haute école
neuchâtel berne jura

Project specifications

PRIVACY PRESERVING OAUTH SERVICE USING TEES

3285_TB

Titus Abele

Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts
Western Switzerland

Abstract

Authentication in online services has evolved in the last decade. Today, it is possible to use so called federated accounts to sign up into a variety of online services. These federated accounts (e.g., Google, Facebook) can facilitate the creation of a new account and further logins into a service, provided that the service allows the usage of such accounts. This raises issues related to online privacy since the providers of federated accounts have access to all data concerning adhesion and connection to online services even though they do not necessarily own the services.

In the prospect of finding a way to obfuscate the data surrounding federated authentication using OAuth, we wish to study the conceptualization and implementation of a blind authentication service by running an OAuth server inside of a trusted execution environment. This would raise the level of confidentiality provided to users. This document introduces the project and outlines the objectives of the study, analysis and conceptualization.

Contents

Figures

Introduction

Using online services often requires authentication. In recent years, methods of authentication have evolved into using more standardised ways of account creation. When being prompted to login, the user often has various choices of either creating a new account or using one of his existing accounts such as a Google or Facebook account. These so-called federated accounts provide a very fast and easy way to sign up to a new service in one single click. Although fast and convenient, the usage of such services has increased the awareness surrounding privacy related issues. Given the generic nature of these accounts, if used in a variety of different applications, the underlying authentication service always stays the same no matter the nature of said application. This in turn is a source of unnecessary data that is somewhat free for the taking. This data contains login details and can be used to generate patterns of service usage, even if the provider of the generic account has nothing to do with the service being used. Knowing the usage patterns of users can be very useful for advertising purposes, even if it violates their privacy.

Mitigation of privacy infringing data collection would require the conception and implementation of a blind authentication service that does not share details with its provider just like illustrated in Figure 1. Such an authenticator would require to be completely impervious to any sort of tampering and logging. A trusted execution environment (TEE) is an adequate solution to this requirement. At the same time, the authenticator needs to respect the Open Authorization standard (OAuth) to be able to easily integrate into production level services. The project surrounding this thesis is aiming to implement this authenticator inside of a trusted execution environment using Intel SGX among other technologies.

The Open Authorization standard outlines specific implementation directives aiming to standardise API access authorizations across devices and technologies. The service is separated into four parties. First, the client, an application or any sort of interactive technology with a user base. Second, a resource owner, a user of said application whose primary goal in this example is to consult one of their resources. Third, the authorization server where the resource owner has an account. Finally, the resource server, a service capable of accepting access tokens instead of login credentials. This last service is usually intrinsically linked to the authorization service.

The sequence that an application must respect is as follows:

1. Authorization request: the client prompts the resource owner if it may or may not request their authorization at one of the authorization servers (e. g. Google or Facebook);
2. The resource owner grants or denies the authorization request;
3. Once granted, the client may send the authorization grant to the authorization server which in turn can check the resource owner’s credentials without the need for the client to witness the exchange of credentials;

4. The authorization service then generates a new access token which provides access to their API without the need for credentials (the authorization server might need to request the credentials directly from the resource owner if a session has expired or isn't present);
5. The client may now access the resource server (API linked to the authorization server) using the access token;
6. Access is granted by the resource server if the authorization server validates the access token. The resource owner may consult their resource on the client.

