Azure AD B2C pre-populate a custom attribute in the SignUp policy

Asked 7 years, 3 months ago Modified 6 years, 8 months ago

Viewed 11k times Part of Microsoft Azure Collective

9

Does Azure AD B2C support pre-populating a custom attribute in the SignUp Policy when called from the Web application (ASP.Net MVC)?

We can create a custom SignUp attribute but we weren't able to find a specification in the documentation how to pass value to populate the custom attribute. If this is not supported out of the box, does anybody found a workaround?

Here are some more details for the context in case somebody has faced a similar scenario and found a useful solution:

We explore the options to solve the following scenario with Azure AD B2C: a registered user invites another person to signup to the application by sending an invitation email which has the url to the application's login page along with a special invitation code(guid) as a query param, so it can click on the link and to be redirected to the Signup page. After the invited person creates an account, we need to use the code in order to associate the newly created user to the user who sent the invitation.

Currently this is implemented in the ASP.Net using the default identity provider (storing the user data in database with AspNet... tables). With replacing the local identity provider with the Azure AD B2C, we are loosing the context during the round-trip to the Azure AD B2C Signup page. The user clicks on the link on the email and gets to the SIgnUp page but the invitation code is not pre-populated.

azure-ad-b2c

Share

Improve this question

Follow

edited Sep 23, 2017 at 16:54

asked Sep 23, 2017 at 14:09

DI. Robin Olivaw

149 1 1 6

Welcome to Stack Overflow. What have you already tried yourself to do this? Please review How do I ask a good question. Stack Overflow is not a coding service. You are expected to ***research your issue and make a good attempt to write the code yourself*** before posting. If you get stuck on something *specific*, come back and include a Minimal, Complete, and Verifiable example and a summary of what you tried, so we can help. – FluffyKitten Sep 23, 2017 at 14:40

3 @FluffyKitten what is not specific about that? The first sentence is a very specific question. And a good question I

might add :) Regards, Mike D. – spottedmahn Sep 25, 2017 at 14:16

1 @spottedmahn The question has been edited since my comment. But just being specific on its own isn't enough - see the rest of requirements in my initial comment. – FluffyKitten Sep 25, 2017 at 14:49

@FluffyKitten I see, thanks for letting me know :) – spottedmahn Sep 25, 2017 at 15:46

1 Answer

Sorted by: Highest score (default)

13

A working sample of an invitation flow is here.

In the `WingTipGamesWebApplication` project, the `InvitationController` controller class has two action methods, `Create` and `Redeem`.

The `Create` action method sends a signed redemption link to the email address for the invited user. This redemption link contains this email address. It could also contain the invitation code.

The `Redeem` action method handles the redemption link. It passes the email address, as the **verified_email** claim in a JWT that is signed with the client secret of the Wingtip Games application (see the `CreateSelfIssuedToken` method in the `Startup` class in the `WingTipGamesWebApplication` project), from the redemption link to the **Invitation** policy. It could also pass the invitation code.

The **Invitation** policy can be found at here.

The **Invitation** policy declares the **verified_email** claim as an input claim:

```
<RelyingParty>
  <DefaultUserJourney ReferenceId="Invitation" />
  <TechnicalProfile Id="Invitation">
    <InputTokenFormat>JWT</InputTokenFormat>
      <CryptographicKeys>
        <Key Id="client_secret" StorageReferenceId="Wi
/>
    </CryptographicKeys>
    <InputClaims>
      <InputClaim ClaimTypeReferenceId="extension_Veri
    </InputClaims>
  </TechnicalProfile>
</RelyingParty>
```

The **extension_verifiedEmail** claim type, which is declared as a read-only field (so that it can't be modified by the end user), is mapped to the **verified_email** input claim:

```
<BuildingBlocks>
  <ClaimsSchema>
    <ClaimType Id="extension_VerifiedEmail">
      <DisplayName>Verified Email</DisplayName>
      <DataType>string</DataType>
      <DefaultPartnerClaimTypes>
        <Protocol Name="OAuth2" PartnerClaimType="veri
        <Protocol Name="OpenIdConnect" PartnerClaimTyp
        <Protocol Name="SAML2"
PartnerClaimType="http://schemas.wingtipb2c.net/identi
/>
      </DefaultPartnerClaimTypes>
      <UserInputType>Readonly</UserInputType>
    </ClaimType>
```

```
  </ClaimsSchema>
</BuildingBlocks>
```

The **Invitation** user journey can be found in here.

