appgate

Appgate SDP and SAML (Security Assertion Markup Language)

Type: Technical guide
Date: May 2021
Applies to: v5.4 and newer

This document provides background on SAML for those of you who have not used this technology before. It goes on to explain how SAML has been applied within the Appgate SDP system; provides an overview of both the user experience and general approach to configuring SAML as an Identity Provider for use with Appgate SDP. For full details about how to configure SAML you should refer to the Appgate SDP administration guide and any separate installation guides.

Background to SAML

Security Assertion Markup Language (SAML) is an XML-based, open-standard data format for exchanging authentication and authorization data between parties, in particular, between an identity provider (IdP), and a service provider (SP).

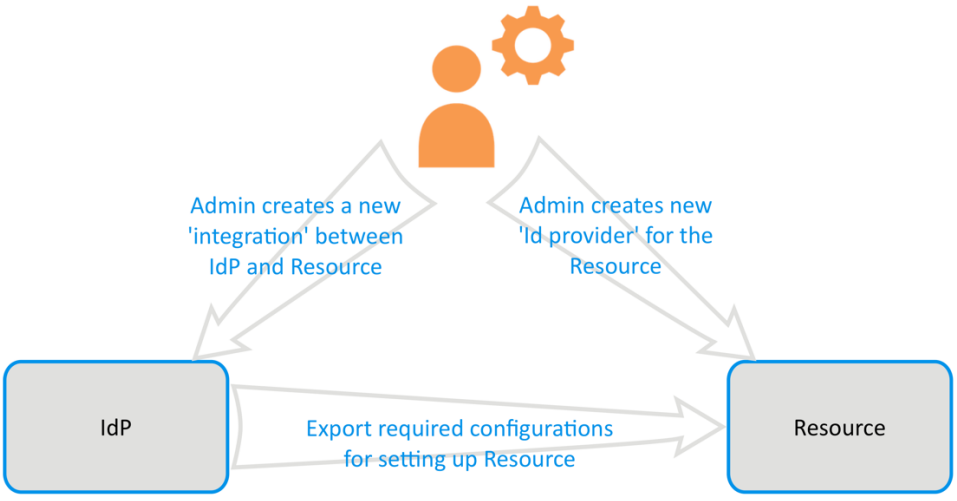
This is important because today's enterprise employees are accessing a lot of applications from a lot of different devices using both browsers and native applications. The issue here is the sheer number of complex passwords a user must remember for all of these - it's enormous! That's where identity federation comes in, by addressing these kinds of challenges it allows enterprises to share identities in a secure fashion – there is no longer the need to keep separate user profiles for every application. This identity federation standard is SAML.

The SAML standard specifies three main elements: assertions, protocols, and bindings. There are three assertions: authentication, attribute, and authorization. There are six protocols of which only the first is explored in this paper:

- Authentication Request Protocol - used when a SP wishes to obtain assertions containing authentication statements from an IdP and get back a message containing these assertions]
- Assertion Query and Request Protocol
- Artefact Resolution Protocol
- Single Logout Protocol
- Name Identifier Management Protocol
- Name Identifier Mapping Protocol

Protocols define how SAML asks for and receives assertions. Binding relates to how SAML message exchanges map to 'Simple Object Access Protocol' exchanges. SAML actually works with various protocols including Hypertext Transfer Protocol [http://] which is the use case explored in this paper.

Any identity provider (IdP) needs to be pre-associated with the service provider (SP) offering the resource by before users will be able to sign-in. This is also the case with SAML, where the administrator would have configured a new 'integration' between the two – usually by taking some information provided by the IdP and seeding the SP (resource) with information as shown below:



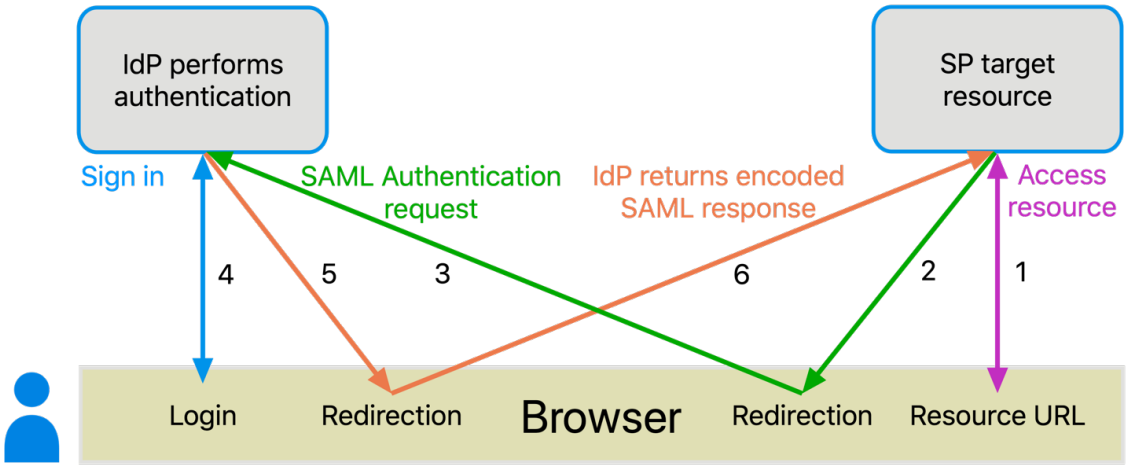
Part of this seeding includes the use of public key authentication. The IdP generates a *key pair*, consisting of a public key (which all SPs know) and a private key (which is kept secret). The private key is able to generate digital signatures for the SAML messages. This signature created using the private key cannot be forged by anybody; but the resources can verify that a particular message is genuine using the pre-seeded public key.

SAML as intended

Normally SAML is designed to work with web apps so in this example: Jo wants to access a web-based resource using her browser (1), typically she would go to https://resource.mycompany.com.

For a SAML configured resource, it won't ask for a username and password directly but instead redirect the browser to the IdP for authentication (2,3), using the seed information provided by the administrator earlier. The URL the user is redirected to might look something like:

```
https://dev-764072.oktapreview.com/app/cryptzonedev764072_xdp3_1/exk6a231yugoUkyax0h7/sso/saml
```



Within this redirect message is the SAML authentication request. As SAML is XML-based, the complete authentication request message is compressed (to save space in the URL) and encoded (because certain characters are not allowed in URLs). If we were to unravel the SAML message it says something like: *this is a request from SP xxxxx; please authenticate the sender of this message, and post the result back.*

When the IdP receives this message, it will authenticate Jo by asking her to enter some credentials (4) in a browser window. This step may not be visible to Jo if 'single sign-on' happens because the request was received while a valid session cookie persists (because Jo already used the SAML IdP to sign in to another resource).

After successful authentication, the SAML protocol message carrying a SAML authentication message (5,6) is sent via Jo's browser to the AssertionConsumerService URL configured within the IdP.

When we unravel this SAML response message in essence it says something like: *this is a message from IdP zzzzz; I have successfully authenticated the user Jo; this message is valid for time window yyyyy; and here is my XML digital signature.*

This last item is used as proof that the message is genuine and that the message has not been tampered with en-route. The digital signature is made using a private/public key algorithm and the public key needed to verify the signature is embedded in a certificate that is already configured within the resource.

The resource reads the SAML authentication response message, it verifies the signature and checks things such as the subject of the message (user's identifier (NameID)), then any: Authentication assertions (Jo was authenticated at a specific time via this type of authentication mechanism); Attribute assertions (Jo is associated with the following claims/attributes); Authorization decision assertions (Jo's access to the resource has been granted or not). The full list of what Appgate SDP requires in the SAML message is shown in the Appendix.

Applying SAML to non-browser-based apps

SAML was never envisaged to be used with non-browser-based resources because of its use of http redirects but, since its invention in 2002, the world has changed and mobile apps are everywhere. Because of this evolution, many apps including the Appgate SDP Client are being developed using 'web view' technologies, which certainly helps in respect of the use of SAML. Using a 'web view' is great in as much as it allows portability across platforms and because it is web-based it is more suited to the use of SAML.

If an embedded in-app browser instance is also included, then the SAML sign in pages can be fully handled within the app presenting a managed and smooth user experience. But there can be downsides to this; it is possible that in-app browsers won't be able to handle some specific types of authentication challenges that the main browser supports (i.e. plug-ins that autofill passwords). Also in-app browsers will not have access to the main browser's cookie storage (i.e. the benefit of SSO between apps using a cached SAML token cannot be realised).

