CSCI 370 Final Report

Safer SSO

Hilmir Arnarsson
Landon Dixon
Avery Overberg
Carter Strate

Revised November 20, 2024

CSCI 370 Fall 2024

Mr. Jensen and Dr. Liu

Table of Contents

I. Abstract

Single sign-on (SSO) is a popular authentication method that allows users to access multiple applications with one set of credentials. However, SSO systems are vulnerable to identity-account inconsistency threats, which allow attackers to reuse email usernames to gain unauthorized access to victim accounts. This paper aims to demonstrate and mitigate identity-account inconsistency threats in SSO systems. The demonstration is presented in a controlled environment, with a custom-built service provider (SP) and identity provider (IdP) to illustrate the vulnerability. The paper also explores potential mitigation strategies to address this threat.

II. Introduction

Client and Stakeholders

The client for this project is Professor Liu, who requires this demonstration for educational purposes. Specifically, he wants to highlight a critical security vulnerability that can occur in SSO systems. Guannan Liu is a new computer science professor at Colorado School of Mines. He has a Ph.D. in Computer Engineering from Virginia Tech and his research focuses on System and Network Security, Human-Factor Security, and User Authentication. He has done research on the vulnerability that this project will focus on. Stakeholders other than Dr. Liu would include Dr. Liu's students who will take advantage of this educational tool, and organizations that could be vulnerable to the attack. The project will help the stakeholders understand how attackers can exploit token-based authentication methods and explore potential defenses against such attacks.

Terminology

- SSO (Single Sign-On) is an authentication process allowing users to access multiple applications with a single login.

- A Service Provider (SP) is an application, service or website on the internet that requires users to log in to store their data. For this project it must be able to utilize SSO. Some common examples are social media websites or GitHub.
- An Identity Provider (IdP) is the third party responsible for creating, maintaining, and managing user identity information. A common example is Google.
- OAuth (open authorization) is an open standard for websites to share user information without sharing passwords. This is how SSO will be implemented for Safer SSO.

Context of the Project

Single Sign-On (SSO) is a widely used and accepted method of online authentication that allows users to access multiple service providers (SPs) with a single set of login credentials for a trusted identity provider (IdP). This project, Safer SSO, aims to demonstrate a critical vulnerability in SSO, the identity-account inconsistency threat. In this threat an attacker can reuse an email username while signing into a SP and gain access to the victim's data on that SP. In other words, if an attacker can reuse an email username from the same IdP as the victim then some SPs will serve the victims data to the attacker. In his research paper on the subject Dr. Liu found that 80% of SPs were vulnerable in some way. The identity-account inconsistency threat will be demonstrated by simulating an attack on a SP created for the purpose utilizing an IdP also created for the purpose. Furthermore, Safer SSO will propose and implement a mitigation of the threat on the SP side. The mitigation is implemented on the SP side as mitigating from the IdP side is trivial. To mitigate from the IdP side the IdP simply must not allow email usernames to be reused.

Software and Hardware Details

There are no previous software revisions applicable to this project. The platform will be developed from scratch using standard containers, ensuring that the project is self-contained and easily replicable in a controlled environment. Docker will be used for containerization, with a Django application serving as the SP and a Salmon-based email server functioning as the IdP also built on Django. The application will run on localhost, and PostgreSQL will serve as the backend database.

III. Requirements

The goal of Safer SSO is to demonstrate a vulnerability associated with using SSO, known as the identity-account inconsistency threat. Since there is no existing codebase, all the code is developed from scratch. To demonstrate this vulnerability, a custom-built SP and IdP are required. Both the SP and the IdP implement SSO, meaning a complete SSO system is developed as part of the project.

Functional Requirements

- The SP includes SSO capabilities
- The SP stores user data that can be exfiltrated by the vulnerability to demonstrate the vulnerability
- The IdP provides tokens necessary for SSO.
- The SP and IdP can communicate effectively to achieve SSO.
- The SP includes a mitigation mechanism to address the identity-account inconsistency threat.
- Both the SP and IdP are locally hosted and containerized to ensure the vulnerability demonstration does not interact with real-world data.

