Using Temporary Security Credentials

AWS STS

API Version 2011-06-15

Amazon Web Services

Using Temporary Security Credentials: AWS STS

Amazon Web Services

Welcome

IAM enables you to grant any user temporary access to your Amazon Web Services (AWS) resources. Temporary access is useful when:

- You need identity federation between AWS and your non-AWS users in your own identity and authorization system.
- You want single sign-on between your identity and authorization system and the AWS Management Console.
- You need enhanced security for mobile environments and for users accessing your AWS resources through a web browser.
- You have IAM users who need temporary security credentials.

To grant users temporary access to your resources, you call the AWS Security Token Service (STS) APIs. The AWS STS APIs return temporary security credentials consisting of a security token, an Access Key ID, and a Secret Access Key. You issue the temporary security credentials to the users who need temporary access to your resources. These users can be existing IAM users, or they can be non-AWS users (*federated identities*). The users might even be systems or applications that need to access your AWS resources. The extent to which a user who has these temporary security credentials can access your AWS resources depends on permissions you set using AWS access policies.

Temporary security credentials provide enhanced security because they have short life spans and cannot be reused after they expire. The Access Key ID and Secret Access Key generated with the token cannot be used without the token, and a user who has these temporary security credentials can access your resources only until the credentials expire. By default, temporary security credentials are valid for 12 hours, but you can specify a different maximum duration. You can create as many sets of temporary security credentials as you need; there is no limit.

This guide describes some common use cases for granting temporary access, explains how to use the AWS STS API to generate temporary security credentials, describes how permissions work, and provides links to information about how to use temporary security credentials with other AWS products. This guide also links to sample applications you can view to learn more about generating temporary security credentials.

Important

Although temporary security credentials are short-lived, you should be aware that users who have temporary access can make lasting changes to your AWS resources. For example, if a user with temporary access launches an Amazon EC2 instance, that instance could continue to

run and incur charges against your AWS account even after the user's temporary security credentials expire.

Note

Not all AWS products support using temporary security credentials. For a list of the products that accept temporary security credentials, see Using Temporary Security Credentials to Access AWS (p. 19).

Use Cases for Granting Temporary Access

Topics

There are several different reasons you might choose to use temporary security credentials. This section describes two of the most common scenarios: granting a mobile application access to your AWS resources, and using your company's own authentication system to grant your employees access to your AWS resources. To illustrate these scenarios we use a fictitious company, Example Corp.

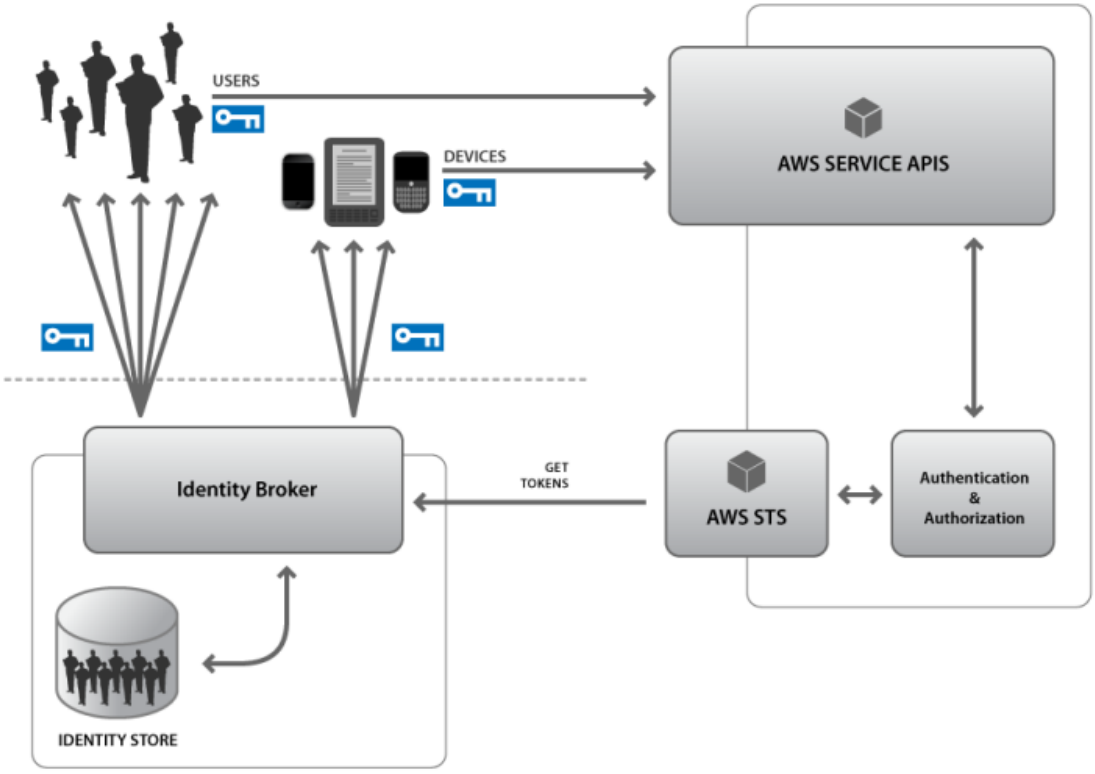
Using a Mobile Application to Access AWS Resources

Example Corp. is building a mobile application that enables the application's registered users to access to the company's AWS resources. Bob is a developer at Example Corp. For security reasons, Bob wants the mobile application to use temporary security credentials to access to the company's AWS resources. Also, because Bob wants to be able to generate temporary security credentials for a large number of devices, temporary security credentials are convenient because there is no limit to the number of temporary security credentials that can be created.

To enable the mobile application to access the company's AWS resources, Bob develops a broker application that will create temporary security credentials for authenticated users. The broker application verifies that users are signed in to the existing Example Corp. identity and authentication system, and then it creates temporary security credentials for the mobile application to use to access the company AWS resources.

The broker application works by verifying that the users are authenticated and then by defining the appropriate permissions in AWS. It passes these permissions to the AWS Security Token Service (STS) GetFederationToken API action so that they are associated with the temporary security credentials. GetFederationToken returns temporary security credentials consisting of an AWS Access Key ID, a Secret Access Key, and a security token. The broker application issues the temporary security credentials to the mobile application. The mobile application can use these temporary credentials to make calls to AWS

directly, until the temporary credentials expire. When the temporary credentials expire, the mobile application contacts the broker application for new credentials. The following diagram illustrates this scenario.



