RADIUS Server Guide

/ AM 5.0.0

Latest update: 5.0.0

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 as a RADIUS Server.

Table of Contents

Preface

This guide shows you how to configure, maintain, and troubleshoot AM when acting as a RADIUS server.

This guide is written for access management designers, developers, and administrators who build, deploy, and maintain AM 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 the RADIUS Server Service

RADIUS is a lightweight, datagram-based protocol formally specified in RFC 2865 that is supported by many devices and servers for external authentication. VPN concentrators, routers, switches, wireless access points, and many other devices have native RADIUS support. Such devices are known as RADIUS clients. Using the RADIUS protocol, they converse with RADIUS servers to authenticate entities, such as users attempting to access their resources.

1.1. The RADIUS Protocol

The RADIUS protocol itself is quite simple. There are four packet types:

- `Access-Request` packets are sent from a client to a server to begin a new authentication conversation, or to respond to a previous response in an existing conversation and provide requested information.
- `Access-Accept` packets are sent from a server to a client to indicate a successful authentication.
- `Access-Reject` packets are sent from a server to a client to indicate a failed authentication.
- `Access-Challenge` packets are sent from a server to a client to solicit more information from the entity being authenticated.

Each packet type defines:

- A set of fields that must be included
- Other fields that can be included to convey:
  - Additional requirements
  - Information about the context of the conversation
  - Attributes of the entity after successful authentication

For example, an `Access-Request` packet should always contain user name and password fields. It can contain other fields that provide information about the client making the request, such as inclusion of the optional `State` field indicates that a packet is part of an authentication conversation already in progress. Its absence indicates the start of a new conversation.

An authentication conversation always begins with an `Access-Request` packet that does not have a `State` field. If the RADIUS server only requires the user name and password for authentication, then conversations will complete after the server sends an `Access-Accept` or `Access-Reject` packet, depending on whether the authentication credentials were valid.

If more information is required by the server, such as an SMS-relayed one-time password sent to the user's phone, the additional requirement can be solicited using an `Access-Challenge` response to the client, followed by an `Access-Request` packet that has a `State` field that associates it with the existing conversation. The conversation completes with an `Access-Accept` or `Access-Reject` packet depending on whether the one-time password supplied in the second request matches the password sent to the user's phone.

This conversational style in which the server accepts, rejects, or solicits more information makes RADIUS an excellent match for leveraging OpenAM's authentication infrastructure. OpenAM performs authentication using chains of authentication modules found in realms.

These modules identify authentication requirements that are conveyed to clients wishing to authenticate. The modules then accept values submitted by the user for verification. The mechanism for modules to convey these requirements to OpenAM is through a finite set of constructs known as callbacks. By leveraging OpenAM's flexible and extensible authentication mechanism, organizations can craft an authentication experience suitable for their needs, while using the same mechanisms for both HTTP and RADIUS authentication.

