Isaac Doesn't Bless Jacob

June 20, 2016

1 The Problem

We need to authenticate users across various systems, both internal and external. Isaac is a service that validates JWT tokens based on our user database, while still avoiding sessions.

The use cases this service is designed to address are.

1. Validate token signatures.
2. Invalidate tokens for non existent or banned users.
3. Invalidate tokens from valid users that were compromised.

2 Use Cases

2.1 Validate Token Signatures

The primary purpose of this service is to identify good tokens from bad. The first line of defense against forged tokens is validating JWT signature.

A JWT token is composed of three parts, the header, payload, and signature. We are using RS256 signing algorithm which uses asymmetric RSA keys. The JWT is signed with the private key and validated with the public key.

Since we are only supporting RS256, the implementation is straight forward and simpler. Simpler is better when it comes to security. Restricting to these kinds of keys will prevent attacks such as forging a JWT token using HMAC and using the public key as the secret.

2.2 Invalidate Tokens for Non Existent or Banned Users

JWT is designed to be stateless. Meaning that the server does not store any session information and the client has the sole responsibility of remembering the token. However, the only feature of JWT provided for invalidating tokens is an expiration date.

We want to support invalidating tokens based on banning and removing users so we can protect our client's data should we have to.

This is why the service will check the user database to validate whether the user is active and able to have access to the system.

2.3 Invalidate Tokens from Valid Users That Were Compromised

Another challenge is how to invalidate JWT tokens from a valid user but their token's were compromised some way. Each user will have something called a "ratchet" which is an integer that can increment to invalidate old tokens.

The service will check and make sure the JWT's ratchet number matches that of the user in the database.

3 Design

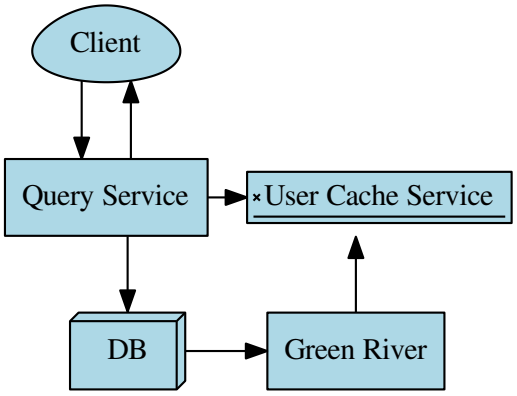
3.1 Overview

The two biggest attributes the service must have is safety and performance.

Safety because all requests to private routes need to be authenticated. We want to maintain JWT's sessionless design while still retaining the ability to invalidate tokens.

Performance because all requests to internal systems will require validation. The service will have to handle a high load.

To maintain performance, we will utilize a cache to limit DB access and to maintain safety, we will use "green river" to prevent a stale cache.



3.2 Safety and Performance

Safety and performance are forces that are constantly pulling on each other. The main feature for performance in this services is a cache so that the service can minimize DB access. The problem with maintaining a cache is obviously that the data goes scale.

For security, we need to be able to invalidate the cache for a user/token set as soon as the DB data changes.

Fortunately one foundational feature of the FoxCommerce system is the event sourcing model. All changes to the DB are handles by "green river" workers. We can then utilize this system to invalidate the cache in the auth service as soon as data in the DB changes.

This means we can maintain invalidate JWT tokens with little latency to maintain safety and prevent a stale cache.