Authentication

User Authentication

The authentication process within the IdP is built on Spring Web Flow, so it's important to understand the basics of Web Flow before proceeding. The use of flows in this manner allows deployers a high degree of flexibility in customizing behavior.

Authentication Flows

Within the IdP the act of authenticating the subject is performed by the execution of an authentication flow. This is a flow definition that contains all the steps for authenticating subjects (e.g., presenting a form to collect credentials, validating those credentials, re-asking for credentials if validation failed or proceeding if it passed). Each flow has a unique ID and the IdP can be configured with multiple flows. Flows may themselves contain more than one subflow or leverage each other in complex ways, but are responsible for producing one and only one final outcome, which is modeled as an AuthenticationResult.

A given flow will have the following properties describing it via an AuthenticationFlowDescriptor object.

- a unique identifier
- whether it supports passive requests (i.e., ones that are not allowed to present a UI)
- whether it supports forced re-authentication
- the serializer to apply to AuthenticationResult objects it produces
- custom Principal objects that the flow is capable of producing for a Subject
- a maximum lifetime that indicates when a successfully completed flow will be considered inactive regardless of whether it's been used since its first usage
- an inactivity timeout that indicates when a successfully completed flow will be considered inactive if not used before the timeout

The result of an authentication flow is data representing the authenticated subject or an error.

Active Authentication Results

The last two properties above refer to active vs inactive/stale results. Active results are those that are available for use in order to avoid having to ask the subject to authenticate again (i.e., for single sign-on), while inactive/stale results are not available for this purpose (but may remain tracked in a session for other reasons).

An authentication result is active if the current moment in time is less than both:

- the initial use of the producing flow plus its lifetime
- the most recent use of the result plus its timeout

If a subject's session expires or is destroyed (e.g., due to a logout or an exceptional condition), all results are automatically considered inactive for that subject.

Use of Principals

The Java Principal interface is used in two distinct ways:

- to capture the usual user-specific information that makes up an identity after authentication
- to capture non user-specific information about the authentication process

The latter is how the IdP deals with ideas like "authentication method", "identity assurance", or in SAML terms, an AuthenticationContext class or declaration.

As an example, SAML has context class URIs that represent authentication technologies, such as a password, Kerberos, or a hardware token. These are represented by custom Principal objects that contain the URIs. Various objects throughout the subsystem capture this information by exposing collections of Principal objects, via the PrincipalSupportingComponent interface:

- AuthenticationFlowDescriptors
- Validation action beans
- AuthenticationResults

On a flow descriptor, the Principal collection represents the potentially supported capabilities of the flow expressed in terms of Principal objects that would match the objects associated with a request, indicating whether a flow can be used to fulfill it. In a Validation action, the collection indicates whether a validator is able to produce a result that satisfies a request, allowing chains of validators. On a result, the collection represents the actual Principal objects associated with the authentication result, which may limit which results can be reused for SSO for particular requests.
In other words, a request expresses specialized requirements (i.e., those other than well-defined concepts like passive or forced authentication) as a set of custom Principals and a matching operator, and various objects involved in the login process expose sets of Principals as a means of evaluating their compatibility with a request.

A request's particular requirements, if any, are captured in a RequestedPrincipalContext subcontext that is attached to the parent AuthenticationContext. In the absence of the subcontext, the request is assumed to allow for use of any flow or result that otherwise meets the need.

Authentication Process

The act of authenticating a subject contains steps that occur before and after the execution of any individual authentication flow. The full authentication process is roughly as follows, and includes some steps that involve the session layer:

1. Retrieve and validate a subject's session, if any (if configured to do so), to establish any active flows.
2. Filter potential/configured flows by any passive/forced requirements, as well as browser/non-browser compatibility, as required.
3. Is specific custom Principal support requested or indicated as a default?
   a. If yes, choose the first potential flow compatible with the request and execute it (or if it's already active, reuse it). An option is provided to prioritize active results over inactive flows (this is a current V2 behavior that is not SAML compliant).
   b. If no, check for a currently active result and reuse it. Otherwise select a potential flow and execute it.
4. To reuse a result, its last activity time is updated and the result is recorded/copied to the AuthenticationContext. No "attempted flow" is recorded.
5. To execute a flow:
   a. Record it as the attempted flow within the AuthenticationContext.
   b. Branch to it.
   c. On success:
      i. An AuthenticationResult is produced and set into the AuthenticationContext.
      ii. A SubjectCanonicalizationContext is created containing the Subject from the AuthenticationResult, and control passes to a SubjectCanonicalization subflow. Failure here would lead to an overall failure of the authentication flow (as in d. below).
      iii. If the identity has switched (as compared to the active session), any existing session is invalidated and the AuthenticationContext and SessionContext objects are cleaned up to reflect this.
      iv. Assuming success, the canonical principal name is copied into the AuthenticationResult.
   d. On failure, return an error. This error may or may not eventually be sent back to the relying party, that's outside the scope of the authentication process.
6. To complete the authentication process successfully, a SubjectContext is created around information copied from the AuthenticationContext.
7. Optionally a session is created or updated with the modified or new AuthenticationResult to enable future SSO.

Flows can be tailored to map specific error events they signal into a "ReselectFlow" event that allows the authentication flow as a whole to retry the selection step and pick another eligible subflow to try. This allows "passive" mechanisms like IP-based or REMOTE_USER-based flows to fail non-fatally. A lot of additional signaling features are available.

Authentication- and Session-Related Context Tree

The layout of the contexts used during authentication follows:

Graphviz output could not be displayed here.

The sections below provide general guidance on how these contexts are used. The API documentation is the best source of detailed information on their contents.

RelyingPartyContext

Contains a reference to the active RelyingPartyConfiguration and ProfileConfiguration, as outlined by ProfileHandling. This is mostly relevant for the user interface, but does allow behavior during authentication to be otherwise tailored by relying party if necessary.

SessionContext

Contains a reference to an active session with the user agent, if any. The entire session mechanism can be disabled by eliding actions from the profile flow that would populate or read from this context. It is a strictly optional part of the authentication subsystem.

AuthenticationContext

Tracks the inputs and outputs of the authentication process, including state during the process. Actions within an authentication flow translate the unvalidated information into validated results that are then "trusted" by other components. Once the authentication process completes, the information that needs to be preserved is copied into the SubjectContext.

The main product of the authentication components is an AuthenticationResult, which represents the information associated with a completed flow. The main product is a Java Subject. If an authentication flow internally consists of a composition of multiple forms of authentication, only a single AuthenticationResult is exposed. Internal state associated with the act of authentication can be maintained by attaching information to the Java Subject.

AuthenticationErrorContext, AuthenticationWarningContext

These contexts track warnings and errors detected during the authentication process, primarily those occurring during the attempt to execute a flow. This allows actions to know why a login attempt might have failed, and captures an ordered list of exceptions as well as a set of flags signifying "important" conditions like an unknown username, invalid password, locked account, etc. The main use of this information is of course during rendering of login forms, to enable error message presentation.
RequestedPrincipalContext

This reflects specific requirements for how authentication should be done. Custom Principals are used to represent protocol-specific requirements affecting the authentication process, and a request may specify an ordered list of Principals to match against, and a matching “operator”, the allowable values of which are unconstrained.

When this subcontext is present, only flows (or results) that evaluate as a “match” to one of the requested Principals will be considered for use. The matching process is accomplished through configured machinery called a PrincipalEvalPredicateFactoryRegistry.

SubjectCanonicalizationContext

Temporary context that holds the state of the canonicalization process. After producing a successful result, an authentication flow populates this context with the Java Subject along with some basic information about the request and executes the c14n subsystem, which must set the canonical principal name inside this context, or fail.

SubjectContext

The overall output of the authentication process contains information about the identity interacting with the IdP in the role of authenticated subject, typically an end user. Within a given profile web flow, the question of “who is the user?” is meant to be answered with this context. Contrast this with the SessionContext, which one might think of as answering “who was the user?”

- principalName – the canonical/normalized identifier for the subject to be used by most components that rely on the subject’s identity
- authenticationResults - a collection of previous AuthenticationResult objects representing the subject’s identity in “raw” form

The model for user identification is a single canonical/normalized identifier representing any authenticated subject. All components other than those dealing with the authentication and identifier normalization process should generally deal only with the normalized identifier, typically only by means of this context. The exception would be a component requiring access to technology-specific extensions to a subject’s identity such as non-neutral identifiers like a DN, or a Kerberos ticket, or some other kind of credential.

