Federated Identity pattern

06/23/2017 • 7 minutes to read • Contributors

In this article

Context and problem
Solution
Issues and considerations
When to use this pattern
Example
Related guidance

Delegate authentication to an external identity provider. This can simplify development, minimize the requirement for user administration, and improve the user experience of the application.

Context and problem

Users typically need to work with multiple applications provided and hosted by different organizations they have a business relationship with. These users might be required to use specific (and different) credentials for each one. This can:

- **Cause a disjointed user experience**. Users often forget sign-in credentials when they have many different ones.
- **Expose security vulnerabilities**. When a user leaves the company the account must immediately be deprovisioned. It's easy to overlook this in large organizations.
- **Complicate user management**. Administrators must manage credentials for all of the users, and perform additional tasks such as providing password reminders.

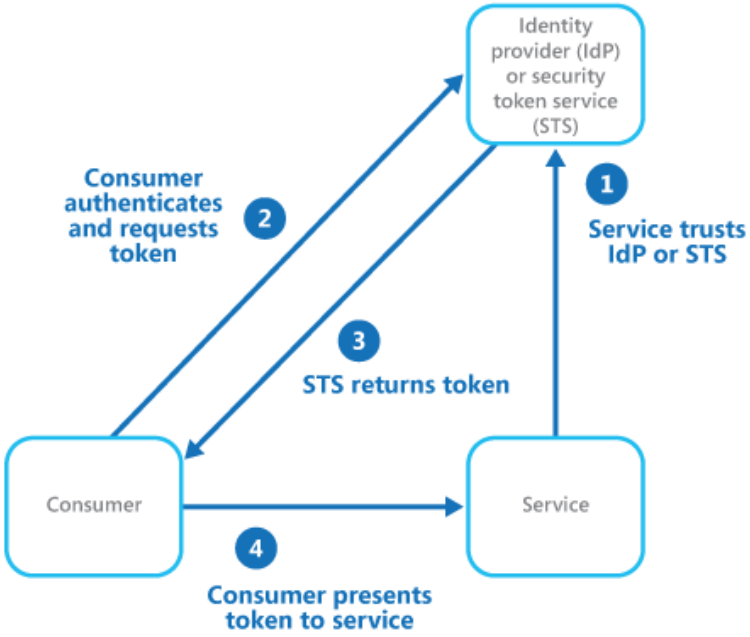
Users typically prefer to use the same credentials for all these applications.

Solution

Implement an authentication mechanism that can use federated identity. Separate user authentication from the application code, and delegate authentication to a trusted identity provider. This can simplify development and allow users to authenticate using a wider range of identity providers (IdP) while minimizing the administrative overhead. It also allows you to clearly decouple authentication from authorization.

The trusted identity providers include corporate directories, on-premises federation services, other security token services (STS) provided by business partners, or social identity providers that can authenticate users who have, for example, a Microsoft, Google, Yahoo!, or Facebook account.

The figure illustrates the Federated Identity pattern when a client application needs to access a service that requires authentication. The authentication is performed by an IdP that works in concert with an STS. The IdP issues security tokens that provide information about the authenticated user. This information, referred to as claims, includes the user's identity, and might also include other information such as role membership and more granular access rights.



This model is often called claims-based access control. Applications and services authorize access to features and functionality based on the claims contained in the token. The service that requires authentication must trust the IdP. The client application contacts the IdP that performs the authentication. If the authentication is successful, the IdP returns a token containing the claims that identify the user to the STS (note that the IdP and STS can be the same service). The STS can transform and augment the claims in the token based on predefined rules, before returning it to the client. The client application can then pass this token to the service as proof of its identity.

> There might be additional STSs in the chain of trust. For example, in the scenario described later, an on-premises STS trusts another STS that is responsible for accessing an identity provider to authenticate the user. This approach is common in enterprise scenarios where there's an on-premises STS and directory.

Federated authentication provides a standards-based solution to the issue of trusting identities across diverse domains, and can support single sign-on. It's becoming more common across all types of applications, especially cloud-hosted applications, because it supports single sign-on without requiring a direct network connection to identity providers. The user doesn't have to enter credentials for every application. This increases security because it prevents the creation of credentials required to access many different applications, and it also hides the user's credentials from all but the original identity provider. Applications see just the authenticated identity information contained within the token.

Federated identity also has the major advantage that management of the identity and credentials is the responsibility of the identity provider. The application or service doesn't need to provide identity management features. In addition, in corporate scenarios, the corporate directory doesn't need to know about the user if it trusts the identity provider. This removes all the administrative overhead of managing the user identity within the directory.

Issues and considerations

Consider the following when designing applications that implement federated authentication:

- Authentication can be a single point of failure. If you deploy your application to multiple datacenters, consider deploying your identity management mechanism to the same datacenters to maintain application reliability and availability.
- Authentication tools make it possible to configure access control based on role claims contained in the authentication token. This is often referred to as role-based access control (RBAC), and it can allow a more granular level of control over access to features and resources.
- Unlike a corporate directory, claims-based authentication using social identity providers doesn't usually provide information about the authenticated user other than an email address, and perhaps a name. Some social identity providers, such as a Microsoft account, provide only a unique identifier. The application usually needs to maintain

some information on registered users, and be able to match this information to the identifier contained in the claims in the token. Typically this is done through registration when the user first accesses the application, and information is then injected into the token as additional claims after each authentication.

