SSO Step-by-Step

Deliverable Version: SSO Step-by-step.docx

MemberSuite Single Sign-on (SSO)

What is it?

Single Sign-on is used with MemberSuite to provide a seamless transition for members from an association's password protected environment into the MemberSuite Portal without the user having to re-enter a password or a completely different set of credentials.

Who owns the credentials?

This is really up to the implementation. MemberSuite's SSO can work whether or not MemberSuite owns the username and password. If MemberSuite owns the credentials, an API method can be called to authenticate a supplied username and password. If an external source is used to authenticate credentials, the assumption by MemberSuite is that the initiator of the SSO call trusts that authentication, so MemberSuite can as well.

What about shared authentication schemes like OAuth or OpenID?

The MemberSuite Portal does not currently support these natively, although it is on our roadmap.

What about synchronizing passwords between an external site and MemberSuite?

For security reasons, MemberSuite does not encourage this. MemberSuite security would not allow us to decrypt a User's password even if we wanted to, so we could not initiate any type of password sync when a password changes on the MemberSuite side. There are API methods that allow setting of a User password, so an external application could possibly push password updates to MemberSuite if that was desired, but then of a change occurred on MemberSuite the credentials would be out-of-sync. Our strong recommendation is to choose one place to control the password, thus eliminating confusion when a password needs to be changed or reset.

How do I get started?

You should start by reviewing our documentation to understand the API:

- Getting Started
  - How to Create a New Access Key
  - How to Locate Your Association ID
  - Signing Certificates
  - Single Sign On

There is also this more technical diagram explaining the step-by-step SSO communication path:

SSO Diagram.pdf

Our recommendation is for the Association to provide the technical resource implementing the SSO at least temporary access to the MemberSuite Console application. With this access, the resource will be able to generate the necessary security keys safely, without having to pass them around via e-mail which could be dangerous.

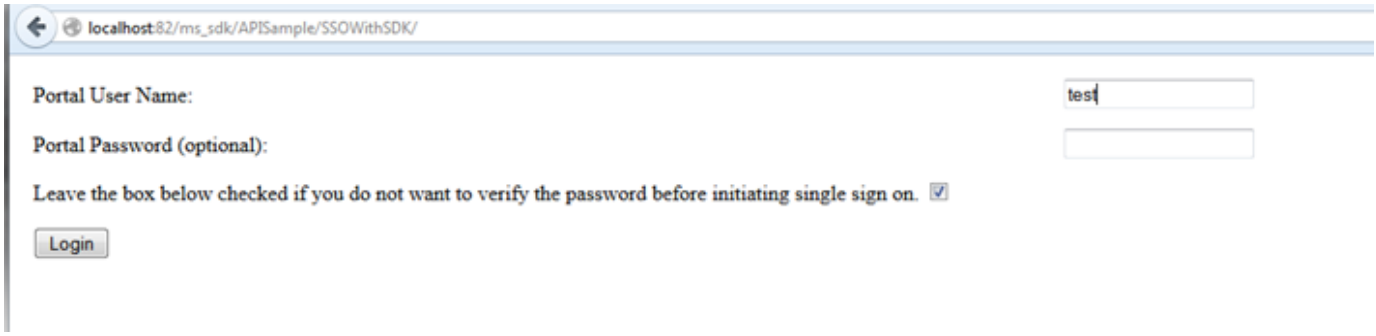
Once you can log into the MemberSuite Console, you will need to create an Access Key ID \ Secret Access Key pair as well as a Signing Certificate ID \ Certificate File pair using the instructions found in the documentation.

The next step would be to pull down the Sample SSO implementations and try to get them running to connect with our Sample Association and test the SSO with the account Username: Test, Password: Test.

This should work with either the .NET Sample or PHP Sample. In both cases, the easiest version would be the SDK version (although direct WSDL and SOAP examples are provided in case you need to convert to a different language). When running, they should look like:

.NET Sample:

PHP Sample:



You can enter the password “test” or just leave the “Do not verify login credentials” checked. When clicking Login, you should be taken into the API Sandbox portal.

Once you have the Sample code working with the default credentials and the MemberSuite API Sandbox association, you will want to swap in the credentials for your site.