The following details enable this scenario:

- The broker application has permission to access the AWS STS API to create temporary security credentials.
- The broker application is able to verify that the mobile application users are authenticated within the existing authentication system, and the broker application can create an unlimited number of temporary security credentials.

To see a sample application similar to the application described in this use case, go to Authenticating Users of AWS Mobile Applications with a Token Vending Machine at *AWS Articles & Tutorials*. For information about creating temporary security credentials, see Creating Temporary Security Credentials (p. 7).

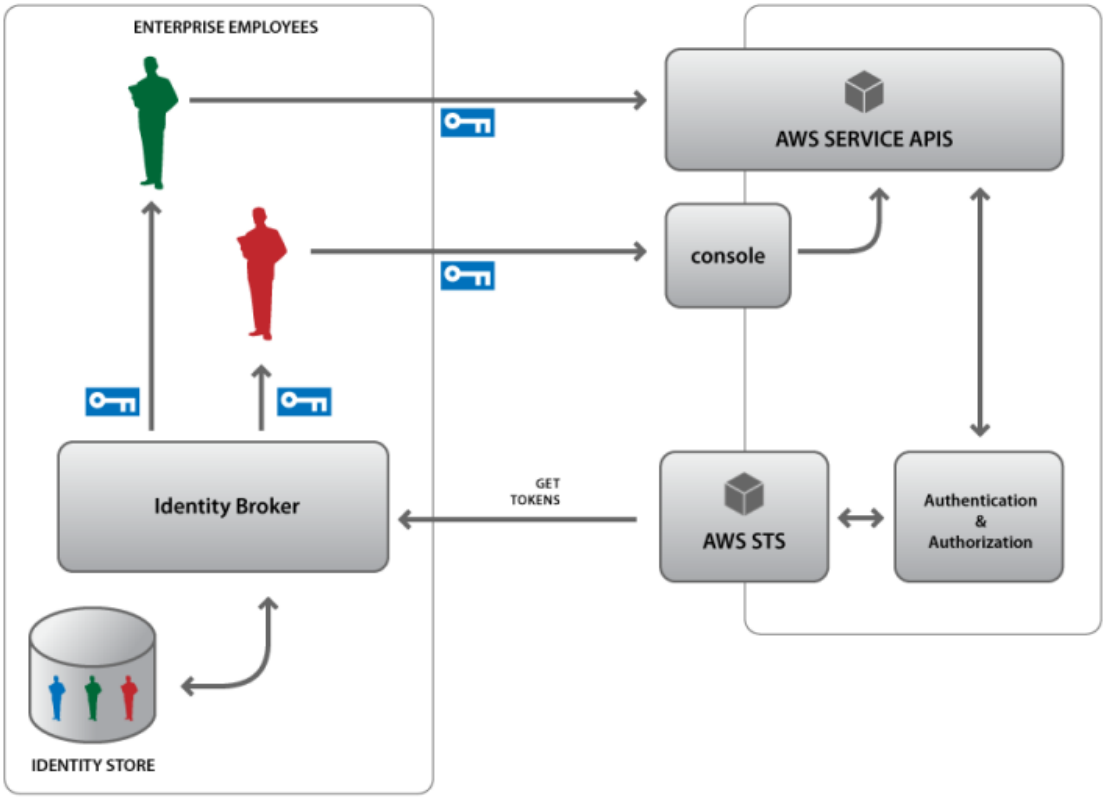
Using Your Company's Own Authentication System to Grant Access to AWS Resources

Example Corp. has many employees who need to access the company's AWS resources, and the employees already have identities in the company identity and authentication system.

Bob is a developer at Example Corp. To enable Example Corp. internal applications to access the company's AWS resources, Bob develops an identity broker application. The application that verifies

employees are signed in to the existing Example Corp. identity and authentication system, then it creates temporary security credentials for the employees.

The broker application works by verifying that the employees are authenticated and then by defining the appropriate permissions in AWS. It passes these permissions to the AWS Security Token Service (STS) GetFederationToken API action so that they are associated with the temporary security credentials. GetFederationToken returns temporary security credentials consisting of an AWS Access Key ID, a Secret Access Key, and a security token. The broker application issues the temporary security credentials to the requesting Example Corp. application. The application can use these temporary security credentials to make calls to AWS directly, until the credentials expire, and without having to create new identities in IAM for every user who needs access. The following diagram illustrates this scenario.



The following details enable this scenario:

- The broker application is able to access the AWS STS API to create temporary security credentials.
- The broker application is able to verify that employees are authenticated within the existing authentication system, and users don't need to have new identities created for them in AWS Identity and Access Management (IAM).

To see a sample application similar to the broker application described in this use case, go to Identity Federation Sample Application for an Active Directory Use Case at *AWS Sample Code & Libraries*. For information about creating temporary security credentials, see Creating Temporary Security Credentials (p. 7).

Ways to Access the AWS Security Token Service

To access the AWS Security Token Service (STS) you can code directly to the AWS STS Query API. This API is a web service interface that accepts HTTPS requests. The AWS STS interface is described in the AWS Security Token Service API Reference.

If you prefer to use AWS STS through an existing programmatic interface, AWS offers SDKs for:

- Java
- .NET
- PHP
- Ruby

Note

AWS also provides a .NET sample application that demonstrates how you might use AWS STS with your existing identity authorization system. To see the sample application, go to Identity Federation Sample Application for an Active Directory Use Case at *AWS Sample Code & Libraries*.

Creating Temporary Security Credentials

Topics

This topic describes using the AWS Security Token Service (STS) API to create temporary security credentials. For information about using one of the supported SDKs to create temporary security credentials, see Ways to Access the AWS Security Token Service (p. 6).

Two types of temporary security credentials are available: credentials for IAM users and credentials for federated users (non-AWS users). The method you use to create the temporary security credentials depends on the type of user they are for. To create temporary security credentials for their own use, IAM users call the AWS STS GetSessionToken API action. Users do not need explicit permission to use `GetSessionToken`; it is available to all IAM users. However, if you want to enable your users to create temporary security credentials for federated users, you must grant those users permission to access GetFederationToken. This section describes how to use these API actions to create temporary security credentials, and it provides policy examples showing how you can use policies to grant IAM users permission to create federation credentials.

For more general information about controlling user permissions, see Managing IAM Policies. The AWS STS API is described in detail in the AWS Security Token Service API Reference.

Important

You should be aware that once you issue temporary security credentials, you cannot revoke them. However, in the rare cases when you need to disable temporary security credentials before they expire, you can control temporary security credential permissions by modifying the permissions of the IAM user who created them. For this reason, we do not recommend using your root account credentials to create temporary security credentials for federated users. For more information, see Disabling Permissions Granted Through Temporary Security Credentials (p. 16).