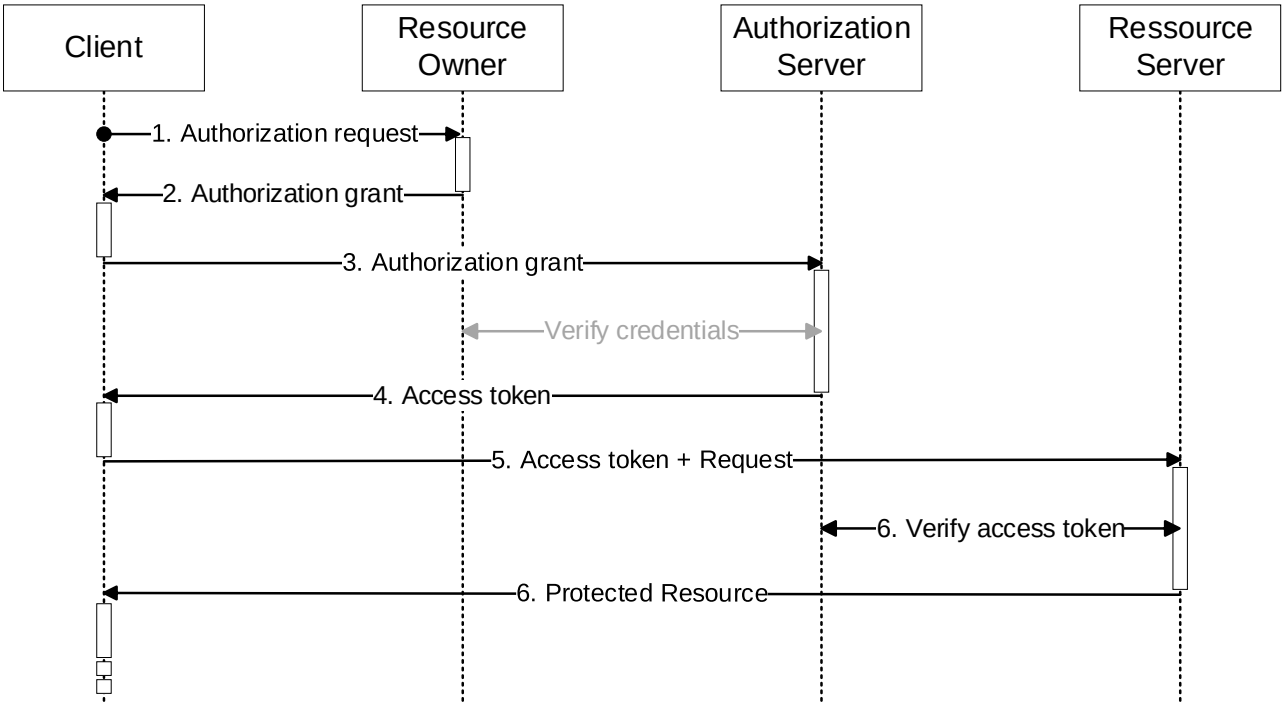

Figure 1 Sequence diagram of the OAuth standard

One important aspect is the lack of kinship between the client application and the authorization and resource servers. It is not necessary for these services to be linked, as a matter of fact, many applications use Google's authorization services to enable easy and fast access to a protected resource, even though they are not owned by Google.

Project Specifications

The main goal of this project is to implement such an OAuth service. To be more precise, the focus lies in the conception and implementation of the authorization server. The latter being able to issue and verify access tokens. Although the implementation of such a service might seem trivial, the project requires the service to be blind. This means that the chosen execution environment is an Intel SGX® enclave running an Integritee worker.

Enclaves

Enclaves represent a trusted execution environment embedded in a process. This environment is part of the application and retains full access to its memory all the while being verified and trusted by hardware at the start of execution. The operation of this environment is only visible within the CPU which makes it obfuscated to the outside. The integrity of the program being executed is thus insured.

