration without any additional security mechanisms, such as email verification or KBA security questions, we recommend configuring the email verification service with user self registration at a minimum.

1. Log in to the AM console as the administrator.
2. Configure the Email Service as described in "Configuring the Email Service".
3. Navigate to Realms > *Realm Name* > Services and select the User Self-Service service.
4. Select the User Registration tab.
5. Enable User Registration.
6. Enable Capcha to turn on the Google reCAPTCHA plugin. Make sure you configured the plugin as described in "Configuring the Google reCAPTCHA Plugin".
7. Enable Email Verification to turn on the email verification service. We recommend you leave Email Verification enabled, so users who self-register must perform email address verification.
8. Enable Verify Email before User Detail to verify the user's email address before requesting the user details.

   By default, the user self-registration flow validates the email address after the user has provided their details. Enable this setting for backwards-compatibility with self-registration flows configured in OpenAM 13 or 13.5.
9. Enable Security Questions to display security questions to the user during the self registration, after which the user must enter their answers to the questions. During the forgotten password or

forgotten username services, the user will be presented with the security questions to be able to reset their passwords or retrieve their usernames if Security Questions is enabled.

10. In the Token LifeTime field, enter an appropriate number of seconds for the token lifetime. If the token lifetime expires before the user self-registers, then the user will need to restart the registration process over again.

    Default: `900` seconds.

11. To customize the User Registration outgoing email, perform the following steps:

    a. In the Outgoing Email Subject field, enter the Subject line of the email.

       The syntax is `lang|subject-text`, where `lang` is the ISO-639 language code, such as `en` for English, `fr` for French, and others. For example, the subject line values could be: **`en|Registration Email`** and **`fr|Inscription E-mail`**.

    b. In the Outgoing Email Body field, enter the text of the email.

       The syntax is `lang|email-text`, where `lang` is the ISO-639 language code. Note that email body text must be all on one line and can contain any HTML tags within the body of the text.

       For example, the email body text could be: **`en|Thank you for registration to our site! Click <a href="%link%">here</a> to register to the site.`**

12. In the Valid Creation Attributes field, enter the user attributes the user can be set during user registration. The attributes are based on the AM identity repository.

13. For Destination After Successful Registration, select one of the following options:

    - **auto-login**. User is automatically logged in and sent to the appropriate page within the system.
    - **default**. User is sent to a success page without being logged in. In this case, AM displays a "You have successfully registered" page. The user can then click the Login link to log in to AM. This is the default selection.
    - **login**. User is sent to the login page to authenticate.

14. Save your changes.

15. Under the Advanced Configuration tab, configure the User Registration Confirmation Email URL for your deployment. The default is: `https://openam.example.com:8443/openam/XUI/?realm=${realm}#register/`.

16. Save your changes.

To Delegate User Self Registration to IDM

IDM offers user self-registration functionality, much like AM, but provides additional onboarding and provisioning features.

You can configure the IDM Provisioning service to allow IDM to complete user registration after authenticating to AM using a social identity authentication module, for example.

1. Verify the following pre-requisites:

   a. The AM and IDM instances are connected to the same user data store.

      For more information, see Reconciliation and AM in the *ForgeRock Identity Management 5.5 Samples Guide*.

   b. The AM instance has a copy of the signing and encryption keys from the IDM installation in its default keystore.

      Follow the instructions in "Copying Key Aliases" in the *Setup and Maintenance Guide* to copy the following key aliases to the AM default keystore:

      - `openidm-selfservice-key`
      - `selfservice`

      Restart AM when completed to apply the changes.

2. Log in to the AM console as the administrator, and navigate to Configure > Global Services > IDM Provisioning.

3. Perform the following actions on the IDM Provisioning page:

   a. Enable the IDM Provisioning service by selecting Enabled.

   b. Enter the URL of the IDM instance in the Deployment URL field, for example `https://openidm.example.com`.

   c. (Optional) If you created new signing or encryption keys, enter their details, ensuring the keys are identical and available in the default keystores of both AM and IDM.

      For more information on IDM security, see Securing and Hardening Servers in the *ForgeRock Identity Management 5.5 Integrator's Guide*.

      If you have copied the `openidm-selfservice-key` and `selfservice` key alises from IDM, you can leave the default values for the key-related properties.

   For more details of the available properties, see "IDM Provisioning".

4. Save your changes.

5. In the AM console, navigate to Realms > *Realm Name* > Authentication > Modules, and create or select a social authentication module in which to enable IDM user registration.

6. On the social authentication module page, perform the following actions on the Account Provisioning tab:

a. Select Use IDM as Registration Service.

b. Ensure Create account if it does not exist is enabled.

7. Save your changes.

   Successfully authenticating to a social authentication module that has IDM as the registration service redirects the user to IDM to complete the user registration.

   For a complete walkthrough, including details on configuring IDM to perform user self registration, see Integrating IDM With the ForgeRock Identity Platform in the *ForgeRock Identity Management 5.5 Samples Guide*.

2.5. Configuring the Forgotten Password Reset Feature

The forgotten password feature allows existing users to reset their passwords when they cannot remember them.

To Configure the Forgotten Password Feature

1. Log in to the AM console as the administrator.
2. Navigate to Realms > *Realm Name* > Services and select the User Self-Service service.
3. Select the Forgotten Password tab.
4. Enable Forgotten Password.
5. Enable Captcha to turn on the Google reCAPTCHA plugin. Make sure you configured the plugin as described in "Configuring the Google reCAPTCHA Plugin".
6. Enable Email Verification to For Email Verification to turn on the email verification service. We recommend that you keep it selected.
7. Enable Security Questions to display security questions to the user during the forgotten password reset process. The user must correctly answer the security questions to be able to reset passwords.
8. In the Token LifeTime field, enter an appropriate number of seconds for the token lifetime. If the token lifetime expires before the user resets their password, then the user will need to restart the forgotten password process over again.

   Default: `900` seconds.

9. To customize the Forgotten Password outgoing email, perform the following steps:

   a. In the Outgoing Email Subject field, enter the subject line of the email.

The syntax is `lang|subject-text`, where `lang` is the ISO-639 language code, such as `en` for English, `fr` for French, and others. For example, the subject line value could be: **`en|Forgotten Password Email`**.

b. In the Outgoing Email Body field, enter the text of the email.

The syntax is `lang|email-text`, where `lang` is the ISO-639 language code. Note that email body text must be all on one line and can contain any HTML tags within the body of the text.

For example, the email body text could be: **`en|Thank you for request! Click <a href="%link %">here</a> to reset your password.`**

10. Save your changes.
11. Under the Advanced Configuration tab, change the default Forgotten Password Confirmation Email URL for your deployment. The default is: `https://openam.example.com:8443/openam/XUI/?realm=${realm}#passwordReset/`.
12. Save your changes.

2.6. Configuring the Forgotten Username Feature

The forgotten username feature allows existing users to retrieve their usernames when they cannot remember them.