Note

You can give your IAM users the ability to create temporary security credentials, but users cannot use these credentials to access IAM or AWS STS.

Creating Temporary Security Credentials to Enable Access for IAM Users

IAM users can use the AWS Security Token Service GetSessionToken API action to create temporary security credentials for themselves. This enables access for IAM users or AWS accounts whose permissions are already defined. Because the credentials are temporary, they provide enhanced security when you have an IAM user who will be accessing your resources through a less secure environment, such as a mobile device or web browser.

By default, temporary security credentials for an IAM user are valid for a maximum of 12 hours, but you can request a duration as short as one hour, or as long as 36 hours. For security reasons, a token for an AWS account's root identity is restricted to a duration of one hour.

`GetSessionToken` returns temporary security credentials consisting of a security token, an Access Key ID, and a Secret Access Key. The following example shows a sample request and response using `GetSessionToken`. The response also includes the expiration time of the temporary security credentials.

Example Request

```
https://sts.amazonaws.com/
?Version=2011-06-15
&Action=GetSessionToken
&DurationSeconds=3600
&AUTHPARAMS
```

Example Response

```
<GetSessionTokenResponse xmlns="https://sts.amazonaws.com/doc/2011-06-15/">
  <GetSessionTokenResult>
    <Credentials>
      <SessionToken>
       AQoEXAMPLEH4aoAH0gNCAPyJxz4BlCFFxWNE1OPTgk5TthT+FvwqnKwRcOIfrRh3c/L
       To6UDdyJwOOvEVPvLXCrrrUtdnniCEXAMPLE/IvU1dYUg2RVAJBanLiHb4IgRmpRV3z
       rkuWJOgQs8IZZaIv2BXIa2R4OlgkBN9bkUDNCJiBeb/AXlzBBko7b15fjrBs2+cTQtp
       Z3CYWFXG8C5zqx37wnOE49mRl/+OtkIKGO7fAE
      </SessionToken>
      <SecretAccessKey>
      wJalrXUtnFEMI/K7MDENG/bPxRfiCYzEXAMPLEKEY
      </SecretAccessKey>
      <Expiration>2011-07-11T19:55:29.611Z</Expiration>
      <AccessKeyId>AKIAIOSFODNN7EXAMPLE</AccessKeyId>
    </Credentials>
  </GetSessionTokenResult>
  <ResponseMetadata>
    <RequestId>58c5dbae-abef-11e0-8cfe-09039844ac7d</RequestId>
  </ResponseMetadata>
</GetSessionTokenResponse>
```

Temporary Security Credentials for IAM Users with Multi-Factor Authentication (MFA)

Optionally, the `GetSessionToken` request can include `SerialNumber` and `TokenCode` values for AWS Multi-Factor Authentication (MFA) verification. If the provided values are valid, STS provides temporary security credentials that include the state of MFA authentication so the temporary security credentials can be used to access the MFA-protected APIs or AWS websites for as long as the MFA authentication is valid.

The following is an example of a `GetSessionToken` request with an MFA verification code and device serial number using the STS Query API.

```
https://sts.amazonaws.com/
?Version=2011-06-15
&Action=GetSessionToken
&DurationSeconds=7200
&SerialNumber=YourMFADeviceSerialNumber
&TokenCode=123456
&AUTHPARAMS
```

Related Topics

- GetSessionToken in the *AWS Security Token Service API Reference*
- Using Multi-Factor Authentication (MFA) Devices with AWS in *Using AWS Identity and Access Management*
- Making Query Requests in *Using AWS Identity and Access Management*
- Controlling Permissions in Temporary Security Credentials (p. 13)

Creating Temporary Security Credentials to Enable Access for Federated Users

To grant temporary access to a non-AWS user whose identity you can authenticate (a federated user) use the AWS STS `GetFederationToken` action. This action is useful if you have non-AWS users that you authenticate with an external service, such as Microsoft Active Directory, LDAP, or Kerberos. You might use `GetFederationToken` in a broker application you build that enables your users to sign in to your authentication system, and to receive temporary security credentials that they can use to access your AWS resources. To see an example of this kind of application, go to Identity Federation Sample Application for an Active Directory Use Case at *AWS Sample Code & Libraries*.

When you create temporary security credentials for a federated user, you specify a user identity and request a maximum duration for the temporary security credentials to remain valid. Credentials created by IAM users are valid for the specified duration, between one and 36 hours; credentials created using account credentials last one hour. You can also use the *`Policy`* parameter to pass in an IAM policy that specifies the permissions to apply to the federated user. The `GetFederationToken` action returns temporary security credentials consisting of the security token, an Access Key ID, and a Secret Access Key.

The following example shows a sample request and response using `GetFederationToken`. In this example, the request includes the name for a federated user named Jean, and it includes a value for the *`Policy`* parameter. This policy grants the federated user permission to access Amazon Simple Storage

Service (Amazon S3) only. In addition to the temporary security credentials, the response includes the Amazon Resource Name (ARN) for the federated user, and the expiration time of the credentials.

Example Request

```
https://sts.amazonaws.com/
?Version=2011-06-15
&Action=GetFederationToken
&Name=Jean
&Policy=%7B%22Statement%22%3A%5B%7B%22Sid%22%3A%22Stmt1%22%2C%22Effect%22%
 3A%22Allow%22%2C%22Action%22%3A%22s3%3A*%22%2C%22Resource%22%3A%22*%22%7D
 %5D%7D
&DurationSeconds=3600
&AUTHPARAMS
```

Note

The policy value shown in the example above is the URL-encoded version of this policy: {"Statement":[{"Sid":"Stmt1","Effect":"Allow","Action":"s3:*","Resource":"*"}]}.