SAML with Appgate SDP

As you will have already seen, SAML links 2 parties together, the IdP and the SP. Because it is hard to get between the user and SaaS based applications, controlling the issuance of SAML assertions is a quite effective way of controlling who can use these applications. Therefore CASBs (Cloud Application Service Broker) often set themselves up as IdPs (in SAML terms) because they need to control authorization towards the SP – which are normally SaaS based applications.

Appgate SDP's primary function is to address the PaaS/IaaS (the SaaS market is secondary). By implementing SAML we are offering enterprises a way to control access to non-web-based resources using the same IdP that they have deployed for their Cloud and web-based apps. With PaaS/IaaS it is more appropriate that we behave as a gateway between the user and the applications, so it makes perfect sense for Appgate SDP to be the SP (in SAML terms).

Appgate SDP Case 1 – Admin SAML

It is possible to use SAML for admin authentication towards the Appgate SDP Controller. In this case we can use SAML straight out of the box. The Assertion Consumer Service (ACS) reply URL should go straight to the Controller: *https://mycontroller.mycompany.com:8443/admin /saml*

Appgate SDP Case 2 – Portal SAML

It is possible to use SAML for Portal user authentication. In this case we can again use SAML straight out of the box. The Assertion Consumer Service (ACS) reply URL should go to the Portal: *https://myportal.com/saml (port 443).* The Portal appliance acts as a redirection server, forwarding the ACS to the requesting (internal) Client instance.

Appgate SDP Case 3 – Client SAML

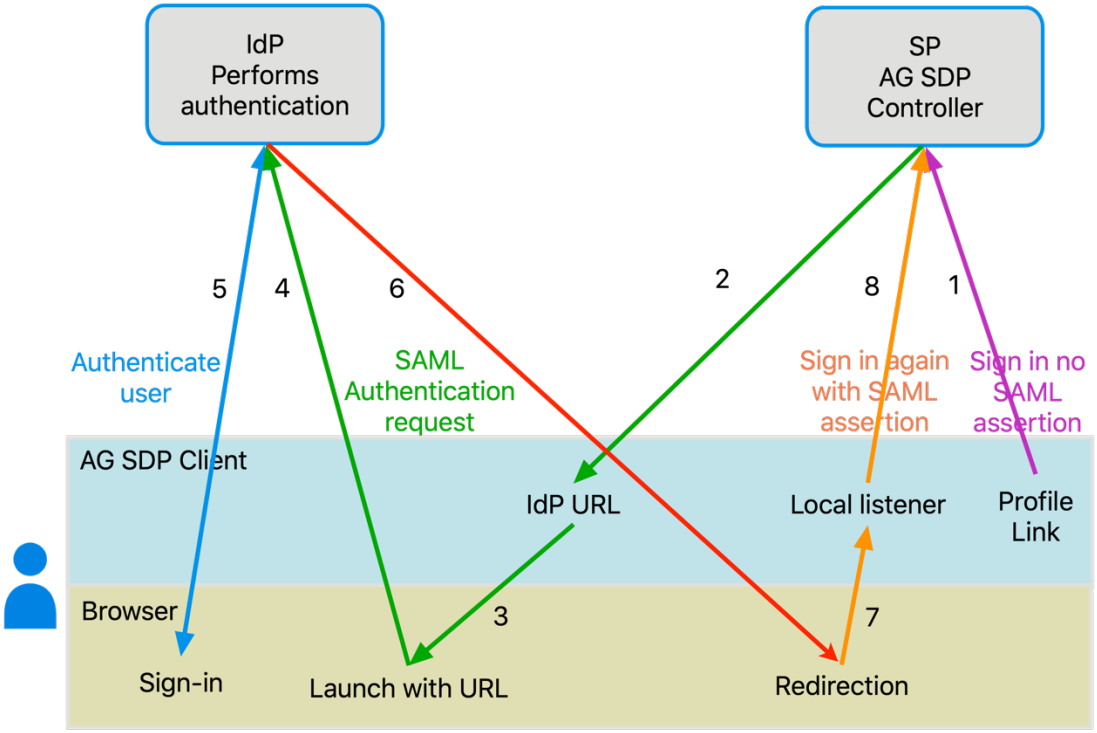
Even though the Appgate SDP Client uses 'web view' technologies this does not make it SAML ready as a browser instance is still required. Because the Client is implemented as a native application, we have ended up with a hybrid arrangement as explained below:

Unlike normal SAML, here Jo wants to sign-in to Appgate SDP, so starts the Client and the connects (1) without any username and password information present because the IdP will request these later on. The Controller (the SP in this case) looks up the pre-seeded information set by the administrator and from this, finds and sends the sign in URL of the IdP to be used for authentication (2).