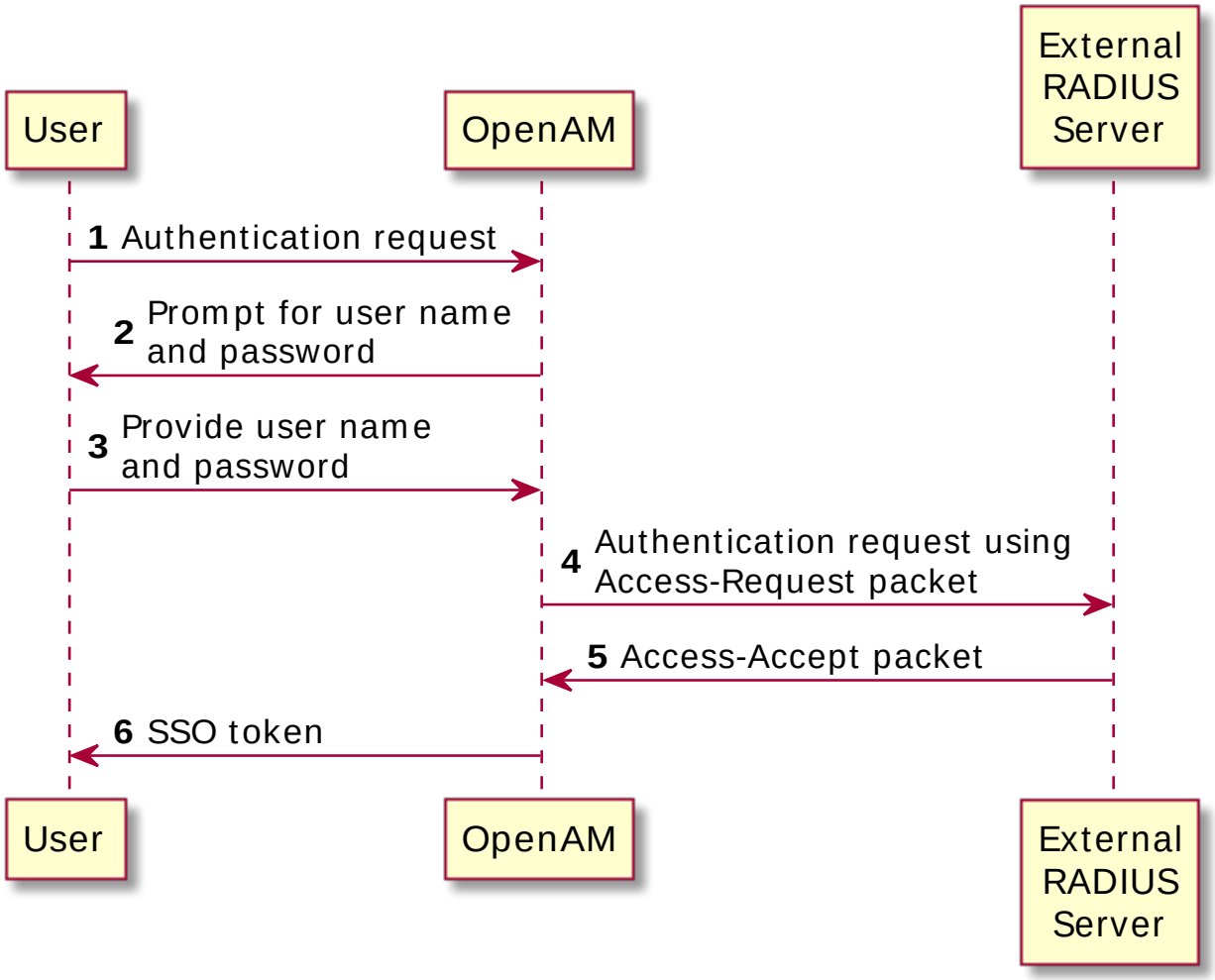
1.2. RADIUS Support

Two AM features support the RADIUS protocol: the RADIUS authentication module and the RADIUS Server service.

1.2.1. RADIUS Authentication Module

The RADIUS authentication module enables OpenAM to act as a RADIUS client, delegating authentication to an external RADIUS server:

RADIUS Authentication Module: Successful Authentication Flow



Use the RADIUS authentication module when you want OpenAM to pass user names and passwords through to an external RADIUS server so that the RADIUS server can authenticate the users.

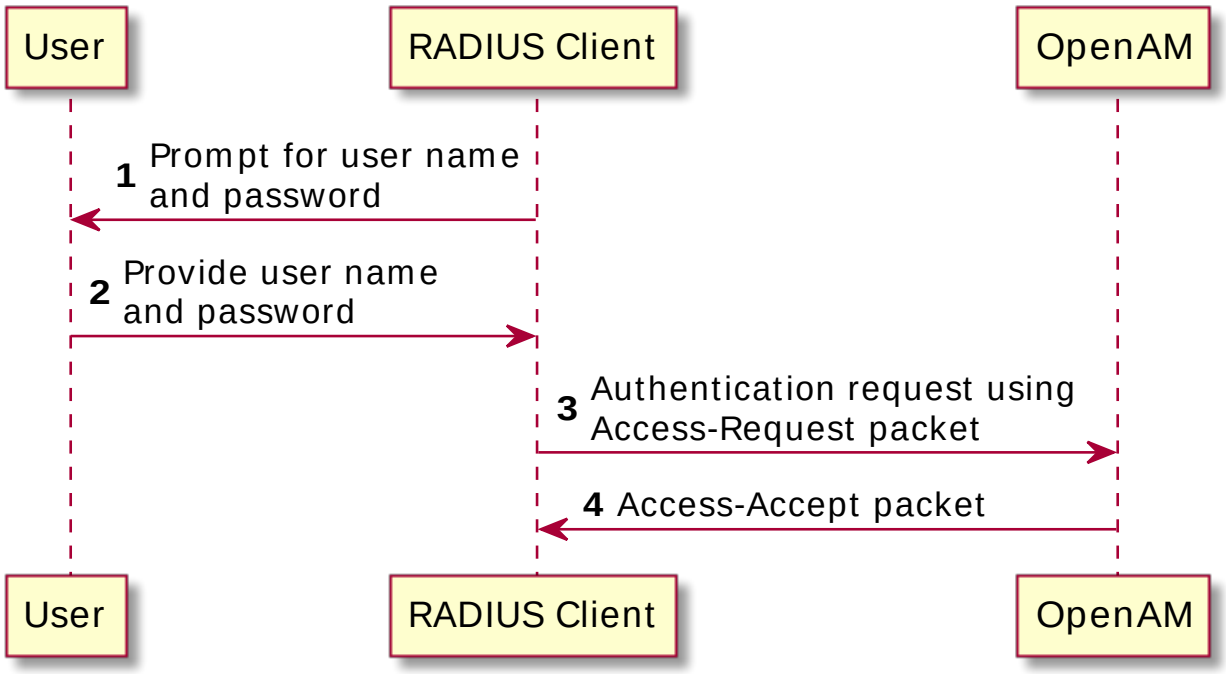
For information about configuring the RADIUS authentication module, see "RADIUS Authentication Module Properties" in the *Authentication and Single Sign-On Guide*.

