

CSCI 370 Final Report

[team name]

[author 1 name]  
[author 2 name]  
[author 3 name]  
[author 4 name]  
[author 5 name]

Revised May 23, 2023

[optional graphic or logo]

CSCI 370 [semester] [year]

Mr./Ms./Dr./Prof. [advisor name]

Table 1: Revision history

|  |  |  |
| --- | --- | --- |
| Revision | Date | Comments |
| New |  |  |
| Rev – 2 |  |  |
| Rev – 3 |  |  |

Table of Contents

[I. Introduction 3](#_Toc135746278)

[II. Functional Requirements 3](#_Toc135746279)

[III. Non-Functional Requirements 3](#_Toc135746280)

[IV. Risks 3](#_Toc135746281)

[V. Definition of Done 3](#_Toc135746282)

[VI. System Architecture 3](#_Toc135746283)

[VII. Software Test and Quality 3](#_Toc135746284)

[VIII. Project Ethical Considerations 3](#_Toc135746285)

[IX. Project Completion Status 3](#_Toc135746286)

[X. Future Work 3](#_Toc135746287)

[XI. Lessons Learned 3](#_Toc135746288)

[XII. Acknowledgments 3](#_Toc135746289)

[XIII. Team Profile 3](#_Toc135746290)

[References 3](#_Toc135746291)

[Appendix A – Key Terms 3](#_Toc135746292)

# I. Introduction

# II. Functional Requirements

# III. Non-Functional Requirements

# IV. Risks

# V. Definition of Done

# VI. System Architecture

There are two main components to our system architecture as discussed before. These are the Service Provider (SP) and the Identity Provider (IdP). These systems are technically independent from each other and are only coupled by the Single Sign On (SSO) protocol we use. The system we have chosen for this is OIDC which is an extension of OAuth 2.0, providing authentication on top of the base protocol. This is not a new design and is commonplace in the current internet ecosystem. Take for example, the Mines suite of services. Before you can access the protected views in Canvas, for example class views, grades, etc., you must first authenticate via the Mines MultiPass system. Here, Canvas is the Service Provider, the Mines login page is the Identity provider, and they use SAML, which is very similar to OIDC, to communicate with each other.

Usually, one would use a well-known social account as an IdP, for example, Facebook, Google, Spotify, etc. However, we want to have fine-grained control over how our IdP behaves to best demonstrate the vulnerability and mitigate it. To properly implement this system, it is vital that we understand the OIDC flow, particularly, we will use the *Authorization Code Flow.* Below is a diagram of the process:

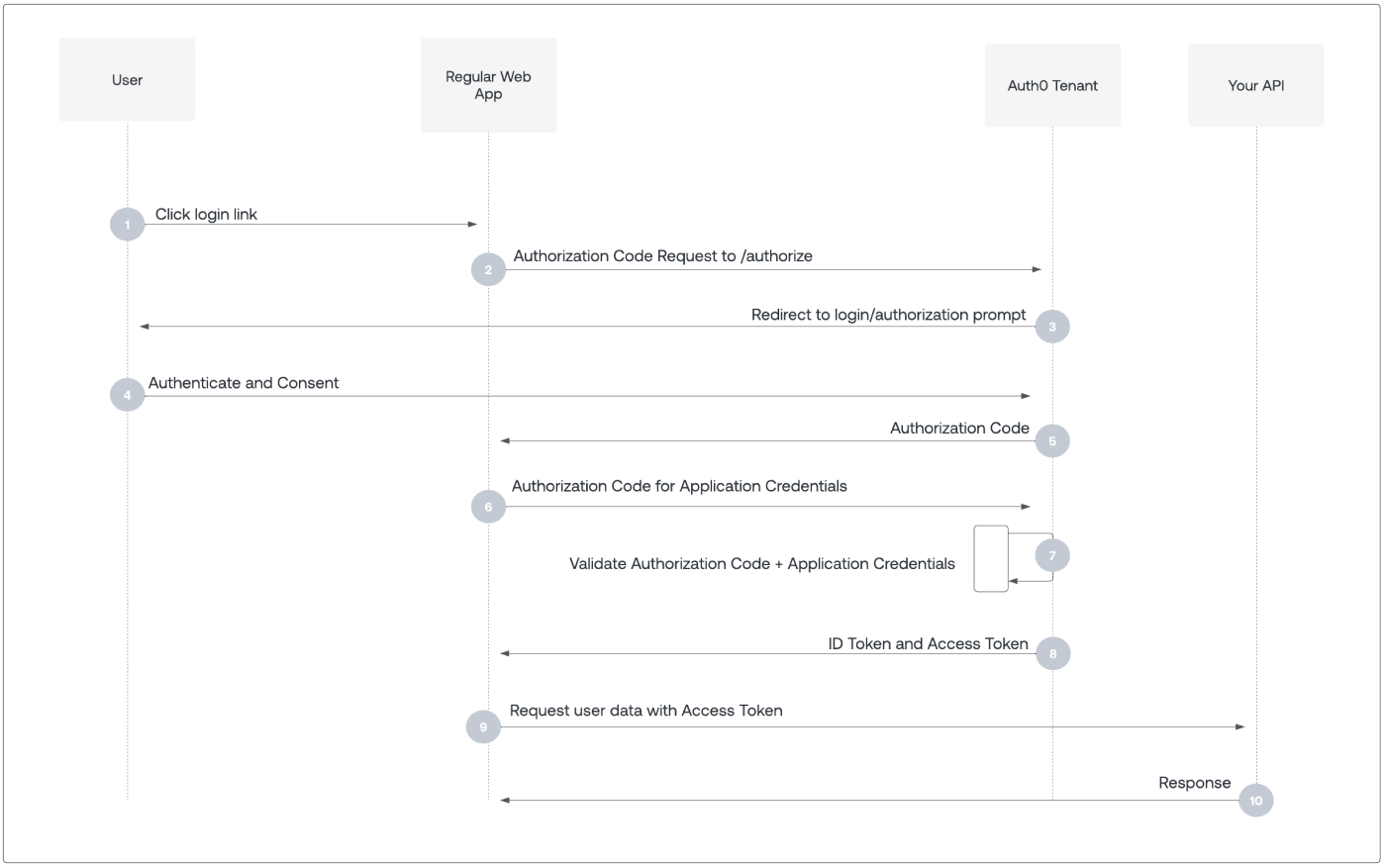


Figure 1 OIDC Authorization Code Flow. Source: [Auth0 by Okta](https://auth0.com/docs/get-started/authentication-and-authorization-flow/authorization-code-flow)

Here, the user begins by trying to log in to the web app which will end up redirecting the user to the IdP for authentication. After authorizing and allowing the web app to use your IdP account, the OAuth process begins. The web app asks for an authorization code which can then be exchanged for an access token to authenticate to the API or service you want to access.

The purpose of our service provider is only to act as the web application in the OIDC process, so it needs to provide the following:

* A login page with a link to sign in via our identity provider.
* A registration process via our identity provider.
* Protected views that cannot be accessed without credentials.
* Maintain a userpool database with client credentials for OIDC.

The last part is critical for mitigating the vulnerability since that is the way we will identify users based on something more than only their email address. The other part of our system, the identity provider, requires the following functionality:

* A login and registration page that can process URL parameters.
* Providing access to OAuth endpoints:
  + */authorize*
  + */token*
  + */revoke*
* Adherence to OIDC requirements.

By tailoring our two providers to these criteria, we can fully demonstrate the vulnerability and then mitigate it. The backends of both of our systems will be handled by Django. It is a framework that fully supports OIDC through libraries, is easily adaptable, and has a fantastic ORM. This brings us to how we store our data. Since the key functionality only requires user data, we will use a traditional relational database system, specifically Postgres. To make our service provider more dynamic, we will use a react webpage to interact with our Django backend. We could do the same for our IdP, but Django templates provide most of the functionality we require and would simplify and expedite our development process. This system can be summarized by the following diagram:

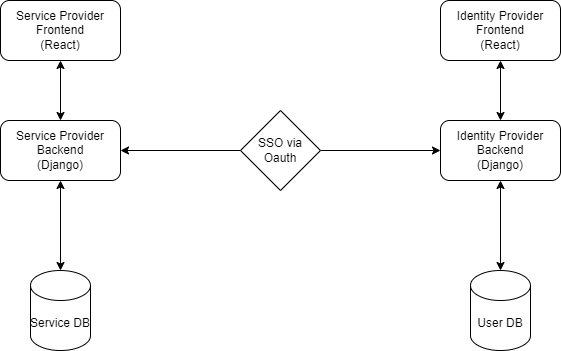


Figure 2 The Architecture of Our System

The arrows here represent direct lines of communication. For example, our frontend has no need to know of our database and will only interact with it through API calls. A key requirement from our client that has not been mentioned so far is that we want to host everything locally, and ideally through containers.

