**CHAPTER 1**

**INTRODUCTION**

* 1. **Introduction**

Currently, Most of the communications between the client and server use REST architectural style through HTTP or simply RESTful API. The client requests resources through URL-encoded params to access the specific resource provided by the server. The request and responses are usually formatted as JSON.

Not all the requests are authorized to access the services. Owner can access only his private information but not others, which requires the owner proving that they are the owner of that resources, which we call “Authentication”. While the process of granting authenticated party a permission to do something is called “Authorization”.

JSON Web Tokens is a new standard that can be used for communicating the information of Authentication and Authorization between client and servers, where server issues one such JWT if the user is a valid owner and JWT can store stateless information about the access rights etc.

While existing standards like OAuth and protocols like Kerberos already provide with simple and usable structure for the developers to implement their own authentication pattern, it can be guaranteed every time to have a stable implementation that’s secure and reliable in all the cases for their service.

* 1. **Problem Statement**

Developing a secure authorization system for the services developed within an organization can prove to be harder and inconsistent as the number of services increases. With Each service having a requirement of using other services and the management of access control for each such service can be messy when viewed from a higher-level.

**1.3 Objectives:**

The main objectives of the proposed system are:

* To centralize the Authorization Process across multiple services
* To centralize the third-party access management
* To develop a library of functions for the preceding objectives

**CHAPTER 2**

**LITERATURE REVIEW**

**[1] "Information technology of user authentication in cross-platform systems" - V. Krylov, N. Volkova and Y. Kozina**

Authors proposed the information technology of user authentication for cross-platform systems has been developed, which is characterized by high reliability and efficiency. Additional research on the appearance of collisions while generating unique keys for user authentication have been carried out. In conditions of information technology growth, systems which are being developed to be used on multiple platforms, gain in importance and popularity. As a rule, such systems have client-server architecture and have strong security requirements for user authentication.

To develop the information technology of such applications, client-server architecture is used, that is, the network architecture in which tasks or network load is distributed between the service providers (servers) and the customers (clients). Typically, they interact through a computer network using network protocols and are located on different computers. Servers expect requests from the client programs and provide the latter with the resources in the form of data. Clients send requests for various operations containing the user object. Server in its turn works with the database and returns the required data to the clients.

The server component is the core of the entire information system. Almost all the functionality is implemented on the server side, namely creation of users, the authentication system, the system of messages and notes, work with the database.

To provide the access to all these features, the server must implement the mechanism, responsible for processing the HTTP requests from the clients and returning the required data with the help of REST API. In REST API, URI, the analogue of hyperlink in a Web browser, is used for requests generation to server. The server recognizes the URI and the HTTP request method, executes the appropriate actions, and then returns the required data to the client.

**[2] "A Single Sign-On Scheme for Cross Domain Web Applications Using Identity-Based Cryptography" - Y. Wang, Q. Wen, and H. Zhang.**

Authors proposed a novel scheme to satisfy the requirement of cross domain application single sign-on. Identity based encryption and signature have been applied in this protocol, which supports the progress of single sign on between heterogeneous target systems. A ticket can roam from a single sign on domain to the other. The communication between application servers, in this scheme, we use identity-based encryption to protect the data safety; the data transfer between user’s browser and servers we choose session key to prevent attacking.

In general case, when a user accesses the web application, it checks the user’s identity credential locally, that is the authenticate progress is accomplished in the system. Meanwhile, the SOA has been applied in more and more systems, we can separate the authentication and authority module from web applications. We proposed a ticket-based scheme to satisfy the single sign-on requirement between cross domain heterogeneous system

**[3] "OAuth Web Authorization Protocol" - Barry Leiba**

This paper discuss about Internet identity management is an umbrella that covers several related problems, all which stem from our use of multiple Internet services that come from different providers and reside in different trust domains. For each domain, we have a separate identity and use separate authentication. Where NSTIC seeks to consolidate these identities through central management, and software such as password managers tries to make it easier to manage authentication credentials for our various identities.