To Configure the Forgotten Username Feature

1. Log in to the AM console as the administrator.
2. Navigate to Realms > *Realm Name* > Services and select the User Self-Service service.
3. Select the Forgotten Password tab.
4. Enable Forgotten Username.
5. Enable Captcha to turn on the Google reCAPTCHA plugin. Make sure you configured the plugin as described in "Configuring the Google reCAPTCHA Plugin".
6. Enable Security Questions to display security questions to the user during the forgotten password reset process. The user must correctly answer the security questions to be able to reset passwords.
7. Enable Email Username for the user to receive the retrieved username by email.
8. Enable Show Username for the user to see their retrieved username on the browser.
9. In the Token LifeTime field, enter an appropriate number of seconds for the token lifetime. If the token lifetime expires before the user resets their password, then the user will need to restart the forgotten password process over again.

Default: `900` seconds.

10. To customize the Forgotten Username outgoing email, perform the following steps:

    a. In the Outgoing Email Subject field, enter the subject line of the email.

    The syntax is `lang|subject-text`, where `lang` is the ISO 639 language code, such as `en` for English, `fr` for French, and others. For example, the subject line value could be: **`en|Forgotten username email`**.

    b. In the Outgoing Email Body field, enter the text of the email.

    The syntax is `lang|email-text`, where `lang` is the ISO 639 language code. Note that email body text must be all on one line and can contain any HTML tags within the body of the text.

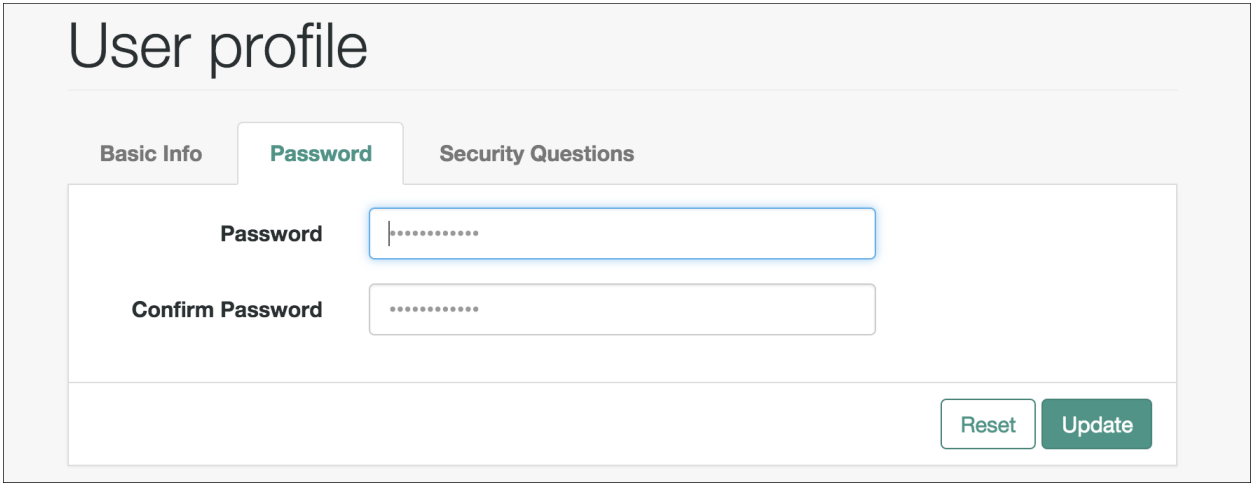
    For example, the email body text could be: `en|Thank you for your inquiry! Your username is %username%.`

11. Save your changes.

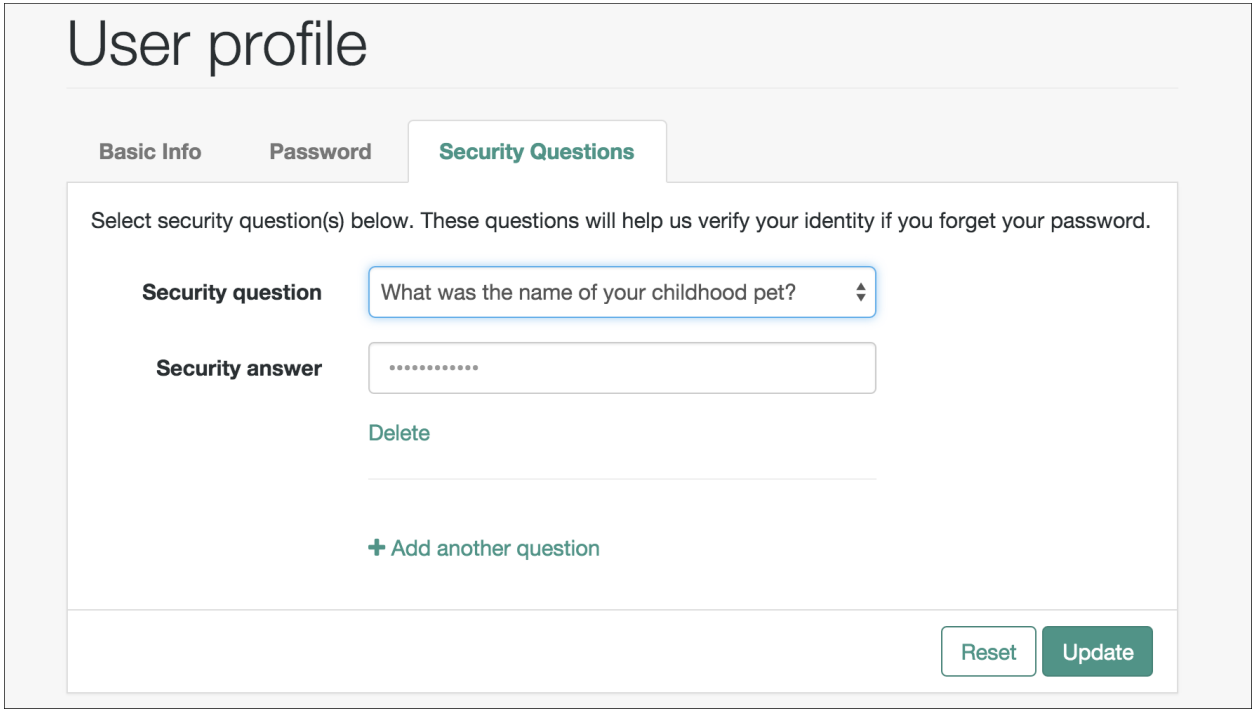
2.7. User Management of Passwords and Security Questions

Once the user has self-registered to your system, the user can change their password and security questions at any time on the user profile page. The user profile page provides tabs to carry out these functions.

User Profile Page Password Tab



User Profile Page Security Questions Tab



Chapter 3

Using User Self Service

This chapter covers client interaction using AM APIs over supported protocols for use with the user self service feature.

3.1. RESTful User Self Service

This section shows how to use the AM RESTful interfaces for user self service functionality: User Self Registration, forgotten password reset, forgotten username retrieval, dashboard configuration, and device profile reset.

