Keycloak Integration

Identity Providers

Keycloak Integration: Identity Providers

After reviewing these slides, you will hopefully:

- Know how Keycloak defines an external Identity Provider (IdP)
- Know which kinds of Identity Providers can be use with Keycloak
- Learn how Keycloak integrates with the example provider (GitHub)
- Learn some of the nuances regarding the system
- (Optional) Follow along and confirm that the example provider works as intended

Outline

This discussion is divided into the following sections:

- Introduction
  - What is Keycloak?
  - What is an Identity Provider?
  - Why bother?
  - (Optional) Getting Started
- Configuring Identity Providers
- Example Using GitHub's API
- Example Using a generic SAML provider
- Integration Nuances

These sections will be accompanied by our example Keycloak instance, but following along with that configuration is optional.

Introduction: Keycloak

- What is Keycloak?
  - Keycloak is an open-source identity and access management system
  - It functions as a Single-Sign On (SSO) system for modern services

- As an ADL engineer, you've probably already encountered it:
  - TLA used it as an SSO system and an identity provider
  - It's typically our go-to for recommending a security configuration with xAPI

- Weeds:
  - Java application that runs on the open-source Wildfly runtime
  - Uses OpenID Connect (a profile of OAuth 2)

Introduction: Identity Providers

Definition:

- *An **identity provider** (abbreviated **IdP**) is a system entity that creates, maintains, and manages **identity** information for principals while providing authentication services to relying applications within a federation or distributed network.*
- TL;DR: A system capable of maintaining and authenticating users

Examples:

- Social Media (Microsoft, Google, Twitter, Facebook, GitHub, etc)
- OpenID Connect (Keycloak)
- SAML v2.0

Introduction: Why Bother?

- Account fatigue:
  - Remembering multiple login credentials
  - Creating a new account for a service you use infrequently

- Increase modularity of the TLA ecosystem
  - Integration with **existing** Identity Providers
  - Instantly gives system access to an entire database users

- SAML and LDAP integration
  - Unlikely that everything will be using OpenID Connect
  - Standards change over time
  - SAML and LDAP are popular ways of interacting with existing user stores

Introduction: Getting Started

We'll start by getting our environment set up. This example will be using an Ubuntu 16.04+ Virtual Machine.

TL;DR:

- `git clone https://github.com/vbhayden/keycloak-federation-examples`
- `sudo ./install-reqs.sh`
- `sudo ./rebuild.sh`

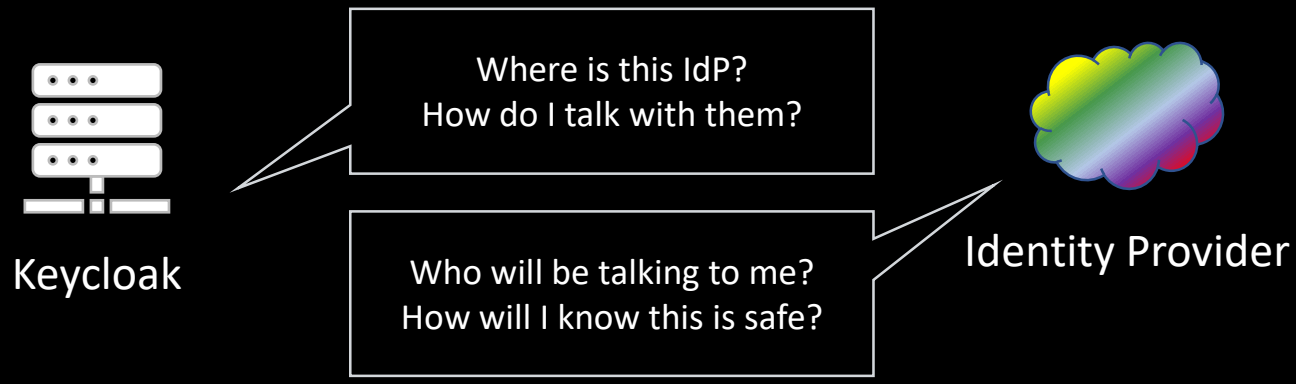
This should have set up the following:

- A Keycloak instance with Wildfly
- A Postgres SQL database
- An OpenLDAP instance with a web-accessible admin UI
- A SimpleSAML instance
- An example service protected by Keycloak

Configuring Identity Providers

To configure an Identity Provider with Keycloak, there are two parts:

- Configuring Keycloak to use another service's users and where to find them
- Configuring that other service to **expect** and **accept** requests from Keycloak



Configuring Identity Providers

1. Register a **client** (sometimes called an **app**) within the Identity Provider

    - Identity Providers usually require you to register some sort of client that will be responsible for the authentication handshakes between it and your OpenID Connect provider (Keycloak). This is for a few different reasons, including client permissions, client revocation, modularity within the provider, etc.