Non-Functional Requirements

- The SP and IdP run on any environment with minimal setup, requiring only the execution of containers.
- The system allows for easy switching between demonstrating the vulnerabilty and the mitigation.
- The databases are accessible and editable to streamline the demonstration process.
- Pre-configured accounts are available to simplify and expedite the vulnerability demonstration.

IV. Risks

With any project there are risks involved. This is especially true of software and even more so of cybersecurity focused projects. It is thus essential that we measure the risks and outline a clear plan to mitigate the risks. Below are the most impactful risks we have analyzed.

Risk	**Likelihood**	**Impact**	**Risk Mitigation Plan**
SP vulnerability exploited beyond the original scope of the project	Likely	Major	Use secure coding practices and perform regular security scans, only implement the attack on the SP designed for the demonstration.
Educating potential malicious actors	Likely	Major	Only deliver codebase to client and keep it private so only students in cybersecurity can utilize the product. Accept the risk to some extent, for vulnerabilities to be patched, people must know about them.
Authentication through OAuth not implemented correctly	Unlikely	Moderate	Do continued testing and quality assurance throughout the creation of the product to ensure correct implementation.
The website design makes it difficult to clearly understand what is happening.	Likely	Minor	Make the implementation clean and clear, provide instructions on how to use the product.

V. Definition of Done

Our definition of done can be thought of in stages. Below is a table with our definitions. We have each facet of our system that we intend to deliver, a stretch factor that ranges from one to five and measures how big of a stretch goal it is (one being very achievable, five being highly unlikely we will finish), an outline of how to test the product, and finally how we will deliver the product.

Goal	**Stretch Factor**	**Testing Outline**	**Method of Delivery**
Identity-account inconsistency threat demonstrated.	2	Test whether logging in to another user's account as a malicious actor is successful.	Docker images and GitHub.
Identity-account inconsistency threat is mitigated.	3	Test whether logging into another user's account as a malicious actor is unsuccessful	Docker images and GitHub
Provide an interactive flow that educates users on exactly how the vulnerability works	5	Provide access to the product in an educational environment, scoring participants on a quiz to test	Provide access to a protected – for

(like OAuth Playground perhaps.)		the effectiveness of the educational tool.	now – hosted website.

VI. System Architecture

There are two main components to our system architecture as previously discussed. 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. Recall that OIDC is an extension of OAuth 2.0, providing an extra layer of verification and authentication. To better illustrate OIDC, consider this analogy. If you are at a concert festival, you do not need to show your ticket each time you go to see a different concert. Usually, you get a wristband that grants you access to each concert, and you receive this wristband upon initial verification. In this analogy, the different concerts are Service Providers, the person who grants you your wristband is an Identity Provider, and your wristband is your OIDC authorization code.

Usually, one would use a well-known social account as an IdP, for example, Facebook, Google, Spotify, et cetera. 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:

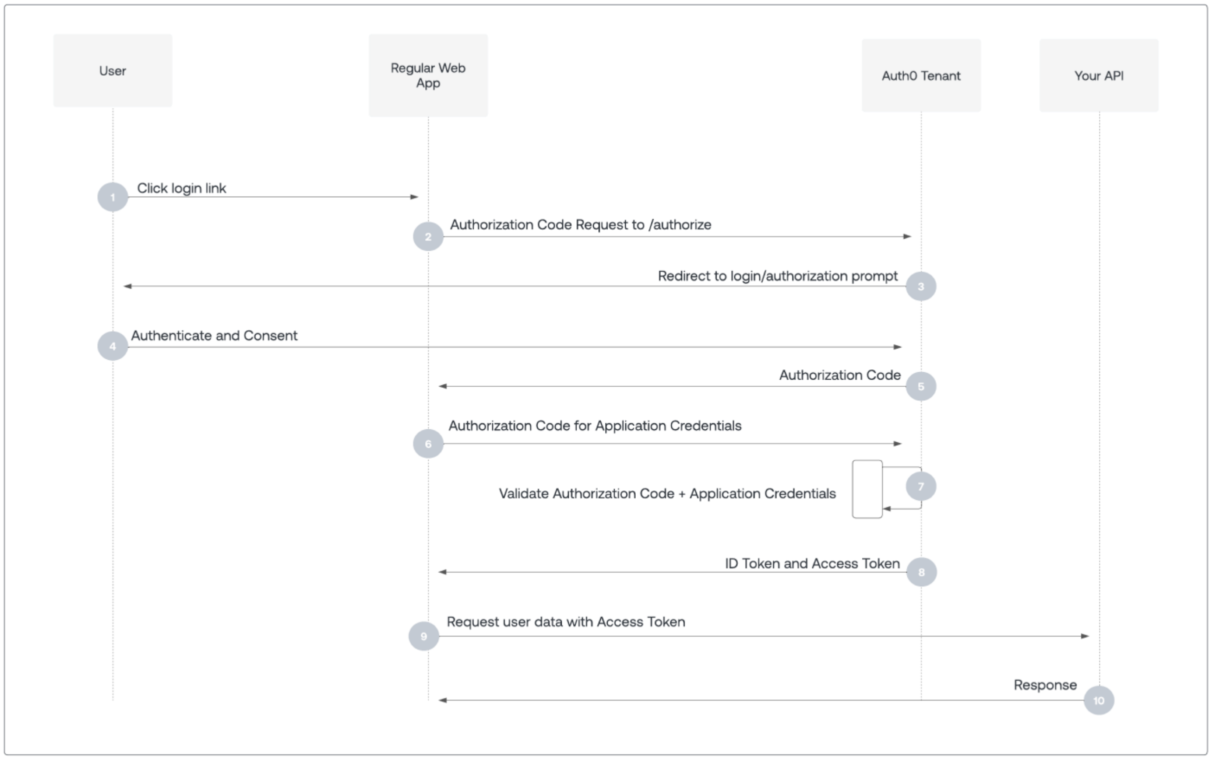

User
Regular Web App
Auth0 Tenant
Your API
1 Click login link
2 Authorization Code Request to /authorize
3 Redirect to login/authorization prompt
4 Authenticate and Consent
5 Authorization Code
6 Authorization Code for Application Credentials
7 Validate Authorization Code + Application Credentials
8 ID Token and Access Token
9 Request user data with Access Token
10 Response

Figure 1 OIDC Authorization Code Flow. Source: Auth0 by Okta

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 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. Django 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:

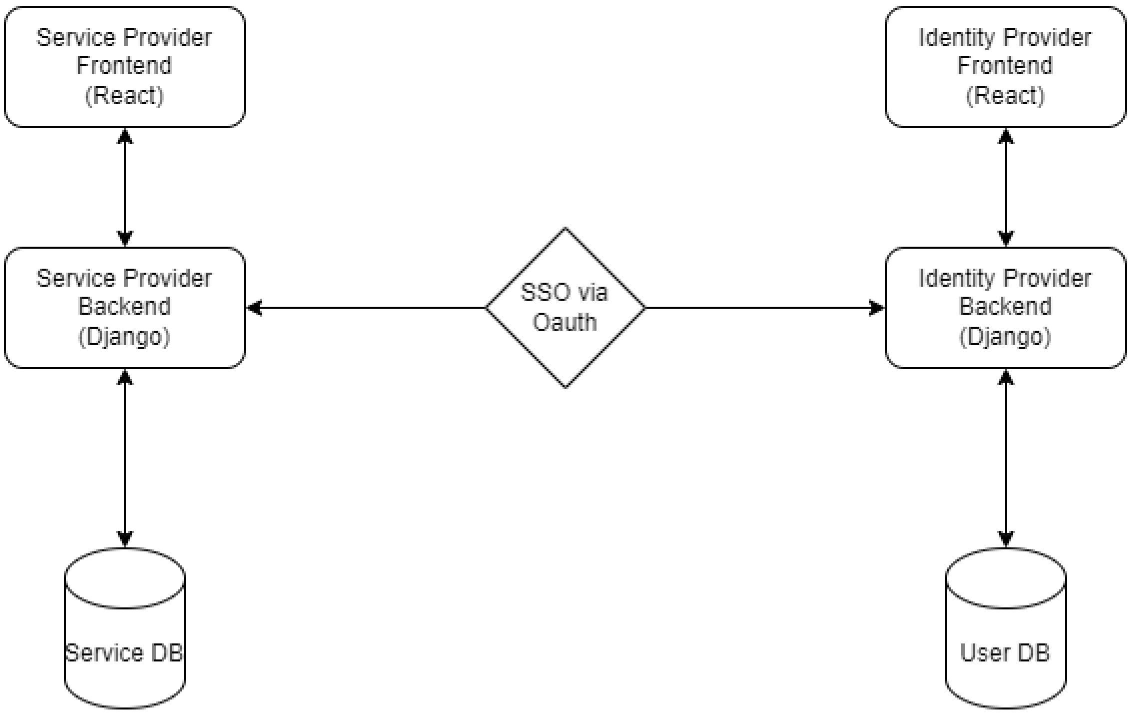

Service Provider Frontend (React)
Identity Provider Frontend (React)
Service Provider Backend (Django)
SSO via Oauth
Identity Provider Backend (Django)
Service DB
User DB

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 cases 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. First, 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. Many service providers make the unsafe choice of not using the unique ID [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, James attempts to log into GitHub and is redirected to the Mines Identity Provider (IdP) for authentication. He signs in and since GitHub only considers the email address, he is granted access to John’s old account.

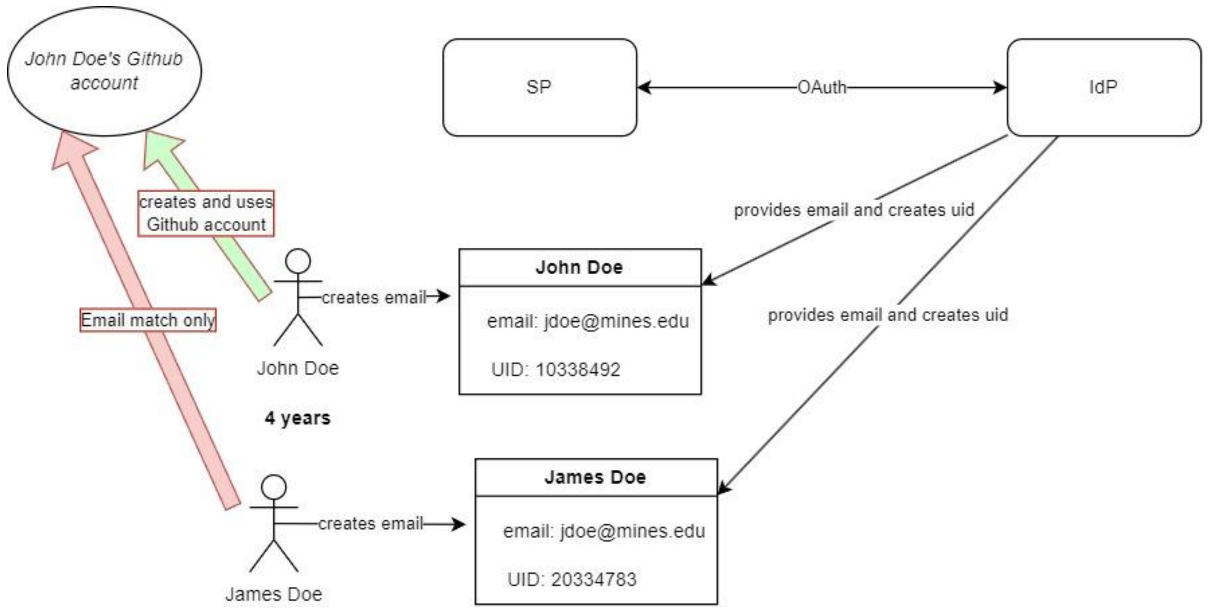
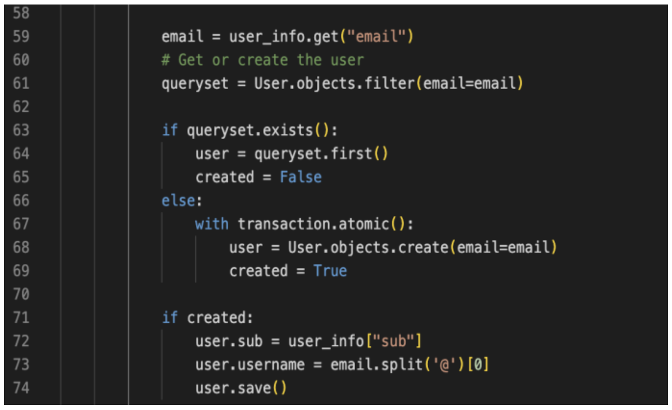