Because SAML is being used, the Client expects this URL so when it receives it opens the browser on the device using this URL which effectively kicks off the SAML authentication request

flow (3,4). This is very similar to the original redirection, but because the browser did not kick off the process then the Client has to invoke the browser.

When the IdP receives this message it will, either authenticate Jo by asking her to enter some credentials (5). This step may not be visible to Jo if 'single sign-on' happens because the request was received while a valid session cookie persists (because Jo already used the SAML IdP to sign in to another resource). After successful authentication, the SAML protocol message carrying a SAML authentication message (6) is sent back to Jo's browser with the AssertionConsumerService URL configured within the IdP.



Here is a significant difference in the Appgate SDP SAML flow. In this case when the SAML IdP was configured for Appgate SDP Client, the redirect URL was set to http://localhost:29001/saml NOT the URL of the SP (the Controller). The Appgate SDP Client has a listener on http://localhost:29001/saml so receives this message (7). The Controller is the service provider so needs the SAML authentication message. The Client grabs this message and re-tries to sign-in to the Controller (8) but this time with the SAML authentication message attached to the request.

The Controller reads the SAML authentication response message, it verifies the signature and checks things such as the subject of the message (user's identifier [NameID]), then any: Authentication assertions (Jo was authenticated at a specific time via this type of authentication mechanism); Attribute assertions (Jo is associated with the following claims/attributes); Authorization decision assertions (Jo's access to the resource has been granted or not).

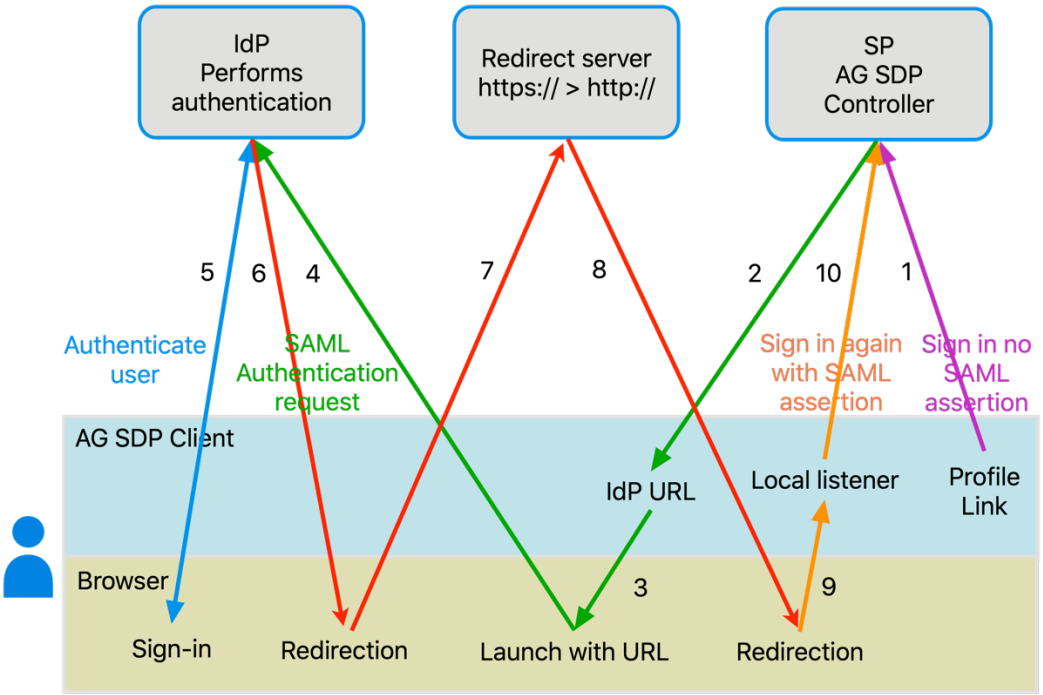
The Controller then maps a number of the SAML assertions into Appgate SDP claims so they can be used for making access decisions downstream. The sign in process is complete and the normal process of issuing Claims and Entitlement tokens continues form here on.

