Authentication & Authorization

Owner	**Maddineni, Sudhakar**
Collaborators	Lourng, Voy Ruwan Jayaweera Prashar, Ankur Thambugala, Bhagya
Status	PROPOSED
Timelines	Q2: DRAFT Q3: PROPOSED Q3: ACCEPTED

1 Abstract

This document provides guidelines and standards for authentication and authorization mechanisms to ensure secure access to systems and resources. The guidelines cover topics such as user/system authentication, roles and permissions management and access patterns for external and internal clients.

2 Table of contents

2.1 Introduction

The traditional client-server authentication model relies on clients requesting access to restricted resources by authenticating with the server using the resource owner's credentials. This authentication process establishes trust between the client and server before granting access to the requested resources. Clients can take various forms such as stateless or stateful applications, native applications, or software components that facilitate interactions. To ensure a secure and reliable authentication process, clients must be identified through standard authentication mechanisms, allowing for centralized authorization procedures to determine the client's permitted access to resources.

2.2 Terminology

- Resource Owner: The entity that owns the protected resources and grants access to them.
- Entity: Any individual, device or application that interacts with a system and needs access to resources or services.
- Client: The application or entity requesting access to the protected resources on behalf of the resource owner.
- Authorization Server: The server responsible for authenticating the client and issuing access tokens. E.g IES
- Resource Server: The server hosting the protected resources that the client wants to access.
- Access Token: A credential representing the authorization granted to the client to access protected resources.
- Refresh Token: A credential used by the client to obtain a new access token without involving the resource owner.
- Grant Type: The type of authorization grant used by the client to obtain an access token (e.g., Authorization Code, Implicit, Client Credentials).
- Scope: A mechanism to limit the access privileges granted to the client by the resource owner.
- ID Token: A JSON Web Token (JWT) that provides identity information about the authenticated user in OpenID Connect.
- Role-Based-Access-Control (RBAC): a method of managing and controlling access to resources with a system based on roles.
- Role: A predefined set of permissions and privileges that define the actions or operations a user or entity can perform within a system.
- Access Control List (ACL): A list that specifies the permissions associated with specific resources in the system.

2.3 Authentication

OpenID Connect (OIDC) is proposed as the standard for authentication due to its robust features and secure design. OIDC is built on top of the OAuth 2.0 framework, providing a powerful identity layer that offers reliable and interoperable authentication capabilities. It uses JSON Web Tokens (JWTs) to securely exchange user identity information between parties.

Note that many existing applications currently use OAuth 1.0a, however support for this protocol is declining with many partners and systems including IMS Global deprecating support in favor of OAuth 2.0.

A JWT (JSON Web Token) is a compact, URL-safe means of representing claims to be transferred between parties. A JWT consists of three parts: a header, a payload, and a signature. More information on the JWT can be found in this RFC.

This document proposes utilizing Identity Enablement Services (IES) as the OIDC provider for authentication purposes within the system, leveraging its capabilities and functionalities. Sample JWT token issued by IES can be found in this document.