Example Response

```
<GetFederationTokenResponse xmlns="https://sts.amazonaws.com/doc/
 2011-06-15/">
  <GetFederationTokenResult>
    <Credentials>
      <SessionToken>
       AQoDYXdzEPT//////////wEXAMPLEtc764bNrC9SAPBSM22wDOk4x4HIZ8j4FZTwdQW
       LWsKWHGBuFqwAeMicRXmxfpSPfIeoIYRqTflfKD8YUuwthAx7mSEI/qkPpKPi/kMcGd
       QrmGdeehM4IC1NtBmUpp2wUE8phUZampKsburEDy0KPkyQDYwT7WZ0wq5VSXDvp75YU
       9HFvlRd8Tx6q6fE8YQcHNVXAkiY9q6d+xo0rKwT38xVqr7ZD0u0iPPkUL64lIZbqBAz
       +scqKmlzm8FDrypNC9Yjc8fPOLn9FX9KSYvKTr4rvx3iSIlTJabIQwj2ICCEXAMPLE==
      </SessionToken>
      <SecretAccessKey>
      wJalrXUtnFEMI/K7MDENG/bPxRfiCYzEXAMPLEKEY
      </SecretAccessKey>
      <Expiration>2011-07-15T23:28:33.359Z</Expiration>
      <AccessKeyId>AKIAIOSFODNN7EXAMPLE;</AccessKeyId>
    </Credentials>
    <FederatedUser>
      <Arn>arn:aws:sts::123456789012:federated-user/Jean</Arn>
      <FederatedUserId>123456789012:Jean</FederatedUserId>
    </FederatedUser>
    <PackedPolicySize>6</PackedPolicySize>
  </GetFederationTokenResult>
  <ResponseMetadata>
    <RequestId>c6104cbe-af31-11e0-8154-cbc7ccf896c7</RequestId>
  </ResponseMetadata>
</GetFederationTokenResponse>
```

Note

`GetFederationToken` stores the policy in a packed format. `GetFederationToken` returns the size so you can adjust the calling parameters. For more information about the size constraints on the policy, go to GetFederationToken in the *AWS Security Token Service API Reference*.

If you prefer to grant permissions at the resource level, you can omit the *Policy* parameter. However, you should be aware that if you do not include a policy for the federated user, the temporary security credentials will not grant any permissions. In this case, you must use resource policies to grant the federated user access to your AWS resources.

For example, if your AWS account number is 111122223333, and you have an Amazon S3 bucket that you want to allow Susan to access even though her temporary security credentials don't include a policy for the bucket, you would need to ensure that the bucket has a policy with an ARN that matches Susan's ARN, such as arn:aws:sts::111122223333:federated-user/Susan.

Related Topics

- GetFederationToken in the *AWS Security Token Service API Reference*
- Making Query Requests
- Controlling Permissions in Temporary Security Credentials (p. 13)
- Disabling Permissions Granted Through Temporary Security Credentials (p. 16)
- Overview of Policies

Granting an IAM User Permission to Create Credentials for a Federated User

IAM users do not have permission to create temporary security credentials for federated users by default. To grant an IAM user or group of users permission to create temporary security credentials for federated users, you attach a policy to the IAM user or group that gives it access to the AWS STS `GetFederationToken` action.

This section includes two examples of granting a user access to `GetFederationToken`. The first example shows a basic policy you might use to grant an IAM user general permission to access `GetFederationToken`. The second example shows a policy you might use when you want to mitigate risk by granting a user limited permissions to use GetFederationToken.

Example of a policy granting a user permission to create temporary security credentials for a federated user

The following example is a simple policy granting an IAM user permission to access `GetFederationToken`.

```
{
 "Statement": [{
 "Effect": "Allow",
 "Action": "sts:GetFederationToken",
 "Resource":"*"
 }]
}
```

Important

When you give an IAM user permission to create temporary security credentials for federated users, you should be aware that this enables the IAM user to delegate its own permissions. For more information on delegating permissions across IAM users and AWS accounts, see Enabling

Cross-Account Access. For more information about controlling permissions in temporary security credentials, see Controlling Permissions in Temporary Security Credentials (p. 13).

Example of granting a user limited permission to create temporary security credentials for federated users

Although you can use the above policy to grant an IAM user permission to create temporary security credentials for federated users, as a best practice you might choose to ensure that the policy you use to grant permission is specific; that is, the policy should restrict the permissions of the IAM user to use `GetFederationToken` to issue temporary security credentials. The policy you use might look like the following example policy.

In this example, the IAM user needs to create temporary security credentials only for federated users whose names start with *Manager*. Because the admin wants to limit the IAM user's permissions so that it can create temporary security credentials that give only the minimum amount of access necessary, the admin attaches the following policy to the IAM user. The Resource ARN answers the question, "What kind of user names can this user create tokens for?" This policy gives the IAM user permission to create temporary security credentials only for federated user names that begin with *Manager*.

```
{
 "Statement": [{
 "Effect": "Allow",
 "Action": "sts:GetFederationToken",
 "Resource":["arn:aws:sts::123456789012:federated-user/Manager*"]
 }]
}
```

Related Topics

- Managing IAM Policies
- Identifiers for IAM Entities

Controlling Permissions in Temporary Security Credentials

Topics

Permissions associated with temporary security credentials are established when the credentials are created, and once the credentials are issued, you cannot revoke them. However, if necessary, you can update permissions after the temporary security credentials are created. This section describes what you need to know about granting permissions in temporary security credentials, and describes how to update or disable permissions after temporary security credentials have been issued.

Permissions in Temporary Security Credentials for IAM Users

When an IAM user calls the AWS Security Token Service (STS) GetSessionToken API action to create temporary security credentials for itself, the credentials returned by GetSessionToken include all the same permissions as those held by that IAM user. The user can access only what it already has permission to access. This is true for any IAM user with AWS credentials.

For more information about IAM user permissions and policies, see Overview of Permissions.

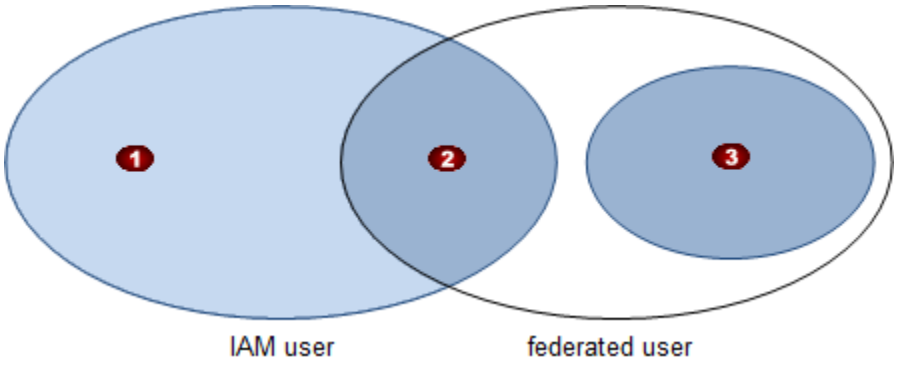
Controlling Permissions in Temporary Security Credentials for Federated Users

When an IAM user calls the GetFederationToken API action to create temporary security credentials for a federated user, the IAM user can optionally pass in a policy to grant the federated user a specific set of permissions. If the temporary security credentials do not include a policy, the federated user has no permissions except any permissions specified in resource policies.