Each component of the service will have its own dedicated container. For example, our SP Postgres database and our IdP Postgres database are separate. For one, this makes sharing extremely simple and spinning up both services is only a matter of composing the containers through a Docker Compose file. Second, it makes publishing simple in case that is the direction the client wants to move forward with the project.

Finally, let’s examine a state diagram of how the vulnerability functions:

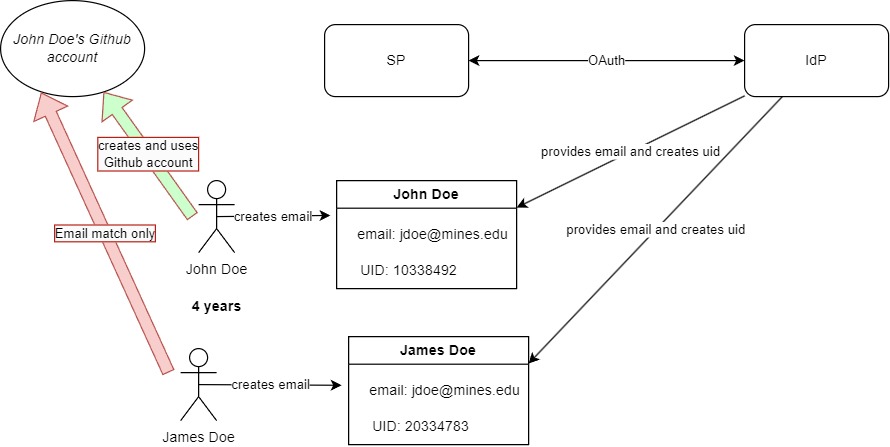


Figure 3 State Diagram Demonstrating Vulnerability

First, a John Doe starts school at the Colorado School of Mines. In this hypothetical, Mines assigns their email address in the format {Initial of First Name}{Last Name}@mines.edu. This might seem strange, but the reality is that many institutions use this scheme or a similar one [1]. In his time at Mines, he creates many accounts, but one of them is a Github account. In this hypothetical, Github uses email addresses as the primary key for a user. Not using the unique ID is an unsafe choice, but many service providers do exactly that [1]. After four strenuous years of University, John Doe graduates. Along comes James Doe and gets assigned the same email as John Doe. For one of his classes, he tries to login to Github and is redirected to the Mines IdP. He signs in and since Github only considers the email address, he is granted access to John’s old account.

# VII. Software Test and Quality

## General Considerations

When we analyzed our code for testing, we found that a traditional unit testing approach would not be effective. One reason for this is most of our code is driven by Django and its Object-relational mapping (ORM) system. However, this is not to say that we don’t unit test; it’s just not our main form of testing. This complements the primary method of testing we use, which is end-to-end testing. The most important part of our product is that the user can easily see how the vulnerability works and how it can be mitigated. The key to our solution is that our systems effectively operate together, and integration testing is therefore more important than unit testing. Ideally, we would achieve this by mocking both our Identity Provider and Service Provider and sending HTTP requests between them, but this is not feasible in our timeframe. Our primary form of integration testing will thus be manual testing.

## Manual Test – Vulnerability

The purpose of this test is to demonstrate that the vulnerability works from the user side. The steps taken to perform the test are the following:

1. Register two users with the identity provider with the same username.
2. Register both users with the service provider using the identity provider.
3. Log in with both users and demonstrate that they access the same account, demonstrating that the service provider does not check for anything but username.

To perform this test, we need our product to have a working prototype. Each time we iterate on our product we can then go back to these steps and perform them to ensure that our core functionality still works. Some edge cases that we need to consider are the following:

* Changing the order of registration between the two users
* Changing the order of logging in between the two users
* Logging in using refresh tokens instead of access tokens

## Manual Test – Mitigation

The purpose of this test is to demonstrate that the vulnerability is mitigated when we are careful with how we authorize users in the service provider. The steps taken to perform the test are the following:

1. Register two users with the identity provider with the same username.
2. Register both users with the service provider using the identity provider.
3. Log in with both users and ensure that neither user can access the others account.