The second orchestration step of the **Invitation** user journey executes the **LocalAccount-Registration-VerifiedEmail** technical profile:

```
<UserJourney Id="Invitation">
  <OrchestrationSteps>
    ...
    <OrchestrationStep Order="2" Type="ClaimsExchange"
      <ClaimsExchanges>
        ...
        <ClaimsExchange Id="LocalAccountRegistrationEx
TechnicalProfileReferenceId="LocalAccount-Registration
      </ClaimsExchanges>
    </OrchestrationStep>
  </OrchestrationSteps>
</UserJourney>
```

The **LocalAccount-Registration-VerifiedEmail** technical profile registers the local account with the verified email address:

```
<TechnicalProfile Id="LocalAccount-Registration-Verifi
  <DisplayName>WingTip Account</DisplayName>
  <Protocol Name="Proprietary"
Handler="Web.TPEngine.Providers.SelfAssertedAttributeP
Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"
  <Metadata>
    <Item
Key="ContentDefinitionReferenceId">api.localaccount.re
    <Item Key="IpAddressClaimReferenceId">IpAddress</I
    <Item Key="language.button_continue">Create</Item>
  </Metadata>
  <CryptographicKeys>
```

```
    <Key Id="issuer_secret" StorageReferenceId="TokenS
  </CryptographicKeys>
  <InputClaimsTransformations>
    <InputClaimsTransformation ReferenceId="CreateEmai
  </InputClaimsTransformations>
  <InputClaims>
    <InputClaim ClaimTypeReferenceId="extension_Verifi
  </InputClaims>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="extension_Verif
/>
    <OutputClaim ClaimTypeReferenceId="newPassword" Re
    <OutputClaim ClaimTypeReferenceId="reenterPassword
    <OutputClaim ClaimTypeReferenceId="displayName" Re
    <OutputClaim ClaimTypeReferenceId="authenticationS
DefaultValue="localAccountAuthentication" />
    <OutputClaim ClaimTypeReferenceId="executed-SelfAs
DefaultValue="true" />
    <OutputClaim ClaimTypeReferenceId="newUser" />
    <OutputClaim ClaimTypeReferenceId="objectId" />
    <OutputClaim ClaimTypeReferenceId="sub" />
    <OutputClaim ClaimTypeReferenceId="userPrincipalNa
  </OutputClaims>
  <ValidationTechnicalProfiles>
    <ValidationTechnicalProfile ReferenceId="AzureActi
WriteUserByEmail-ThrowIfExists" />
  </ValidationTechnicalProfiles>
  <UseTechnicalProfileForSessionManagement ReferenceId
AzureActiveDirectory" />
</TechnicalProfile>
```

Before the local account is registered by the **AzureActiveDirectoryStore-WriteUserByEmail-ThrowIfExists** validation technical profile, the **CreateEmailFromVerifiedEmail** claims transformation copies the **verified_email** claim to the **email** claim:

```
<ClaimsTransformation Id="CreateEmailFromVerifiedEmail
TransformationMethod="FormatStringClaim">
  <InputClaims>
```

```
    <InputClaim ClaimTypeReferenceId="extension_Verifi
TransformationClaimType="inputClaim" />
  </InputClaims>
  <InputParameters>
    <InputParameter Id="stringFormat" DataType="string
  </InputParameters>
  <OutputClaims>
    <OutputClaim ClaimTypeReferenceId="email"
TransformationClaimType="outputClaim" />
  </OutputClaims>
</ClaimsTransformation>
```

To save the invitation code against the local account, you must:

- Add the "extension_InvitationCode" claim to the claims schema
- Add it as an input claim to the **Invitation** policy
- Add it as an input claim to the **LocalAccount-Registration-VerifiedEmail** technical profile
- Add it as a persisted claim to the **AzureActiveDirectoryStore-WriteUserByEmail-ThrowIfExist** technical profile

Share Improve this answer Follow

edited Apr 1, 2018 at 1:17

answered Sep 26, 2017 at 10:59

Chris Padgett
14.6k 2 19 33

Thanks for the very detailed explanation. To confirm our understanding - the suggested solution is to create a custom

policy where a new attribute can be added as a claim following the model for verified email and this attribute will be stored in Azure AD B2C and returned as claim to the Web application after the guest user signed-up? Practically, following this model, many custom attributes can be added, correct? To deploy it, is it just uploading these files: .._B2C_1A_base.xml .._B2C_1A_base_extensions.xml .._B2C_1A_invitation.xml or other steps are required?
– DI. Robin Olivaw Sep 28, 2017 at 2:42

1 You are correct for all of the above questions. Issuing the custom claim to the web application will enable the web application to link the invited user with the inviter user. Alternatively, you can add an orchestration step to the user journey, which invokes a RESTful service (such as an Azure Function) to link them. – Chris Padgett Sep 29, 2017 at 10:27

1 @spottedmahn Updated. – Chris Padgett Apr 1, 2018 at 1:17

1 This is really interesting, but there is something I'm not quite clear on...this talks about "local" accounts, by which I think it means users who directly register in the B2C tenant by signing up with their email address and creating a password. Can the same process be used to invite users whose account would be a "work" account (i.e. an account already associated with another Azure AD)? – ADyson Mar 29, 2019 at 14:00

1 Hi @ADyson. You are right. It is referring to accounts that are managed by the Azure AD B2C tenant. You might be able to implement an invitation process for an external user that's managed by another Azure AD tenant by passing the **extension_VerifiedEmail** claim through as the `login_hint` parameter that's supported by Azure AD. You might also have to ensure that the e-mail address of the Azure AD user matches that of the invited user.
– Chris Padgett Apr 1, 2019 at 7:15