The steps to perform user self service via the REST APIs varies depending on the configured User Self Service process flow. For more information, see "User Self Service Process Flows".

When performing user self service functions, you can enable one or more security methods, such as email validation, Google reCAPTCHA, knowledge-based authentication, or custom plugins. Each configured security method requires requests to be sent from AM to the client, and completed responses returned to AM to verify.

Important

At least one security method should be enabled for each user self service feature.

A unique token is provided in the second request to the client that must be used in any subsequent responses, so that AM can maintain the state of the user self service process.

In this section, long URLs are wrapped to fit the printed page, and some of the output is formatted for easier reading.

3.2. Registering Users

This section explains how to use the REST APIs for registering a user in AM.

By default, the user self-registration flow validates the email address after the user has provided their details. AM also provides a backwards-compatible mode for user self-registration flows configured in OpenAM 13 and 13.5 that allows AM to validate the email address before the user has provided their details.

- For information about registering users when AM requires the user details before validating the email address, see "To Register a User with the REST APIs".
- For information about registering users when AM validates the email address before requiring the user details, see "To Register a User with the REST APIs (Backwards-Compatible Mode)".

To Register a User with the REST APIs

Before performing the steps in this procedure, ensure that Verify Email before User Detail (Realms > *Realm Name* > Services > User Self-Service > User Registration) is *disabled*.

1. Create a GET request to the `/selfservice/userRegistration` endpoint. Notice that the request does not require any form of authentication.

```
$ curl \
https://openam.example.com:8443/openam/json/realms/root/selfservice/userRegistration
{
   "requirements": {
     "$schema": "http://json-schema.org/draft-04/schema#",
     "description": "New user details",
     "properties": {
        "user": {
            "description": "User details",
            "type": "object"
        }
     },
     "required": [
        "user"
     ],
     "type": "object"
   },
   "tag": "initial",
   "type": "userDetails"
}
```

   AM sends a request to complete the user details. The `required` array defines the data that must be returned to AM to progress past this step of the registration. In the example, the required type is a `user` object that contains the user details.

2. Create a POST response back to the `/selfservice/userRegistration` endpoint with a query string containing `_action=submitRequirements`. In the POST data, include an `input` element in the JSON structure, which should contain values for each element in the `required` array of the request.

   In this example, AM requests an object named `user`. Ths object should contain values for the `username`, `givenName`, `sn`, `mail`, `userPassword`, and `inetUserStatus` properties.

```
$ curl \
--request POST
 \
--header "Content-Type: application/json"
 \
--data \
'{
    "input": {
```

```
       "user": {
         "username": "DEMO",
         "givenName": "Demo User",
         "sn": "User",
         "mail":"demo@example.com",
         "userPassword": "forgerock",
         "inetUserStatus": "Active"
       }
    }
}' \
 https://openam.example.com:8443/openam/json/realms/root/selfservice/userRegistration?
_action=submitRequirements
{
    "requirements": {
      "$schema": "http://json-schema.org/draft-04/schema#",
      "description": "Verify emailed code",
      "properties": {
         "code": {
             "description": "Enter code emailed",
             "type": "string"
         }
      },
      "required": [
          "code"
      ],
      "type": "object"
    },
    "tag": "validateCode",
    "token": "eyJ0eXAiOiJKV.....QiLCJjmqrlqUfQ",
    "type": "emailValidation"
 }
```

If the response is accepted, AM continues with the registration process and sends the next request for information.

The value of the `token` element should be included in this and any subsequent responses to AM for this registration; AM uses this information to track which stage of the registration process is being completed.

Note that the request for information is of the type `emailValidation`. Other possible types include:

- `captcha`, if the Google reCAPTCHA plugin is enabled
- `kbaSecurityAnswerDefinitionStage`, if knowledge-based security questions are required

For an example of Google reCAPTCHA validation, see "Retrieving Forgotten Usernames".

3. Return the information required by the next step of the registration, along with the `token` element.

   In this example, the user information was accepted and a code was emailed to the email address. AM requires this code in the response in an element named `code` before continuing:

```
$ curl \
--request POST
 \
--header "Content-Type: application/json"
 \
--data \
  '{
     "input": {
       "code": "cf53fcb6-3bf2-44eb-a437-885296899699"
     },
     "token": "eyJ0eXAiOiJKV.....QiLCJjmqrlqUfQ"
  }' https://openam.example.com:8443/openam/json/realms/root/selfservice/userRegistration\
 ?_action=submitRequirements

{
  "type": "selfRegistration",
  "tag": "end",
  "status": {
      "success": true
  },
  "additions": {}
}
```

When the process is complete, the response from AM has a `tag` property with value of `end`. If the `success` property in the `status` object has a value of `true`, then self-registration is complete and the user account was created.

In the example, AM only required email verification to register a new user. In flows containing Google reCAPTCHA validation or knowledge-based security questions, you would continue returning POST data to AM containing the requested information until the process is complete.

To Register a User with the REST APIs (Backwards-Compatible Mode)

Before performing the steps in this procedure, ensure that Verify Email before User Detail (Realms > *Realm Name* > Services > User Self-Service > User Registration) is *enabled*.

1. Create a GET request to the `/selfservice/userRegistration` endpoint. Notice that the request does not require any form of authentication.

```
$ curl \
--header "Accept-API-Version: resource=1.0, protocol=1.0" \
https://openam.example.com:8443/openam/json/realms/root/selfservice/userRegistration
{
   "type":"emailValidation",
   "tag":"initial",
   "requirements":{
      "$schema":"http://json-schema.org/draft-04/schema#",
      "description":"Verify your email address",
      "type":"object",
      "required":[
         "mail"
      ],
      "properties":{
         "mail":{
            "description":"Email address",
            "type":"string"
         }
      }
   }
}
```

AM sends the first request for security information. In this example, the first request is of type `emailValidation`, but other types include `captcha` if the Google reCAPTCHA plugin is enabled, and `kbaSecurityAnswerDefinitionStage` if knowledge-based authentication is required.

The `required` array defines the data that must be returned to AM to progress past this step of the registration.

The `properties` element contains additional information about the required response, such as a description of the required field, or the site key required to generate a reCAPTCHA challenge.

2. Create a POST response back to the `/selfservice/userRegistration` endpoint with a query string containing `_action=submitRequirements`. In the POST data, include an `input` element in the JSON structure, which should contain values for each element in the `required` array of the request.

   In this example, a `mail` value was requested.

```
$ curl \
--request POST
 \
--header "Content-Type: application/json"
 \
--header "Accept-API-Version: resource=1.0, protocol=1.0"
 \
--data \
'{
    "input": {
        "mail": "demo.user@example.com"
    }
}' \
https://openam.example.com:8443/openam/json/selfservice/userRegistration
\
?_action=submitRequirements
{
   "type":"emailValidation",
   "tag":"validateCode",
   "requirements":{
      "$schema":"http://json-schema.org/draft-04/schema#",
      "description":"Verify emailed code",
      "type":"object",
      "required":[
         "code"
      ],
      "properties":{
         "code":{
            "description":"Enter code emailed",
            "type":"string"
         }
      }
   },
   "token":"eyAicHis...PIF-lN4s"
}
```

If the response was accepted, AM continues with the registration process and sends the next request for information. In this example, the email address was accepted and a code was emailed to the address, which AM requires in the response in an element named `code` before continuing.

