



AWS IAM CyberArk Authentication Integration Technical Documentation

Name of Company: AWS

Website: <https://aws.amazon.com/>

Name of Product: IAM

Date: Feb 2019

FEATURE OVERVIEW

The Conjur IAM Authenticator allows an AWS resource to use its AWS IAM role to authenticate with Conjur. This approach enables EC2 instances and Lambda functions to access credentials stored in Conjur without a pre-configured Conjur identity.

SETUP

To enable an IAM Authenticator called `prod`, we'll set the following environment variable when we start Conjur:

```
CONJUR_AUTHENTICATORS=authn-iam/prod
```

In this context, `prod` is called the `service ID`. Multiple authenticators can be configured with different service IDs. Each one implements a separate "zone" of authentication.

After Conjur is started, we'll create a policy to enable our `prod` IAM Authenticator:

```
# policy id needs to match the convention `conjur/authn-iam/<service ID>`
- !policy
  id: conjur/authn-iam/prod
  body:
    - !webservice

    - !group clients

    - !permit
      role: !group clients
      privilege: [ read, authenticate ]
      resource: !webservice
```

Next, let's create an application policy to provide a database username and password to an AWS services. The IAM role in this example is `011915987442:MyApp`:

```
- !policy
  id: myapp
  body:
```

```

- &variables
  - !variable database/username
  - !variable database/password

# Create a group that will have permission to retrieve variables
- !group secrets-users

# Give the `secrets-users` group permission to retrieve variables
- !permit
  role: !group secrets-users
  privilege: [ read, execute ]
  resource: *variables

# Create a layer to hold this application's hosts
- !layer

# The host ID needs to match the AWS ARN of the role we wish to authenticate.
- !host 011915987442/MyApp

# Add our host into our layer
- !grant
  role: !layer
  member: 011915987442/MyApp

# Give the host in our layer permission to retrieve variables
- !grant
  member: !layer
  role: !group secrets-users

```

Important: Note above host has an ID composed of a prefix/namespace followed by the AWS account ID followed by the name of the AWS IAM role. The AWS Account ID and name of the role is extracted from the `getCallerIdentity` by the authenticator

Finally, let's give our `myapp` host permission to authenticate using the IAM Authenticator:

```
- !grant  
  role: !group conjur/authn-iam/prod/clients  
  member: !host myapp/011915987442/MyApp
```

WORKFLOW

Now that the IAM Authenticator has been configured and we've permitted an IAM role to authenticate, let's look at the authentication flow of an EC2 instance or Lambda function.

1. The instance or function starts, assuming the IAM role it was provided.
2. From the instance or function, generate a signed request to the STS service, to get the identity of the requestor. This request is signed using the instance or function's access key. The signed request is valid for five minutes. Below is an example of how a signed request would be generated (using Ruby):

```
require 'aws-sigv4'
require 'aws-sdk'

request = Aws::Sigv4::Signer.new(
  service: 'sts',
  region: 'us-east-1',
  credentials_provider: Aws::InstanceProfileCredentials.new
).sign_request(
  http_method: 'GET',
  url: 'https://sts.amazonaws.com/?Action=GetCallerIdentity&Version=2011-06-15'
).headers
```

3. Using the signed request, the instance or function authenticates with Conjur using the following:

```
require 'conjur-api'

Conjur.configuration.account = 'my-account'
Conjur.configuration.appliance_url = 'https://conjur.mydomain.com/authn-iam/prod'
Conjur.configuration.cert_file = '<cert>/<path>/conjur-yourorg.pem'
Conjur.configuration.apply_cert_config!

conjur = Conjur::API.new_from_key 'host/aws/011915987442:assumed-role/MyApp',
request.to_json

variable = conjur.resource("#{Conjur.configuration.account}:variable:database/password")
```

4. When Conjur receives this authentication request, it performs the following:

- i. Validates the host myapp/011915987442/MyApp has permission to authenticate using the prodIAM Authenticator.
- ii. Extracts the signed request from the POST body.
- iii. Creates a request to the AWS STS service, using the provided signed request as its header.
- iv. If the STS request is successful, the requesting instance or function's IAM role is returned. The role is Validated against the requesting role. If the two roles match, an authentication token is returned. Below is an example of a successful STS response:

```

v.    <GetCallerIdentityResponse xmlns="https://sts.amazonaws.com/doc/2011-06-15/">
vi.    <GetCallerIdentityResult>
vii.    <Arn>arn:aws:sts::011915987442:MyApp/i-0a5702a5a078e1a00</Arn>
viii.    <UserId>AROAJYAZ7DBLU2PE4DWOW:i-0a5702a5a078e1a00</UserId>
ix.    <Account>011915987442</Account>
x.    </GetCallerIdentityResult>
xi.    <ResponseMetadata>
xii.    <RequestId>88278d14-4f3e-11e8-88a1-a9dc7b9cbe6c</RequestId>
xiii. </ResponseMetadata>

      </GetCallerIdentityResponse>

```

Authentication may fail for a number of reasons. In each case, a 401 Unauthorized response will be returned.

Reasons for failing authentication include:

- Signed request is invalid (signed by an unknown AWS Access Key or older than 5 minutes)
- Role ARN from signed request does not match the Conjur host ARN

Host does not have the privilege to authenticate using the IAM Authenticator.