viewds identity solutions

White Paper:

Making Office 365 Work with an External SAML Identity Provider

Edward Sia

Recently I was asked to integrate our Cobalt Identity Server with Office 365 (O365) using SAML 2.0 for web SSO. I managed to get everything working in the end but not without some confusion and frustration along the way. This post attempts to capture the issues that I encountered and provides a straightforward step-by-step guide to configuring O365 to use SAML SSO, which I hope might help others attempting to integrate O365 with an external SAML IdP.

The main document that I referred to came from Microsoft MSDN and is called "Use SAML to Implement Single Sign-On". I also referred to some MSDN PowerShell cmdlet pages to learn more about how they should be used.

I must confess that prior to this integration with O365, I had done SAML SSO integration with other Service Providers (SPs), including Salesforce. However, I had fairly limited experience with Microsoft product integration. My lack of Microsoft-specific product knowledge (things like PowerShell and ADFS) may explain why I had trouble with the integration. But I have a sneaky feeling that the MSDN documentation itself is a little misleading or at least is missing a few crucial steps. Hopefully this document will fill in those gaps and provide a complete description of how to get O365 working with an external SAML 2.0 IdP.

The tasks that must be performed include executing numerous PowerShell cmdlets, verifying the registered domain by making changes to its DNS records, creating and configuring an application on the IdP side, and optionally, creating a sample user in Azure AD for testing purposes.

Before you get started you should ensure that you have the following prerequisites:

- An administrator account for Office 365
- A domain name that you own
- Windows PowerShell with the Azure AD PowerShell module installed

Connect to Windows Azure AD using Windows Powershell

Open Windows Powershell 4.0 and make sure the module Windows Azure Active Directory has been installed (see the prerequisites listed above). Run the following command to connect to Windows Azure AD using your Office 365 administrator account.

```
PS C:\> Connect-MsolService
```

Verify your domain in Azure AD

Verifying your domain is a one-time task; it basically confirms to Azure that you have administrative control of the DNS domain. Unless you plan to use a different domain in Azure AD, you will not need to perform this again in most situations. For the purposes of this guide, assume you own the domain “dev.companydomain.com”.

1. Add the domain that you own to Azure AD.

```
PS C:\> New-MsolDomain -Name dev.companydomain.com -Authentication Federated
```

2. In order to confirm ownership, Microsoft requires the domain owner to add a custom TXT DNS record for the domain to the domain server. This command is used to retrieve details of the DNS record that must be set.

```
PS C:\> Get-MsolDomainVerificationDns -DomainName dev.companydomain.com -Mode DnsTxtRecord
```

3. Once the DNS record has been added to the domain server, you need to execute the Confirm-MsolDomain command, along with its mandatory parameters, in order to confirm ownership of the domain, as well as set up the federated domain.

```
PS C:\> $domainname = "dev.companydomain.com"

PS C:\> $logoffuri = "https://localhost:9900/login/" # Landing page when user logs out

PS C:\> $passivelogonuri = "https://localhost:9900/identity/saml" # Identity Provider
SAML HTTP-POST endpoint

PS C:\> $cert =

"MIIEWTCCA0GgAwIBAgIJAJk28/BKoqaCMA0GCSqGSIb3DQEBBQUAMHcxCzAJBgNVBAYTAkFVMTYwNAYDV
QQKEy1lTml0aWF0aXZlcy5jb20gUHR5LiBMdGQuLCBBQk4gMTkgMDkyIDQyMiA0NzYxFDASBgNV.......
...I1hKLFfsiQgSfjGsQNcbNEAuF6DEb8LWhzZ34s7Qd/c5pjfCOAUGPTKIOOz5es2mPngwe5dl6355Z22
yTAaYjpQq92Pyjs7gJ0DCw+Quct5gos3Tw2be3aG6+KOcmtNB91AF/39qvS0jA8bWQGVi9jVaYSC/Fu/IH
CoHGBi/Y8F5j3zBQhxn7Zz5cspcVYfF3yYz2dB8J1Fm6YbFGc4kNO2zTR5bw/c3aA7nlK5B8KWGJTfSFg==" #
Server certificate, pem file. Remove spaces and newlines

PS C:\> $issueruri = "https://localhost:9900/identity/saml" # Issuer URI set by your
Identity Provider

PS C:\> $protocol = "SAMLP" # To ensure domain uses SAML SSO
```

```
PS C:\> Confirm-MsolDomain -DomainName $domainname -IssuerUri $issueruri -
FederationBrandName $domainname -LogOffUri $logoffuri -PassiveLogOnUri
$passivelogonuri -SigningCertificate $cert -PreferredAuthenticationProtocol $protocol
```