1.2.2. RADIUS Server Service

The RADIUS Server service provides a RADIUS server within OpenAM. The server authenticates RADIUS clients that are external to OpenAM. The server is backed by OpenAM's authentication chains and modules, thereby providing the possibility of multi-factor authentication in addition to simple user name and password authentication.

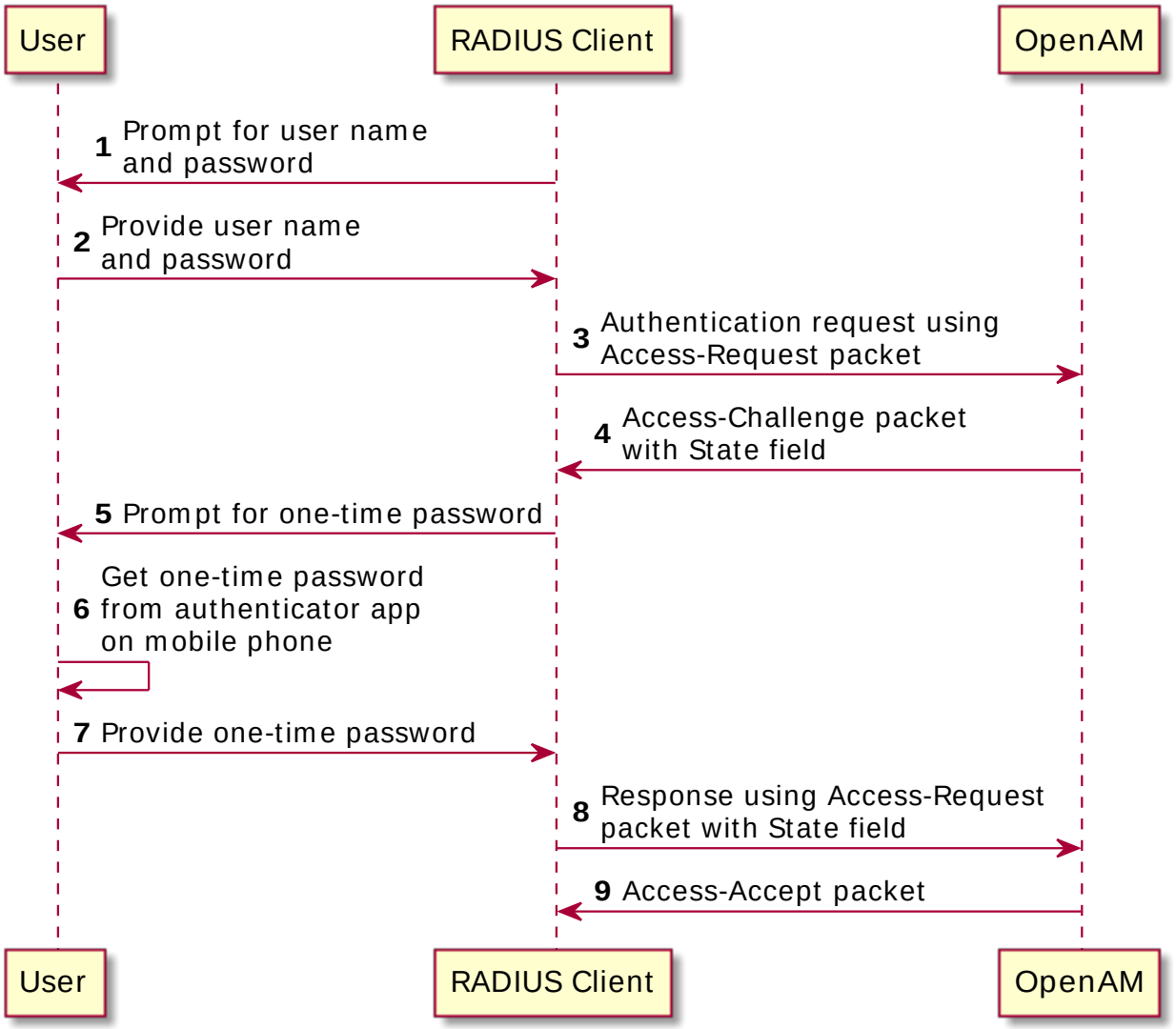
The following example shows the flow of a successful simple user name and password authentication attempt from a RADIUS client:

