Claims-Based Single Sign-On for Microsoft SharePoint 2010

* 1. This chapter walks you through an example of integrating two SharePoint Web applications into single-sign on (SSO) environment for intranet and extranet Web users who all belong to a single security realm. These users can already access other ASP.NET Web applications in the SSO environment. You'll see examples of SharePoint applications that Adatum has made claims-aware so that Adatum employees can access the SharePoint applications from the company intranet or from the Web.
  2. This basic scenario doesn't show how to establish a trust relationship between enterprises that would allow uses from another company to access the SharePoint site (that is discussed in chapter 12, "Federated Identity for SharePoint Applications.") It focuses on how to implement single sign-on and single sign-off within a security domain as a preparation for sharing resources with other security domains, and on how to configure SharePoint to use claims-based authentication and authorization. In short, this scenario contains the commonly used elements that will appear in all claims-aware SharePoint applications. For further information about integrating ASP.NET Web applications into an SSO environment and making them claims-aware, you should read chapter 3, "Claims-Based Single Sign-On for the Web."
  3. For additional information about SharePoint and claims-based identity, see Appendix F, "SharePoint 2010 Authentication Architecture and Considerations."

# The Premise

* 1. Adatum is a medium sized company that uses Microsoft Active Directory® AD to authenticate the employees in its corporate network. Adatum is planning to implement two applications as SharePoint 2010 Web applications that employees will access from both the intranet and the internet:
  2. A portal, named a-Portal, where Adatum stores the product documentation that's used by its sales force when they engage with customers. This SharePoint Web application consists of a single site collection based on the "Team Site" template.
  3. A web application, named a-Techs, where field staff access scheduling information, tasks, and technical data. It also includes a blog, where field technicians can capture tips and techniques to share with other team members (and possibly partners in the future). This SharePoint Web application consists of two site collections; one based on the "Team Site" template, and one based on the "Blog" template. This Web application also uses SharePoint user profile data.
  4. Adatum has already established an SSO environment that includes existing ASP.NET Web applications such as the a-Order and a-Expense applications. As part of this environment, Adatum has configured Active Directory Federation Services (ADFS) to act as an identity provider (IdP).

# Goals and Requirements

The goals of this scenario are to show how to configure a SharePoint environment to use a claims-based identity model to control access, and how to customize SharePoint to provide a way for a SharePoint farm administrator to effectively manage access to the claims-enabled SharePoint applications.

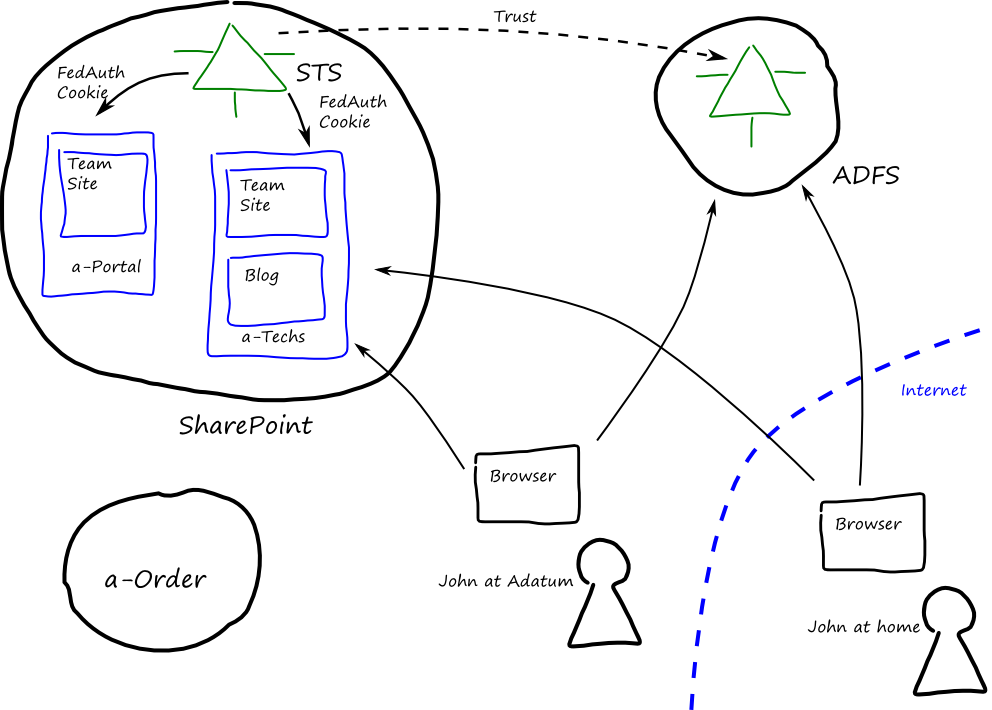
Configuring a SharePoint environment to use claims includes configuring the trust relationship between SharePoint and ADFS and configuring which claims ADFS passes to SharePoint.

Users must be able to access the SharePoint Web applications from both the intranet and internet as part of an SSO realm that includes other ASP.NET Web applications. The environment should also support single sign-out, so that logging out from any ASP.NET or SharePoint Web application logs the user out from all applications that are part of the SSO domain.

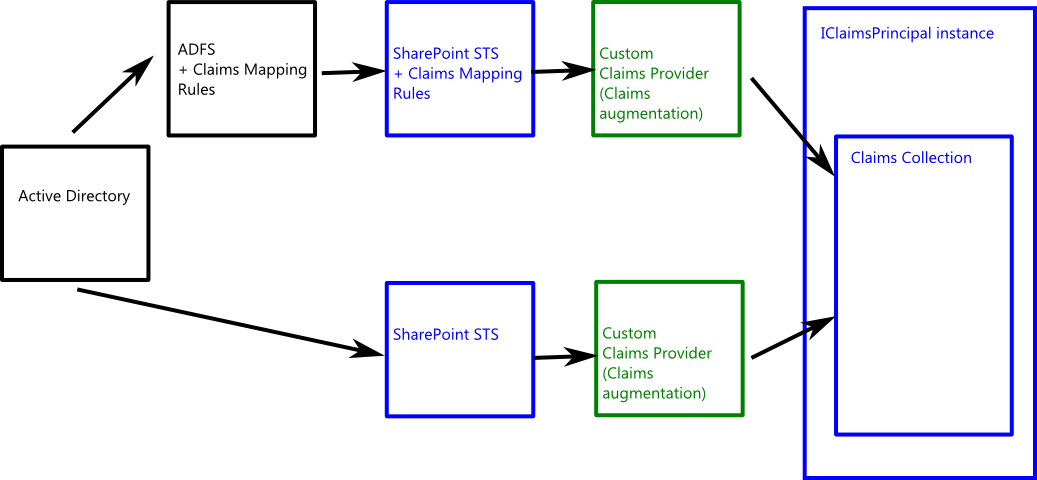
SharePoint site collection administrators should be able to control access to site collections and sites based on role memberships defined in AD. For example, only users in the Sales role should have access to the a-Portal Web application and only users in the Team Leader role should be able to post to the blog in the a-Techs application.

