STScI | SPACE TELESCOPE SCIENCE INSTITUTE

EXPANDING THE FRONTIERS OF SPACE ASTRONOMY

OAuth and Shibboleth with MAST

Tom Donaldson – Christian Mesh

IVOA Interop, College Park, November 2018

Default Shibboleth Setup

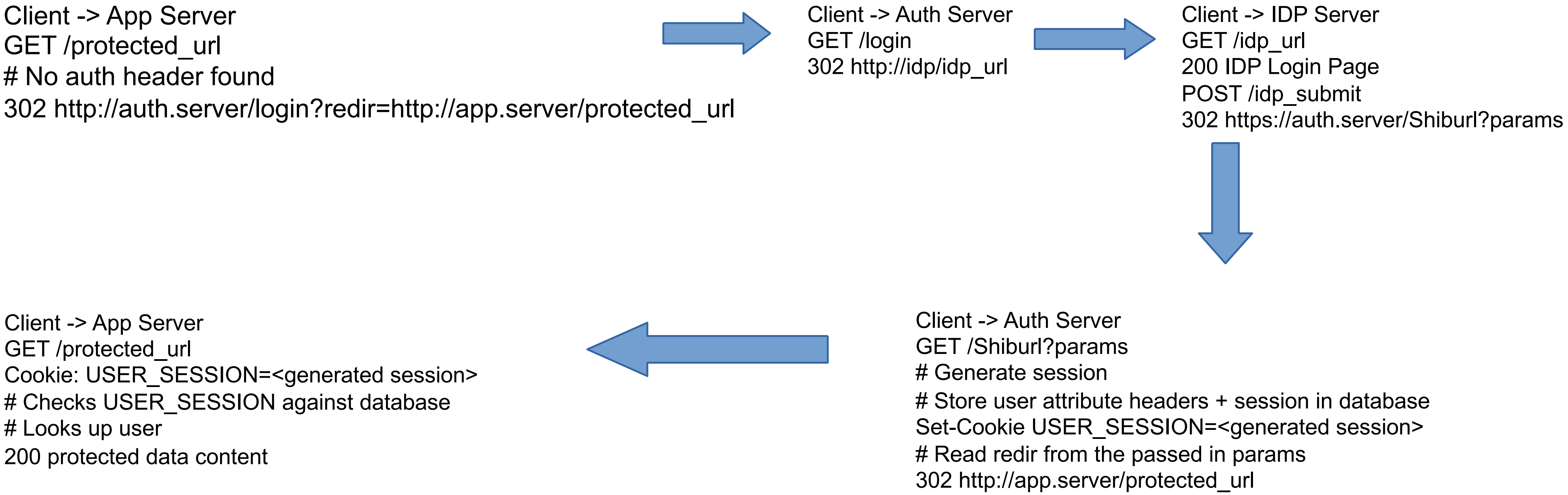
- Each server has an instance of Shibboleth integrated with the webserver (IIS, Apache, etc.)
- Shibboleth is configured to protect specific routes based on user attributes.
  - Routes can require an authenticated session, which forces a login,
  - Or be configured to allow anonymous traffic.
- Shibboleth adds user attributes to headers for each request on configured routes.
- The application reads the headers to identify the authenticated user, if any.

Implementation Issues

- Shib sessions are established per server.
  - Requires sticky routing if more than one server is used.
  - After authN on one server, if you get load balanced to another server, shib won’t know your identity unless another interaction with the IdP is forced.
  - Worse for routes that allow anonymous access, since shib won’t trigger the IdP interaction.
- Non-browser clients are not set up to handle the myriad of redirects that happen during authN.
  - Lack of API token support, so a programmatic client needs to go through a full login process.
  - Enhanced Client Protocol (ECP) helps, but still requires an absurd amount of client code.
- Lack of first class support in many applications (E.g., Jupyterhub)

Hybrid Approach

- Place a sidecar authentication application behind Shibboleth to store the user attrib headers and generates a session
    - Uses Shibboleth for initial login flow
    - Stores the user info and session in a database and returns a Set-Cookie directive for the session
- Applications can check the headers passed in for a session cookie and look up the user in the database.
- If a user is not found, redirect to shibboleth sidecar for routes that require authN
- Applications can also be configured to use a shared service for looking up user info from headers



Client -> App Server
GET /protected_url
# No auth header found
302 http://auth.server/login?redir=http://app.server/protected_url

Client -> Auth Server
GET /login
302 http://idp/idp_url

Client -> IDP Server
GET /idp_url
200 IDP Login Page
POST /idp_submit
302 https://auth.server/Shiburl?params

Client -> Auth Server
GET /Shiburl?params
# Generate session
# Store user attribute headers + session in database
Set-Cookie USER_SESSION=<generated session>
# Read redir from the passed in params
302 http://app.server/protected_url

Client -> App Server
GET /protected_url
Cookie: USER_SESSION=<generated session>
# Checks USER_SESSION against database
# Looks up user
200 protected data content

Improving developer / user workflow

- Instead of having every application talk to the auth database, it can instead make a request to a route on the authorization server with all of the headers it received
- This route can return a serialized user object (we use json) that the application can then utilize. It's much easier to retrofit existing applications using this technique
- Adding support for API tokens. Entries can be added to the auth database which point at the user info normally set by a session
- Users can be sent to a site on the authorization server which exposes a token creation interface

OAuth Support

- Now that we have the concept of API tokens, it’s a small amount of work to build an OAuth provider service to live on the auth server.
- OAuth is supported by most web applications / web frameworks and is an industry standard.
- Web app integration is as easy as using a 3rd party library for most languages
    - No per-server installation/configuration as was required with shibboleth.
- Supports scoped access
    - The user only authorizes partial account access for the OAuth token
- If an OAuth token is exposed, it is easy to revoke and limited in scope

MAST Deployment

- We are deploying support for Auth.MAST in the Portal on Monday
- Our implementation includes all of the techniques mentioned above
  - Shibboleth running on a sidecar host, proxying certain requests to our auth application
  - Existing MAST applications (such as the Portal) are being changed to ask the auth application for information about the current user via a service (by passing along the headers it received)
  - All MAST applications are under the mast subdomain and can share cookies. This allows the above two points to function.
  - New MAST applications are built to query the auth database directly
  - Both internal and external applications can be configured to use our OAuth provider.
    - This has been a few lines of configuration for each instance.
    - Includes Jupyterlab instances on AWS.

Thank you!