    - Some providers are better than others with their documentation, so finding out how to actually create one of these clients can be tricky.

    - You might want your Keycloak instance running in order to provide a proper, navigable redirect URL for the provider, but this isn't required for our example.

You should now have a set of credentials to use for the Keycloak side.

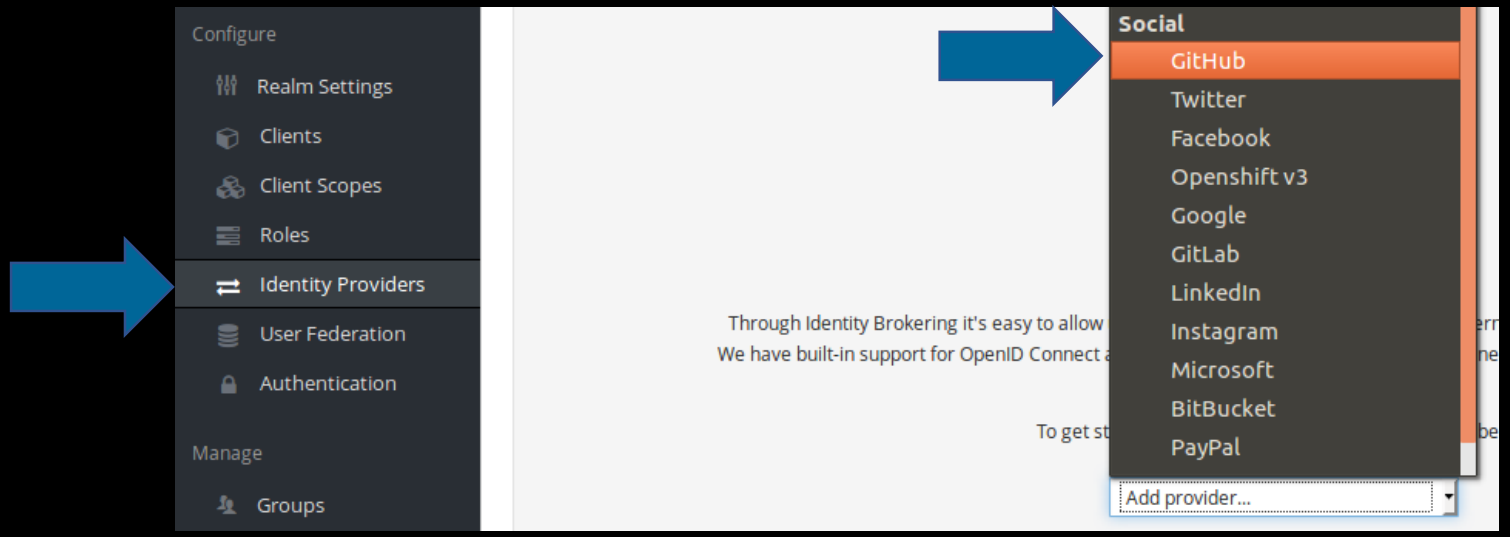
Configuring Identity Providers

2. Use those credentials to enable the Identity Provider within Keycloak

- As this process is actually a transaction of user data from the Identity Provider's API to Keycloak, we will require some sort of permission (usually through the form of a "client secret") to get that user information.
- Once you have those credentials, Keycloak will use them to interact with the Identity Provider and complete the login transactions.
- There are presets for popular Identity Providers (Facebook, Twitter, etc), but you can configure anything that uses the OpenID Connect or SAML v2.0 specifications.