# Overview of the Solution

Adatum has created two claims-enabled SharePoint Web applications one for sales persons and one for field technical employees. These applications are available on the intranet and internet. The following diagram shows the main components of the solution suggested by Adatum.

* 1. 
  2. Figure 1
  3. Claims enabled SharePoint applications at Adatum
  4. **#!CALLOUT!#**
  5. In SharePoint, you configure an STS by creating a SharePoint trusted identity token issuer.

## Authentication Mechanism

1. Adatum has configured both SharePoint Web applications to use ADFS as a Trusted Identity Provider. Adatum has also configured ADFS to use different authentication types depending on where the user is accessing the applications from: intranet users will sign-in automatically using Integrated Windows Authentication (IWA) and internet users will enter their Adatum Windows credentials on a Web form. In this way, all users authenticate with AD through ADFS.
   1. #!SPOKENBY(Markus)!#
   2. During development, it's useful to be able to see the set of claims that a user has. See the section "Displaying Claims in a Web Part" for one way to do this.
2. An alternative approach that Adatum considered was to configure two authentication types in each Web application in SharePoint. SharePoint 2010 allows you to configure multiple authentication mechanisms for a single Web application; for example, you could configure a SharePoint Web application to use both Windows Authentication and a Trusted Identity Provider. Figure 2 shows the two alternative routes by which user attributes from Active Directory become claims belonging to a SharePoint user in this alternative scenario. The SharePoint STS is an instance of a SharePoint trusted identity token issuer, the custom claims providers are optional components.
   1. 
   2. Figure 2
   3. Building a user's claims collection
   4. #!SPOKENBY(Jana)!#
   5. You can use the claims augmentation offered by the custom claims providers to programmatically add additional claims to a user's claims set.
   6. The difficulty with this approach is that although both authentication mechanisms result in a set of claims for the **IClaimsPrincipalInstance** associated with the user, without additional code they are unlikely to generate the same types of claims. For example, the claims from Windows authentication will include **groupsid** claims, while the claims from the trusted identity provider will include **role** claims. An additional complexity of this approach is that you'll probably want to customize the page that SharePoint displays offering users a choice of authentication provider.

For an example of how a custom claims provider that converts SIDs to group names, see this blog post: http://blogs.technet.com/b/speschka/archive/2010/09/12/a-sharepoint-2010-claims-provider-to-convert-role-sids-to-group-names.aspx.

* 1. For an example of how to customize the default SharePoint page that presents a choice of authentication providers to the user, see this blog post: http://blogs.msdn.com/b/brporter/archive/2010/05/10/temp.aspx.
  2. For these reasons, Adatum selected the first approach that uses a single Trusted Identity Provider in SharePoint so that they can use the claims mapping rules in ADFS and ensure that a consistent set of claims reach SharePoint.

## End-to-End Walkthroughs

The following sections outline two scenarios for a user who accesses a claims-enabled SharePoint environment: the first scenario describes what happens when a user accesses two different site collections in the same SharePoint Web application, the second scenario describes what happens when a user accesses two SharePoint Web applications hosted in the same domain.

* 1. The walkthroughs below describe the experience of internet users who must provide their username and password to ADFS in order to authenticate. ADFS will not prompt intranet users (inside the corporate firewall) for their credentials, but will authenticate them using Windows Integrated Authentication: intranet users will not see the sign-in page at ADFS.

### Visiting two Site Collections in a SharePoint Web Application

* 1. In this walkthrough, John visits the Document Library and then the Team Site in the a-Techs SharePoint Web application.
  2. John browses to the Team site in the a-Techs SharePoint Web application.
  3. John has not yet been authenticated so SharePoint redirects John's browser to ADFS. This goes through several intermediate steps: the SharePoint authentication endpoint, and the SharePoint sign-in endpoint, before it arrives at ADFS.
  4. John enters his Adatum domain credentials; ADFS validates the credentials, creates a token that contains John's claims, and redirects the browser to the SharePoint STS (the "/\_trust/" endpoint in the SharePoint Web application references the trusted identity token issuer).
  5. The SharePoint STS validates the token from ADFS and issues a FedAuth cookie for the a-Techs SharePoint Web application. This cookie contains a reference to the token that contains John's claims: the token itself is stored in the SharePoint token cache.
  6. SharePoint checks that John has permissions to access to the Team site collection, and redirects his browser to the site (the "/\_layouts/Authenticate.aspx" endpoint in the SharePoint web application performs the permissions check).
  7. John browses to the Blog site in the a-Techs SharePoint Web Application. John does not require a new token for this site collection because it is part of the same SharePoint Web application.
  8. In chapter 12, "Federated Identity for SharePoint Applications," you can see a sequence diagram that illustrates this process in relation to sliding sessions.