Issues: Firstly, I realized that I had to provide more arguments than were indicated in the MSDN Confirm-MsolDomain page (https://msdn.microsoft.cohm/en-us/library/dn194117.aspx). As a minimum, -IssuerUri, -LogOffUri, -PassiveLogOnUri, and -SigningCertificate were required to get this cmdlet to execute successfully, rather than just the -DomainName that the documentation suggests. Secondly, spaces and newlines in the PEM certificate had to be removed. It took me a while to figure all this out and I can't remember the exact error message but I recall complaints about an invalid value for parameter federationSettings, which I had not provided.

Configure the domain in your Office 365 for federation

The following cmdlet is provided by Microsoft MSDN for configuring SSO with a third party IDP. You might notice that the parameters are identical to the ones used to verify the domain in the previous section.

```
PS C:\> $domainname = "dev.companydomain.com"

PS C:\> $logoffuri = "https://localhost:9900/login/" # Landing page when user logs out

PS C:\> $passivelogonuri = "https://localhost:9900/identity/saml" # Identity Provider
SAML HTTP-POST endpoint

PS C:\> $cert =

"MIIEWTCCA0GgAwIBAgIJAJk28/BKoqaCMA0GCSqGSIb3DQEBBQUAMHcxCzAJBgNVBAYTAkFVMTYwNAYDV
QQKEy1lTml0aWF0aXZlcy5jb20gUHR5LiBMdGQuLCBBQk4gMTkgMDkyIDQyMiA0NzYxFDASBgNV.......
...I1hKLFfsiQgSfjGsQNcbNEAuF6DEb8LWhzZ34s7Qd/c5pjfCOAUGPTKIOOz5es2mPngwe5dl6355Z22
yTAaYjpQq92Pyjs7gJ0DCw+Quct5gos3Tw2be3aG6+KOcmtNB91AF/39qvS0jA8bWQGVi9jVaYSC/Fu/IH
CoHGBi/Y8F5j3zBQhxn7Zz5cspcVYfF3yYz2dB8J1Fm6YbFGc4kNO2zTR5bw/c3aA7nlK5B8KWGJTfSFg==" #
Server certificate, pem file. Remove spaces and newlines

PS C:\> $issueruri = "https://localhost:9900/identity/saml" # Issuer URI set by your
Identity Provider

PS C:\> $protocol = "SAMLP" # To ensure domain uses SAML SSO
```

```
PS C:\> Set-MsolDomainAuthentication -DomainName $domainname -FederationBrandName
$domainname -Authentication Federated -IssuerUri $issueruri -LogOffUri $logoffuri -
PassiveLogOnUri $passivelogonuri -SigningCertificate $cert -
PreferredAuthenticationProtocol $protocol
```

Issues: I am not entirely sure why this command duplicates all the arguments from the previous cmdlet. There may be sound reasons for this from an implementation perspective or there may not be but in any event, I did not have time to mess around with the cmdlets and arguments to investigate further, so I decided just to run it anyway. In fact, I had to run it to get things working.

Tip: If for any reason you want to change the parameters that you configured previously, you can use the Set-MsolDomainAuthentication cmdlet but change the value of -Authentication to "Managed", then change it back to "Federated" with new values. You may find that the Microsoft MSDN documentation suggests using Convert-MsolDomainToFederated and Convert-MsolDomainToStandard to do this but AFAIK they are both ADFS-related cmdlets and hence not really relevant, as our goal here is to use an external IdP that isn't ADFS.

Create a user for testing

The ImmutableId used here must match a user's unique identifier in your IdP. And we must provide this unique identifier as the value in the NameID element in the SAML assertion.

```
PS C:\> New-MsolUser -UserPrincipalName john.doe@dev.companydomain.com -ImmutableId
2e28f6ce-4e3b-4538-b284-1461f9379b48 -DisplayName "John Doe" -FirstName John -LastName
Doe -AlternateEmailAddresses "john.doe@company.com"
```

Assign license to the user

A license should be assigned to the test user, so they can access O365 apps.