RADIUS Server Service: Successful Simple Authentication Flow



The following example shows the flow of a successful multi-factor authentication scenario in which the RADIUS Server service is backed by an authentication chain that includes the LDAP and the ForgeRock Authenticator (OATH) authentication modules. First, the LDAP authentication module requires the user to provide a user name and password. Then the ForgeRock Authenticator (OATH) module requires the user to enter a one-time password obtained from the authenticator app on a mobile phone:

RADIUS Server Service: Successful Multi-Factor Authentication Flow



The rest of this chapter covers the configuration of the RADIUS Server service in an OpenAM deployment.

Chapter 2

Implementing the RADIUS Server Service

This chapter describes how to configure OpenAM as a Remote Authentication Dial-In User Service (RADIUS) server that can accept authentication requests from RADIUS clients.

2.1. Configuring the RADIUS Server Service

The OpenAM RADIUS server is disabled by default. To enable it, perform the following steps:

To Enable and Configure the RADIUS Server

1. Login to the AM console as the top-level administrative user, such as `amadmin`.
2. Navigate to Configure > Global Services, and then click RADIUS Server.
3. Under Secondary Configuration Instance, click New.

   OpenAM uses secondary configuration instances in the RADIUS Server service to encapsulate RADIUS clients. You must configure one secondary configuration instance, also known as a subconfiguration, for each client that will connect to the RADIUS Server.
4. Configure attributes for the subconfiguration. See "RADIUS Server" for information about configuring the subconfiguration attributes.
5. Click Add to add the configuration for the RADIUS client to the overall RADIUS Server service's configuration.
6. If you have multiple RADIUS clients that will connect to the OpenAM RADIUS server, add a subconfiguration for each client. It is not necessary to configure all your RADIUS clients when you configure the RADIUS Server service initially—you can add and remove clients over time as you need them.
7. Configure global attributes of the RADIUS Server service. At a minimum, set the Enabled field to `YES` to start the RADIUS server immediately after you save the RADIUS Server service configuration.

   See "RADIUS Server" for information about configuring the RADIUS Server service's global attributes.
8. On the main configuration page for the RADIUS Server service, click Save.

The RADIUS server starts immediately after you save the configuration if the Enabled field has the value `YES`. Any time you make changes to the RADIUS Server service configuration, they take effect as soon as you save the changes.

Chapter 3

Troubleshooting the RADIUS Server Service

This chapter describes how to troubleshoot OpenAM as a Remote Authentication Dial-In User Service (RADIUS) server that can accept authentication requests from RADIUS clients.

3.1. Troubleshooting the RADIUS Server Service

This section covers how to configure OpenAM to troubleshoot the RADIUS Server service, describes how to run a sample client included with OpenAM, and provides details about some specific issues that you might run into when using the RADIUS Server service.

3.1.1. Configuring for Troubleshooting the RADIUS Server Service

If you need to troubleshoot the RADIUS Server service, enable message-level debugging. For information about enabling debug logging, see "Debug Logging" in the *Setup and Maintenance Guide*.