If the temporary security credentials do include a policy, the permissions applied to the federated user are based on the following:

- The policy associated with the temporary security credentials
- Any resource policies that explicitly give access to the federated user
- Any policies associated with the IAM user who created the temporary security credentials

The following diagram illustrates how permissions work for federated users.



1	The IAM user who creates the temporary security credentials for the federated user is represented by the first oval, on the left. The federated user is the second oval, on the right. The IAM user's permissions are represented by the entire area within the oval, including the area of intersection (areas 1 and 2).
2	The IAM user can delegate permissions to the federated user only to the extent that he has permissions. In this diagram, the IAM user delegates only a portion of his permissions to the federated user. The delegated permissions are indicated by the intersection of the ovals (area 2).
3	In this diagram, there is a resource policy that explicitly grants permissions to the federated user, indicated by the third oval (area 3). As a result, the federated user's permissions include the permissions granted by the IAM user (area 2), as well as the permissions granted in the resource policy (area 3).

These permissions are evaluated together at the time of authorization, and the evaluation logic never results in a conflict. There is always a true/false result that either allows or denies the requested access. For more information about how permissions are evaluated, see Evaluation Logic in *Using AWS Identity and Access Management*.

Example of using policies to control how temporary security credentials are created and how permissions are delegated

Suppose that an IAM user named Issuer wants to grant read-only permissions to federated users for the AWS account's Amazon S3 buckets. The AWS account administrator (the admin) could scope the Amazon S3 permissions of Issuer with the following IAM user policy.

```
{
  "Statement": [{
    "Effect": "Allow",
    "Action": ["s3:GetObject", "sts:GetFederationToken"]
```

```
    "Resource": "arn:aws:s3:::mybucket/federated-user/*"
  }]
}
```

When Issuer wants to delegate permissions to a federated user, Issuer can call GetFederationToken with the name of the federated user, the duration the token is valid, and a policy granting access to Amazon S3, minimally. Any temporary security credentials Issuer creates will enable a federated user to read from the Amazon S3 bucket as long as the temporary security credentials are valid.

Issuer can restrict the permissions he delegates. For example, if Issuer wants to restrict the permissions he delegates based on the name of the user, so that a user named Jill is able to read her own files but not the files of any other federated user, then Issuer could do this by passing in a policy when he calls GetFederationToken. The policy that Issuer passes in might look like the following example.

```
{
  "Statement": [{
    "Effect": "Allow",
    "Action": "s3:GetObject",
    "Resource": "arn:aws:s3:::mybucket/federated-user/Jill/*"
  }]
}
```

In this way, Issuer can delegate an arbitrary number of permissions, leveraging the hierarchical structure of Amazon S3 object names.

Note

If conditions exist in either policy, the conditions must be satisfied by the request for authorization to succeed. So, for example, if Issuer can make Amazon S3 requests only subject to an aws:SourceIp condition, that condition also applies to calls made with temporary security credentials issued by Issuer.

Additionally, because AWS checks permissions at authorization time rather than when the temporary security credentials are created, permissions are impacted by the policies in effect when a request is made, regardless of the policies in effect when the temporary security credentials were created. So in this example, Issuer might create temporary security credentials with the following general wildcard permissions.

```
{
  "Statement": [{
    "Effect": "Allow",
    "Action": "*",
    "Resource": "*"
  }]
}
```

As a result, even though Issuer doesn't have these general permissions, Issuer can successfully call GetFederationToken and receive temporary security credentials with this policy. Jill will then have whatever permissions Issuer has when her temporary security credentials are authenticated by AWS, but no more. In this case, if Jill were to attempt to delete an Amazon S3 bucket, the request would fail because Issuer does not have permission to delete a bucket.

If after issuing the temporary security credentials to Jill, the AWS account owner wanted to revoke Jill's permissions, but no one else's, the account owner could use a deny policy like the one in the following example. To effect this policy, the AWS account owner would attach this policy to Issuer (the IAM user who created the temporary security credentials). For the purposes of this policy, it doesn't matter that

IAM doesn't know who Jill is; the policy works because it denies the permission of Issuer to enable the Amazon S3 GetObject action for Jill.

```
{
  "Statement": [{
    "Effect": "Deny",
    "Action": "s3:GetObject",
    "Resource": "arn:aws:s3:::mybucket/federated/Jill/*"
  }]
}
```

Note

You should also be aware that delegation works across accounts, from an account to an IAM user under the account, and from an IAM user to a set of temporary security credentials. For more information on delegating permissions across accounts, see Enabling Cross-Account Access in *Using AWS Identity and Access Management*.

Related Topics

- Overview of Policies in *Using AWS Identity and Access Management*
- Disabling Permissions Granted Through Temporary Security Credentials (p. 16)

Disabling Permissions Granted Through Temporary Security Credentials

Temporary security credentials are valid until they expire, and they cannot be revoked. However, because policies are evaluated at the time of authentication, and because the permissions delegated through temporary security credentials cannot exceed the permissions of the IAM user who created them, there are ways you can disable access rights of a user who possesses temporary security credentials after those credentials have already been issued. These are a couple of the methods you use to disable a user's access rights:

- Create a resource policy that denies access to the user who created the temporary security credentials.
- Create an IAM user policy for the user who created the temporary security credentials that denies access to a specific resource by a federated user.

Each of these methods is described in the following sections.

Note

When you update existing policy permissions, or when you apply a new policy to a user or a resource, in some cases the permissions might not take effect immediately. For some AWS products, it may take a few minutes for policy updates to take effect.

For more information about how IAM evaluates permissions in the context of temporary security credentials, see Controlling Permissions in Temporary Security Credentials (p. 13). For general information about how IAM evaluates permissions, see Overview of Permissions in *Using AWS Identity and Access Management*.

Example of creating a resource policy that denies access to the user who created the temporary security credentials

To deny access to a user who has temporary security credentials that have not already expired, you can deny access to the IAM user who created the credentials. In the case of IAM users, the holder of the temporary security credentials and the creator are the same identity. In the case of federated users, denying access to the user who *created* the temporary security credentials is effective because the permissions granted to the federated user cannot exceed the permissions of the IAM user who created the temporary security credentials.