- If there's more than one identity provider configured for the STS, it must detect which identity provider the user should be redirected to for authentication. This process is called home realm discovery. The STS might be able to do this automatically based on an email address or user name that the user provides, a subdomain of the application that the user is accessing, the user's IP address scope, or on the contents of a cookie stored in the user's browser. For example, if the user entered an email address in the Microsoft domain, such as user@live.com, the STS will redirect the user to the Microsoft account sign-in page. On later visits, the STS could use a cookie to indicate that the last sign in was with a Microsoft account. If automatic discovery can't determine the home realm, the STS will display a home realm discovery page that lists the trusted identity providers, and the user must select the one they want to use.

When to use this pattern

This pattern is useful for scenarios such as:

- **Single sign-on in the enterprise**. In this scenario you need to authenticate employees for corporate applications that are hosted in the cloud outside the corporate security boundary, without requiring them to sign in every time they visit an application. The user experience is the same as when using on-premises applications where they're authenticated when signing in to a corporate network, and from then on have access to all relevant applications without needing to sign in again.

- **Federated identity with multiple partners**. In this scenario you need to authenticate both corporate employees and business partners who don't have accounts in the corporate directory. This is common in business-to-business applications, applications that integrate with third-party services, and where companies with different IT systems have merged or shared resources.

- **Federated identity in SaaS applications**. In this scenario independent software vendors provide a ready-to-use service for multiple clients or tenants. Each tenant authenticates using a suitable identity provider. For example, business users will use their corporate credentials, while consumers and clients of the tenant will use their social identity credentials.

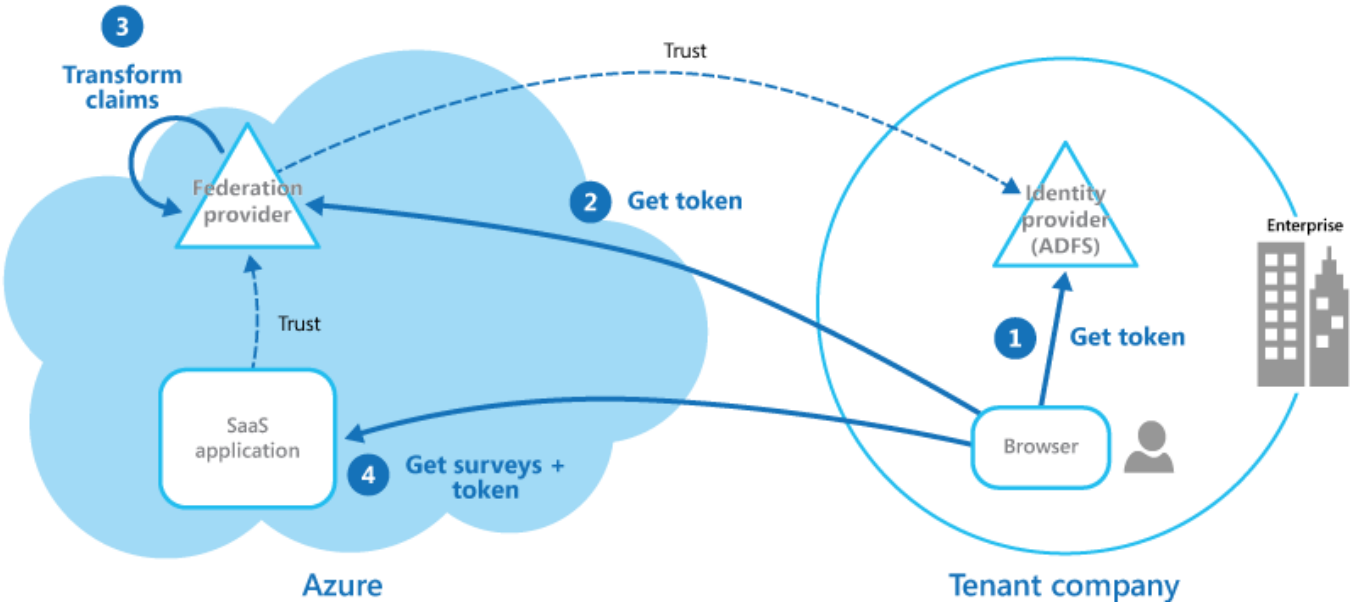
This pattern might not be useful in the following situations:

- All users of the application can be authenticated by one identity provider, and there's no requirement to authenticate using any other identity provider. This is typical in business applications that use a corporate directory (accessible within the application) for authentication, by using a VPN, or (in a cloud-hosted scenario) through a virtual network connection between the on-premises directory and the application.

- The application was originally built using a different authentication mechanism, perhaps with custom user stores, or doesn't have the capability to handle the negotiation standards used by claims-based technologies. Retrofitting claims-based authentication and access control into existing applications can be complex, and probably not cost effective.

Example

An organization hosts a multi-tenant software as a service (SaaS) application in Microsoft Azure. The application includes a website that tenants can use to manage the application for their own users. The application allows tenants to access the website by using a federated identity that is generated by Active Directory Federation Services (ADFS) when a user is authenticated by that organization's own Active Directory.



The figure shows how tenants authenticate with their own identity provider (step 1), in this case ADFS. After successfully authenticating a tenant, ADFS issues a token. The client browser forwards this token to the SaaS application's federation provider, which trusts tokens issued by the tenant's ADFS, in order to get back a token that is valid for the SaaS federation provider (step 2). If necessary, the SaaS federation provider performs a transformation on the claims in the token into claims that the application recognizes (step 3) before returning the new token to the client browser. The application trusts tokens issued by the SaaS federation provider and uses the claims in the token to apply authorization rules (step 4).

Tenants won't need to remember separate credentials to access the application, and an administrator at the tenant's company can configure in its own ADFS the list of users that can access the application.

Related guidance

- Microsoft Azure Active Directory
- Active Directory Domain Services
- Active Directory Federation Services
- Identity management for multitenant applications in Microsoft Azure
- Multitenant Applications in Azure