With message-level debug logging enabled, OpenAM writes messages to the `Radius` debug log file when notable events occur, including the following:

- RADIUS server startup
- Changes to the RADIUS server configuration
- Successful and unsuccessful client connections
- Various error events

You can also configure the RADIUS Server service to log the packets sent between RADIUS clients and OpenAM. To enable packet logging, use the Log Packet Contents for this Client property when configuring RADIUS clients in the RADIUS Server service.

3.1.2. Running the Sample RADIUS Client

The `openam-radius-server-5.0.0.jar` includes a sample client that you can use to test simple connectivity to the RADIUS Server service.

The following procedure describes how to set up and run the sample client:

To Run the Sample RADIUS Client

1. Configure the RADIUS Server service. Be sure to enable the service. Include a secondary configuration instance for the sample client as part of the service configuration.

   For more information on the RADIUS Server service configuration properties, see "RADIUS Server".

2. Create a file named `radius.properties` in the current working directory. The file consists of the following key-value pairs:

   - `secret` - Mandatory property specifying the RADIUS client's shared secret. This property's value must be identical to the value of the Client Secret property for the RADIUS client in the OpenAM RADIUS Server service configuration.
   - `host` - Mandatory property specifying the host name or IP address of the OpenAM server.
   - `port` - Mandatory property specifying the port number on which OpenAM's RADIUS server listens. This property's value must be identical to the Listener Port property in the OpenAM RADIUS Server service configuration.
   - `show-traffic` - Optional property specifying whether to show traffic packet during client operation. Valid values are `true` and `false`. Packet traffic is not shown if this property is not specified.

   The following is an example `radius.properties` file:

   ```
   secret=cangetin
   host=openam.example.com
   port=1812
   show-traffic=true
   ```

3. Make sure that your current working directory is the directory in which you created the `radius.properties` file, then execute the sample client. Messages from the sample client indicate success or failure authenticating. If you specify `show-traffic=true` in the `radius.properties` file, the packets to and from the OpenAM RADIUS server appear in standard output:

   ```
   $ java -jar //path/to/tomcat/webapps/openam/WEB-INF/lib/openam-radius-server-5.0.0.jar
   ? Username: demo
   ? Password: changeit
   Packet To openam.example.com:1812
     ACCESS_REQUEST [1]
       - USER_NAME : demo
       - USER_PASSWORD : *******
       - NAS_IP_ADDRESS : openam.example.com/192.168.10.10
       - NAS_PORT : 0

   Packet From openam.example.com:1812
     ACCESS_ACCEPT [1]

   ---> SUCCESS! You've Authenticated!
   ```

3.1.3. Solutions to Common RADIUS Server Service Issues

This section offers solutions to issues that you might encounter when configuring communication between RADIUS clients and the RADIUS Server service. The solutions assume that you have enabled message-level debugging for the RADIUS Server service in OpenAM and have access to the debug logs.

Client Cannot Connect

When a RADIUS client connects to OpenAM's RADIUS server and hangs without receiving a response, the problem could be one of four possible issues:

- The OpenAM RADIUS Server service is not enabled.

  An entry similar to the following in the `Radius` debug log indicates that OpenAM's RADIUS Server was started:

  ```
  amRadiusServer:10/12/2015 12:00:14:814 PM PDT: Thread[RADIUS-1812-Listener,5,main]:
  TransactionId[27350419-8c21-429e-b580-35abf64604cf]
  RADIUS Listener is Active.
  Port              : 1812
  Threads Core      : 2
  Threads Max       : 10
  Thread Keep-alive : 10 sec
  Request Queue     : 10
  ```

  If no such entry exists in the debug log, re-examine the configuration for the RADIUS Server service and correct the problem.

- The client is not defined.

  An entry similar to the following in the `Radius` debug log indicates the inability of a client to connect:

  ```
  amRadiusServer:10/12/2015 04:05:53:681 PM PDT: Thread[RADIUS-1812-Listener,5,main]:
  TransactionId[270084d5-b7d0-42e4-8709-eeaeaf435aff]
  WARNING: No Defined RADIUS Client matches IP address /192.168.10.10. Dropping request.
  ```

  To fix the problem, correct the client configuration in the RADIUS Server service.

- The handler class for the client is incorrect.