### Visiting Two SharePoint Web Applications

* 1. In this walkthrough, John visits the a-Portal SharePoint Web application and then visits the a-Techs SharePoint Web application.
  2. John visits the a-Portal SharePoint Web application.
     1. John browses to the Team site in the a-Portal SharePoint Web application.
     2. John has not yet been authenticated so SharePoint redirects John's browser to ADFS.
     3. John enters his Adatum domain credentials; ADFS validates the credentials, issues a SAML token that contains John's claims, and redirects the browser to the SharePoint STS (the "/\_trust/" endpoint in the SharePoint Web application). ADFS also creates an SSO cookie so that it can recognize if it has already authenticated someone.
     4. The SharePoint STS validates the token from ADFS and issues a FedAuth cookie for the a-Portal SharePoint Web application that contains a reference to John's claims in the SharePoint token cache.
     5. SharePoint checks that John has access to the Team site collection, and redirects his browser to the site.
  3. John visits the a-Techs SharePoint Web application.
     1. John browses to the Team site in the a-Techs SharePoint Web application.
     2. John has not yet been authenticated for this SharePoint Web application so SharePoint redirects John's browser to ADFS.
     3. ADFS detects the SSO cookie that it issued in step 1.c, and redirects the browser with a new SAML token to the SharePoint STS.
     4. The SharePoint STS validates the token from ADFS and issues a FedAuth cookie for the a-Techs SharePoint Web application that contains a reference to John's claims in the SharePoint token cache.
     5. SharePoint checks that John has permissions to access to the Team site collection, and redirects his browser to the site.
  4. In this example, it's important to ensure that each SharePoint Web application uses its own FedAuth token. If the Web applications have different host names, this will happen automatically. However, if in a test environment the Web applications share the same host name, the second Web application will try to use the existing FedAuth token that is not valid in that Web application. Each Web application must have its own FedAuth token. See the section, "Setup and Physical Deployment," in this chapter for more details.

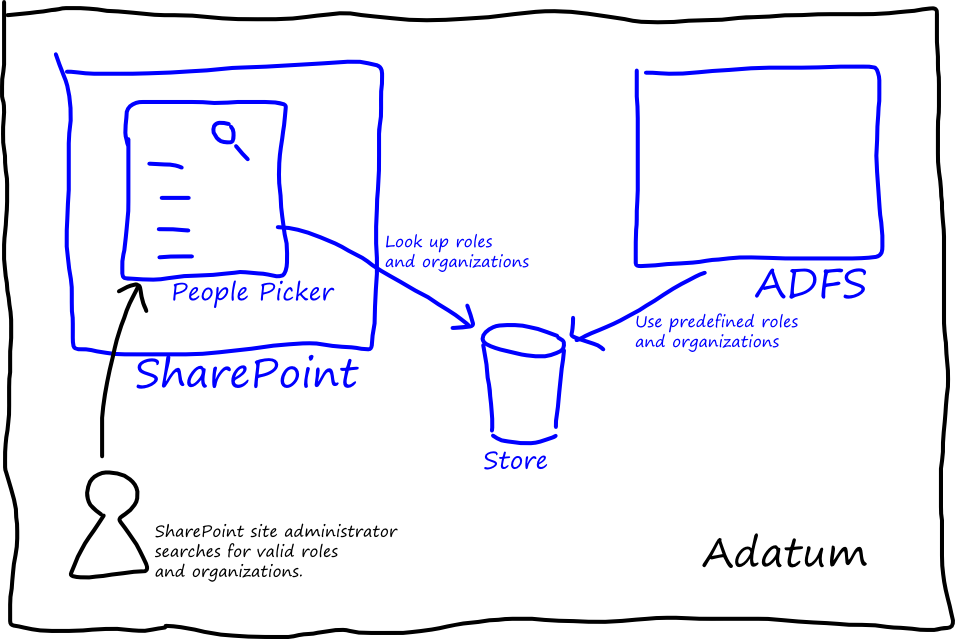
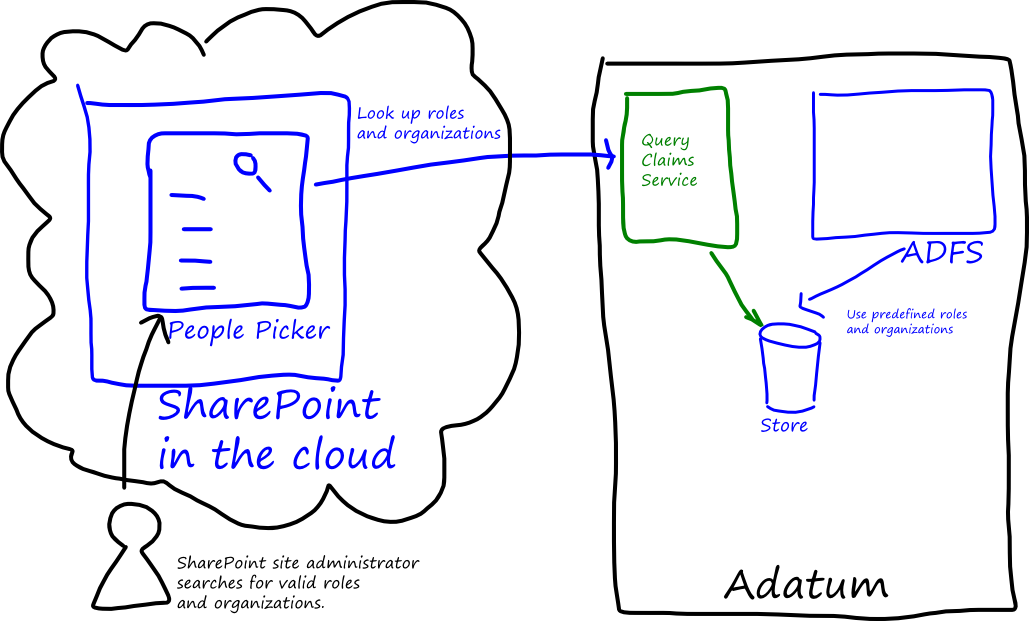
## Authorization in SharePoint

* 1. This scenario uses standard SharePoint groups to control access to the sites in the two SharePoint Web applications. The following table summarizes the permissions.

|  |  |  |  |
| --- | --- | --- | --- |
| Site | SharePoint Group | Permission level | Role Claim |
| a-Portal Team site | SalesSite Members | Contribute | sales |
| a-Techs Team site | TechSite Members | Contribute | techleaders |
| a-Techs Team site | TechSite Members | Contribute | techs |
| a-Techs Blog site | TechBlog Members | Contribute | techleaders |
| a-Techs Blog site | TechBlog Visitors | Read | techs |

* 1. In SharePoint, a site administrator can add users to a SharePoint group to grant those users the permissions associated with the group. In a claims-based environment, a site administrator can add users to a SharePoint group based on the user's claims; for example, a site administrator could add all authenticated users in the **sales** role to the SharePoint Site Members group by using the Site Permissions Tools.
  2. Mapping claims to SharePoint groups simplifies the administration tasks in SharePoint. There is no need to add individual users to SharePoint groups.
  3. Adatum has modified the SharePoint People Picker to make it easier for site administrators to map role and organization claims to SharePoint groups.
  4. If your identity provider does not provide the claims that you need to implement your authorization rules, you can use claims augmentation in the SharePoint STS to modify existing claim values or to add additional claims to an authenticated user.

