SAML CSRF Prevention

Overview

This document captures design, considerations, and (eventually) decisions regarding the mitigation of CSRF attacks in the context of mod-login-saml. Here are some helpful links providing background information on CSRF attacks and SAML in general:

- https://wiki.shibboleth.net/confluence/display/CONCEPT/FlowsAndConfig
- https://tools.ietf.org/html/rfc6749#section-10.12
- There's a lot of information out there about these things - if you want/need more background, google can help.

Approach

In order to prevent CSRF attacks, we need some way to check if the user making the call to POST /saml/login is the same user making the call to POST /saml/callback. At a high level, we're looking at something like this:

1. user clicks "sign in via sso", triggering a call to POST /saml/login.
2. mod-saml-login generates a csrfToken, sets a cookie with that value. Also includes the csrfToken in the RelayState param returned along with the SAMLRequest
3. stripes either redirects or POSTS to the IdP - SAMLRequest + RelayState (depending on which binding is being used)
4. the IdP challenges the user for credentials and generates an auto-submitting form that POSTS to /saml/callback with the SAMLResponse and original RelayState (and the csrfToken cookie set for folio's domain)
5. mod-login-saml compares the csrfToken from RelayState to the value in the csrfToken cookie, if they match we proceed, otherwise it suggests that this could be a CSRF attempt and we reject

Here's where things break:

- Stripes calls POST /saml/login via fetch and needs to specify `credentials: 'include'` so the cookie isn't silently ignored
- Even with that the cookie will still be ignored if `Access-Control-Allow-Origin: *`, which it currently is for all cross-site requests. For this to work we need two things:
    - Set `Access-Control-Allow-Origin` to <origin> and whitelist the stripes origin and IdP's origin. Okapi uses Vertx's CorsHandler, which expects a regex to be provided in the `create(pattern)` method, so we'd have to craft a regex that includes the origins we want to whitelist
    - `Set Access-Control-Allow-Credentials: true`

mod-login-saml

- `POST /saml/login`
    - Generate a CSRF token (really just a nonce; pac4j-vertx's `DefaultCsrfTokenGenerator` can be used)
    - Include the CSRF token in the RelayState response property... `RelayState = <stripes URL to send user to once logged in>?csrfToken=<csrfToken>`
    - Set a cookie `csrfToken=<csrfToken>; Path=/; Domain=<Stripes domain>; SameSite=Lax`
        - Is there a need to use `SameSite: None; Secure`?
- `POST /saml/callback`
    - Compare the CSRF token value from the csrfToken cookie and parsed from RelayState. If they don't match, return a 401, otherwise proceed as usual.

stripes-core

- CORS - In order to set a cookie, the fetch must specify `credentials: 'include'`

OKAPI

- CORS - In order to set a cookie, `Access-Control-Allow-Origin` can't be `*` and we need `Access-Control-Allow-Credentials: true`.
    - Options:
        1. Allow this to be configurable?
            - Would need to be configurable on a per-tenant basis
            - Would need to be dynamic to handle new tenants
            - Need to be careful not to make it difficult for UI/edge module developers
        2. Skip CORS handling for the `/_/invoke/tenants/<tenantId>/<path>` route - delegate this to the module being called

        - Easier to configure - there are already tenant-specific SSO settings that are read by this module
        - Isolates the changes to a much smaller portion of FOLIO (really only mod-login-saml?)
    3. Make mod-login-saml directly accessible (a la the edge modules) and handle CORS in the module.
        - this module would need to log in as an institutional/system user in order to make the necessary calls to users /configuration/etc.
        - How this would work requires additional thought and would likely require a fair amount of refactoring.
    4. Introduce a way to specify whether or not CORS handling should be enabled or not in the module descriptor definition for a given endpoint
        - Default would be true (current behavior)
        - In OKAPI, when handling requests to /_/invoke/tenant/<tenantId>/<path>, check if the target endpoint wants CORS handled by OKAPI or delegated to the module.
        - Example:
            - Here, if delegateCORS == true, OKAPI would

```
...elided...
{
  "methods": [
    "POST"
  ],
  "pathPattern": "/saml/login",
  "modulePermissions": [
    "configuration.entries.collection.get"
  ],
  "delegateCORS": true
},
{
  "methods": [
    "POST"
  ],
  "pathPattern": "/saml/callback",
  "modulePermissions": [
    "auth.signtoken",
    "configuration.entries.collection.get",
    "users.collection.get"
  ],
  "delegateCORS": true
},
...elided...
```

- NOTE: Depending no how we choose to implement refresh tokens, these changes may be applicable to our that conversation as well.

Frontend Redirect/Callback URL

Once the user authenticates with the IdP, the IdP generages an auto-submitting form that posts to a callback URL. Currently this is POST /saml/callback handled by mod-login-saml. One idea was to use a stripes URL instead and essentially proxy the callback to the backend. The idea here is that it simplifies the CORS handling for the FOLIO backend, which would only need to whitelist the stripes origin, not the IdP origin as well. My opinion is that this isn't really worth it, but I thought it should be mentioned here anyway.

Pros:

- The FOLIO backend wouldn't need to whitelist the IdP's origin

Cons:

- Another "hop" in the flow
- I think we're really just moving the CORS issue from okapi to stripes

Other Considerations:

- We have to build in support for specifying an origin whitelist anyway to accommodate stripes/okapi being on different hosts/domains, so adding the IdP origin to the whitelist isn't all that difficult.
- Running front-end and back-end on the same host like https://folio.example.com/ and https://folio.example.com/okapi/ (sample proxy) avoids many pre-flight CORS requests and the latency they cause.

JIRAs

- MODLOGSAML-59 - Getting issue details... STATUS

Other Considerations

- Assuming we go with this basic approach described here, we could take it one step further to fix MODLOGSAML-58 - Getting issue details... STATUS . More specifically:
  - Saving the entire RelayState in the cookie, and comparing the entire RelayState value in POST /saml/callback.
  - We might also want to do some validation against the "stripesURL" value provided to POST /saml/login to ensure that it at least matches the origin and that origin is one that's whitelisted in CORS.