Programming Guide to Extending Authentication

Extending the authentication features of the IdP generally involves creating a custom login flow for the system to execute. There is virtually no limit on what such a flow can do, as it has full control of the IdP once it has been run.

A login flow is a subflow that is assigned a flow ID that starts with “auth.” and is further defined to the system with a bean of type net.shibboleth.idp.authn.AuthenticationFlowDescriptor in a list in conf/authn/general-authn.xml. It must generally be, further, enabled via the idp.authn.flows property.

While you may deliver a custom flow in a relatively “drop-in”, self-contained jar, you MAY NOT manipulate the state of the IdP at runtime to install the necessary descriptor bean because it is impossible to guarantee that your modification will take place early enough to be seen by other objects in the system. There is no publically supported mechanism to extend any of the beans defined inside the “root” web app context, and so you MUST rely on the deployer making the necessary adjustments to define custom flows to the system via the associated type of FlowDescriptor.

Internal Contract

Login flows must interact with the system by accessing and mutating the context tree in specific ways.

The input contract is as follows:

1. An AuthenticationContext will exist and be populated (e.g. isPassive, isForceAuthn, etc.).
2. A RequestedPrincipalContext may exist and will be populated if present.
3. A called login flow will be present in the collection of available login flows (that is, if a flow is run, it can assume it was supposed to run).
4. If a pre-existing result of a called login flow is present as an active result, it could not have satisfied any RequestedPrincipalContext present, or forced authentication is required. This inference holds because otherwise the previously active result would be used for SSO.
5. There may be intermediate state present in the form of a previously produced AuthenticationResult if a flow is called as part of some higher level orchestration, but flows should generally assume that they are free to act as if no such state exists unless their logic is dependent on recognizing and acting on that state. In other words, if you don’t care about intermediate results, you don’t have to care about intermediate results.

The output contract is as follows:

1. If a login flow completes with a "proceed" event, then it MUST satisfy the following requirements:
   a. An AuthenticationResult that represents the desired outcome MUST be set in the AuthenticationContext. It is not strictly required that the result satisfy any RequestedPrincipalContext present, but failure to ensure this will result in an error later.
   b. A SubjectCanonicalizationContext MUST be present below the root context and MUST contain the Subject to canonicalize from the login process (generally this will be from the AuthenticationResult but does not strictly have to be).
   c. Starting with V3.3, the login flow SHOULD run the SubjectCanonicalization master subflow, but it MAY leave that for the authentication master flow. The signal for this is whether the principalName field in the SubjectCanonicalizationContext is populated after the flow completes.
2. If a login flow completes with any other event, it should assume that its outcome will be treated as unsuccessful. Flows may signal specific behavior back to the authentication master flow:
   a. ReselectFlow — tells the master authentication flow to choose another eligible flow to run (i.e., fall through)  
   b. Flow ID of another login flow — tells the master authentication flow to run that flow next (see the following section)
c. Anything Else – authentication will fail and the event will be reflected back as the result of the authentication master flow, to be interpreted by the calling parent flow (signaling a custom event requires that conf/authn/authn-events-flow.xml be modified)

General Considerations for Custom Login Flows

Composing and Combining Flows

It's possible to pass control from one login flow to another. An example use case would be a button on a login form that triggers a more advanced mechanism such as certificate, token, or SPNEGO authentication that might require a user to opt-in to using it. Another use case would be to create a special "pseudo" login flow that doesn't do any work itself, but presents a list of options to the user for selection.

There are a few different ways to make this happen, with different formalism and impact on the rest of the configuration. If a flow calls another flow internally, the second flow is generally being executed in service of the first. That is, the IdP considers the "attempted login flow" to be the original one and any results produced by the called flow, if left in place, will be associated with the original flow. As a consequence, the called flow doesn't have to be "known" to the IdP as an available flow for it to run because only the calling flow is "real" to it. This can be very powerful to enable composition of flows while hiding the complexity within the calling flow to manage the results.