In the following policy example, if the AWS account owner applied this resource policy to his Amazon Simple Queue Service (Amazon SQS) queue, the IAM user named John could not send messages from the queue, and neither could any federated users who have temporary security credentials created by John.

```
{
  "Version": "2008-10-17",
  "Id": "Queue1_Policy_UUID",
  "Statement": [{
    "Principal": "arn:aws:iam::111122223333:user/John"
    "Effect": "Deny",
    "Action": "sqs:SendMessage",
    "Resource": "arn:aws:sqs:us-east-1:111122223333:myqueue"
  }]
}
```

Important

For this reason, never use your root credentials to create temporary security credentials. This way, in the case of an emergency during which you need to deny access to your resources, as a last resort you can disable the IAM user who created the temporary security credentials, and you will deny access via any temporary security credentials the IAM user created. You can modify permissions for the issuing user without affecting your root account. For information about modifying user permissions, see Managing IAM Policies in *Using AWS Identity and Access Management*.

Example of creating an IAM user policy that denies access to a specific resource by a federated user

If, after issuing temporary security credentials to a federated user, the AWS account owner wants to revoke the user's permissions, but no one else's, the account owner could use a deny policy like the one in the following example.

To effect this policy, the AWS account owner would attach this policy to the IAM user who *created* the temporary security credentials. For this policy, it doesn't matter that IAM doesn't know who Jill is; the policy works because it denies the permission of the IAM user who created the temporary security credentials to delegate access to the Amazon S3 GetObject action for Jill.

```
{
  "Statement": [{
    "Effect": "Deny",
    "Action": "s3:GetObject",
    "Resource": "arn:aws:s3:::mybucket/federated-user/Jill/*"
  }]
}
```

Note

You should also be aware that delegation works across accounts, from an account to an IAM user under the account, and from an IAM user to temporary security credentials. For more information on delegating permissions across accounts, see Enabling Cross-Account Access in *Using AWS Identity and Access Management*.

Using Temporary Security Credentials to Access AWS

As described in Creating Temporary Security Credentials (p. 7), when you create temporary security credentials, the AWS Security Token Service (STS) API returns temporary security credentials consisting of a token, an Access Key ID, and a Secret Access Key. To give a user access to your resources, you distribute the temporary security credentials to the user you are granting temporary access to. You can distribute these credentials manually or programmatically. When the user makes calls to your resources, the user passes in the token and Access Key ID, and signs the request with the Secret Access Key. The token will not work with different access keys.

How the user passes in the token depends on the API and version of the AWS product the user is making calls to. The following table describes the AWS products that currently support temporary security credentials, and provides links to information about how to make a call with temporary security credentials using each product API. If service-specific information is not available, general information about using temporary security credentials is provided in the following section, Using Temporary Security Credentials to Authenticate an AWS Request (p. 20).

Note

The AWS services listed in the following table also support using temporary security credentials to access AWS through the AWS Management Console.

AWS product	For more information, see...
Amazon CloudFront	Using Temporary Security Credentials to Authenticate an AWS Request (p. 20), below
Amazon CloudWatch	Using Temporary Security Credentials to Authenticate an AWS Request (p. 20), below
Amazon DynamoDB	Using Temporary Security Credentials to Authenticate an AWS Request (p. 20), below
Amazon EC2	Using Temporary Security Credentials in the *Amazon Elastic Compute Cloud User Guide*
Amazon ElastiCache	Using Temporary Security Credentials to Authenticate an AWS Request (p. 20), below

AWS product	For more information, see...
Amazon RDS	Using Temporary Security Credentials to Authenticate an AWS Request (p. 20), below
Amazon Route 53	Using Temporary Security Credentials to Authenticate an AWS Request (p. 20), below
Amazon S3	Making Requests Using IAM User Temporary Credentials or Making Requests Using Federated User Temporary Credentials in the *Amazon Simple Storage Service Developer Guide*
Amazon SES	Using Temporary Security Credentials to Authenticate an AWS Request (p. 20), below
Amazon SimpleDB	Using Temporary Security Credentials in the *Amazon SimpleDB Developer Guide*
Amazon SNS	Using Temporary Security Credentials in the *Amazon Simple Notification Service Getting Started Guide*
Amazon SQS	Using Temporary Security Credentials in the *Amazon Simple Queue Service Developer Guide*
Amazon SWF	Using Temporary Security Credentials to Authenticate an AWS Request (p. 20), below
Amazon VPC	Using Temporary Security Credentials to Authenticate an AWS Request (p. 20), below
AWS Storage Gateway	Using Temporary Security Credentials to Authenticate an AWS Request (p. 20), below
Elastic Load Balancing	Using Temporary Security Credentials to Authenticate an AWS Request (p. 20), below

Note

You can give your IAM users the ability to create temporary security credentials, but for security reasons, users cannot use temporary security credentials to access IAM or AWS STS.

Using Temporary Security Credentials to Authenticate an AWS Request

The way you format a request for an AWS service varies depending on the service. For most services, you will need to do the following:

- Replace your usual AWSAccessKeyId parameter with the user Access Key ID provided by IAM
- Add the IAM SecurityToken as a new parameter
- Sign the request with the user SecretKeyId provided by IAM

If you send requests using expired credentials AWS denies the request.

Your request might look like the following example. This example shows the wire protocol for using temporary security credentials to authenticate an Amazon SimpleDB request over HTTPS.

```
https://sdb.amazonaws.com/
?Action=GetAttributes
&AWSAccessKeyId=Access Key ID provided by AWS Security Token Service
&DomainName=MyDomain
&ItemName=MyItem
&SignatureVersion=2
&SignatureMethod=HmacSHA256
&Timestamp=2010-01-25T15%3A03%3A07-07%3A00
&Version=2009-04-15
&Signature=Signature calculated using the SecretKeyId provided by AWS Security
 Token Service
&SecurityToken=Security Token Value
```

Note

AWS provides support for temporary security credentials and session tokens in the AWS SDKs so you can implement temporary security credentials or session tokens with a specific programming language. Each SDK has its own instructions for implementing this feature. For a current list of AWS SDKs that support this feature, see Ways to Access the AWS Security Token Service (p. 6).

Giving Federated Users Direct Access to the AWS Management Console

Topics

You can give your federated users single sign-on access to the AWS Management Console through your identity and authorization system, without requiring users to sign into Amazon Web Services (AWS). To give your federated users single sign-on access, you create a URL that gives them secure and direct access to the AWS Management Console.

To create the URL you need to complete the following tasks:

- Verify that the user is authenticated.
- Create temporary security credentials for the user.
- Construct the URL that passes the temporary security credentials to the AWS Management Console.
- Distribute the URL to the user.

The URL is valid for 15 minutes from the time it is created. The temporary security credentials associated with the URL are valid for the duration you specified when you created them, starting from the time they were created.