Appgate SDP Case 4 – Client SAML over TLS

As already stated, the primary trust mechanism is for the SP and IdP to have a pre-existing trust relationship, typically involving a Public Key Infrastructure (PKI). So, when an assertion is sent then it is mandated that the response message be digitally signed using the XML digital signature. And where message integrity and message confidentiality are required, then TLS is recommended. This whole flow makes the assumption that the browser trusts the IdP and SP as they will have been set up with properly signed certificates that the browser trusts. But unlike browsers / web servers the Appgate SDP system lives in a world of its own relying on the self-signed certificate issued by the Controller. In order to avoid all these potential trust issues, thus far the Appgate SDP Client has not interfaced to the outside world, however with SAML this situation no longer applies.

Some SAML IdPs may force the use of https:// for the AssertionConsumerServiceUrl, which specifies the SP's URL (localhost in our case). Because of the aforementioned trust issues we have not implemented a https:// web service for the localhost listener so this will not work.

To mitigate this situation when it arises Appgate SDP requires the flow to take a slightly different double redirection route.



The difference with this case is that instead of redirecting Jo's browser to localhost the redirect is to an intermediate redirect server (6,7). In this case the AssertionConsumerService reply URL should be set to https://myredirectserver.mycompany.com when configuring the IdP.

Our SAML implementation has the Client expecting the SAML authentication message on localhost. So, the redirect server does what the IdP did in case 1 and issues another redirect to http://localhost:29001/saml (8) where the Client is still listening (9). The Client is happy because the message was duly delivered on its http://localhost listener. There is no difference as far as the Client is concerned for this case - it can't tell which case is being used. Once the message is received the sign-in process (10) continues as it did in the previous SAML case.

The Portal appliance already performs a redirection service for its (internal) Clients. It can also perform this redirection as it cannot tell the difference between internal and external Clients. So, if you are using a Portal appliance you get this service for free!

Authentication user experiences

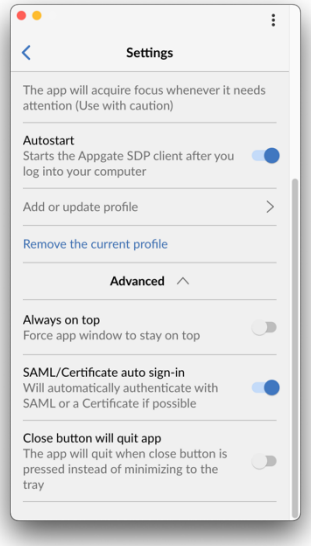
The user experience differs between desktop and mobile operating systems:

Desktop

On desktop operating systems the default browser is used maintain a consistent SAML experience across all the user's apps and preserve SSO between apps.

- There are two user options which affect the user experience:
  - *Autostart* means the Client launches when the device boots and the "Sign in with provider" button will be shown.
  - *SAML auto-launch browser* means the "Sign in with provider" page is skipped and the browser opens with the SAML IdPs sign-in page.

When the browser launches, the user enters any SAML credentials that the IdP requires (unless they have already done so recently).



Mobile

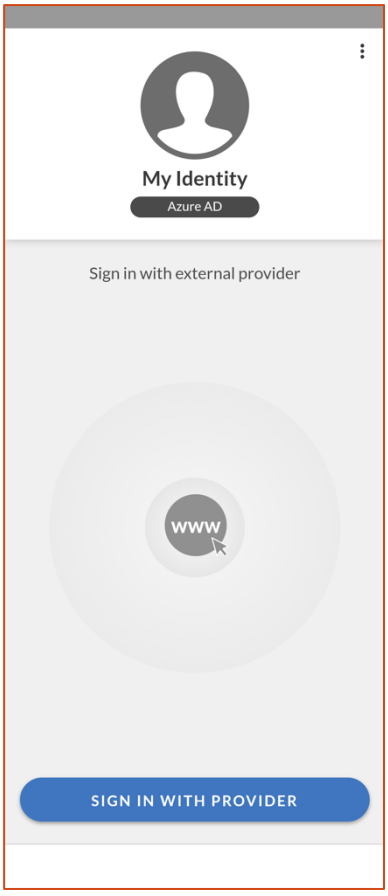
On mobile operating systems, the two options above are not available. The starting and stopping of apps is handled very differently on mobile and the user will always start the app when required – so *Autostart* is not required.