A similar strategy is for the calling flow to reset the attemptedFlow property to the flow it is about to call, which will cause the IdP to associate the results with the called flow and to evaluate the results it produces based on the configuration of that flow. This works well if the called flow is very distinct in nature from the calling flow (e.g. branching from a password form to certificate authentication).

A more formal way to branch to a new flow is by signaling to the IdP that it should run a different login flow by returning an event. To do this signaling, a login flow must end in a WebFlow event using the name of the login flow to run next (which must be prefixed by "authn"). For example, to run a flow named "authn/SPNEGO", a button or link might submit a parameter named "/_eventId_authn/SPNEGO" (the parameter value can be anything).

This signaling approach has limitations when it comes to error handling. The signaling flow gives up control and if the signaled flow fails or signals for a reselection, control passes back to the master selection process and not to the original flow.

Flow Cancellation

It's possible to provide users with a way to "cancel" the use of a login flow, perhaps to indicate that can't successfully complete it, or to just cancel the request outright.

Flows use event IDs to signal their outcome back to the master subflow that orchestrates AuthenticationFlowSelection. The built-in login flows include configuration support for mapping various conditions they encounter into specific event IDs. If a login flow produces the "ReselectFlow" event, that causes the master subflow to try and find a new login flow to attempt, but any other unsuccessful event will essentially cancel the login process.

You can take advantage of that machinery to include a button or link a user can click to signal a custom (or built-in) event ID and then you can choose how to handle that event as the overall result of the authentication process with either a local error handling view or by returning control to a requesting service (see the ErrorHandlingConfiguration topic for a discussion of error event handling).

Programming Guide to Using Authentication

If you're creating a custom profile/protocol feature that needs authentication, a web flow that wishes to invoke the Authentication subsystem must do the following:

1. Create an AuthenticationContext and populate any of the fields derived from the protocol-specific context in which the caller is operating. For example, a SAML profile handler would populate the the passive and forced authentication flags based on the IsPassive and ForceAuthn attributes in the <AuthnRequest> message. Note that no fields need be populated in order to invoke the subsystem; the default content is sufficient.
2. Attach the AuthenticationContext as a child of the ProfileRequestContext.
3. If applicable to the profile request, create, populate, and attach a RequestedPrincipalContext as a child of the AuthenticationContext.
4. Transfer control to the "authn" subflow and transition on the possible results back to your own flow.

The following events worthy of special note may occur as a result of invoking the subsystem:

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>proceed</td>
<td>Successful authentication.</td>
</tr>
<tr>
<td>RestartAuthentication</td>
<td>Authentication should be repeated from scratch (including creation of a new AuthenticationContext and related content).</td>
</tr>
<tr>
<td>NoPotentialFlow</td>
<td>No authentication flow is configured for use. This can also indicate that a request for passive authentication prevented use of all the possible flows.</td>
</tr>
<tr>
<td>RequestUnsupported</td>
<td>No flow or active result met the requirements of a RequestedPrincipalContext.</td>
</tr>
<tr>
<td>NoCredentials</td>
<td>A flow was unable to extract or obtain credentials from the subject.</td>
</tr>
<tr>
<td>InvalidCredentials</td>
<td>A flow was unable to successfully validate credentials from the subject. Often this event and the previous event may be isolated within a flow because the subject is expected to keep trying, but there may be retry limits or other factors that will expose this condition.</td>
</tr>
<tr>
<td>SubjectCanonicalizationError</td>
<td>The Subject resulting from authentication couldn't be turned into a canonical principal name.</td>
</tr>
</tbody>
</table>
Various other events signifying more low-level error conditions may also occur.

In any case other than "proceed", the caller **MUST NOT** expect any results to be valid, except for the possible presence of an AuthenticationErrorContext as a child of the AuthenticationContext, and the attempted flow, if processing reached that point.

In the case of "proceed", all fields of the AuthenticationContext are populated as documented, including:

- the attempted flow, in the event that authentication was freshly performed
- the AuthenticationResult produced or reused

Success **will** populate a SubjectContext as a child of the ProfileRequestContext. Use of the authenticated subject **MUST** be based on the content of this subcontext.

Success **may** populate a SessionContext as a child of the ProfileRequestContext.