1. To check if a user is licensed

```
PS C:\> Get-MsolUser
```

2. To check if you still have licenses available

```
PC C:\> Get-MsolAccountSku
```

3. To assign a license to a user. Assuming the license is "companyDom:O365_BUSINESS_ESSENTIALS"

```
PS C:\> Set-MsolUserLicense -UserPrincipalName john.doe@dev.companydomain.com -
AddLicenses companyDom:O365_BUSINESS_ESSENTIALS
```

Additional commands that might be useful

Check if the domain is Federated enabled.

```
PS C:\> Get-MsolDomain
```

Get the federation settings of a domain. Retrieve everything.

```
PS C:\> Get-MsolDomainFederationSettings -DomainName dev.companydomain.com | Format-List *
```

Configuring an app in the identity provider

Defining the app (or service provider, SP) in your IdP is the final stage in the integration. Really what we are doing here is configuring the IdP so that it sends a SAML assertion that can be consumed sensibly by Azure AD.

Key things that are worth pointing out when configuring the app for Azure AD in your IdP:

- Because Azure AD publishes and consumes AssertionConsumingServiceIndex, the app needs to be configured to recognize Azure AD AssertionConsumingServiceIndex and map it to its callback URL. The callback URL is the HTTP-Post binding URL found in the Azure AD's metadata.
- Azure AD's entityID is "urn:federation:MicrosoftOnline" (see Azure AD's metadata).

- Azure AD expects the IdP to provide an extra attribute with the name "IDPEmail" in the SAML Assertion that will be used to map the federated identity in Azure AD (see Use SAML to implement Single Sign-On).

Below is a sample of the Azure AD app configuration that I registered with the ViewDS Cobalt IdP. Critical information is highlighted in bold while the format is irrelevant in this context.

```
{
        "name" : "Azure AD",
        "type" : "RS256",
        "identifier" : "urn:federation:MicrosoftOnline",
        "entity" : "urn:federation:MicrosoftOnline",
        "secret" : "********",
        "callback" : [{
        "index": 0,
        "location": "https://login.microsoftonline.com/login.srf"
        }],
        "duration" : 3600,
        "samlAttributeConsumingService" : [{
        "index" : 1,
        "default" : true,
        "name" : "Azure AD",
        "description" : "Azure AD with Cobalt",
        "attributes" : [{
                "name" : "IDPEmail",
                "nameformat" : "urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified",
                "attribute" : "mail",
                "required" : true
        }]
   }]
}
```

Including the right email address in the SAML assertion

Note (from Use SAML to implement Single Sign-On)

The "UserPrincipalName" value must match the value that you send for "IDPEmail" in your SAML 2.0 claim and the "ImmutableID" value must match the value sent in your "NameID" assertion.

In this example, domain "dev.companydomain.com" has been configured in Azure AD as the federated domain. Therefore, you must ensure that the SAML assertion being constructed in your IdP includes a user email with that domain. The reason being that Azure AD is only able to match the user in its directory using an email attribute with that domain (and NameID) in the SAML assertion.

```
- john.doe@dev.companydomain.com
```

Verifying SAML connectivity

One of the simplest and quickest ways to verify that single sign-on has been set up correctly is to attempt SP-initiated login to the Office 365 Portal.

1. Access the Office 365 Portal.

   ```
   https://portal.office.com/
   ```

   A login screen is displayed.

2. Provide the user's email address on the Office 365 login screen.

   ```
   john.doe@dev.companydomain.com
   ```

3. If your external IdP has been setup correctly on the Azure AD side, then you will be redirected to the external IdP's login screen. Provide the appropriate login credentials on the external IdP's login screen.
4. Once the correct credentials have been provided, the external IdP will send a SAML assertion (that contains attributes and the ImmutableID) to Office 365 and redirect the

browser to the Office 365 Portal in a logged-in state. This indicates that external authentication has worked successfully.

5. If Office 365 login is unsuccessful, then an error code will be displayed. Should this happen, you can use the Microsoft MSDN documentation to identify the meaning of the error code and help troubleshoot the problem.

Conclusion

In conclusion, the procedures are not too technical after all, once you get past the misleading documentation. Of course, understanding what needs to happen at different points in the exercise is crucial and I do hope that this article has provided clear guidance about how to implement your external Identity Provider with Azure AD. Finally, one other resource that really contributed towards the success of my implementation was the document by the third party Identity Provider Salesforce. I used it to verify the PowerShell cmdlets I was executing contained all the mandatory parameters and the correct values.