Similarly to the previous outlined test, we need a working prototype. More edge cases that need to be considered are the following:

* Ensuring that a user cannot change their principal id
* The user cannot ascertain that another user has the same username as himself in the identity provider

## Key Unit Tests

Some key functionality needs to be more thoroughly tested than the rest. We will achieve this through unit testing. The following processes and parts of our application that need more testing are the following:

* User registration functionality in identity provider
* User registration functionality in service provider
* OAuth requests and responses in service provider
* Restricting access to pages and endpoints in service provider based on authentication status
* Sign out and revoke token functionality

# VIII. Project Ethical Considerations

## Implications of Publishing

A classic quandary that presents itself when working on academical cybersecurity papers is the following: “Who does our research affect and how might it damage them?” While we are not working on an academic paper, the nature of our project remains the same. We are creating a learning environment for a vulnerability that potentially affects users of large service providers and identity providers. If it were not for Professor Liu’s paper and work, then we would have an ethical responsibility of informing vulnerable companies. In his paper, he outlines what SP´s are vulnerable and notified them. Furthermore, since our product is contained locally and does not depend on outside services and is conducted in a controlled environment, we have no further obligation to notify affected parties.

## Key Ethical Principles

This does not mean that we have no further ethical responsibility. The team outlined three principles from the ACM code of ethics that we thought were the most important to our project:

* 1.3 Be honest and trustworthy.
* 2.5 Give comprehensive and thorough evaluations of computer systems and their impacts, including analysis of possible risks.
* 2.7 Foster public awareness and understanding of computing, related technologies, and their consequences.

### Principle 1.3

As we expose vulnerabilities in a widely used authentication protocol, maintaining honesty and trustworthiness is essential to the integrity of our project. Our system simulates potential security flaws in the Authorization Code Flow of OIDC, and by openly acknowledging the limits and risks of the vulnerability, we create a transparent learning environment. Being trustworthy in our demonstration means ensuring that our simulations are contained and safe, avoiding any real-world exploitation of sensitive data.

Furthermore, honesty in the presentation of the vulnerability is crucial. Misrepresenting the scope or impact of the flaw could mislead learners, leading to improper security assumptions. For example, in our system’s architecture, which includes a Service Provider (SP) and an Identity Provider (IdP) communicating via OIDC, we must clearly explain how the vulnerability manifests and the exact circumstances under which it can be exploited. Trustworthiness also requires that we provide accurate methods for mitigating this vulnerability, reinforcing best practices in secure authentication design.

### Principle 2.5

The goal of our project is to demonstrate vulnerabilities in OIDC and how these can be mitigated. This principle drives the need for a thorough evaluation of the system’s security. By walking through the entire OIDC Authorization Code Flow, we analyze where and how vulnerabilities can occur, such as the interception of authorization codes or token leakage.

A thorough evaluation includes not only identifying the vulnerability but also contextualizing its results. The consequences of the identity-account inconsistency threat are severe. For example, if a vulnerability in our demo system were exploited in a real-world application, it could lead to unauthorized access to sensitive resources, such as the user's profile or data stored within the Service Provider.

### Principle 2.7

The heart of our project is to encourage people to understand the identity-account inconsistency threat and how it could affect them. This principle is thus the most important one to the team. Furthermore, by making the experience interactive to the user, we promote a higher mode of learning. The goal is for a layman to be able to understand the vulnerability without any prior knowledge of OAuth or OIDC. This principle is the one we will pursue the most.

# IX. Project Completion Status

# X. Future Work

# XI. Lessons Learned

# XII. Acknowledgments

# XIII. Team Profile

# References

[1] G. Liu, X. Gao, and H. Wang, “An investigation of identity-account inconsistency in single sign-on,” in *Proceedings of the Web Conference 2021*, 2021, pp. 105–117.

# Appendix A – Key Terms

Include descriptions of technical terms, abbreviations and acronyms

|  |  |
| --- | --- |
| **Term** | **Definition** |
| *[Insert Term]* | *[Provide definition of the term used in this document.]* |
| *[Insert Term]* | *[Provide definition of the term used in this document.]* |
| *[Insert Term]* | *[Provide definition of the term used in this document.]* |