The value of the `token` element should be included in this and any subsequent responses to AM for this registration.

3. Continue returning POST data to AM containing the requested information, in the format specified in the request. Also return the `token` value in the POST data, so that AM can track which stage of the registration process is being completed.

```
$ curl \
--request POST
 \
--header "Content-Type: application/json"
 \
--header "Accept-API-Version: resource=1.0, protocol=1.0"
 \
--data \
'{
```

```
    "input": {
        "code": "cf53fcb6-3bf2-44eb-a437-885296899699"
    },
    "token": "eyAicHis...PIF-lN4s"
}' \
https://openam.example.com:8443/openam/json/selfservice/userRegistration
\
?_action=submitRequirements
{
   "type":"userDetails",
   "tag":"initial",
   "requirements":{
      "$schema":"http://json-schema.org/draft-04/schema#",
      "description":"New user details",
      "type":"object",
      "required":[
         "user"
      ],
      "properties":{
         "user":{
            "description":"User details",
            "type":"object"
         }
      }
   },
   "token":"eyAicHis...PIF-lN4s"
}
```

4. When requested—when the `type` value in the request is `userDetails`—supply the details of the new user as an object in the POST data.

```
$ curl \
--request POST
 \
--header "Content-Type: application/json"
 \
--header "Accept-API-Version: resource=1.0, protocol=1.0"
 \
--data \
'{
    "input": {
        "user": {
            "username": "demo",
            "givenName": "Demo User",
            "sn": "User",
            "userPassword": "d3m0",
            "inetUserStatus": "Active"
        }
    },
    "token": "eyAicHis...PIF-lN4s"
}' \
https://openam.example.com:8443/openam/json/selfservice/userRegistration
\
?_action=submitRequirements
{
     "type": "selfRegistration",
     "tag": "end",
     "status": {
     "success": true
     },
     "additions": {}
}
```

When the process is complete, the `tag` element has a value of `end`. If the `success` element in the `status` element has a value of `true`, then self-registration is complete and the user account was created.

The User Self-Service feature provides options to set the user's destination after a successful self-registration. These options include redirecting the user to a 'successful registration' page, to the login page, or automaticatically logging the user into the system. Use the `Destination After Successful Self-Registration` property to set the option (on the console: *Realm Name* > Services > User Self-Service > User Registration). When you select `User sent to 'successful registration' page` or `User sent to login page`, the JSON response after a successful registration is as follows:

```
{
   "type": "selfRegistration",
   "tag": "end",
   "status": {
       "success": true
   },
   "additions": {}
}
```

If you select **User is automatically logged in**, the JSON response is:

```
{
    "type": "autoLoginStage",
    "tag": "end",
    "status": {
        "success": true
    },
    "additions": {
        "tokenId": "AQIC5...MQAA*",
        "successUrl": "/openam/console"
    }
}
```

3.3. Retrieving Forgotten Usernames

This section explains how to use the REST APIs to retrieve a forgotten username.

To Retrieve a Forgotten Username with the REST APIs

1. Create a GET request to the `/selfservice/forgottenUsername` endpoint. Notice that the request does not require any form of authentication.

```
$ curl \
https://openam.example.com:8443/openam/json/realms/root/selfservice/forgottenUsername
{
  "type": "captcha",
  "tag": "initial",
  "requirements": {
     "$schema": "http://json-schema.org/draft-04/schema#",
     "description": "Captcha stage",
     "type": "object",
     "required": [
       "response"
     ],
     "properties": {
       "response": {
         "recaptchaSiteKey": "6Lfr1...cIqbd",
         "description": "Captcha response",
         "type": "string"
       }
      }
  }
}
```

   In this example, the Google reCAPTCHA plugin is enabled, so the first request is of the `captcha` type.

2. Create a POST response back to the `/selfservice/forgottenUsername` endpoint with a query string containing `_action=submitRequirements`. In the POST data, include an `input` element in the JSON structure, which should contain values for each element in the `required` array of the request.

In this example, a `response` value was requested, which should be the user input as provided after completing the Google reCAPTCHA challenge.

```
$ curl \
 --request POST \
 --header "Content-Type: application/json" \
 --data \
   '{
      "input": {
        "response": "03AHJ...qiE1x4"
      }
   }' https://openam.example.com:8443/openam/json/realms/root/selfservice/forgottenUsername\
      ?_action=submitRequirements
   {
     "type": "userQuery",
     "tag": "initial",
     "requirements": {
       "$schema": "http://json-schema.org/draft-04/schema#",
       "description": "Find your account",
       "type": "object",
       "required": [
         "queryFilter"
       ],
      "properties": {
        "queryFilter": {
          "description": "filter string to find account",
          "type": "string"
        }
      }
   },
   "token": "eyAicHis...PIF-lN4s"
 }
```

If the response was accepted, AM continues with the username retrieval process and sends the next request for information. In this example, the Google reCAPTCHA was verified and AM is requesting details about the account name to retrieve, which must be provided in a `queryFilter` element.

The value of the `token` element should be included in this and all subsequent responses to AM for this retrieval process.

3. Create a POST response to AM with a `queryFilter` value in the POST data containing the user's email address associated with their account.

   For more information on query filters, see "Query" in the *Development Guide*.

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--data \
  '{
     "input": {
       "queryFilter": "mail eq \"demo.user@example.com\""
```

```
     },
     "token": "eyAicHis...PIF-lN4s"
  }' https://openam.example.com:8443/openam/json/realms/root/selfservice/forgottenUsername\
     ?_action=submitRequirements
  {
    "type": "kbaSecurityAnswerVerificationStage",
    "tag": "initial",
    "requirements": {
      "$schema": "http://json-schema.org/draft-04/schema#",
      "description": "Answer security questions",
      "type": "object",
      "required": [
        "answer1"
      ],
    "properties": {
      "answer1": {
        "systemQuestion": {
          "en": "What was the model of your first car?"
        },
      "type": "string"
      }
    }
  },
  "token": "eyAicHis...PIF-lN4s"
}
```

If a single subject is located that matches the provided query filter, the retrieval process continues.

If KBA is enabled, AM requests answers to the configured number of KBA questions, as in this example.

If a subject is not found, an HTTP 400 Bad Request status is returned, and an error message in the JSON data:

```
{
 "code": 400,
 "reason": "Bad Request",
 "message": "Unable to find account"
}
```

4. Return a POST response with the answers as values of the elements specified in the `required` array, in this example `answer1`. Ensure the same `token` value is sent with each response.