  An entry similar to the following in the `Radius` debug log indicates an incorrect handler class:

  ```
  ERROR: Configuration setting handlerClass in RADIUS Client configuration named
  'TestClient' is invalid. Requests from this client will be ignored.
  ```

  To fix the problem, correct the client configuration in the RADIUS Server service.

- Traffic is not arriving at the OpenAM server.

No specific debug log entries appear for this problem.

This is likely a network communication problem. Investigate the route for traffic between the RADIUS client and the OpenAM RADIUS server to see where communication is lost.

Authentication Always Fails

When authentication always fails, the probable cause is one of the following three issues:

- The client secret configured for the client in the RADIUS Server service is incorrect.

  In an `Access-Request` packet, the shared secret is used along with the random value sent in the request authenticator field to encrypt the password field value that is passed across the wire. If the client and server's shared secrets are not identical, the password expected by the server will not match the password sent by the client, resulting in authentication always failing. The user's password is always incorrect in such a scenario and there is no way for the server to differentiate between the client secret being incorrect and the password sent from the client being incorrect. The log file indicates that OpenAM has sent an `Access-Reject` packet to the client, similar to the action that would be taken if the shared secret matched on the client and server and the user entered an invalid password:

  ```
  amRadiusServer:10/12/2015 04:27:55:785 PM PDT: Thread[RADIUS-1812-Listener,5,main]:
  TransactionId[270084d5-b7d0-42e4-8709-eeaeaf435aff]
  finalPacketType sent in response to auth request: 'ACCESS_REJECT'
  ```

  Since the shared secret is specific to each client, such messages might appear for one RADIUS client, while other clients can authenticate successfully.

  To fix this problem, correct the configuration for your client in the RADIUS Server service.

- The realm configured for the client in the RADIUS Server service is incorrect.

  An entry similar to the following in the `Radius` debug log indicates an invalid realm in the RADIUS Server service configuration:

  ```
  ERROR: Unable to start login process. Denying Access.
  com.sun.identity.authentication.spi.AuthLoginException: Domain is invalid|
  invalid_domain.jsp
  ```

  If the realm is missing from the configuration, an error similar to the following appears:

  ```
  ERROR: Unable to initialize declared handler class
  'org.forgerock.openam.radius.server.spi.handlers.OpenAMAuthHandler' for RADIUS
  client ''. Rejecting access.
  java.lang.IllegalStateException: Configuration property 'realm' not found in
  handler configuration. It must be added to the Configuration Properties for this
  class in the Radius Client's configuration.
  ```

  To fix this problem, correct the client configuration in the RADIUS Server service.

- The authentication chain configured for the client in the RADIUS Server service is incorrect.

  An entry similar to the following in the `Radius` debug log indicates an invalid authentication chain in the RADIUS Server service configuration:

  ```
  amRadiusServer:10/12/2015 05:32:21:771 PM PDT: Thread[pool-5-thread-2,5,main]:
  TransactionId[378a41cf-0581-4b62-a92f-be2b008ab4d3] ERROR: Unable to start login
  process. Denying Access.
  ```

  If the chain is missing from the configuration, an error similar to the following appears:

  ```
  ERROR: Unable to initialize declared handler class
  'org.forgerock.openam.radius.server.spi.handlers.OpenAMAuthHandler' for RADIUS
  client ''. Rejecting access.
  java.lang.IllegalStateException: Configuration property 'chain' not found in
  handler configuration. It must be added to the Configuration Properties for this
  class in the Radius Client's configuration.
  ```

  To fix this problem, correct the client configuration in the RADIUS Server service.

Configuration Is Correct but Authentication Fails

In this case, you might have a client-specific problem. OpenAM provides a tool that you can use to eliminate OpenAM and its configuration as the cause of the problem. You can declare an alternate handler class implementation in the RADIUS Server service configuration. Two test handlers are available for troubleshooting purposes:

- The `org.forgerock.openam.radius.server.spi.handlers.AcceptAllHandler` handler always returns an `Access-Accept` packet, indicating successful authentication for all requests.
- The `org.forgerock.openam.radius.server.spi.handlers.RejectAllHandler` handler always returns an `Access-Reject` packet, indicating failed authentication for all requests.