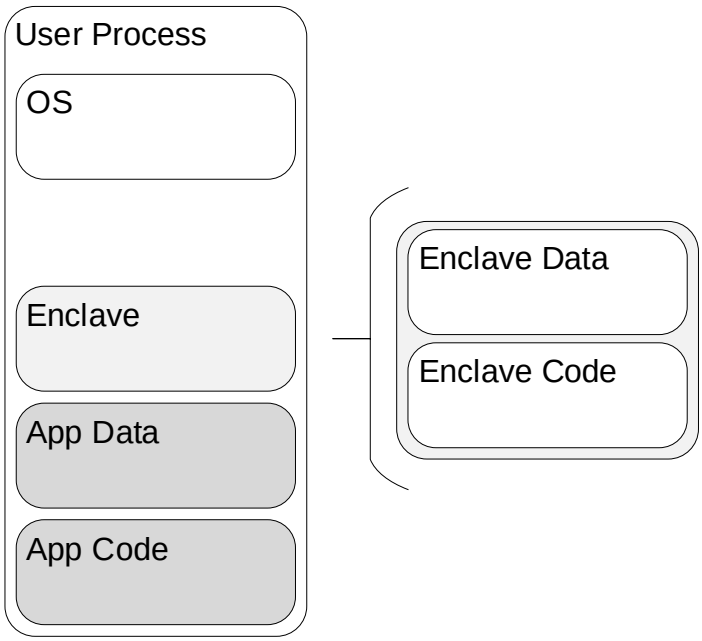

Figure 2 Schema of an enclave in a User Process context

The integrity of the contents of the enclave is ensured with a cryptographic hash value called the MRENCLAVE. This SHA-256 digest is the result of the initialization of the enclave and combines the contents of the pages (code, data, stack and heap). Once the enclave has been initialized and the MRENCLAVE value has been set, nothing inside the enclave may change without compromising the MRENCLAVE value thus breaking the trust.

Integritee

Integritee is a “scalable public blockchain solution”[1] whose aim is to facilitate the usage of trusted execution environments. It functions by using two separate entities.

1. The integritee-node is a blockchain based on the substrate framework that maintains a registry of remote attested integritee-service enclaves;
2. The integritee-worker represents an entity executed on an enclave which has been attested, verified and subscribed to the accompanying node.

This project’s the goal is to implement the OAuth service inside an integritee-worker. This is meant to ensure scalability, interoperability and confidentiality all while keeping high standards of performance and trust. Accompanying this implementation is a clear description of the technologies and a complete documentation. All these objectives are specified below.

[1] Courtesy of https://integritee.network/

Main Objectives

i. Fork the worker repository and implement OAuth logic

   i.1. This service must be capable to serve as an authorization server and provide access tokens to clients.

ii. Implementation of an OAuth service inside the enclave

   ii.1. Conception

   ii.2. Development

iii. Analysis and familiarisation

   iii.1. Documenting about the usage of IN

   iii.2. Documenting about existing services [2]

   iii.3. Familiarisation with Rust

iv. Thesis on a system answering the detailed specifications.

Secondary Objectives

i. Documenting feasibility

ii. Evaluating the tools

   ii.1. Suggest alternate solutions if need be

   ii.2. Document the procedure to follow for implementation.

To be noted

The objectives listed above are not in chronological order but rather in order of importance. It is also important to note that this list of objectives is subject to change. The secondary objectives are not dependant on the completion of the main objectives and will serve as a solidification of the knowledge required to complete the main objectives.

References

[1] SCS, Privacy Preserving OAuth Service, *Masterarbeiten,* viewed on 04.01.2023, URL https://www.scs.ch/studien-masterarbeit/privacy-preserving-oauth-service-with-tees/

[2] GitHub, scs, *ma-thesis-no-data-collection*, viewed on 04.01.2023, URL https://github.com/scs/ma-thesis-no-data-collection

[3] Wikipedia, OAuth, *Open-Authorisation Standard*, viewed on 04.01.2023, URL https://en.wikipedia.org/wiki/OAuth

[4] J. Ménétrey, M. Pasin, P. Felber and V. Schiavoni, "Twine: An Embedded Trusted Runtime for WebAssembly," *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, Chania, Greece, 2021, pp. 205-216, doi: 10.1109/ICDE51399.2021.00025.

[5] GitHub, UNINE, *unine-twine*, viewed on 27.02.2023, URL https://github.com/JamesMenetrey/unine-twine.

[6] Overview of Intel SGX® Enclaves, intel.com, viewed on 24.05.2023, URL https://www.intel.com/content/dam/develop/external/us/en/documents/overview-of-intel-sgx-enclave-637284.pdf