Figure 3 State Diagram Demonstrating Vulnerability

VII. Technical Design and Details

To better clarify the responsibility of each system in the OIDC flow, let us examine what JWT tokens are being exchanged and where. A JWT token is just a form of a token that can be used for authentication and authorization. It is essentially a JSON object with some sensitive fields. For our purposes, let us assume that it grants access and verifies identity. Initially, the frontend redirects the user to login and passes the client ID of the backend along with a code verifier and challenge that will be used later when verifying the response in the backend. After successfully logging in, an authorization code will be delivered to the backend of the SP which the SP uses to gain information about the user. The code is exchanged for a JWT token from the IdP which when decrypted using public-key cryptography simultaneously verifies the identity of the user and provides information about them. Upon verification, the SP generates a new JWT token that will be passed to the frontend. The frontend saves this on the client-side of the application and can then authenticate using that token to make requests to the backend’s API.

In order to understand where the identity-account inconsistency threat stems from, we must look at how the backend looks up the user in its own database upon verification of the token. In the vulnerable version, the backend uses the email in the information provided by the IdP as the primary key. However, the IdP also provides a “sub” field, which is guaranteed to be unique among the IdP. Using this as our primary key secures the system. This can also be configured on the IdP to be a universally unique identifier which is safer since then the vulnerability cannot rise even if the SP provides multiple options to sign in. Below is the key line of code that creates the vulnerability and then the mitigated version:

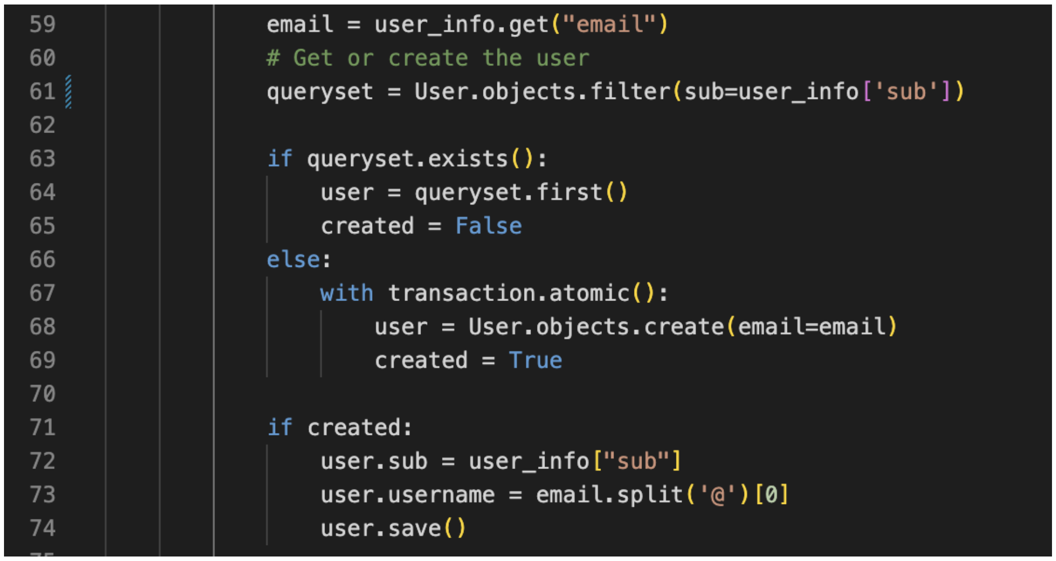


```python
email = user_info.get("email")
# Get or create the user
queryset = User.objects.filter(email=email)

if queryset.exists():
    user = queryset.first()
    created = False
else:
    with transaction.atomic():
        user = User.objects.create(email=email)
        created = True

if created:
    user.sub = user_info["sub"]
    user.username = email.split('@')[0]
    user.save()
```

Figure 4 Vulnerable piece of code



```python
email = user_info.get("email")
# Get or create the user
queryset = User.objects.filter(sub=user_info['sub'])

if queryset.exists():
    user = queryset.first()
    created = False
else:
    with transaction.atomic():
        user = User.objects.create(email=email)
        created = True

if created:
    user.sub = user_info["sub"]
    user.username = email.split('@')[0]
    user.save()
```

Figure 5 Mitigated line of code

This single line of code is the crux of the problem and makes 80% of service providers vulnerable.

VIII. 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

IX. Ethical Considerations

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 [2}:

- 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.

X. Results and Future Work

The goal of the project was to implement a demonstration in the browser—hosted locally—for the identity-account inconsistency threat. The team successfully implemented a demonstration of the vulnerability and created both a service provider and an identity provider. These services were developed locally and fully containerized. Additionally, we managed to mitigate the vulnerability, meeting the client's core functional requirements.

To validate our implementation, we conducted extensive testing. Through manual tests, we confirmed that the vulnerability functioned as expected, allowing two users with the same username in the identity provider to access the same account in the service provider. We also conducted edge case tests, such as altering registration and login orders and using refresh tokens instead of access tokens, ensuring the vulnerability held under various scenarios.

Following the implementation of the mitigation, additional testing demonstrated that the solution successfully addressed the issue. Users were no longer able to access each other's accounts, even under edge case conditions, such as attempts to modify a user's principal ID or detect shared usernames in the identity provider. These results validate the reliability and effectiveness of our prototype for both demonstrating and mitigating the vulnerability.

Some stretch goals we aimed to implement but were unable to complete due to time constraints remain excellent candidates for future work. These include:

- Prettifying the interface to make the demonstration more visually appealing.
- Providing an interactive interface that guides users step-by-step through the process to enhance learning.
- Simplifying installation to make it easier for future users to set up the project.
- Making it simpler to switch between the vulnerable and mitigated versions of the system for demonstration purposes.

The team believes that the current codebase, paired with this document, provides a strong foundation for implementing these enhancements in the future. By focusing on these stretch goals, future efforts could greatly improve the usability and accessibility of the demonstration.

XI. Lessons Learned

While the team learned many lessons, there are a few that we would like to outline that are actionable and the most useful to future teams:

- Start coding early. Initially it might seem like the class is frontloaded with work not related to coding and that there will be more time later in the semester to focus more on coding. However, midterms and other common academic challenges impeded us in coding when the schedule of the class told us to. Luckily, we had started early, which helped us immensely in our project.