.NET web.config:

```
<add key="AssociationID" value="2537d8c3-0004-4ddb-b7e4-a6d76c09d3f9"/>
<add key="AccessKeyID" value="AAAAAPIA/UObVXfWokrswA"/>
<add key="SecretAccessKey" value="PislBM1IXpWjshA/IQagSzrJ/FiXbuJu80zaKctvLO/CzHcZXYC9F8MbatI2jtFY4TZRU3TsmYflg3HsSYdWOQ=="/>
<add key="SigningCertificateId" value="AAAAAPMAnESfVm9oock2Kw"/>

<!-- Portal for SSO -->
<add key="PortalUrl" value="https://customer14517d3f9.portal.production.membersuite.com" />
```

PHP config.php:

```
public static function read($name){

$config = array('AccessKeyId' => 'AAAAAPIA/UObVXfWokrswA',
                'AssociationId' => '2537d8c3-0004-4ddb-b7e4-a6d76c09d3f9',
                'SecretAccessKey' => 'PislBM1IXpWjshA/IQagSzrJ/FiXbuJu80zaKctvLO/CzHcZXYC9F8MbatI2jtFY4TZRU3TsmYflg3HsSYdWOQ==',
                'SigningcertificateId' => 'AAAAAPMAnESfVm9oock2Kw',
                'SigningcertificatePath' => 'bin/signingcertificate.xml',
                'PortalUrl' => 'https://customer14517d3f9.portal.production.membersuite.com/'
                );
```

The “PortalUrl” value can be found in the Console by going to “Setup \ Portal Settings” Also place the “signingcertificate.xml” in the “bin” directory with the one you downloaded from or registered with MemberSuite.

1.6 SSO Setup Checklist

1. Log into Console and create a new API User for this specific integration and then create a new Access Key ID \ Secret Access Key pair a nd download or copy the ID and Secret Access key somewhere.
2. On this same user create a Signing Certificate and be sure to save the XML file.
3. Get the Association ID for this association.
4. Get the Portal URL from Setup \ Portal Settings.

5. Set up either the .NET or PHP Sample SSO code and test using the credentials **test** \ **test**.
6. Go back to your Console login and create a valid Portal Login and test it out by making sure you can log in to the PortalUrl (you can also create a new account from the Portal's "Create Account" link).
7. Swap in your credentials and signingcertificate.xml:
    a. AssociationId
    b. AccessKeyId
    c. SecretAccessKey
    d. SigningcertificateId
    e. PortalUrl
    f. \bin\signingcertificate.xml
    g. Now test the sample again with the Portal login you created above.
    h. Once you have confirmed that the credentials are working properly, you can start adding the code into your own login pages.

Reverse SSO

What is it?

If instead of using an external login system for users and then jumping into the MemberSuite portal, you would like to have them log into MemberSuite and then jump to one or more external sites seamlessly, you would need to implement Reverse SSO. This works the same way as SSO, except that MemberSuite generates the SSO Token and then the external sites must implement a page which validates that token.

Who owns the credentials?

In this case, typically, MemberSuite would own the credentials and users would log in through our login page, however it would be possible to use regular SSO where another site owns the credentials and forwards the user through SSO to the MemberSuite portal. In this case, Reverse SSO could still be used to jump back to the originating site (if there was concern about session expiration) or to a completely different site.

How do I implement this?

There are 2 steps to this process:

1) You must build the page that will receive the SSO Token on the target site

2) You must add links into either your Portal Skin or other content in MemberSuite

Building the Receive SSO Token page

A sample "ReceiveSSOToken" page can be found in the C# sample code. This page requires many of the same prerequisites and setup as the regular SSO process including Access Keys and Signing Certificates. This page simply validates that the Token passed to it is valid through a call to the MemberSuite API.

Note: This token is only valid for a very short amount of time. You would not be able to cache it and use it later. You would want to make sure to generate the proper login session for the new site immediately and then continue with the process.

Creating Reverse SSO Links

To test that the new "ReceiveSSOToken" works, you would need to add a link somewhere in the MemberSuite Portal site content. This could be in the Portal Skin, a generic Portal Link, or in any other content exposed in the site (like an Event description or Portal Text Override).

The format of the link would need to be:

javascript:JumpToExternalLink('http://mysite.org/ReceiveSSOToken.aspx', 'http://google.com', true);

This is a custom JavaScript function that will post the URLs appropriately to the page that will generate the Portal Security Token and redirect to the Client's page. The parameters are:

1) The URL of the page the client has built to receive and process the token

2) The URL of the page to pass as the "NextUrl" to the client page (for redirecting after login)

3) “true” to open in a new tab or “false” to keep in the same tab\window

We also pass to the client page the current URL as the “ReturnUrl” in case the receiving page wants to provide a “return to MemberSuite” link that takes the user back to exactly where they were.

Optional Keep-Alive Process

The simplest way to handle linking out of MemberSuite would be to push the external link to a new tab\window in the browser. This would allow the MemberSuite portal to remain open and continue to manage its own session timeout. If, however, the client requires a more seamless transition between MemberSuite and non-MemberSuite pages all within the same tab\window we would need to implement keep-alive pages. Effectively, what this means is that every time a MemberSuite portal page loads it will dynamically load a URL on the non-MemberSuite site to keep that session from expiring and then any pages on the non-MemberSuite site would need to also dynamically load a URL on the MemberSuite site.

You would want to put something like this in the header or footer of your site HTML:

```
<div style="display: none">
    <iframe frameborder="0" height="0" src="https://site.membersuite.com/KeepAlive.aspx" width="0"></iframe>
</div>
```