The apps themselves are containerised, so working with other apps (the browser) is not always supported. For the best user experience an in-app browser instance is used which renders the SAML sign-in page served by the SAML IdP.

Since the user experience remains in-app then there is no point in the *SAML auto-sign-in* browser feature.

The in-app user experience is exactly the same as desktop except that the user will always have to enter their credentials the first time because this is a new browser instance. Thereafter they may be available for re-use depending on the SAML provider settings.

This would mean you may not see the actual providers screen – only the one shown here:



Authentication events

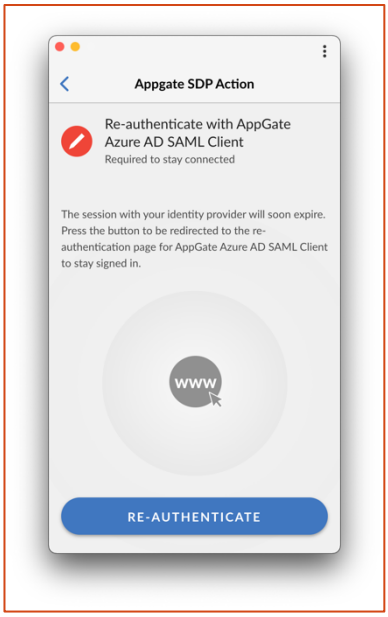
There are 3 occasions while using the Appgate SDP system where the Client may ask the user to perform a SAML authentication:

At Client sign in

The profile link will dictate when the user will be using SAML to sign-in to the Appgate SDP system. The Client itself will not be asking for any credentials such as username and password;

Because the sign-in process is managed by the SAML IdP, Appgate SDP will have no prior knowledge of how the user has been authenticated. However, part of the SAML configuration will include attributes – so the AuthnContextClassRef attribute can be mapped to a Claim in Appgate SDP.

When the Claims token expires



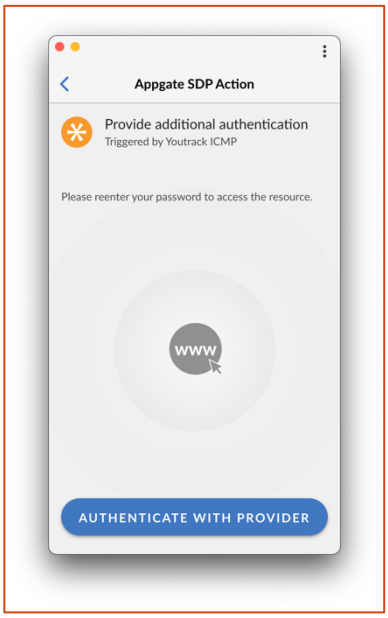
When a claims token expires, the Appgate SDP system wants to re-authenticate you to the IdP before issuing a new one. Without SAML this is done by re-submitting cached username/password credentials to the IdP silently. In the case of SAML, as a SP, Appgate SDP has no knowledge of the credentials so can't do this. This requires a slightly changed process - so in this case you will be prompted by Appgate SDP to go to the SAML provider sign in page.

You have to click "RE-AUTHENTICATE" and the browser will open and the (re)authenticate process begins.

The same process as before is repeated with the user required to enter their SAML credentials unless they are still valid (either because of some other recent SAML authentication or because the SAML token has a longer timeout than the Claims token).

After successful re-authentication the Client captures the SAML protocol message carrying a SAML authentication and sends it to the Controller. A new claims token will be issued.

With a password user interaction



The Appgate SDP system sometimes requires real time confirmation of the user's credentials. This might happen when you try to access a specific resource which is linked to a "retype password" Condition. In the SAML case Appgate SDP must refer to the IdP again - you will be prompted by Appgate SDP to go to the SAML provider sign in page.

You have to click "AUTHENTICATE WITH PROVIDER" and the browser will open in order to (re)authenticate the user.

The same process as before is repeated with the user required to enter their SAML credentials unless they are still valid (because of some other recent SAML authentication or because the SAML token has a longer timeout than the Claims token).

After successful re-authentication the Client captures the SAML protocol message carrying a SAML authentication and sends it to the Controller. A new claims token will be issued.

Configuring SAML

SAML may seem like a complex solution, but it is actually very simple to configure. It is no harder than linking up any other type of identity or authentication solution.