In a case where you believe that configuration is correct but authentication always fails, you could specify the `org.forgerock.openam.radius.server.spi.handlers.AcceptAllHandler` handler class in the RADIUS Server service configuration for your client. With packet logging enabled, all requests received from the client should log packet contents traffic similar to the following even if the password is incorrect:

```
WARNING:
Packet from TestClient:
  ACCESS_REQUEST [1]
    - USER_NAME : demo
    - USER_PASSWORD : *******
    - NAS_IP_ADDRESS : /127.0.0.1
    - NAS_PORT : 0
```

This is followed by:

```
WARNING:
Packet to TestClient:
  ACCESS_ACCEPT [1]
```

If the client still indicates that authentication has failed, refer to the documentation for the client to determine why the `Access-Accept` response is rejected. Most likely, the client expects specific fields in the `Access-Accept` response that are not provided by OpenAM. There is currently no facility in OpenAM to return fields in `Access-Accept` responses.

Authentication Always Succeeds, Even With a Bad Password

This would be a very unusual situation, probably due to the `org.forgerock.openam.radius.server.spi.handlers.AcceptAllHandler` handler being left in place after troubleshooting an error scenario in which authentication always suceeds.

To resolve the problem, verify that the correct handler class is specified in the RADIUS Server service configuration for the client. If it is not specified, review the authentication modules in the chain that authenticates users and determine whether one of the modules might be accepting all authentication requests. This situation could also occur because of incorrectly-specified module criteria in the chain's definition.

3.2. RADIUS Server Limitations

Deploying OpenAM's RADIUS server lets an organization consolidate RADIUS and HTTP authentication into a single solution, facilitating reuse of existing authentication mechanisms between both types of clients. However, there are several limitations:

- Because RADIUS authentication attempts always start with a user name and password transmitted in an `Access-Request` packet, the first module in an authentication chain used for RADIUS clients must accept a user name and a password.
- By default, OpenAM caches up to 5,000 RADIUS clients concurrently.

  If necessary, you can change the maximum number of RADIUS clients that can be cached concurrently on an OpenAM server by configuring the `org.forgerock.openam.radius.server.context.cache.size` advanced server property.

  See "Configuring Servers" in the *Reference* for information about how to configure advanced server properties.
- Some OpenAM callback types are not applicable to RADIUS clients. For example, a `RedirectCallback` directs HTTP clients, such as browsers, to HTTP resources to be used for some aspect of authentication. Redirects make no sense to RADIUS clients and cannot be consumed in any meaningful way.

  A `ConfirmationCallback` also presents challenges for RADIUS clients.

  As a result, some OpenAM authentication modules cannot be used with RADIUS clients. Before attempting to use an authentication module with RADIUS clients, review the module's callbacks to determine whether the module will support RADIUS clients. You can use the REST API to determine the callbacks for an authentication module as described in "Authentication and Logout" in the *Development Guide*.

- Some client mechanisms leveraged by authentication modules might not be applicable to RADIUS clients. For example, suppose a customized SMS one-time password module sends a one-time password over an SMS service, and then provides a `ChoiceCallback` that enables the user to set a cookie in their browser that expires after 30 days. Such a module might first determine whether the cookie was available, still valid, and applicable to the current user before reissuing a new one-time and soliciting the value from the user.

  RADIUS clients are unable to process HTTP cookies. Therefore, although RADIUS clients can support a `ChoiceCallback`, the customized feature described in the previous paragraph would not function correctly for RADIUS clients and therefore should not be deployed with RADIUS clients. As a result, some callback sets within an authentication module will differ depending on the type of client being authenticated.

- The RADIUS Server service logs *only* to the ForgeRock common audit logger introduced in OpenAM 13. It does *not* log to the classic OpenAM audit logs that were available prior to OpenAM 13.

When building custom authentication modules, consider their suitability to handle the types of clients that might use them, and make adjustments to callbacks as needed.

Chapter 4
RADIUS Reference

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

4.1. RADIUS Server

amster type ID: `RadiusServerService`

4.1.1. Configuration

The following settings appear on the **Configuration** tab:

Enabled

Enables the OpenAM RADIUS server to listen for requests on the listener port and to handle the requests.

The possible values for this property are:

```
YES
NO
```

Default value: `NO`

amster data attribute: `radiusListenerEnabled`

Listener Port

The UDP port on which each OpenAM server will listen for RADIUS Access-Request packets

According to the RADIUS Authentication Specification, RFC 2865, the officially assigned port number for RADIUS is `1812`. Specify a value from `1024` to `65535`. All client requests are handled through the same port.