```
$ curl \
--request POST \
--header "Content-Type: application/json" \
--data \
  '{
     "input": {
       "answer1": "Mustang"
     },
     "token": "eyAicHis...PIF-lN4s"
  }' \
  https://openam.example.com:8443/openam/json/realms/root/selfservice/forgottenUsername\
      ?_action=submitRequirements
  {
     "type": "retrieveUsername",
     "tag": "end",
     "status": {
       "success": true
     },
     "additions": {
       "userName": "demo"
     }
  }
```

When the process is complete, the `tag` element has a value of `end`. If the `success` element in the `status` element has a value of `true`, then username retrieval is complete and the username is emailed to the registered address.

If the Show Username option is enabled for username retrieval, the username retrieved is also returned in the JSON response as the value of the `userName` element, as in the example above.

3.4. Replacing Forgotten Passwords

This section explains how to use the REST APIs to replace a forgotten password.

To Replace a Forgotten Password with the REST APIs

1. Create a GET request to the `/selfservice/forgottenPassword` endpoint. Notice that the request does not require any form of authentication.

```
$ curl \
  https://openam.example.com:8443/openam/json/realms/root/selfservice/forgottenPassword
{
  "type": "captcha",
  "tag": "initial",
  "requirements": {
    "$schema": "http://json-schema.org/draft-04/schema#",
    "description": "Captcha stage",
    "type": "object",
    "required": [
      "response"
    ],
    "properties": {
      "response": {
        "recaptchaSiteKey": "6Lfr1...cIqbd",
        "description": "Captcha response",
        "type": "string"
      }
    }
  }
}
```

In this example the Google reCAPTCHA plugin is enabled, so the first request is of the `captcha` type.

2. Create a POST response back to the `/selfservice/forgottenPassword` endpoint with a query string containing `_action=submitRequirements`. In the POST data, include an `input` element in the JSON structure, which should contain values for each element in the `required` array of the request.

   In this example, a `response` value was requested, which should be the user input as provided after completing the Google reCAPTCHA challenge.

```
$ curl \
--request POST
 \
--header "Content-Type: application/json"
 \
--data \
  '{
     "input": {
       "response": "03AHJ...qiE1x4"
     }
  }' https://openam.example.com:8443/openam/json/realms/root/selfservice/forgottenPassword\
      ?_action=submitRequirements
{
  "type": "userQuery",
  "tag": "initial",
  "requirements": {
    "$schema": "http://json-schema.org/draft-04/schema#",
    "description": "Find your account",
    "type": "object",
    "required": [
      "queryFilter"
    ],
    "properties": {
      "queryFilter": {
        "description": "filter string to find account",
        "type": "string"
      }
    }
  },
  "token": "eyAicHis...PIF-lN4s"
}
```

If the response was accepted, AM continues with the password reset process and sends the next request for information. In this example, the Google reCAPTCHA was verified and AM is requesting details about the account with the password to replace, which must be provided in a `queryFilter` element.

The value of the `token` element should be included in this and all subsequent responses to AM for this reset process.

3. Create a POST response to AM with a `queryFilter` value in the POST data containing the username of the subject with the password to replace.

   For more information on query filters, see "Query" in the *Development Guide*.

```
$ curl \
--request POST
 \
--header "Content-Type: application/json"
 \
--data \
  '{
     "input": {
       "queryFilter": "uid eq \"demo\""
     },
     "token": "eyAicHis...PIF-lN4s"
```

```
  }' https://openam.example.com:8443/openam/json/realms/root/selfservice/forgottenPassword\
      ?_action=submitRequirements
{
  "type": "kbaSecurityAnswerVerificationStage",
  "tag": "initial",
  "requirements": {
     "$schema": "http://json-schema.org/draft-04/schema#",
     "description": "Answer security questions",
     "type": "object",
     "required": [
       "answer1"
     ],
     "properties": {
       "answer1": {
         "systemQuestion": {
           "en": "What was the model of your first car?"
         },
         "type": "string"
       }
     }
  },
  "token": "eyAicHis...PIF-lN4s"
}
```

If a single subject is located that matches the provided query filter, the password reset process continues.

If a subject is not found, an HTTP 400 Bad Request status is returned, and an error message in the JSON data:

```
{
  "code": 400,
  "reason": "Bad Request",
  "message": "Unable to find account"
}
```

4. Continue returning POST data to AM containing the requested information, in the format specified in the request. Also return the `token` value in the POST data, so that AM can track which stage of the password reset process is being completed.

```
$ curl \
--request POST
 \
--header "Content-Type: application/json"
 \
--data \
  '{
     "input": {
       "answer1": "Mustang"
     },
     "token": "eyAicHis...PIF-lN4s"
  }' https://openam.example.com:8443/openam/json/realms/root/selfservice/forgottenPassword\
      ?_action=submitRequirements
{
  "type": "resetStage",
  "tag": "initial",
  "requirements": {
     "$schema": "http://json-schema.org/draft-04/schema#",
     "description": "Reset password",
     "type": "object",
     "required": [
        "password"
     ],
     "properties": {
        "password": {
          "description": "Password",
          "type": "string"
        }
     }
  },
  "token": "eyAicHis...PIF-lN4s"
}
```

5. When requested—when the `type` value in the request is `resetStage`—supply the new password in the POST data.

```
$ curl \
--request POST
 \
--header "Content-Type: application/json"
 \
--data \
  '{
      "input": {
        "password": "User1234"
      },
      "token": "eyAicHis...PIF-lN4s"
  }' \
  https://openam.example.com:8443/openam/json/realms/root/selfservice/forgottenPassword\
    ?_action=submitRequirements
{
  "type": "resetStage",
  "tag": "end",
  "status": {
     "success": true
  },
  "additions": {}
}
```

When the process is complete, the `tag` element has a value of `end`. If the `success` element in the `status` element has a value of `true`, then password reset is complete and the new password is now active.

If the password is not accepted, an HTTP 400 Bad Request status is returned, and an error message in the JSON data:

```
{
  "code": 400,
  "reason": "Bad Request",
  "message": "Minimum password length is 8."
}
```

Chapter 4

User Self Service Reference

This chapter covers AM configuration properties for the user self service feature, which is accessible through the Configure tab of the AM console, most of which can also be set by using the **ssoadm** command. The chapter is organized to follow the AM console layout.

4.1. User Self-Service

amster service name: `selfService`

4.1.1. General Configuration

The following settings appear on the General Configuration tab:

Encryption Key Pair Alias

An encryption key alias in the OpenAM server's JCEKS keystore. Used to encrypt the JWT token that OpenAM uses to track end users during User Self-Service operations.

For example, you might set this property to: *selfserviceenctest*

amster attribute: `encryptionKeyPairAlias`

Signing Secret Key Alias

A signing secret key alias in the OpenAM server's JCEKS keystore. Used to sign the JWT token that OpenAM uses to track end users during User Self-Service operations.

For example, you might set this property to: *selfservicesigntest*

amster attribute: `signingSecretKeyAlias`

Google reCAPTCHA Site Key

Google reCAPTCHA plugin site key.

amster attribute: `captchaSiteKey`

Google reCAPTCHA Secret Key

Google reCAPTCHA plugin secret key.

amster attribute: `captchaSecretKey`

Google Re-captcha Verification URL

Google reCAPTCHA plugin verification URL.

Default value: `https://www.google.com/recaptcha/api/siteverify`

amster attribute: `captchaVerificationUrl`

Security Questions

Specifies the default set of knowledge-based authentication (KBA) security questions. The security questions can be set for the User Self-Registration, forgotten password reset, and forgotten username services, respectively.

Format is `unique key|locale|question`.

Default value:

```
4|en|What is your mother's maiden name?
3|en|What was the name of your childhood pet?
2|en|What was the model of your first car?
1|en|What is the name of your favourite restaurant?
```

amster attribute: `kbaQuestions`

Minimum Answers to Define

Specifies the minimum number of KBA answers that users must define.

Default value: `1`

amster attribute: `minimumAnswersToDefine`

Minimum Answers to Verify

Specifies the minimum number of KBA questions that users need to answer to be granted the privilege to carry out an action, such as registering for an account, resetting a password, or retrieving a username. Specify a value from `0` to `50`.

Default value: `1`

amster attribute: `minimumAnswersToVerify`

Valid Query Attributes

Specifies the valid query attributes used to search for the user. This is a list of attributes used to identify your account for forgotten password and forgotten username.

Default value:

```
uid
mail
givenName
sn
```

amster attribute: `validQueryAttributes`

4.1.2. User Registration

The following settings appear on the User Registration tab:

User Registration

If enabled, new users can sign up for an account.

Default value: `false`

amster attribute: `userRegistrationEnabled`

Captcha

If enabled, users must pass a Google reCAPTCHA challenge during user self-registration to mitigate against software bots.

Default value: `false`

amster attribute: `userRegistrationCaptchaEnabled`

Email Verification

If enabled, users who self-register must perform email address verification.

Default value: `true`

amster attribute: `userRegistrationEmailVerificationEnabled`

Security Questions

If enabled, users must set up their security questions during the self-registration process.

Default value: `false`

amster attribute: `userRegistrationKbaEnabled`

Token Lifetime (seconds)

Maximum lifetime of the token allowing User Self-Registration, in seconds.

Default value: `900`

amster attribute: `userRegistrationTokenTTL`

Outgoing Email Subject

Customize the User Self-Registration verification email subject text. Format is `locale|subject text`.

Default value: `en|Registration email`

amster attribute: `userRegistrationEmailSubject`

Outgoing Email Body

Customize the User Self-Registration verification email body text. Format is: `locale|body text`.

Default value: `en|<h2>Click on this <a href="%link%">link</a> to register.</h2>`

amster attribute: `userRegistrationEmailBody`

Valid Creation Attributes

Specifies a whitelist of user attributes that can be set during user creation.

Default value:

```
userPassword
mail
kbaInfo
givenName
inetUserStatus
sn
username
```

amster attribute: `userRegistrationValidUserAttributes`

Destination After Successful Self-Registration

Specifies the action to be taken after a user successfully registers a new account. Choose from:

- `default`. User is sent to a success page without being logged in.
- `login`. User is sent to the login page to authenticate.
- `autologin`. User is automatically logged in and sent to the appropriate page.

The possible values for this property are:

```
User sent to 'successful registration' page
User sent to login page
User is automatically logged in
```

Default value: `default`

amster attribute: `userRegisteredDestination`

4.1.3. Forgotten Password

The following settings appear on the Forgotten Password tab:

Forgotten Password

If enabled, users can reset their forgotten password.

Default value: `false`

amster attribute: `forgottenPasswordEnabled`

Captcha

If enabled, users must pass a Google reCAPTCHA challenge during password reset to mitigate against software bots.

Default value: `false`

amster attribute: `forgottenPasswordCaptchaEnabled`

Email Verification

If enabled, users who reset passwords must perform email address verification.

Default value: `true`

amster attribute: `forgottenPasswordEmailVerificationEnabled`

Security Questions

If enabled, users must answer their security questions during the forgotten password process.

Default value: `false`

amster attribute: `forgottenPasswordKbaEnabled`

Token Lifetime (seconds)

Maximum lifetime for the token allowing forgotten password reset, in seconds.

Specify a value from `0` to `2147483647`.

Default value: `900`

amster attribute: `forgottenPasswordTokenTTL`

Outgoing Email Subject

Customize the forgotten password email subject text. Format is `locale|subject text`.

Default value: `en|Forgotten password email`

amster attribute: `forgottenPasswordEmailSubject`

Outgoing Email Body

Customize the forgotten password email body text. Format is `locale|body text`.

Default value: `en|<h2>Click on this <a href="%link%">link</a> to reset your password.</h2>`

amster attribute: `forgottenPasswordEmailBody`

4.1.4. Forgotten Username

The following settings appear on the Forgotten Username tab:

Forgotten Username

If enabled, users can retrieve their forgotten username.

Default value: `false`

amster attribute: `forgottenUsernameEnabled`

Captcha

If enabled, users must pass a Google reCAPTCHA challenge during the forgotten username retrieval process to mitigate against software bots.

Default value: `false`

amster attribute: `forgottenUsernameCaptchaEnabled`

Security Questions

If enabled, users must answer their security questions during the forgotten username process.

Default value: `false`

amster attribute: `forgottenUsernameKbaEnabled`

Email Username

If enabled, users receive their forgotten username by email.

Default value: `true`

amster attribute: `forgottenUsernameEmailUsernameEnabled`

Show Username

If enabled, users see their forgotten username on the browser page.

Default value: `false`

amster attribute: `forgottenUsernameShowUsernameEnabled`

Token LifeTime (seconds)

Maximum lifetime for the token allowing forgotten username, in seconds.

Default value: `900`

amster attribute: `forgottenUsernameTokenTTL`

Outgoing Email Subject

Customizes the forgotten username email subject text. Format is `locale|subject text`.

Default value: `en|Forgotten username email`

amster attribute: `forgottenUsernameEmailSubject`

Outgoing Email Body

Customizes the forgotten username email body text. Format is `locale|body text`.

Default value: `en|<h2>Your username is <span style="color:blue">%username%</span>.</h2>`

amster attribute: `forgottenUsernameEmailBody`

4.1.5. Profile Management

The following settings appear on the Profile Management tab:

Protected Update Attributes

Specifies a profile's protected user attributes, which causes re-authentication when the user attempts to modify these attributes.

amster attribute: `profileProtectedUserAttributes`

4.1.6. Advanced Configuration

The following settings appear on the Advanced Configuration tab:

User Registration Confirmation Email URL

Specifies the confirmation URL that the user receives during the self-registration process. The `${realm}` string is replaced with the current realm.

Default value: `http://openam.example.com:8080/openam/XUI/?realm=${realm}#register/`

amster attribute: `userRegistrationConfirmationUrl`

Forgotten Password Confirmation Email URL

Specifies the confirmation URL that the user receives after confirming their identity during the forgotten password process. The `${realm}` string is replaced with the current realm.

Default value: `http://openam.example.com:8080/openam/XUI/?realm=${realm}#passwordReset/`

amster attribute: `forgottenPasswordConfirmationUrl`

User Registration Service Config Provider Class

Specifies the provider class to configure any custom plugins.

Default value: `org.forgerock.openam.selfservice.config.flows.UserRegistrationConfigProvider`

amster attribute: `userRegistrationServiceConfigClass`

Forgotten Password Service Config Provider Class

Specifies the provider class to configure any custom plugins.

Default value: `org.forgerock.openam.selfservice.config.flows.ForgottenPasswordConfigProvider`

amster attribute: `forgottenPasswordServiceConfigClass`

Forgotten Username Service Config Provider Class

Specifies the provider class to configure any custom plugins.

Default value: `org.forgerock.openam.selfservice.config.flows.ForgottenUsernameConfigProvider`

amster attribute: `forgottenUsernameServiceConfigClass`

4.2. Legacy User Self Service

amster service name: `security`

4.2.1. Realm Defaults

The following settings appear on the Realm Defaults tab:

Legacy Self-Service REST Endpoint

Specify whether to enable the legacy self-service endpoint.

OpenAM supports two User Self-Service components: the Legacy User Self-Service, which is based on a Java SDK and is available in OpenAM versions prior to OpenAM 13, and a common REST-based/XUI-based User Self-Service available in OpenAM 13 and later.

The Legacy User Self-Service will be deprecated in a future release.

Default value: `false`

amster attribute: `selfServiceEnabled`

Self-Registration for Users

If enabled, new users can sign up using a REST API client.

Default value: `false`

amster attribute: `selfRegistrationEnabled`

Self-Registration Token LifeTime (seconds)

Maximum life time for the token allowing User Self-Registration using the REST API.

Default value: `900`

amster attribute: `selfRegistrationTokenLifetime`

Self-Registration Confirmation Email URL

This page handles the HTTP GET request when the user clicks the link sent by email in the confirmation request.

Default value: `http://openam.example.com:8080/openam/XUI/confirm.html`

amster attribute: `selfRegistrationConfirmationUrl`

Forgot Password for Users

If enabled, users can assign themselves a new password using a REST API client.

Default value: `false`

amster attribute: `forgotPasswordEnabled`

Forgot Password Token Lifetime (seconds)

Maximum life time for the token that allows a user to process a forgotten password using the REST API.

Default value: `900`

amster attribute: `forgotPasswordTokenLifetime`

Forgot Password Confirmation Email URL

This page handles the HTTP GET request when the user clicks the link sent by email in the confirmation request.

Default value: `http://openam.example.com:8080/openam/XUI/confirm.html`

amster attribute: `forgotPasswordConfirmationUrl`

Destination After Successful Self-Registration

Specifies the behavior when self-registration has successfully completed.

The possible values for this property are:

```
User is sent to a 'successful registration' page, without being logged in.
User is sent to the login page, to authenticate.
User is automatically logged in and sent to the appropriate page within the system.
```

Default value: `default`

amster attribute: `userRegisteredDestination`

Protected User Attributes

A list of user profile attributes. Users modifying any of the attributes in this list will be required to enter a password as confirmation before the change is accepted. This option applies to XUI deployments only.

amster attribute: `protectedUserAttributes`

4.3. IDM Provisioning

amster service name: `idm-integration`

4.3.1. Realm Defaults

The following settings appear on the Realm Defaults tab:

Enabled

Default value: `false`

amster attribute: `enabled`

Deployment URL

URL of the IDM deployment.

For example, you might set this property to: *https://openidm.example.com*

amster attribute: `idmDeploymentUrl`

Signing Key Alias

Alias of the signing symmetric key in AM's default keystore. Must be a duplicate of the symmetric key used by IDM.

Default value: `openidm-selfservice-key`

amster attribute: `provisioningSigningKeyAlias`

Encryption Key Alias

Alias of the encryption asymmetric key in AM's default keystore. Must be a duplicate of the asymmetric key used by IDM.

Default value: `selfservice`

amster attribute: `provisioningEncryptionKeyAlias`

Signing Algorithm

JWT signing algorithm.

Default value: `HS256`

amster attribute: `provisioningSigningAlgorithm`

Encryption Algorithm

JWT encryption algorithm.

Default value: `RSAES_PKCS1_V1_5`

amster attribute: `provisioningEncryptionAlgorithm`

Encryption Method

JWT encryption method.

Default value: `A128CBC_HS256`

amster attribute: `provisioningEncryptionMethod`

Appendix A. Getting Support

For more information or resources about AM and ForgeRock Support, see the following sections:

A.1. Accessing Documentation Online

ForgeRock publishes comprehensive documentation online:

- The ForgeRock Knowledge Base offers a large and increasing number of up-to-date, practical articles that help you deploy and manage ForgeRock software.

  While many articles are visible to community members, ForgeRock customers have access to much more, including advanced information for customers using ForgeRock software in a mission-critical capacity.

- ForgeRock product documentation, such as this document, aims to be technically accurate and complete with respect to the software documented. It is visible to everyone and covers all product features and examples of how to use them.

A.2. Using the ForgeRock.org Site

The ForgeRock.org site has links to source code for ForgeRock open source software, as well as links to the ForgeRock forums and technical blogs.

If you are a *ForgeRock customer*, raise a support ticket instead of using the forums. ForgeRock support professionals will get in touch to help you.

A.3. Getting Support and Contacting ForgeRock

ForgeRock provides support services, professional services, training through ForgeRock University, and partner services to assist you in setting up and maintaining your deployments. For a general overview of these services, see https://www.forgerock.com.

ForgeRock has staff members around the globe who support our international customers and partners. For details on ForgeRock's support offering, including support plans and service level agreements (SLAs), visit https://www.forgerock.com/support.

Glossary

Access control	Control to grant or to deny access to a resource.
Account lockout	The act of making an account temporarily or permanently inactive after successive authentication failures.
Actions	Defined as part of policies, these verbs indicate what authorized subjects can do to resources.
Advice	In the context of a policy decision denying access, a hint to the policy enforcement point about remedial action to take that could result in a decision allowing access.
Agent administrator	User having privileges only to read and write agent profile configuration information, typically created to delegate agent profile creation to the user installing a web or Java agent.
Agent authenticator	Entity with read-only access to multiple agent profiles defined in the same realm; allows an agent to read web service profiles.
Application	In general terms, a service exposing protected resources. In the context of AM policies, the application is a template that constrains the policies that govern access to protected resources. An application can have zero or more policies.
Application type	Application types act as templates for creating policy applications. Application types define a preset list of actions and functional logic, such as policy lookup and resource comparator logic.

	Application types also define the internal normalization, indexing logic, and comparator logic for applications.
Attribute-based access control (ABAC)	Access control that is based on attributes of a user, such as how old a user is or whether the user is a paying customer.
Authentication	The act of confirming the identity of a principal.
Authentication chaining	A series of authentication modules configured together which a principal must negotiate as configured in order to authenticate successfully.
Authentication level	Positive integer associated with an authentication module, usually used to require success with more stringent authentication measures when requesting resources requiring special protection.
Authentication module	AM authentication unit that handles one way of obtaining and verifying credentials.
Authorization	The act of determining whether to grant or to deny a principal access to a resource.
Authorization Server	In OAuth 2.0, issues access tokens to the client after authenticating a resource owner and confirming that the owner authorizes the client to access the protected resource. AM can play this role in the OAuth 2.0 authorization framework.
Auto-federation	Arrangement to federate a principal's identity automatically based on a common attribute value shared across the principal's profiles at different providers.
Bulk federation	Batch job permanently federating user profiles between a service provider and an identity provider based on a list of matched user identifiers that exist on both providers.
Circle of trust	Group of providers, including at least one identity provider, who have agreed to trust each other to participate in a SAML v2.0 provider federation.
Client	In OAuth 2.0, requests protected web resources on behalf of the resource owner given the owner's authorization. AM can play this role in the OAuth 2.0 authorization framework.
Conditions	Defined as part of policies, these determine the circumstances under which which a policy applies. Environmental conditions reflect circumstances like the client IP address, time of day, how the subject authenticated, or the authentication level achieved.

	Subject conditions reflect characteristics of the subject like whether the subject authenticated, the identity of the subject, or claims in the subject's JWT.
Configuration datastore	LDAP directory service holding AM configuration data.