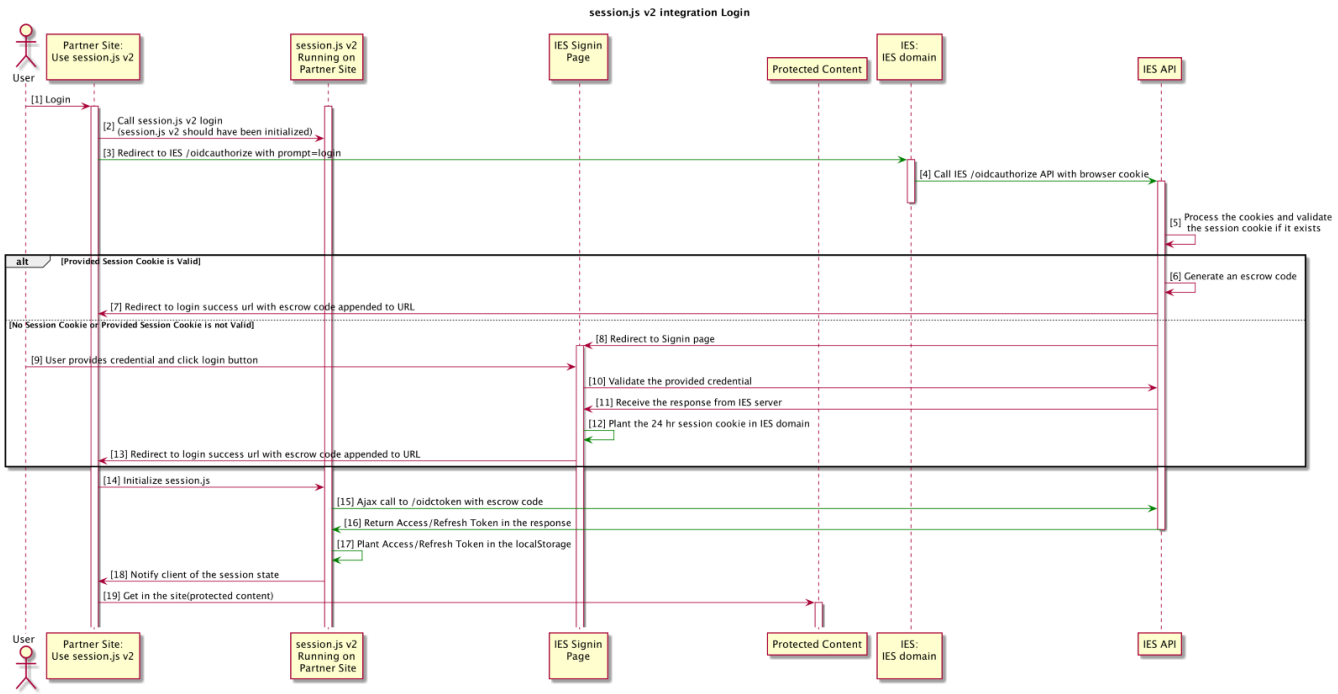
2.3.1 End User Authentication flow

This document proposes the utilization of the OAuth 2.0 Authorization Code Grant flow as the standard for end-user authentication.

The following steps outline the Authorization Code Grant flow:

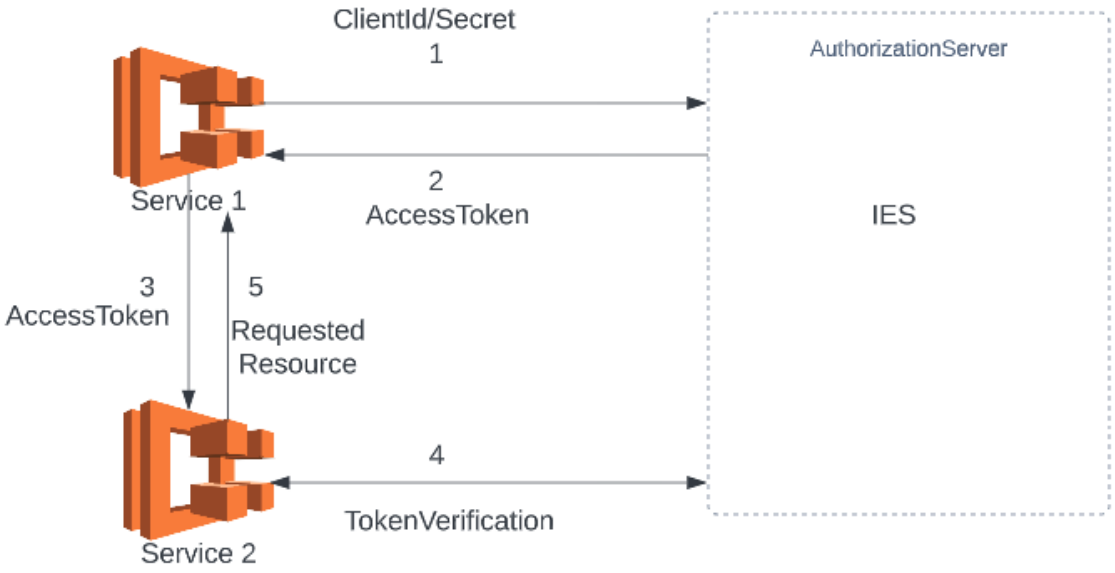
- Client Initiation: The client application redirects the user to the authorization server's authorization endpoint, including the client identifier and requested scopes.
- User Authorization: The user grants permissions and authenticates through an authorization prompt presented by the authorization server.
- Authorization Grant: The authorization server generates an authorization code and redirects the user back to the client application's redirect URI.
- Authorization Code Exchange: The client application sends the authorization code, client identifier, client secret, redirect URI, and grant type to the authorization server's token endpoint.
- Access Token Retrieval: The authorization server validates the request and provides an access token, optional refresh token, and additional information.
- Access Token Usage: The client application includes the received access token in API requests to access user resources.
- Token Refresh: If necessary, the client application employs the refresh token to acquire a new access token without user involvement. This is done by sending a request to the authorization server's token endpoint with the refresh token, client credentials, and grant type.