> **Important**
>
> Keep in mind that the URL grants access to your AWS resources through the AWS Management Console, to the extent that you have enabled permissions in the associated temporary security credentials. For this reason, you should treat the URL as a secret. We recommend returning the URL through a secure redirect, for example, by using a 302 HTTP response status code over an SSL connection. For more information about the 302 HTTP response status code, go to RFC 2616, section 10.3.3.

To complete these tasks you can use the HTTPS Query APIs for AWS Identity and Access Management (IAM) and the AWS Security Token Service (STS). Or, you can use the Java or Ruby programming languages. Each of these methods is described in the following sections.

Constructing the URL for the AWS Management Console (Query APIs)

This topic describes how to construct a URL that gives your federated users direct access to the AWS Management Console. This task uses the AWS Identity and Access Management (IAM) and AWS Security Token Service (STS) HTTPS Query APIs. For more information about making Query requests, go to Making Query Requests in *Using IAM*.

Note

The following procedure contains examples of text strings. To enhance readability, line breaks have been added to some of the longer examples. When you create these strings for your own use, you should omit any line breaks.

To give a federated user access to your resources from the AWS Management Console

1. Authenticate the user in your identity and authorization system.
2. Create temporary security credentials for the user. The credentials consist of an Access Key ID, a Secret Access Key, and a security token. For more information about creating temporary credentials, see Creating Temporary Security Credentials (p. 7).

   Important

   When you create temporary security credentials you must specify the permissions the credentials will grant to the user who holds them. For more information about controlling permissions in temporary security credentials, see Controlling Permissions in Temporary Security Credentials (p. 13).
3. After you obtain the temporary security credentials, you format them as a JSON string so that you can exchange them for a sign-in token. The following example shows how to encode the temporary security credentials. You replace the red text with the appropriate values from the temporary security credentials that you create.

```
{"sessionId":"*** AWS Access Key ID ***",
"sessionKey":"*** AWS Secret Access Key ***",
"sessionToken":"*** AWS security token ***"}
```

4. Next, make a request to the AWS federation endpoint (https://signin.aws.amazon.com/federation) with the Action and Session parameters, shown in the following example.

```
Action = getSigninToken
Session = *** the JSON string described in Step 3, form-urlencoded ***
```

   The following string is an example of what your request might look like.

```
https://signin.aws.amazon.com/federation?
Action=getSigninToken
&Session=%7B%22sessionId%22%3A%22ASIAEXAMPLEMDLUUAEYQ%22%2C%22sessionKey%22
```

```
%3A%22tpSl9thxr2PkEXAMPLETAnVLVGdwC5zXtGDr%2FqWi%22%2C%22sessionToken%22%3A
%22AQoDYXdzEXAMPLE4BrM96BJ7btBQRrAcCjQIbg55555555OBT7y8h2YJ7woJkRzsLpJBpklC
qPXxS2AjRorJAm%2BsBtv1YXlZF%2FfHljgORxOevE388GdGaKRfO9W4DxK4HU0fIpwL%2BQ7oX
2Fj%2BJa%2FAb5u0cL%2BzI1P5rJuDzH%2F0pWEiYfiWXXH20rWruXVXpIIO%2FPhMHlV3Jw%2B
gDc4ZJ0WItuLPsuyP7BVUXWLcAVyTFbxyLy36FBSXF1z8a%2FvJN7utcj0mJRGIiIZSV7FQuepa
WP5YARYMrOUMqBB3v308LKBU8Z0xYe2%2FqthrLXf1nX0njbU%2FJTrct%2BEdG9PRb3907qa5n
VbnnnxdVQJ3mPgQchAZpDI9LsDDbGsa67JHUyFYnyUUUkMRfe7G70gjvbz9gQ%EXAMPLE
```

The response is a JSON document with an `SigninToken` value. It will look similar to the following example.

```
{"SigninToken":"*** the SigninToken string ***"}
```

5. Finally, you create the URL that your federated users will use to access the AWS Management Console. The URL is the federation URL endpoint (https://signin.aws.amazon.com/federation), plus the following parameters.

```
Action = login
Issuer = *** the form-urlencoded URL for your internal sign-in page ***
Destination = *** the desired AWS Management Console URL, also
 form-urlencoded ***
SigninToken = *** the value of SigninToken from the JSON document returned
 in Step 4 ***
```

The following example shows what the final URL might look like. The URL is valid for 15 minutes from the time it is created. The temporary security credentials associated with the URL are valid for the duration you specified when you created them.

```
https://signin.aws.amazon.com/federation?
Action=login
&Issuer=https%3A%2F%2Fexample.com
&Destination=https%3A%2F%2Fconsole.aws.amazon.com%2Fs
&SigninToken=VCQgs5qZZt3Q6fn8Tr5EXAMPLEmLnwB7JjUc-SHwnUUWabcRdnWsi4DBn-dvC
CZ85wrD0nmldUcZEXAMPLE-vXYH4Q__mleuF_W2BE5HYexbe9y4Of-kje53SsjNNecATfjIzpW1
WibbnH6YcYRiBoffZBGExbEXAMPLE5aiKX4THWjQKC6gg6alHu6JFrnOJoK3dtP6I9a6hi6yPgm
iOkPZMmNGmhsvVxetKzr8mx3pxhHbMEXAMPLETv1pij0rok3IyCR2YVcIjqwfWv32HU2Xlj471u
3fU6uOfUComeKiqTGX974xzJOZbdmX_t_lLrhEXAMPLEDDIisSnyHGw2xaZZqudm4mo2uTDk9Pv
9l5K0ZCqIgEXAMPLEcA6tgLPykEWGUyH6BdSC6166n4M4JkXIQgac7_7821YqixsNxZ6rsrpzwf
nQoS14O7R0eJCCJ684EXAMPLEZRdBNnuLbUYpz2Iw3vIN0tQgOujwnwydPscM9F7foaEK3jwMkg
Apeb1-6L_OB12MZhuFxx55555EXAMPLEhyETEd4ZulKPdXHkgl6T9ZkIlHz2Uy1RUTUhhUxNtSQ
nWc5xkbBoEcXqpoSIeK7yhje9Vzhd61AEXAMPLElbWeouACEMG6-Vd3dAgFYd6i5FYoyFrZLWvm
0LSG7RyYKeYN5VIzUk3YWQpyjP0RiT5KUrsUi-NEXAMPLExMOMdoODBEgKQsk-iu2ozh6r8bxwC
RNhujg
```

Constructing the URL for the AWS Management Console (Java)