## The People Picker

* 1. It is difficult for site administrators at Adatum to use the default people picker to reliably assign permissions in the a-Portal and a-Techs Web applications. The default behavior of the people picker is to allow the user to enter part of a user name or group name and then use the search function to locate the user or group. In a claims enabled SharePoint Web application this does not work as expected because there is no repository of users and groups for the people picker to search: the only information SharePoint has is the claims data associated with the current user. The default people picker implementation works around this by always finding a match and resolving the name even if the name is incorrect, which makes it easy for an administrator to make a mistake. For example, the site administrator would like to assign a permission to anyone in the **techs** role, if the site administrator makes a typing mistake and searches for **techz** in the people picker he will get a match and be able to assign a permission to a non-existent role.
  2. #!SPOKENBY(Bharath)!#
  3. In a claims-enabled application, the application receives a set of claims from a trusted issuer about the person accessing the application. This contrasts with the approach where the application queries a directory service to discover information about the user. The claims-based approach is much more flexible: the claims can come from many different issuers and be used in a federated identity environment. However, in a claims-based scenario the application may not have direct access to lists of users in a directory.
  4. To prevent this type of error, Adatum implemented a custom **SPClaimsManager** component that can search for role and organization values in a pre-defined list of valid values. Figure 3 shows the overall architecture of the solution that Adatum adopted. There is a central store of valid role and organization names that both ADFS and the SharePoint people picker use: this way Adatum can configure ADFS to issue **role** and **organization** claims that the SharePoint people picker will recognize.
  5. 
  6. Figure 3
  7. Architecture of the Adatum people picker solution.
  8. SharePoint and ADFS both run inside the Adatum corporate network. If SharePoint is running in a separate network from ADFS and the store, then a slightly more complex solution is needed. This might arise if SharePoint is running in the cloud, or if SharePoint needs to resolve values used by a partner' directory services. In this case, the architecture might include a lookup service as shown in Figure 4: in SharePoint you can use Business Connectivity Services to make the call to the lookup service — this introduces a useful layer of indirection into the architecture.
  9. 
  10. Figure 4
  11. People picker solution architecture including a query claims lookup service.
  12. Adatum plans to use **role** and **organization** claims to assign permissions in SharePoint, and wants to avoid assigning permissions to individual users. However, some organizations may prefer to use names or email addresses to assign permissions in some circumstances. It is still possible to do this in a claims-enabled SharePoint site, but with the standard people picker component site administrators will face the same problem of the people picker resolving both valid and invalid names. To work around this problem you can again create a custom people picker component that resolves **name** and **emailaddress** claim values against your directory service.
  13. #!SPOKENBY(Poe)!#
  14. In the long run, it's more maintainable to manage permissions based on roles (and organizations) rather than on individuals in SharePoint. You can use AD and ADFS to manage individual's role and organization membership, while in SharePoint you can focus on mapping roles and organizations to SharePoint groups.

## Single Sign-Out

* 1. For a SharePoint Web application to participate in the single sign-out process, it must be able to handle the following scenarios. For more information about single sign-out and the WS-Federation protocol see chapter 3, "Claims-Based Single Sign-On for the Web and Windows Azure."
  2. The user should be able to initiate the single sign-out from within the SharePoint Web application. Adatum modified the behavior of the standard sign-out process to send the WS-Federation **wsignout** message to the token issuer. In the Adatum scenario, this token issuer is ADFS.
  3. SharePoint Web applications should handle WS-Federation **wsignoutcleanup** messages from the issuer and invalidate any security tokens for the application. For this to work in SharePoint you must configure the SharePoint security token service to use session cookies rather than persistent cookies.
  4. If the user is signing in using Windows authentication in ADFS, then after they have signed out if they revisit the Web application they will be signed in automatically and silently. Although the single sign-out has happened, the user won't be aware of it.
  5. By default, SharePoint uses persistent cookies to store the session token, and this means that a user can close the browser, and then re-open it and get back to the SharePoint Web application provided the cookie has not expired. The consequence of changing to session cookies is that if a user closes the browser, she will always be required to authenticate again when she next visits the SharePoint Web application. Adatum prefers this behavior because it offers better security.
  6. #!SPOKENBY(Poe)!#
  7. The default name for the session cookie is FedAuth.

# Inside the Implementation

The following sections describe the key configuration steps that Adatum performed in order to implement the scenario that this chapter describes.

## Relying Party Configuration in ADFS

Each SharePoint Web application is a separate relying party (RP) from the perspective of ADFS. Adatum has configured each of the RPs to use the WS-Federation protocol and to issue the **emailaddress** and **role** claims for users that it authenticates: it passes the values of these claims through from Active Directory. The following table shows the mapping rules that Adatum configured for each RP in ADFS.

|  |  |
| --- | --- |
| LDAP Attribute | Outgoing claim type |
| E-Mail-Addresses | E-Mail Address |
| Token-Groups – Unqualified Names | Role |

It's important that the claims issued to SharePoint by ADFS (or any other claims issuer) are SAML 1.x compliant. For a description of the correct name format for claims that will be consumed by SharePoint, see this blog post: <http://social.technet.microsoft.com/wiki/contents/articles/ad-fs-2-0-the-admin-event-log-shows-error-111-with-system-argumentexception-id4216.aspx>.

