thiss-jquery-plugin Documentation

Release 1.0.4

Leif Johansson

Mar 10, 2020

Contents:

This is a jquery plugin for interacting with a thiss.io discovery and persistence service infrastructure. Specifically the jquery.thiss.ds-widget.js plugin turns an input field and a bit of surrounding markup into a typeahead search with optional display of previous selections - all fetched from an MDQ server.

CHAPTER 1

Introduction

The Identity Selector Software (thiss.io) is an implementation of an identity selector supported by the Coalition for Seamless Access. It implements a discovery service using the RA21.org recommended practices for discovery UX.

The Identity Selector Software suite is a front-channel identity selector for distributed identity ecosystems aka Federated Identity Management. The objective is to simplify the process of choosing an "identity provider" by having the browser remember the users choice in browser local store. Currently the system has been used for large-scale SAML-based identity federations but there are no intrinsic dependencies to SAML as such and the system could be easily adapted to other protocols that follow the common pattern of federation by relying on redirecting the user to an authentication provider of some sort.

The system was designed with privacy as the number one focus. No information is shared with the relying party during the identity provider choice process. This is ensured by relying on the browser security model and judicious use of inter-domain communicatiton using post-message.

This package (thiss-jquery-plugin) contains a jQuery plugin for building your own discovery service on top of a thiss-js service (eg use.thiss.io or service.seamlessaccess.org).

1.1 Architecture

The Identity Selector Software (thiss.io) is a set of front-channel (aka browser-based) cross-domain APIs using post-message (built using the post-robot package):

- A persistence API that allows store & retrieval of information about the last N (3) identity providers used to authenticate a user. Unlike simlilar project (eg google account chooser) the information stored does not include any PII (eg email-addresses) but only identifies the identity provider used in a way consistent with the authentication protocol used.
- A discovery API that implements SAML identity provider discovery layered on top of the persistence API

Both of these APIs have a *server* and a *client* component. The client components can be found in this library and can be imported (using npm) in existing projects. The server components can be found in the thiss-js repository. The server component is implemented as javascript running in an iframe fetched from a service URI. This ORIGIN (in the sense of the w3c security model) protects access to the browser local store and ensures that the calling page only has

access to the intended API. The calling page (aka the client) is responsible for initializing the iframe but after this no longer has any control over the code executing inside it. The *server* iframe, while executing in the client browser, is therefore sandboxed from the calling page.

The persistence API is completely protocol agnostic eg has no dependency on SAML, all of which are in the discovery API. Future versions are expected to provide similar APIs for OpenID Connect supporting OpenID connect federation and possibly other protocols.

A relying party (aka SP) will typically not integrate directly with these APIs but will rely on higher-level services built using these APIs, eg those provided by and instance of thiss-js such as use.thiss.io or service.eamlessaccess.org

1.2 Audience

This documentation is targeted at developers who want to build their own identity provider selector service but don't want to deal directly with the the low-level APIs. Most relying parties do not want to do this but should instead relying on the highlevel services provided by an instance of thiss-js. Possible uses for this package include federation operators and/or relying parties with highly complex metadata infrastructure that need to operate their own discovery service but still want to take advantage of the persistence service offered by an existing thiss.io infrastructure. Readers are assumed to have a working knowledge of front channel development and associated tooling (eg webpack, babel, npm etc).

CHAPTER 2

Installing thiss-jquery-plugin

Install via npm is straight-forward:

```
# npm install [--save] @theidentityselector/thiss-jquery-plugin
```

The thiss-ds package supports both CommonJS-style and ES6 import aswell as old-school CDN delivery. Note that jQuery is not bundled and needs to be provided as an external dependency.

CommonJS:

```
require("thiss-jquery-plugin");
```

ES6-style

```
import '@theidentityselector/thiss-jquery-plugin';
```

CDN (thanks to unpkg.com)

```
<script src="//unpkg.com/browse/@theidentityselector/thiss-jquery-plugin" />
```

CHAPTER 3

Using thiss-jquery-plugin

3.1 Overview

The package contains (at present) a single jQuery plugin that does several things:

1. display an identity selector based on a search-box and type-ahead. Should be combined with a search-capable MDQ server.
2. display information about a relying party based on MDQ lookup (eg title, icon etc)

Combined these capabilities form the basis upon which an identity UX can be built. Note that this plugin does not enforce any markup/layout - such things must be provided by the calling application.

3.2 Discovery Service

There is a simple demo application in index.html/demo.js/demo.css also provided as a standalone application in the thiss-examples repository. Initializing the plugin is pretty straight-formward. This example illustrates the minimal set of required attributes:

```
$(selector).discovery_client({
  persistence: 'https://use.this.io/ps',
  search: 'https://md.thiss.io/entities/',
  mdq: 'https://md.thiss.io/entities/',
  render: function (item) { ... },
  search_result_selector: '#ds-search-list',
  saved_choices_selector: '#ds-saved-choices',
  input_field_selector: 'input',
  entity_selector: '.identityprovider'
});
```

Typically this is done inside a standard jQuery `$(document).ready { ... }` wrapper. The persistence is either an instance of the `PersitenceService` class from thiss-ds-js or the URL of a persistence service as in the example above. The search and mdq paramaters are either URLs or callable javascript functions which take a single

string argument (entityID or search string) and return an entity or list of entities as apropriate. The format of the returned objects must observe the schema from thiss-ds-js.

This call would initialize an identity selector on `selector` which is assumed to be a `div` or similar. Search results are rendered using the render function (must return DOM nodes matching the `entity_selector` selector) in the `search_result_selector` element. If present any saved choices from the persistence service are show in `saved_choices_selector`. The other attributes should be pretty-self-explanatory.

The element indentified by `input_field_selector` (by default any input-field) is used to create the search field. Typical markup might look something like this:

```
<ul class="list-unstyled" id="ds-saved-choices"></ul>
<div id="ds">
   <form>
      <div class="form-content">
         <input autocomplete="off" type="search" placeholder="Search for an identity␣
↪provider..." autofocus>
      </div>
   </form>
   <ul class="list-unstyled" id="ds-search-list"></ul>
</div>
```

3.3 SP Metadata

The plugin can also lookup and extract SP metadata based on the `entityID` parameter which is present in the query string of the URL if the application is beeing called as a SAML Identity Provider Discovery service. Here is how to call the plugin chained after the example call above:

```
$(selector).discovery_client({...}).discovery_client('sp').then(entity => $(some_div).
↪text(entity.title)).
```

The call to `discovery_client('sp')` returns a promise resolving to an object that contains the display-string of the SP. Use a call to `then` to resolve the promise and pass to a function that can do something like populating a UI element with the information received. This can be called multiple times but each will result in a separate call to the underlying MDQ service which probably results in network traffic - caveat emptor.

CHAPTER 4

Indices and tables

- genindex
- search