OAuth addresses this exposure by providing an alternative mechanism through which we can authorize specific actions, and only those specific actions, without giving unrestricted or permanent access. It has the target service create an access token that we can give out that allows only limited access we’ve authorized, perhaps for a limited time or on one-time basis

The OAuth Working Group is developing a set of token types, that will allow implementations to choose different security characteristics that might be appropriate for different use cases and operational environments. Another working group document describes the “Threat Model and Security Considerations” in some detail.

**CHAPTER 3**

**SYSTEM REQUIREMENTS**

**3.1 Hardware Requirements:**

* Memory: 8GB RAM or above
* Processor: intel 8th gen, AMD Zen 2 or above
* Disk: Minimum 32 GB of free space
  1. **Software Requirements:**

* Operating System: Linux kernel v3.2 / Windows build 1809 or higher
* Processor: intel 8th gen, AMD Zen 2 or above
* Tools and Runtimes: NodeJS v14+, Git

**CHAPTER 4**

**Existing and Proposed System**

**4.1 EXISTING SYSTEM:**

Existing System includes the developer of a web service to implement Authorization on upon the service API, using existing protocols like OAuth or using any third-party providers like Auth0. Same goes for every web service developed in an organization, each implementing their way of auth process for other services to access their services. Similarly, each such service needs to implement a way of monitoring their service access control by other services.

Drawbacks of existing system:

* Inconsistent Implementations across organization
* Additional Cost, Time, and Resources for developers
* Might be insecure and harder to fix any known security issues

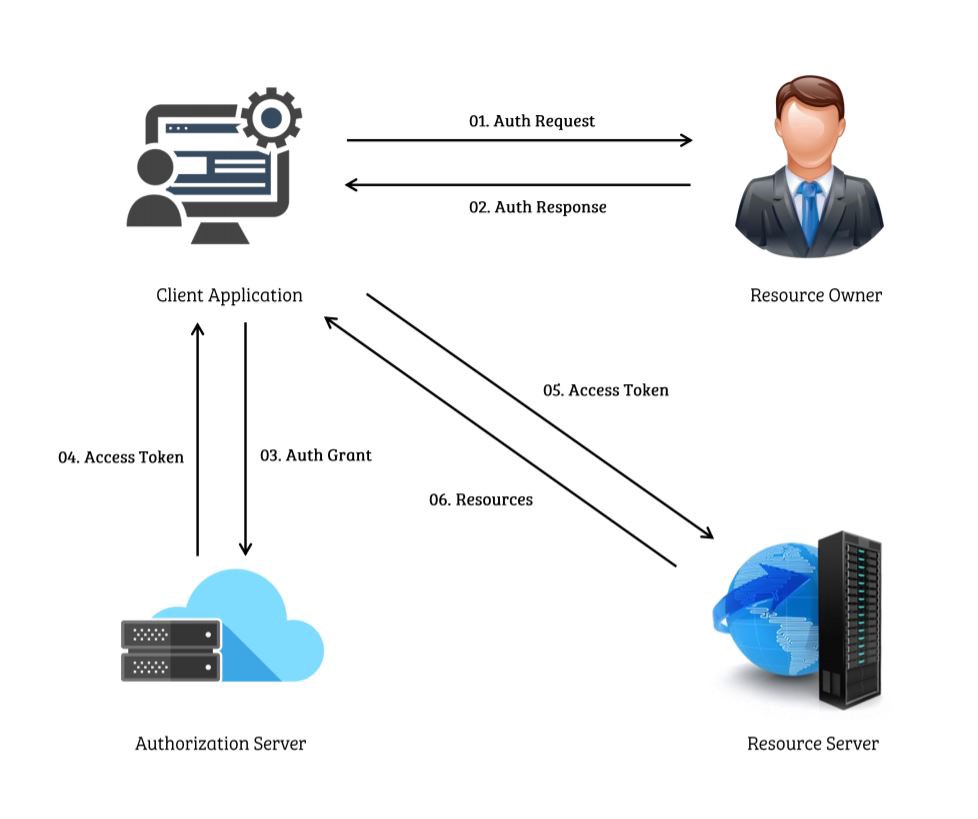


Fig 4.1: OAuth Working Example

**4.2 PROPOSED SYSTEM:**

In this proposal, we aim to develop a library of functions that can help the developers of both client and web service to quickly get started and integrate their respective things to a central auth server, while also supporting the development of central auth server.