ADFS must be able to identify which relying party a request comes from so that it can issue the correct set of rules. The sample scenario uses the identifiers shown in the following table:

|  |  |
| --- | --- |
| Relying Party | Identifiers |
| a-Portal SharePoint Web application | urn:adatum-portal:sharepoint |
| a-Techs SharePoint Web application | urn:adatum-techs:sharepoint |

* 1. **#!SPOKENBY(Poe)!#**
  2. SharePoint will send these identifier values in the **wtrealm** parameter. It's important to make sure that these identifiers match the configuration in SharePoint. These examples show the recommended format for these identifiers, however there is no specific required format.
  3. As part of the configuration in ADFS, you must specify the URL of the relying party WS-Federation protocol endpoint: this URL will be the "/\_trust/" path in your SharePoint Web application.
  4. You must enter the required information in ADFS manually (or create PowerShell scripts): SharePoint does not expose a FederationMetadata.xml document that you can use to automate the configuration.

## SharePoint STS Configuration

* 1. You must configure the SharePoint STS to trust the ADFS issuer, and the map the incoming claims from ADFS to claims that your SharePoint applications will use. The following sections describe the steps you must perform to complete this configuration.
  2. Remember to install the SharePoint Snapin before attempting to run any SharePoint PowerShell scripts. You can do this with the following PowerShell command:
  3. **Add-PSSnapin Microsoft.Sharepoint.Powershell**

### Create a New SharePoint Trusted Root Authority

* 1. ADFS signs the tokens that it issues with a token signing certificate. You must import into SharePoint a certificate that it can use to validate the token from ADFS. You can use the following PowerShell commands to import a certificate from the adfs.cer file:
  2. $cert = New-Object System.Security.Cryptography.X509Certificates.X509Certificate2("C:\adfs.cer ")
  3. New-SPTrustedRootAuthority
  4. -Name "Token Signing Cert"
  5. -Certificate $cert
  6. You can export this certificate from ADFS using the certificates node in the ADFS 2.0 Management console.
  7. If the signing certificate from ADFS has one or more parent certificates in its certificate chain, you must add these to SharePoint as well. You can use the same SharePoint command to do this.
  8. Notice that you must import any certificates that SharePoint uses into SharePoint — SharePoint does not use the trusted root authorities in the certificate store in the local machine.

### Create the Claim Mappings in SharePoint

* 1. To map the incoming claims from ADFS to claims that SharePoint uses, you must create some mapping rules. The following PowerShell commands shows how to create rules to pass through the incoming **emailaddress** and **role** claims.
  2. $map = New-SPClaimTypeMapping
  3. -IncomingClaimType "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress"
  4. -IncomingClaimTypeDisplayName "EmailAddress"
  5. -SameAsIncoming
  6. $map2 = New-SPClaimTypeMapping
  7. -IncomingClaimType "http://schemas.microsoft.com/ws/2008/06/identity/claims/role" -IncomingClaimTypeDisplayName "Role"
  8. –SameAsIncoming
  9. You can choose to perform your claims mapping either as a part of the RP definition in ADFS, or in the SharePoint STS. However, the rules mapping language in ADFS is the more flexible of the two.
  10. For an example of how to add additional claim types, see the "People Picker Customizations" section later in this chapter.

### Create a New SharePoint Trusted Identity Token Issuer

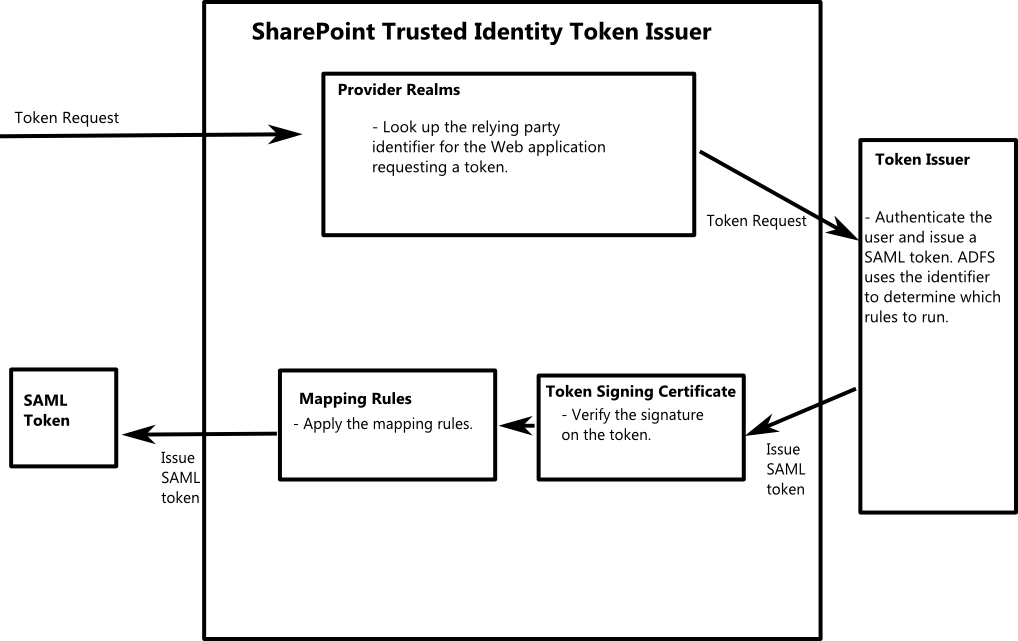
* 1. A SharePoint trusted identity token issuer binds together the details of the identity provider and the mapping rules to associate them with a specific SharePoint Web application. The following PowerShell commands show how to add the configuration settings for the scenario that this chapter describes. This script uses the **$cert**, **$map**, and **$map2** variables from the previous script snippets.
  2. $ap = New-SPTrustedIdentityTokenIssuer
  3. -Name "SAML Provider"
  4. -Description "Uses Adatum ADFS as an identity provider"
  5. -Realm "urn:adatum-portal:sharepoint"
  6. -ImportTrustCertificate $cert
  7. -ClaimsMappings $map,$map2
  8. -SignInUrl "https://DC-adatum/adfs/ls/"
  9. -IdentifierClaim http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress
  10. $uri = New-Object System.Uri("https://adatum-sp:31242/")

$ap.ProviderRealms.Add($uri, "urn:adatum-techs:sharepoint")

$ap.Update()