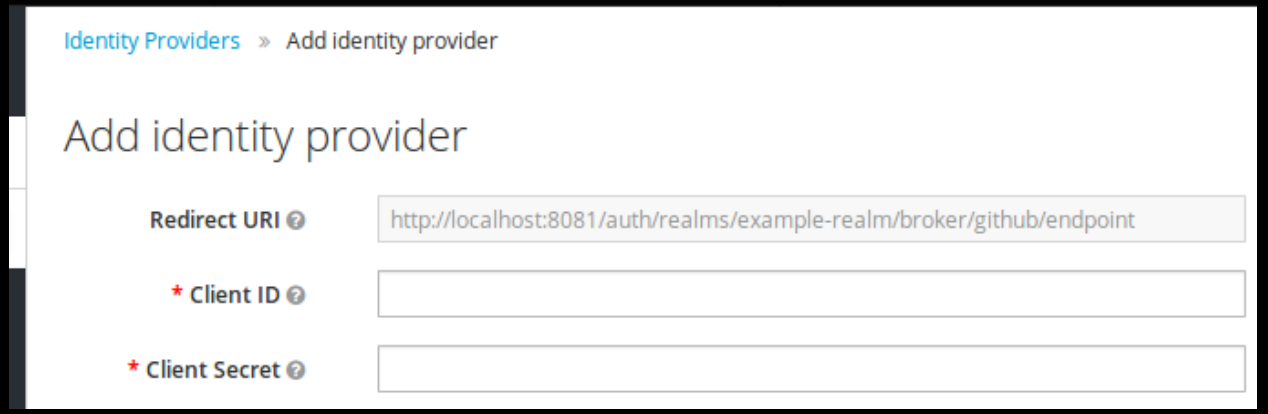
GitHub Example: Keycloak Setup

- Since the GitHub API will require our Keycloak address, let's set that up first.
  - If you followed the instructions earlier, you should already have a Keycloak instance up!
  - If not, just enjoy the show
- Click on Identity Providers
- Select **GitHub** from the list



GitHub Example: Keycloak Setup

You should now see a screen for configuring a GitHub Identity Provider



For this example, we're only interested in these 3 topmost fields

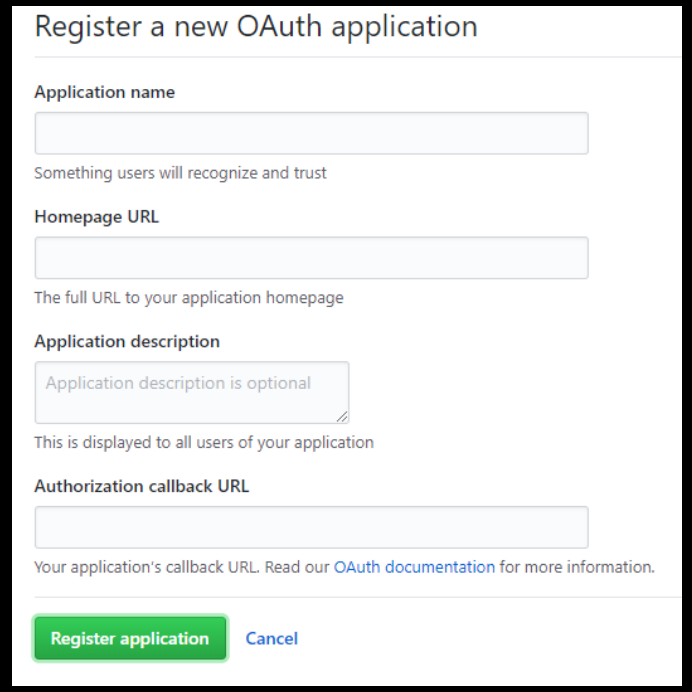
- Client ID and Client Secret we will get from GitHub
- We use the Redirect URI to configure the GitHub client itself

And with that, our Keycloak configuration is finished. We'll set up the GitHub client now.

GitHub Example: Client Setup

- Before we start, what is actually going on with this process?
    - First, we (someone GitHub knows and trusts) will need to register our Keycloak service as an OAuth app. In addition to letting GitHub know that we will be taking user information from GitHub's API using this app, we will also tell GitHub where to redirect the user once the login transaction is complete.
    - In our case, this will be our Keycloak server's location. It can either be the DNS name or an IP address. Since the redirect happens on the browser, **local IPs are valid for testing purposes**.
        - In the real world, you will need your Keycloak instance's URL to complete this stage.
- Browse to: **https://github.com/settings/developers** and click **New OAuth App**