If there exists a set of services, we centralize the access control process and management for easier management of services and development. So, the client developer can request access to any of these web services specifically. Once the service provider grants the permission after he requests an access permission, The client developer links his client application with those services.

Also, a core set of developers in the organization can rapidly initialize the authorization server to let service providers to integrate with this central auth server, and it’s up to the service provider to decide whether all the client requests should pass through this centralized server to access the service.

Diagram

Description automatically generated

Fig 4.2: Proposed System Recipe Architecture

This is one of such application of this library, can be furthermore enhanced with additionally features like central monitoring, client-service management, rate-limiting, paid services etc.

**Chapter 5**

**SYSTEM DESIGN AND ARCHITECTURE**

**5.1 Actors Involved**

The system assumes there are at max 3 actors the entire authorization process.

**Client Developer**: one who is developing the client application while also authenticating his client application with central auth server beforehand, so they can access all the services they need for their application. Here, the client application can also be a service which the other services and applications can use.

**Auth Server**: or simply called as Authorization Server or in some cases an Authentication Server, if the service provider decides develop authorization part himself, but generally we call it an “Auth Server”. In this project, the server is centralized so there exists only one server that deals with all the auth related stuff, unlike each service having their own auth process built in.

**Service Provider:** are the ones that develop and maintain the web service. This web service provides a public web API which a client application uses to access the resources it needs after being authorized by the auth server. On beforehand, the service provider must authenticate the service to the auth server.

Often there can be a single actor that takes the multiple roles. For example, the developer of client application, auth server and all services can be single individual.

**5.2 Software Development Kit**

The system or in technical term called “Software Development Kit” (SDK) is divided into 2 components, namely client developer libraries and Auth Server Libraries as shown in below figure.

Fig 5.1: Components of SDK

Each of this component or set of libraries provide Application Programming Interface (API) which are reusable functions written in a programming language.

**5.3 Client Developer Libraries**

This component of SDK provides functions in a specific programming language that can be used by client application developer to call the Web API provided by the central auth server to access the services for which the client is authorized to (or simply called Resource Fetching). For the scope of project, all such functions are planned to be implemented in the language of Javascript/TypeScript which are currently used across all over the web and has vast supporting ecosystem.

Additionally, also automatically helps with managing the session tokes stored in the client context and refreshing the expired tokens.

**5.4 Auth Server Development Libraries**

The other component of this SDK which provides various set of APIs for the development of Centralized Auth Server. The component is divided into subcomponents for more modular behavior.

**Authenticated Client Management**: provides basic functions that are helpful with gathering client, his application information and to monitor and log his access of services.

**Session Management**: For establishing and maintaining sessions between the registered services.

**Deployment**: To help with initializing and deployment of central auth server and while also managing the load balancing and crash handling with fallback mechanisms.

And additionally provides some other utility functions for the ease of development, while also including all the interfaces for the data transfer objects (DTO) for the external communication including to clients and services.

This component helps with self-hosting the so developed auth server which could be used for multiple scenarios through configuration.

**5.5 Methodology**

We use an authentication protocol that works based on tickets/keys to allow nodes communicating over a non-secure network to prove their identity to one another in a secure manner. Its designers aimed it primarily at a client–server model, and it provides mutual authentication—both the user and the server verify each other's identity.

We use existing cryptographic algorithms, both symmetric and asymmetric in the various phases of authentication. For example, AES-256 and RSA-512 algorithms respectively. While a hashing algorithm is used for key verification and storage. SHA-3 is the current latest and most secure hash algorithm.

The client authenticates itself to the Authentication Server (AS) which issues a key, which is time stamped and contains other metadata for AS to identify and verify the validity of the metadata and returns the base64 encoded result to the client application. the key expires at some point although it may be transparently renewed by the user's session manager while they are logged in.

Server that provides with services is then requested by the AS with a different protocol which also guarantees that request is sent to the right client. This protocol request uses http headers for specifying the exact type of service, which the service can declare and documented as a part of their service API documentation.