This topic describes how to programmatically construct a URL that gives your federated users direct access to the AWS Management Console. The following code snippet uses the AWS SDK for Java. You replace the red text with the appropriate values for your use case.

```
import java.net.URLEncoder;
import java.net.URL;
import java.net.URLConnection;
import java.io.BufferedReader;
import java.io.InputStreamReader;
// Available at http://www.json.org/java/index.html
import org.json.JSONObject;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.services.securitytoken.AWSSecurityTokenServiceClient;
import com.amazonaws.services.securitytoken.model.Credentials;
import com.amazonaws.services.securitytoken.model.GetFederationTokenRequest;
import com.amazonaws.services.securitytoken.model.GetFederationTokenResult;

AWSCredentials credentials = new BasicAWSCredentials(
  "*** Access Key ID ***",
  "*** Secret Key ***");
AWSSecurityTokenServiceClient stsClient =
  new AWSSecurityTokenServiceClient(credentials);

GetFederationTokenRequest getFederationTokenRequest =
  new GetFederationTokenRequest();
getFederationTokenRequest.setDurationSeconds(3600);
getFederationTokenRequest.setName("UserName");

// A sample policy for accessing Amazon SNS in the console.
String policy = "{\"Statement\":[{\"Action\":\"sns:*\"," +
  "\"Effect\":\"Allow\",\"Resource\":\"*\"}]}";

getFederationTokenRequest.setPolicy(policy);

GetFederationTokenResult federationTokenResult =
  stsClient.getFederationToken(getFederationTokenRequest);

Credentials federatedCredentials = federationTokenResult.getCredentials();

// The issuer parameter specifies your internal sign-in
// page, for example https://mysignin.internal.mycompany.com/.
// The console parameter specifies the URL to the destination console of the
// AWS Management Console. This example goes to the Amazon SNS console.
// The signin parameter is the URL to send the request to.
String issuerURL = "https://mysignin.internal.mycompany.com/";
String consoleURL = "https://console.aws.amazon.com/sns";
String signInURL = "https://signin.aws.amazon.com/federation";

// Create the sign-in token using temporary credentials,
// including the Access Key ID,  Secret Access Key, and security token.
String sessionJson = String.format(
  "{\"%1$s\":\"%2$s\",\"%3$s\":\"%4$s\",\"%5$s\":\"%6$s\"}",
  "sessionId", federatedCredentials.getAccessKeyId(),
  "sessionKey", federatedCredentials.getSecretAccessKey(),
  "sessionToken", federatedCredentials.getSessionToken());

String getSigninTokenURL = signInURL + "?Action=getSigninToken" +
  "&SessionType=json&Session=" + URLEncoder.encode(sessionJson,
  "UTF-8");
URL url = new URL(getSigninTokenURL);
URLConnection conn = url.openConnection ();
```

```
BufferedReader bufferReader = new BufferedReader(new
  InputStreamReader(conn.getInputStream()));
String returnContent = bufferReader.readLine();
String signinToken = new JSONObject(returnContent).getString("SigninToken");

String signinTokenParameter = "&SigninToken=" +
  URLEncoder.encode(signinToken,"UTF-8");

// The issuer parameter is optional, but recommended. Use it to direct users
// to your sign-in page when their session expires.
String issuerParameter = "&Issuer=" + URLEncoder.encode(issuerURL, "UTF-8");
String destinationParameter = "&Destination=" +
  URLEncoder.encode(consoleURL,"UTF-8");
String loginURL = signInURL + "?Action=login" + signinTokenParameter +
  issuerParameter + destinationParameter;
```

Constructing the URL for the AWS Management Console (Ruby)

This topic describes how to programmatically construct a URL that gives your federated users direct access to the AWS Management Console. This code snippet uses the AWS SDK for Ruby.

```
require 'rubygems'
require 'json'
require 'open-uri'
require 'cgi'
require 'aws-sdk'

# Normally, the temporary credentials will come from your identity
# broker, but for this example we create them here
sts = AWS::STS.new(:access_key_id => "*** Your AWS Access Key ID ***",
  :secret_access_key => "*** Your AWS Secret Access Key ***")

# A sample policy for accessing Amazon SNS in the console.
policy = AWS::STS::Policy.new
policy.allow(:actions => "sns:*",:resources => :any)

session = sts.new_federated_session(
  "UserName",
  :policy => policy,
  :duration => 3600)

# The issuer parameter specifies your internal sign-in
# page, for example https://mysignin.internal.mycompany.com/.
# The console parameter specifies the URL to the destination console of the
# AWS Management Console. This example goes to the Amazon SNS console.
# The signin parameter is the URL to send the request to.
issuer_url = "https://mysignin.internal.mycompany.com/"
console_url = "https://console.aws.amazon.com/sns"
signin_url = "https://signin.aws.amazon.com/federation"

# Create the sign-in token using temporary credentials,
# including the Access Key ID, Secret Access Key, and security token.
```

```
session_json = {
  :sessionId => session.credentials[:access_key_id],
  :sessionKey => session.credentials[:secret_access_key],
  :sessionToken => session.credentials[:session_token]
}.to_json

get_signin_token_url = signin_url + "?Action=getSigninToken" +
  "&SessionType=json&Session=" + CGI.escape(session_json)
returned_content = URI.parse(get_signin_token_url).read
signin_token = JSON.parse(returned_content)['SigninToken']
signin_token_param = "&SigninToken=" + CGI.escape(signin_token)

# The issuer parameter is optional, but recommended. Use it to direct users
# to your sign-in page when their session expires.
issuer_param = "&Issuer=" + CGI.escape(issuer_url)
destination_param = "&Destination=" + CGI.escape(console_url)

login_url = signin_url + "?Action=login" + signin_token_param +
  issuer_param + destination_param
```

AWS Security Token Service Sample Applications

The following sample applications are available for your reference. We will list additional links to sample applications as they become available.

- Identity Federation Sample Application for an Active Directory Use Case at *AWS Sample Code & Libraries*
- Authenticating Users of AWS Mobile Applications with a Token Vending Machine at *AWS Articles & Tutorials*

Document History

The following table describes the documentation for this release of the AWS Security Token Service.

- **API version—**2011-06-15
- **Last document update—**July 10, 2012

Change	Description	Release Date
MFA-Protected API Access	Introduced MFA-protected API access, a feature that enables you to add an extra layer of security over AWS APIs using AWS Multi-Factor Authentication (MFA), see Temporary Security Credentials for IAM Users with Multi-Factor Authentication (MFA) (p. 9).	July 10, 2012
Fixed API version in documentation	Corrected the API version displayed in Using Temporary Security Credentials. The API version of the AWS Security Token Service is not the same as the AWS Identity and Access Management IAM API version.	April 26, 2012
New Guide	This release introduces *Using Temporary Security Credentials*.	January 19, 2012