GitHub Example: Client Setup

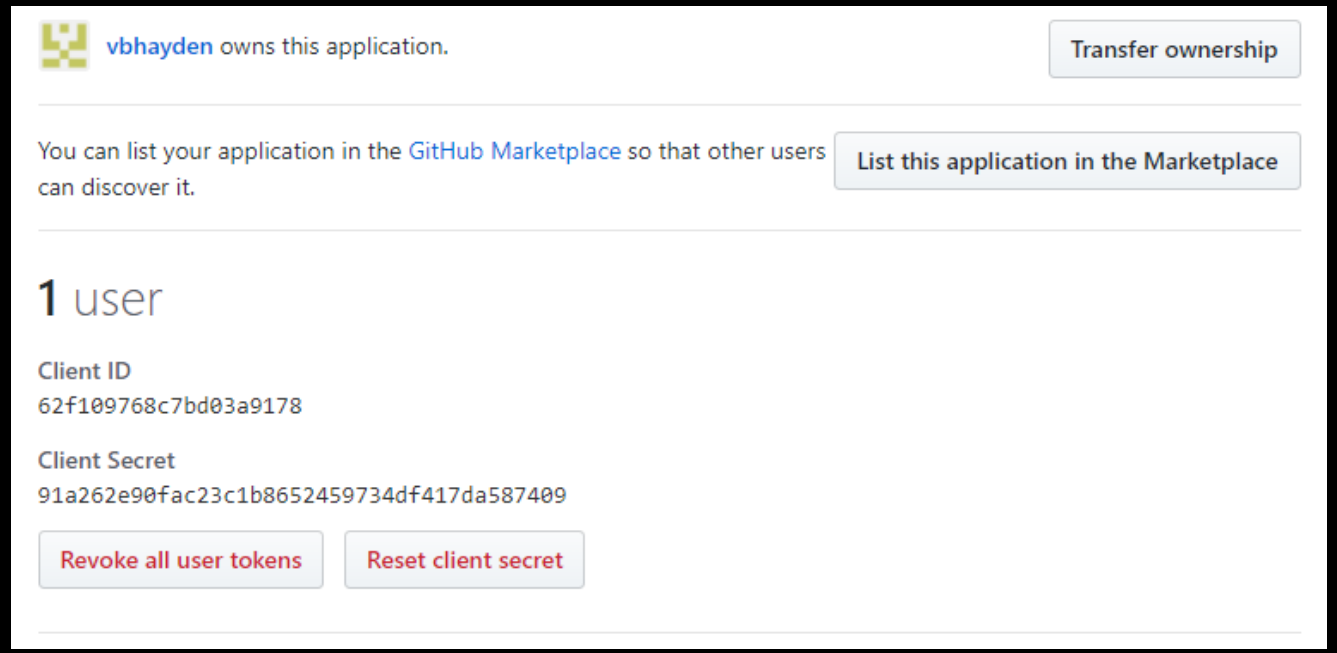


- Application name (Required; String)
- Homepage (Required; URL)
- Description (Optional; String)
- Callback URL (Required, Important)
  - This is how GitHub will know to send the user back to Keycloak once they've logged in on GitHub

Keep in mind that this redirect happens on the browser. We will use localhost in this example, but it can be any publicly (or privately for testing purposes) accessible URL.

GitHub Example: Client Setup

Once that application is registered, GitHub will assign it an ID and a Secret



And with that, the GitHub portion of our integration is done, back to Keycloak …

GitHub Example: Client Setup

We now have the Client ID and Secret values to use for our Identity Provider:

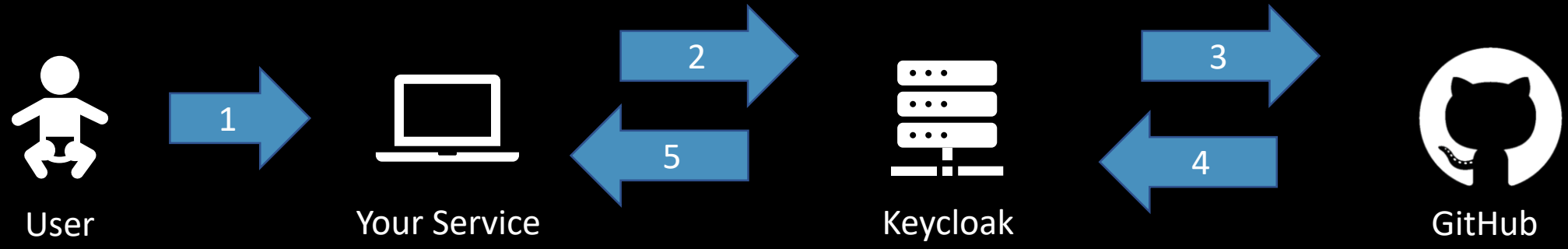
Save the settings and you're done.

GitHub Example: Confirmation

If everything went according to plan, users can now

- 1) Browse to your Keycloak-protected service
- 2) Be redirected to your Keycloak login page
- 3) Choose to sign-in with their GitHub account
- 4) Be redirected back to the Keycloak service
- 5) Be redirected to your service

If you were following along, navigate to **localhost:3000** in your browser to check it out.



Questions/Comments

Trey Hayden
Software Engineer
trey.hayden.ctr@adlnet.gov

Thank You!