All you need to do it to tell each party about the existence of the other and then to share some 'secret' between them to ensure the security of the overall solution.

Below is a quick summary of the steps required however you should refer to the admin guide and the more detailed integration guides which detail this process.

IdP configuration

To configure SAML for Appgate SDP you should first configure the IdP. In this case we use OKTA as the example. The main things to configure are:

- The Single Sign On URL – the URL of the SP (http://localhost:29001/saml, https://mycontroller.myco.com:8443/saml or https://myportal.myco.com/saml)
- The Audience URI – an attribute that is checked by the SP (Controller) and needs to match.

Most SAMP providers allow the configurations to be exported as XML metadata which Appgate SDP can consume.

Appgate SDP uses claims extensively to make decisions for assignment of Policies and in Conditions. It is therefore important to capture attributes from the IdP for use as claims. There should be a section on Attributes where they can be selected from those available in the user database and specified as to how they will appear in the SAML attributes.

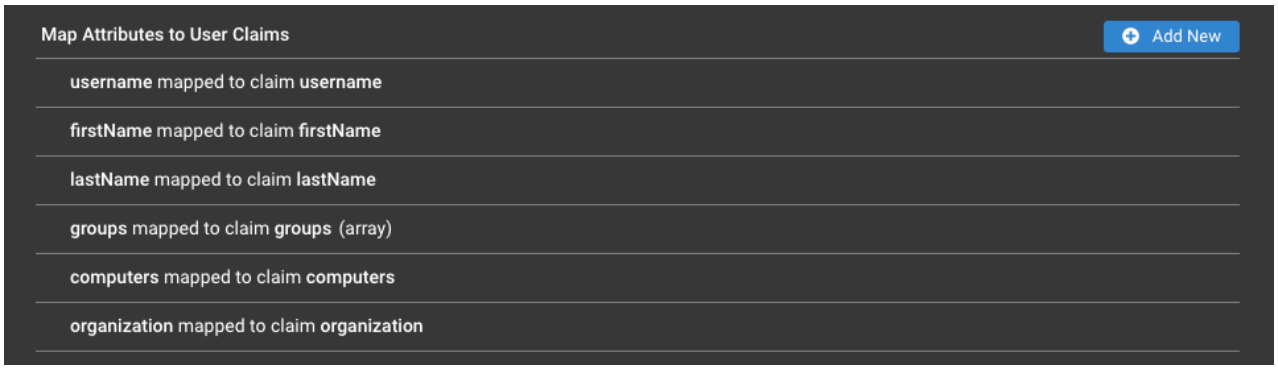
SP (Controller) configuration

Configuring the Controller is also quite straightforward.

Add a new Identity Provider and choose SAML:

- Import the XML meta data
- Add the the Audience is the value you entered into the first screen.
- ForceAuthn provides the option that requires users to enter their credentials every time Client requires SAML authentication.

You must then map SAML attributes set up earlier to Appgate SDP claims:



Resources

You'll find additional resources on the Appgate website.

The Appgate SDP product documentation is available here:

- Admin Guide: https://sdphelp.appgate.com/adminguide/saml-idp.html
- Client User Guide: https://sdphelp.appgate.com/userguide

Access to our support services (including further articles) is via the customer portal.

Appendices

1. Appgate SDP Controller SAML message requirements

These are the requirements that need to be taken into consideration when configuring the IdP to ensure compatibility with Appgate SDP:

- The SAML Response token received must be base64 encoded XML assertion. (Standard)
- Status code must be "urn:oasis:names:tc:SAML:2.0:status:Success"
- At least one assertion must be present. Either regular assertion or encrypted assertion (if there is a decryption key configured in the identity provider).
- Only the first assertion will be validated and used. The rest will be discarded.
- Assertion.Subject.NameID must exist. This value will be used as the "username" in Appgate SDP.
- NameID value will also be added as an attribute with the hardcoded name 'samlNameId', which can then be mapped as a claim.
- The assertion must have an 'AuthnStatement'.
- The assertion must have an 'AttributeStatement'.
- The assertion must have audience defined under 'Assertion.Conditions.AudienceRestrictions[0].Audiences[0].AudienceURI', and must exactly match the value entered in Identity Provider configuration.
- Assertion must have an issuer tag and its value must match the value entered in the Identity Provider configuration.
- Assertion condition's 'NotBefore' and 'NotOnOrAfter' must be valid compared to the system time.
    - The public certificate entered in the Identity Provider Configuration must have been used to create either the assertion signature in `Response > Assertion > Signature > SignatureValue` or the token signature in `Response > Signature > SignatureValue`