IES is used for End User Authentication. Following is the IES Login Flow Diagram.



Further details on IES API can be found here.

2.3.2 Client Credentials grant



The Client Credentials grant type is an OAuth 2.0 grant type for enabling secure service-service communication. It is suitable to establish communication between the services when there is no user involved in the process.

In this flow,

1. The client service/application (Service1) sends a request to the authorization server (IES) with its credentials (client ID and secret) using the “client credentials” grant type.
2. The Authorization Server authenticates the client by verifying the client credentials. If the client is authenticated, the authorization server issues an Access Token (JWT) to the client.
3. Client can then use the Access Token to make requests to the protected resource on Resource Server (Service 2) by including the access token in the authorization header (Authorization: Bearer <JWT>) of the request.
4. The resource server verifies the access token with authorization server to ensure that it is valid and has not expired.
5. If the access token is valid, the resource server responds with the requested resource.

2.4 Authorization

Authorization helps protect confidential data or service by ensuring that only identified specific individuals or systems can access it. By implementing access controls and permissions, organizations can prevent unauthorized users from viewing, modifying, or deleting critical information, thereby reducing the risk of data breaches, limit access privileges to specific resources, reducing the potential for data misuse, insider threats.

RBAC (Role-Based Access Control) is a widely adopted industry standard for authorization. It assigns roles to users based on their responsibilities and grants permissions accordingly. This approach simplifies the management of access controls by associating permissions with roles rather than individual users.

Group Manager (aka *Group Auth*) is a multi-tenant Access Control List (ACL) under Registrar’s suite of services. When a particular system is onboarded to Group Manager (GM) an entity called a *‘system’* is created inside GM. This system is the boundary/scope of a single tenant. GM also identifies a super user to each system (provided at the time of onboarding). This super user can manage roles, permissions, and role assignments to other users. A role in GM is referred to as a Group. A group can have one or many permissions. A permission is a text which can uniquely identify a single ability in a system (ex. CREATE_COURSE, READ_SECTION, etc.). Once a particular role is assigned to a user, that user will carry all the permissions assigned to that group. A User in GM is identified by an identifier. Currently all the users of GM are IES users. GM has a fully developed RESTful API which can be leveraged by other systems to manage its roles, permissions, and users.

For this ACL to work as an authorization model, each onboard system should implement its control mechanisms which respect these granted/revoked permissions. Systems can check whether a particular user is assigned a particular permission (before allowing the user to perform some function) by invoking certain APIs of GM.

2.4.1 Principle of Least Privilege

Users and roles should have minimal permissions by default. This ensures that a user is not accidentally provided more access than intended in the present or coupled with future work.

What it is and why it's important and examples.

The Principle of Least Privilege is a security concept that advocates granting users and processes only the minimum permissions necessary to perform their required tasks. It is based on the idea of restricting access rights to the bare minimum required to carry out specific actions or access specific resources.

By implementing the least privilege, organizations reduce the potential attack surface available to attacker or malware. If a user or process is compromised, the damage that can be done is limited because they only have access to a restricted set of resources.

When each user or process has restricted privileges, it becomes easier to track and audit their actions. With a fine-grained authorization model based on least privilege, organizations can identify the actions taken by specific users or processes, detect anomalies, and establish accountability for any security incidents or policy violations.

Enforcing Access Control Lists (ACLs) at different layers of an application is an effective way to control and restrict access to resources.

2.4.2 Presentation Layer

User Interface (UI): Implement access control logic within the UI components to control user interactions and visibility of specific features based on their permissions.

Client-side Validation: Use client-side code (JavaScript) to enforce ACLs on user inputs, ensuring that only authorized actions are performed.

2.4.3 Application Layer

Business Logic: Integrate ACL checks into the application's business logic layer to enforce authorization rules for specific operations or functionalities.

Controllers/Handlers: Apply ACL validation within the controllers or request handlers to ensure that only authenticated and authorized users can access specific endpoints or perform certain actions.

2.4.4 Service Layer

API Services: Implement ACL checks within the service layer that exposes APIs, validating user permissions before executing requested operations or returning sensitive data. To enhance the security of Spring Boot-based Java microservices, it is recommended to leverage the existing PSF security module. This allows for the utilization of the module's robust security features and capabilities within the microservices architecture.