* 1. **#!SPOKENBY(Markus)!#**
  2. Don't forget to call the **Update** method to save the changes that the **ProviderRealms.Add** method makes.
  3. The following table describes the key parameters in the PowerShell commands.

|  |  |
| --- | --- |
| Parameter/command | Notes |
| -Realm | The realm is the value of the RP identifier in ADFS. In this example, the realm parameter identifies the a-Portal SharePoint Web application. The **Add** method of the **ProviderRealms** object adds the identifier for the a-Techs SharePoint Web application. The Uri is the address of the SharePoint Web application. |
| -ImportTrustCertificate | This associates the token signing certificate from ADFS with the token issuer. |
| -ClaimsMappings | This associates the claims mapping rules with the token issuer. |
| -SignInUrl | This identifies the Url where the user can authenticate with ADFS. |
| -IdentifierClaim | This identifies which claim from the IdP uniquely identifies the user. |

* 1. This example uses the email address as the identifier. You may want to consider alternative unique identifiers because of the possibility that email addresses can change.
  2. Figure 5 summarizes how the SharePoint trusted identity token issuer uses the configuration data to issue a SAML token to the SharePoint Web application.
  3. 
  4. Figure 5
  5. The SharePoint Trusted Identity Token Issuer
  6. When a SharePoint Web application requests a token from a trusted identity provider, the SharePoint trusted token issuer first looks up the unique identifier the Web application. It passes this identifier to the external token issuer in the **wtrealm** parameter of the request. When the external token issuer returns a SAML token, the SharePoint trusted identity token issuer verifies the signature, applies any mapping rules, and places the new SAML token in the SharePoint token cache. It also creates a FedAuth cookie that contains a reference to the SAML token in the cache. Whenever the user access a page in the SharePoint Web application, SharePoint first checks if a valid SAML token exists for the user, and then uses the claims in the token to perform any authorization checks.
  7. There is a one-to-one mapping between SharePoint trusted identity token issuers and trust certificates from the external token issuer. You cannot configure a new SharePoint trusted identity token issuer using a token signing certificate that an existing SharePoint trusted identity token issuer uses.

## SharePoint Web Application Configuration

* 1. Each Web application in SharePoint defines which authentication mechanisms it can use. In the scenario described in this chapter, Adatum has configured both SharePoint Web applications to use a SAML-based trusted identity provider. Both intranet and internet users use the SAML-based Trusted Identity provider.
  2. **#!SPOKENBY(Poe)!#**
  3. You can configure the authentication methods that SharePoint will use for a Web application in the "SharePoint 2010 Central Administration" site: navigate to **Manage Web Applications**, select the application you want to change, and click on **Authentication Providers**.

## People Picker Customizations

* 1. To customize the behavior of the standard people picker to enable site administrators to reliably select role and organization claims, Adatum created a custom claim provider to deploy to SharePoint. The Visual Studio® solution SampleClaimsProvider in the 10‑SharePoint folder from <http://claimsid.codeplex.com> includes a custom claim provider that demonstrates how Adatum extended the behavior of the people picker. For reasons of simplicity, this sample does not use a store to maintain the list of **role** and **organization** claims that Adatum uses, the lists of valid claims are maintained in memory. In a production quality claims provider you should read the permissible claims values from a store shared with the identity provider. For more information, see the section "The People Picker" earlier in this chapter.
  2. Use a custom **SPClaimProvider** class to override the default people picker behavior.
  3. The **SampleClaimsProvider** class extends the abstract **SPClaimProvider** class and overrides the methods such as **FillHierarchy**, **FillResolve**, and **FillSearch**. The **SPTrustedClaimsIssuer** class, which derives from the **SPClaimProvider** class, implements the default UI behavior in the people picker.
  4. The **GetPickerEntry** method is responsible for building an entry that will display in the people picker. The following code sample shows this method.
  5. private PickerEntity GetPickerEntity(string ClaimValue, string claimType, string GroupName)
  6. {
  7. PickerEntity pe = CreatePickerEntity();
  8. var issuer = SPOriginalIssuers.Format(
  9. SPOriginalIssuerType.TrustedProvider, TrustedProviderName);
  10. pe.Claim = new SPClaim(claimType, ClaimValue,
  11. Microsoft.IdentityModel.Claims.ClaimValueTypes.String, issuer);
  12. pe.Description = claimType + "/" + ClaimValue;
  13. pe.DisplayText = ClaimValue;
  14. pe.EntityData[PeopleEditorEntityDataKeys.DisplayName] = ClaimValue;
  15. pe.EntityType = SPClaimEntityTypes.Trusted;
  16. pe.IsResolved = true;
  17. pe.EntityGroupName = GroupName;
  18. return pe;
  19. }
  20. This method uses the **ClaimValue**, **claimType**, and **GroupName** strings to create a claim that the people picker can display. The **TrustedProviderName** variable refers to the name of the SharePoint trusted identity token issuer that you are using: the **SPOriginalIssuers.Format** method returns a string with the full name of the original valid issuer that you must use when you create a new claim.
  21. Notice that a claim definition includes the claim issuer as well as the claim type and value. SharePoint will check the source of a claim as a part of any authorization rules.
  22. If you are creating an identity claim, you must ensure that the **claimType** that you pass to the **SPClaim** constructor matches the identity claim type of your trusted identity token issuer, and that you set the **EntityType** property to **SPClaimEntityTypes.User**.
  23. The people picker uses the value of the **Description** property to display in a tooltip in the UI when a user hovers the mouse over a resolved claim.
  24. If you deploy this solution to SharePoint, then the people picker will display search results from this custom claim provider in addition to results from the default, built-in claim provider. This means that if a site administrator searches for a non-existent **role** or **organization** claim, then the default claim provider will continue to resolve this non-existent claim value. To prevent this behavior, you can make your custom claim provider the default claim provider. If the name of the trusted identity token issuer is "SAML Provider" and the name of the custom claim provider is "ADFSClaimProvider," then the following PowerShell script will make the custom claim provider the default.
  25. **#!CALLOUT!#**
  26. Adatum made the custom claim provider the default claim provider in the SharePoint Web applications.
  27. $ti = Get-SPTrustedIdentityTokenIssuer "SAML Provider"
  28. $ti.ClaimProviderName = "ADFSClaimsProvider"
  29. $ti.Update()
  30. It's also important to ensure that the claim types that the site administrator will use in the custom people picker exist in the trusted identity token issuer. You can use the following PowerShell script to list the claims which are present in the configuration.
  31. $i = Get-SPTrustedIdentityTokenIssuer "SAML Provider"
  32. $i.ClaimTypes
  33. You can add claim types to an existing trusted identity token issuer using the technique shown in the following PowerShell script.
  34. $map = New-SPClaimTypeMapping -IncomingClaimType
  35. "http://schemas.microsoft.com/ws/2008/06/identity/claims/organization"
  36. -IncomingClaimTypeDisplayName "Organization" -LocalClaimType
  37. "http://schemas.microsoft.com/ws/2008/06/identity/claims/organization"
  38. $ti = Get-SPTrustedIdentityTokenIssuer "SAML Provider"
  39. $ti.ClaimTypes.Add(
  40. "http://schemas.microsoft.com/ws/2008/06/identity/claims/organization")
  41. Add-SPClaimTypeMapping -Identity $map -TrustedIdentityTokenIssuer $ti
  42. This script maps an incoming claim and defines the new claim type in the trusted identity token issuer.