Token signature

```
<samlp:Response Destination="http://127.0.0.1:29001/saml".....
  <saml:Issuer>.....</saml:Issuer>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>.....</ds:SignedInfo>
      <ds:SignatureValue>
          xkZmtxWrm2BI7BktR9meBNfUasL.....
```

Assertion Signature

```
<samlp:Response xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol".....
<samlp:Status>
  <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
</samlp:Status>
<Assertion xmlns=.....
  <Issuer>.....</Issuer>
  <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>.....</ds:SignedInfo>
  <ds:SignatureValue>
      ka1ZflX26Ad19Sqbx5gq7DM.....
```

2. Configuring the IdP:

SAML 2.0 supports a number of different bindings of which we use two:

HTTP Redirect Binding

HTTP POST Binding

These are the most commonly used for web SSO. For example, the SP may use HTTP Redirect to send a request while the IdP uses HTTP POST to transmit the response.

For Client sign in

- Clients only support HTTP POST binding: http://localhost:29001/saml
- For HTTPS POST binding a redirection server must be used such as the Portal appliance.

For Portal Client sign in

- POST: https://myportal/saml

For Admin UI sign in

- POST: https://mycontroller:8443/admin/saml
- Redirect: https://mycontroller:8443/ui/sign-in.html
- Port 8443 is set in the appliance configuration 'Admin/API TLS Connection' field.

3. Diagnostics

If anyone wants to see SAML in action then there is a useful diagnostic tool that lets you see what SAML is doing under the covers; SAML tracer is a Firefox plugin which adds a viewer window to Firefox that automatically decodes and shows SAML messages.

4. Example SAML Assertion

The interesting values are highlighted.

```
<?xml version="1.0" encoding="UTF-8"?>
<saml2:Assertion
    xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion" ID="id592873769033074679005651 8"
IssueInstant="2016-05-11T14:52:14.821Z" Version="2.0">
    <saml2:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-
format:entity">http://www.okta.com/Issuer</saml2:Issuer>
    <saml2:Subject>
        <saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-
format:transient">userName</saml2:NameID>
        <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
            <saml2:SubjectConfirmationData NotOnOrAfter="2016-05-11T14:57:14.821Z"
Recipient="http://127.0.0.1:29001/saml"/>
        </saml2:SubjectConfirmation>
    </saml2:Subject>
    <saml2:Conditions NotBefore="2016-05-11T14:47:14.821Z" NotOnOrAfter="2016-05-11T14:57:14.821Z">
        <saml2:AudienceRestriction>
            <saml2:Audience>okta_test</saml2:Audience>
        </saml2:AudienceRestriction>
    </saml2:Conditions>
    <saml2:AuthnStatement AuthnInstant="2016-05-11T14:52:14.821Z">
        <saml2:AuthnContext>

<saml2:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport</saml
2:AuthnContextClassRef>
        </saml2:AuthnContext>
    </saml2:AuthnStatement>
    <saml2:AttributeStatement>
```

```
        <saml2:Attribute Name="username" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
            <saml2:AttributeValue
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="xs:string">user.sign in
            </saml2:AttributeValue>
        </saml2:Attribute>
        <saml2:Attribute Name="id" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
            <saml2:AttributeValue
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="xs:string">user.sign in
            </saml2:AttributeValue>
        </saml2:Attribute>
        <saml2:Attribute Name="firstName" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
            <saml2:AttributeValue
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="xs:string">user.firstName
            </saml2:AttributeValue>
        </saml2:Attribute>
        <saml2:Attribute Name="lastName" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
            <saml2:AttributeValue
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="xs:string">user.lastName
            </saml2:AttributeValue>
        </saml2:Attribute>
        <saml2:Attribute Name="emails" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
            <saml2:AttributeValue
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="xs:string">user.email
            </saml2:AttributeValue>
        </saml2:Attribute>
        <saml2:Attribute Name="groups" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
            <saml2:AttributeValue
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="xs:string">GroupName Match Contains "test" (ignores case)
            </saml2:AttributeValue>
        </saml2:Attribute>
    </saml2:AttributeStatement>
</saml2:Assertion>
```