- Give yourself time for setup and set reachable goals when working on a new codebase. The nature of our project meant that we started from scratch. While this was an exciting prospect and created more freedom and learning opportunities, it was also significantly harder. There was a lot of time spent on getting things set up initially along with getting the different applications to communicate effectively. Do not underestimate the challenges presented when starting new projects.
- Focus on the skills outside of coding. While the project itself was important to us and it helped us learn good practices when working on larger-scale projects, this was not the most important part of the class. From our perspective, the most helpful skills we learned were presentations and public speaking, providing good documentation, doing the design process correctly and in-order, and communicating with the client effectively. We especially want to emphasize the importance of presentations. They might seem auxiliary but are one of the most important skills to have in industry, regardless of whether you work at a multibillion corporate entity or a three-person startup.
- Break your project up early. Given the fact that most teams have four to five people working together at the same time, it is imperative that everybody has something to be working on at all times. If you consider yourself the strongest coder on the team, you might realistically output twice as much as your peers. If you can get everybody to work effectively at the same time, your team will output four to five times as much than otherwise
- Make ample use of online documentation and tutorials. Many of the things being done have been done in some form or another before, and many of them have been documented somewhere online. By making use of these resources, especially the video tutorials, many of the headaches of early development are avoided. These can also be extremely helpful in finding and fixing bugs. Leveraging the vast amount of knowledge available online is imperative when working on a project this large.

XII. Conclusion

The Safer SSO project successfully achieved its primary goal of demonstrating and mitigating the identity-account inconsistency threat in Single Sign-On systems. By implementing a custom Service Provider (SP) and Identity Provider (IdP), the project provided a robust framework for educating stakeholders on this critical security vulnerability. Through thorough testing and validation, we confirmed the efficacy of the mitigation strategies, ensuring the security of user accounts even in scenarios that could exploit vulnerabilities.

Despite the project's accomplishments, there remains room for enhancement. Future efforts could focus on refining the user interface, creating more interactive educational tools, and simplifying the installation and setup process to make the demonstration accessible to a broader audience. These improvements would not only strengthen the project’s educational impact but also support its continued use as a teaching tool in cybersecurity education.

Overall, this project has underscored the importance of rigorous authentication protocols in modern digital systems. By addressing a widespread vulnerability and providing a practical solution, Safer SSO contributes to advancing secure authentication practices and raising awareness of potential risks in identity management systems.

XIII. Acknowledgments

We would like to extend our sincere gratitude to Dr. Liu, our client, for his invaluable guidance and insight throughout this project. His expertise and support have been instrumental in helping us understand the complexities of our work and achieve our objectives. We are also deeply thankful to our advisor, Scott Jensen, for his continuous encouragement and technical expertise. His advice and mentorship have been crucial in overcoming challenges and advancing our understanding. Their combined knowledge and dedication have greatly enriched our learning experience, and we are truly appreciative of their contributions.

XIV. Team Profile

Hilmir Arnarsson

Senior in Computer Science

Team Lead

Landon Dixon

Senior in Computer Science

Research and Writing Lead

Avery Overberg

Senior in Computer Science

Advisor POC

Carter Strate

Senior in Computer Science

Client Liaison

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.

[2] Association for Computing Machinery, *ACM Code of Ethics and Professional Conduct*, 2018. [Online]. Available: https://www.acm.org/code-of-ethics

Appendix A – Key Terms

Include descriptions of technical terms, abbreviations and acronyms

Term	**Definition**
IdP	*The IdP is responsible for creating, maintaining, and managing user identity information.*
SP	*It is the application or service relying on the IdP to authenticate users*
SSO (single sign-on)	*An authentication process allowing users to access multiple applications with a single login*
OAuth (open authorization)	*An open standard for websites to share user information without sharing passwords. This is how SSO will be implemented for safer SSO.*