Cross-domain single sign-on (CDSSO)	AM capability allowing single sign-on across different DNS domains.
Delegation	Granting users administrative privileges with AM.
Entitlement	Decision that defines which resource names can and cannot be accessed for a given subject in the context of a particular application, which actions are allowed and which are denied, and any related advice and attributes.
Extended metadata	Federation configuration information specific to AM.
Extensible Access Control Markup Language (XACML)	Standard, XML-based access control policy language, including a processing model for making authorization decisions based on policies.
Federation	Standardized means for aggregating identities, sharing authentication and authorization data information between trusted providers, and allowing principals to access services across different providers without authenticating repeatedly.
Fedlet	Service provider application capable of participating in a circle of trust and allowing federation without installing all of AM on the service provider side; AM lets you create Java Fedlets.
Hot swappable	Refers to configuration properties for which changes can take effect without restarting the container where AM runs.
Identity	Set of data that uniquely describes a person or a thing such as a device or an application.
Identity federation	Linking of a principal's identity across multiple providers.
Identity provider (IdP)	Entity that produces assertions about a principal (such as how and when a principal authenticated, or that the principal's profile has a specified attribute value).
Identity repository	Data store holding user profiles and group information; different identity repositories can be defined for different realms.
Java agent	Java web application installed in a web container that acts as a policy enforcement point, filtering requests to other applications in the container with policies based on application resource URLs.

Metadata	Federation configuration information for a provider.
Policy	Set of rules that define who is granted access to a protected resource when, how, and under what conditions.
Policy agent	Java, web, or custom agent that intercepts requests for resources, directs principals to AM for authentication, and enforces policy decisions from AM.
Policy Administration Point (PAP)	Entity that manages and stores policy definitions.
Policy Decision Point (PDP)	Entity that evaluates access rights and then issues authorization decisions.
Policy Enforcement Point (PEP)	Entity that intercepts a request for a resource and then enforces policy decisions from a PDP.
Policy Information Point (PIP)	Entity that provides extra information, such as user profile attributes that a PDP needs in order to make a decision.
Principal	Represents an entity that has been authenticated (such as a user, a device, or an application), and thus is distinguished from other entities. When a Subject successfully authenticates, AM associates the Subject with the Principal.
Privilege	In the context of delegated administration, a set of administrative tasks that can be performed by specified subjects in a given realm.
Provider federation	Agreement among providers to participate in a circle of trust.
Realm	AM unit for organizing configuration and identity information. Realms can be used for example when different parts of an organization have different applications and user data stores, and when different organizations use the same AM deployment. Administrators can delegate realm administration. The administrator assigns administrative privileges to users, allowing them to perform administrative tasks within the realm.
Resource	Something a user can access over the network such as a web page. Defined as part of policies, these can include wildcards in order to match multiple actual resources.
Resource owner	In OAuth 2.0, entity who can authorize access to protected web resources, such as an end user.

Resource server	In OAuth 2.0, server hosting protected web resources, capable of handling access tokens to respond to requests for such resources.
Response attributes	Defined as part of policies, these allow AM to return additional information in the form of "attributes" with the response to a policy decision.
Role based access control (RBAC)	Access control that is based on whether a user has been granted a set of permissions (a role).
Security Assertion Markup Language (SAML)	Standard, XML-based language for exchanging authentication and authorization data between identity providers and service providers.
Service provider (SP)	Entity that consumes assertions about a principal (and provides a service that the principal is trying to access).
Session	The interval that starts with the user authenticating through AM and ends when the user logs out, or when their session is terminated. For browser-based clients, AM manages user sessions across one or more applications by setting a session cookie. See also Stateful session and Stateless session.
Session high availability	Capability that lets any AM server in a clustered deployment access shared, persistent information about users' sessions from the CTS token store. The user does not need to log in again unless the entire deployment goes down.
Session token	Unique identifier issued by AM after successful authentication. For a Stateful session, the session token is used to track a principal's session.
Single log out (SLO)	Capability allowing a principal to end a session once, thereby ending her session across multiple applications.
Single sign-on (SSO)	Capability allowing a principal to authenticate once and gain access to multiple applications without authenticating again.
Site	Group of AM servers configured the same way, accessed through a load balancer layer. The load balancer handles failover to provide service-level availability. Use sticky load balancing based on `amlbcookie` values to improve site performance. The load balancer can also be used to protect AM services.
Standard metadata	Standard federation configuration information that you can share with other access management software.
Stateful session	An AM session that resides in the Core Token Service's token store. Stateful sessions might also be cached in memory on one or more

	AM servers. AM tracks stateful sessions in order to handle events like logout and timeout, to permit session constraints, and to notify applications involved in SSO when a session ends.
Stateless session	An AM session for which state information is encoded in AM and stored on the client. The information from the session is not retained in the CTS token store. For browser-based clients, AM sets a cookie in the browser that contains the session information.
Subject	Entity that requests access to a resource When a subject successfully authenticates, AM associates the subject with the Principal that distinguishes it from other subjects. A subject can be associated with multiple principals.
User data store	Data storage service holding principals' profiles; underlying storage can be an LDAP directory service or a custom `IdRepo` implementation.
Web Agent	Native library installed in a web server that acts as a policy enforcement point with policies based on web page URLs.