Default value: `1812`

amster data attribute: `radiusServerPort`

Thread Pool Core Size

When a RADIUS request is received and fewer than `corePoolSize` threads are running, a new thread is created to handle the request, even if other worker threads are idle. If there are more than "Thread Pool Core Size" but less than "Thread Pool Max Size" threads running, a new thread will be created only if the queue is full. By setting "Thread Pool Core Size" and "Thread Pool Max Size" to the same value, you create a fixed-size thread pool. Specify a value from `1` to `100`.

Default value: `1`

amster data attribute: `radiusThreadPoolCoreSize`

Thread Pool Max Size

Maximum number of threads allowed in the pool. See also "Thread Pool Core Size".

Default value: `10`

amster data attribute: `radiusThreadPoolMaxSize`

Thread Pool Keep-Alive Seconds

If the pool currently has more than Thread Pool Core Size threads, excess threads will be terminated if they have been idle for more than the Keep-Alive Seconds. Specify a value from `1` to `3600`.

Default value: `10`

amster data attribute: `radiusThreadPoolKeepaliveSeconds`

Thread Pool Queue Size

The number of requests that can be queued for the pool before further requests will be silently dropped. See also "Thread Pool Core Size" and "Thread Pool Max Size". Specify a value from `1` to `1000`.

Default value: `20`

amster data attribute: `radiusThreadPoolQueueSize`

4.1.2. Secondary Configurations

This service has the following Secondary Configurations.

4.1.2.1. radiusClient

Client IP Address

The IP Address of the client.

Section 5.4 of the RADIUS Authentication Specification, RFC 2865, indicates that the source IP address of the Access-Request packet *MUST* be used to identify a configured client and hence determine the shared secret to use for decrypting the User-Password field.

This property should hold the source IP address of the client. This should match the value obtained from Java's `InetSocketAddress.getAddress().toString()` function.

To verify the value, send an Access-Request packet to OpenAM's RADIUS port and watch for a message stating: `"No Defined RADIUS Client matches IP address '/127.0.0.1'. Dropping request."`. The value used in this property should match the IP address returned in the single quotes.

Default value: `/127.0.0.1`

amster data attribute: `clientIpAddress`

Client Secret

This secret shared between server and client for encryption of the user password.

This secret must be conveyed to the RADIUS client and entered into its configuration before the User-Password field of incoming Access-Request packets can be decrypted to validate the password for the represented by that packet. A default value is generated for you but you can enter a custom value if desired.

Default value: `ov1Q0Bpw+urxFczT`

amster data attribute: `clientSecret`

Log Packet Contents for this Client

Indicates if full packet contents should be dumped to the log.

When troubleshooting issues with RADIUS it is helpful to know what was received in a given packet. Enabling this feature will cause packet contents to be logged in a human consumable format. The only caveat is that the USER_PASSWORD field will be obfiscated by replacing with asterisks. This should only be enabled for troubleshooting as it adds significant content to logs and slows processing.

Default value: `NO`

amster data attribute: `clientPacketsLogged`

Handler Class

The fully qualified name of a class to handle incoming RADIUS Access-Requests for this client.

This class must implement the `com.sun.identity.authentication.modules.radius.server.spi.AccessRequestHandler` interface to handle incoming Access-Request packets and provide a suitable response. An instance of this class is created when configuration is first loaded to validate the class and then once for each new request. The configuration properties will only be passed for the request handling instances and not when validating the class.

Default value: `org.forgerock.openam.radius.server.spi.handlers.OpenAMAuthHandler`

amster data attribute: `handlerClass`

Handler Class Configuration Properties

Properties needed by the handler class for its configuration.

These properties are provided to the handler via its `init` method prior to the call to handle the request packet. If these values are changed the next handler instance created for an incoming request will receive the updated values. Each entry assumes that the first '`=`' character incurred separates a key from its value. All entries are placed in a properties file handed to each handler instance.

Default value:

```
realm=/
chain=ldapService
```

amster data attribute: `handlerConfig`

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 policy agent profile configuration information, typically created to delegate policy agent profile creation to the user installing a policy 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 EE policy agent	Java web application installed in a web container that acts as a policy agent, 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	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, a relational database, or a custom `IdRepo` implementation.
Web policy agent	Native library installed in a web server that acts as a policy agent with policies based on web page URLs.