## Single Sign-Out Control

* 1. To implement single sign-out behavior, you must be able to send the WS-Federation **wsignout** message to the token issuer when the user clicks either the "Sign out" or "Sign in with a different user" link on any page in the a-Portal or a-Techs SharePoint Web applications. Adatum implemented the single sign-out logic in the **SessionAuthenticationModule's** **SignedIn** and **SigningOut** events. The Visual Studio® solution SingleSignOutModule in the 10‑SharePoint folder from <http://claimsid.codeplex.com> includes a custom HTTP module to deploy to your SharePoint Web application that includes this functionality.
  2. The following code sample shows the **DoFederatedSignOut** method that the **SigningOut** event handler invokes to perform the sign-out.
  3. private void DoFederatedSignOut()
  4. {
  5. string providerName = GetProviderNameFromCookie();
  6. SPTrustedLoginProvider loginProvider = null;
  7. SPSecurity.RunWithElevatedPrivileges(delegate()
  8. {
  9. loginProvider = GetLoginProvider(providerName);
  10. });
  11. if (loginProvider != null)
  12. {
  13. string returnUrl = string.Format(
  14. System.Globalization.CultureInfo.InvariantCulture,
  15. "{0}://{1}/\_layouts/SignOut.aspx",
  16. HttpContext.Current.Request.Url.Scheme,
  17. HttpContext.Current.Request.Url.Host);
  18. HttpCookie signOutExpiredCookie =
  19. new HttpCookie(SignOutCookieName, string.Empty);
  20. signOutExpiredCookie.Expires = new DateTime(1970, 1, 1);
  21. HttpContext.Current.Response.Cookies.Remove(SignOutCookieName);
  22. HttpContext.Current.Response.Cookies.Add(signOutExpiredCookie);
  23. WSFederationAuthenticationModule.FederatedSignOut(
  24. loginProvider.ProviderUri, new Uri(returnUrl));
  25. }
  26. }
  27. **#!SPOKENBY(Markus)!#**
  28. This method reads the provider name from a custom sign-out cookie rather than from the **IClaimsIdentity** object associated with the current user: this is because if the user's session has expired, there will be no **IClaimsIdentity** object. Also, it's not safe to read the provider name from the FedAuth cookie.
  29. This method performs the sign-out by calling the SharePoint **SPFederationAuthenticationModule.FederatedSignOut** method, passing the address of the claims provider and the address of the SharePoint Web application's sign-out page as parameters. To discover the address of the claims provider, it uses an **SPTrustedLoginProvider** object: however, to get a reference to the **SPTrustedLoginProvider** object it needs its name, and it discovers the name by reading the custom sign-out cookie.
  30. This method uses the **SPSecurity.RunWithElevatedPrivileges** method to invoke the **GetLoginProvider** method with "Full Control" permissions.
  31. The following code sample shows how Adatum creates the custom sign-out cookie in the **Session\_SignedIn** event.
  32. private const string SignOutCookieName = "SPSignOut";
  33. void WSFederationAuthenticationModule\_SignedIn(object sender, EventArgs e)
  34. {
  35. IClaimsIdentity identity =
  36. HttpContext.Current.User.Identity as IClaimsIdentity;
  38. if (identity != null)
  39. {
  40. foreach (Claim claim in identity.Claims)
  41. {
  42. if (claim.ClaimType == SPClaimTypes.IdentityProvider)
  43. {
  44. int index = claim.Value.IndexOf(':');
  45. string loginProviderName = claim.Value.Substring(
  46. index + 1, claim.Value.Length - index - 1);
  47. HttpCookie signOutCookie = new HttpCookie(SignOutCookieName,
  48. Convert.ToBase64String(
  49. System.Text.Encoding.UTF8.GetBytes(loginProviderName)));
  50. signOutCookie.Secure = FederatedAuthentication.SessionAuthenticationModule
  51. .CookieHandler.RequireSsl;
  52. signOutCookie.HttpOnly = FederatedAuthentication
  53. .SessionAuthenticationModule.CookieHandler.HideFromClientScript;
  54. signOutCookie.Domain = FederatedAuthentication
  55. .SessionAuthenticationModule.CookieHandler.Domain;
  56. HttpContext.Current.Response.Cookies.Add(signOutCookie);
  57. break;
  58. }
  59. }
  60. }
  61. }
  62. The custom sign-out cookie is not encrypted or signed. It is transported using SSL, and only contains the name of the user's login provider.
  63. **#!SPOKENBY(Jana)!#**
  64. One of the key reasons that Adatum selected this approach for handling single sign-out was its compatibility with the sliding-sessions implementation that Adatum chose the sign-out process must be initiated when the user is inactive for more than the defined period of inactivity and when the user's SAML token **ValidTo** time is reached. For details about how Adatum implemented sliding sessions in the a-Portal Web application see Chapter 12, "Federated Identity for SharePoint Applications."
  65. You can find a complete listing of the global.asax file that Adatum use in the a-Portal Web application at the end of this chapter.