**CHAPTER 6**

**Advantages and disadvantages**

**6.1 Advantages of Proposed System**

There are several advantages to using a centralized server for cross-application web api authorization:

Improved security: By centralizing the authentication process, you can improve the security of your system by using stronger authentication methods and implementing more robust security controls.

Single sign-on: A centralized authentication server allows users to log in once and gain access to multiple applications without having to remember separate login credentials for each one. This can improve user experience and reduce the risk of password-related security breaches.

Reduced complexity: By centralizing the authentication process, you can reduce the complexity of your system and make it easier to manage. This can save time and resources, as you only need to manage a single authentication server rather than implementing separate authentication systems for each application.

Enhanced control: Using a centralized server allows you to have more control over the authentication process and who has access to your applications. This can help you ensure that only authorized users have access to sensitive data and resources.

Better scalability: A centralized authentication server can handle a larger number of users and applications than a decentralized system, making it easier to scale your system as your user base grows.

**6.2 Disadvantages of the Proposed System**

There are a few potential disadvantages to using a centralized server for cross-application web authorization:

Dependency on the centralized server: The entire system relies on the availability and reliability of the centralized server. If the server goes down or experiences issues, it can disrupt the entire system.

Single point of failure: The centralized server represents a single point of failure in the system. If it goes down, the entire system is affected.

Performance: The centralized server may become a bottleneck if it is not able to handle the volume of requests from client applications. This can lead to slower performance and a poor user experience.

Security risks: The centralized server stores sensitive information, such as access tokens and user credentials, which can make it a target for attacks. If the server is compromised, it can expose sensitive information and potentially compromise the security of the entire system.

Maintenance and management: The centralized server require ongoing maintenance and management to ensure it is running smoothly. This can be time-consuming and costly.

Overall, while a centralized server can offer some benefits in terms of centralized control and enabling multiple client applications to share the same authorization server and authentication server, it also introduces additional complexity and potential risks.

**Future Enhancements**

* Supporting already pre-existing protocols or auth providers
* Going beyond Web
* Monetization of API Requests
* Rate-limiting
* Email & Other forms of User Verification, including Security Device
* Existing Social Logins

**CONCLUSION**

In Conclusion, the System Proposed will separate the Authentication and Authorization Processes making it simpler for the Clients who are having different applications in different domains to use a Single Login and Verification to Access different types of resources.

The Library Provided saves the developer (Client) huge amount of time which would be needed to develop the Centralized Server that increases the Cost of the system and the resources that are required to maintain the Authentication and Authorization working Properly

without compromising the security of the data that is being accessed by the User using the System.

The usage of Tokens for Authentication and Authorization increases the Security of the data by usage of Sessions to keep track of users and help in providing the Role Based Authentications which also further enhances the Security of the data by categorizing the people for read access and write access, admin access. It also further eases the task of the user using the resources by restricting the number of login credentials required to be remembered.

**REFERENCES**

1. IEEE Papers:
   1. Y. Wang, Q. Wen and H. Zhang, "A Single Sign-On Scheme for Cross Domain Web Applications Using Identity-Based Cryptography", 2010, pp. 483-485,

doi: 10.1109/NSWCTC.2010.120

* 1. B. Leiba, "OAuth Web Authorization Protocol," in IEEE Internet Computing, vol. 16, no. 1, pp. 74-77, Jan.-Feb. 2012,

doi: 10.1109/MIC.2012.11.

* 1. V. Krylov, N. Volkova and Y. Kozina, "Information technology of user authentication in cross-platform systems", 2017, pp. 952-954, doi: 10.1109/IDAACS.2017.8095227.

1. Websites
   1. <https://en.wikipedia.org/wiki/Representational_state_transfer>
   2. <https://en.wikipedia.org/wiki/Kerberos_(protocol)>
   3. <https://oauth.net/2/>
   4. <https://auth0.com/docs/get-started/auth0-overview>
   5. <https://www.rfc-editor.org/rfc/rfc7519>
   6. <https://auth0.com/blog/what-is-an-authentication-server/>