## Displaying Claims in a Web Part

* 1. When you're developing a claims enabled SharePoint solution, it's useful to be able to view the set of claims that a user has when the user visits a SharePoint Web application. The Visual Studio® solution DisplayClaimsWebPart in the 10‑SharePoint folder from <http://claimsid.codeplex.com> includes a SharePoint Web Part that displays claims data for the current user. The Web Part displays the following claims data:
  + The claim type.
  + The claim issuer (this is typically SharePoint).
  + The original claim issuer (this might be a trusted provider or the SharePoint STS).
  + The claim value.
  1. This is a standard Web Part that you can deploy to a SharePoint Web application directly from Visual Studio or through the SharePoint UI. After the Web Part is deployed to SharePoint you can add it to any SharePoint Web page. It does not require any further configuration.

## User Profile Synchronization

* 1. A claims-enabled SharePoint environment can synchronize user profile data stored in the SharePoint profile store with profile data that is stored in directory services and other business systems in the enterprise. The important difference in the way that user profiles work in a claims-enabled Web application such as the Adatum a-Techs SharePoint application is how SharePoint identifies the correct user profile from the claims data associated with an **SPUser** instance.
  2. To make sure that SharePoint can match up a user profile from the current **SPUser** instance, you must ensure that three user properties are correctly configured.

|  |  |
| --- | --- |
| Property name | Property value |
| Claim User Identifier | This is the unique identifier for a user. For Adatum, this is the value it used for the *IdentifierClaim* parameter when it configured the SharePoint trusted identity token issuer: http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress. |
| Claim Provider Identifier | This identifies the trusted identity token issuer. For Adatum this value is "SAML Provider." This value is set automatically when you configure the user profile synchronization service. |
| Claim Provider Type | This specifies the token provider type. For Adatum this value is "Trusted Claims Provider Authentication." This value is set automatically when you configure the user profile synchronization service. |

* 1. **#!SPOKENBY(Poe)!#**
  2. To test this, you must have SharePoint 2010 Server (not Foundation) installed in Farm (not Standalone) mode.

# Setup and Physical Deployment

To run this scenario in a lab environment you may want to change some of the default configuration options in SharePoint and ADFS.

## FedAuth Tokens

* 1. Each SharePoint Web application must have its own FedAuth cookie if it is to function correctly in an SSO environment. In a production environment, this is not normally an issue because each SharePoint Web application has a separate host name: for example, a-portal.adatum.com, and a-techs.adatum.com. However, in a lab environment you may not want to configure the necessary DNS infrastructure to support this: if your SharePoint Web applications share the same host name, for example lab-sp.adatum.com:31242 and lab-sp.adatum.com:40197, then you must make a configuration change to make sure that each application uses a different name for the FedAuth cookie. You can change the name of the FedAuth cookie in the microsoft.IdentityModel section of the web.config file. The following snippet shows how to change the token name to "techsFedAuth" from its default name of "FedAuth."
  2. <federatedAuthentication>
  3. …
  4. <cookieHandler mode="Custom" path="/" name="techsFedAuth">
  5. …
  6. </federatedAuthentication>

## ADFS Default Authentication Method

* 1. By default, an ADFS server installation uses Integrated Windows Authentication, and an ADFS proxy installation uses an ASP.NET form to collect credentials. In a lab environment, if you do not have an ADFS proxy installation, you may want to change the default behavior of the ADFS server to use an ASP.NET form. To change this, edit the web.config file in the /adfs/ls folder. The following snippet shows "Forms" at the top of the list, making it the default. This means that in a simple lab environment you will always need to sign-in explicitly.
  2. <microsoft.identityServer.web>
  3. <localAuthenticationTypes>
  4. <add name="Forms" page="FormsSignIn.aspx" />
  5. <add name="Integrated" page="auth/integrated/" />
  6. <add name="TlsClient" page="auth/sslclient/" />
  7. <add name="Basic" page="auth/basic/" />
  8. </localAuthenticationTypes>
  9. …
  10. </microsoft.identityServer.web>

## Server Deployment

* 1. ADFS enables you to deploy a proxy instances that are intended to handle authentication requests from the Web rather than the internal corporate network which are handled by the main ADFS Server instances. This provides an addition layer of security because the main ADFS Server instances can be kept inside the corporate firewall. For more information about deploying ADFS Servers and ADFS Server Proxies, see this section on the TechNet Web site: <http://technet.microsoft.com/en-us/library/gg982491(WS.10).aspx>. You will also need to ensure that your SharePoint Web application is exposed to the internet to allow Adatum employees to access it remotely.

# More Information

* 1. For more information about SharePoint and claims-based identity, see Appendix F, "SharePoint 2010 Authentication Architecture and Considerations."
  2. For a detailed, end-to-end walkthrough that describes how to configure SharePoint and ADFS, see this blog post: <http://blogs.technet.com/b/speschka/archive/2010/07/30/configuring-sharepoint-2010-and-adfs-v2-end-to-end.aspx>.
  3. The following resources are useful if you are planning to create a custom people picker component for your SharePoint environment:
  + People Picker overview (SharePoint Server 2010): <http://technet.microsoft.com/en-us/library/gg602068.aspx>
  + Custom claims providers for People Picker (SharePoint Server 2010): <http://technet.microsoft.com/en-us/library/gg602072.aspx>
  + Creating Custom Claims Providers in SharePoint 2010: <http://msdn.microsoft.com/library/gg615945.aspx>
  + Claims Walkthrough: Writing Claims Providers for SharePoint 2010: <http://msdn.microsoft.com/en-us/library/ff699494.aspx>
  + How to Override the Default Name Resolution and Claims Provider in SharePoint 2010: <http://blogs.technet.com/b/speschka/archive/2010/04/28/how-to-override-the-default-name-resolution-and-claims-provider-in-sharepoint-2010.aspx>
  1. For further information about using profiles in a claims-enabled SharePoint environment, see this blog post: [http://blogs.msdn.com/b/brporter/archive/2010/07/19/trusted-identity-providers-amp-user-profile-synchronization.aspx](http://blogs.msdn.com/b/brporter/archive/2010/07/19/trusted-identity-providers-amp-user-profile-synchronization.aspx%20).