Middleware: Utilize middleware components to intercept incoming requests and enforce ACL rules before they reach the underlying service logic.

2.4.5 Data Layer

Database Access: Apply ACL rules at the database level, ensuring that users or roles have appropriate privileges to read, write, or modify specific data records or tables.

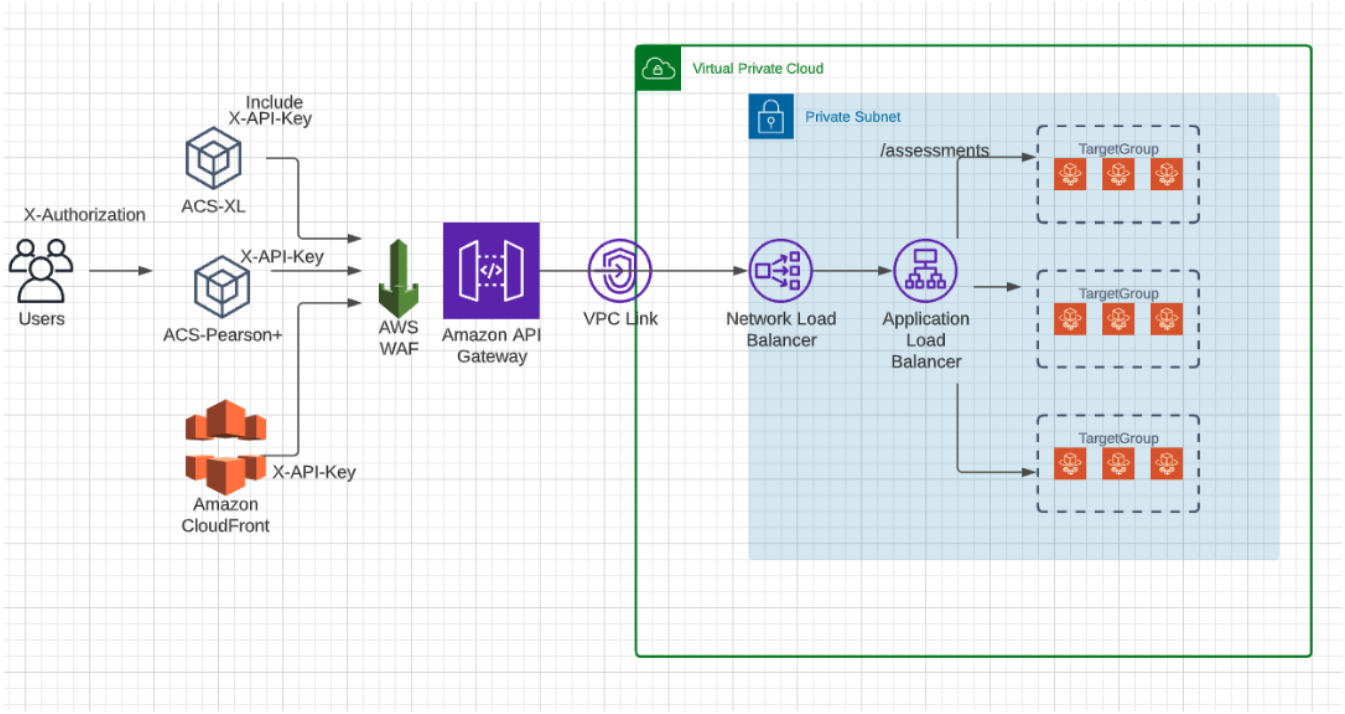
Row-Level Security: Utilize database features or extensions that provide row-level security, allowing fine-grained access control at the individual record level based on user attributes or roles.

The below ADRs contain examples of integration patterns that enforce authentication and authorization at different layers.

API Gateway, BFF/AppSync, Service-Service -1, 2

2.5 External Traffic via API Gateway

Reference Architecture Diagram:



API Gateway and VPC Link:

- Leverage API Gateway as the entry point for external clients to access domain microservices securely and efficiently.
- Establish a VPC link between API Gateway and VPC, allowing direct communication with the backend services while keeping them isolated from the public internet.
- Ensure that the gateway header “X-API-Key" is properly secured and protected against unauthorized access.
- Ensure that for authenticated requests, clients include the user JWT token in the header as Authorization Bearer token, adhering to the standard authentication protocol.

Load Balancing:

- Use a combination of Network Load Balancer (NLB) and Application Load Balancer (ALB) for routing requests to domain microservices.
- ALB provides advanced features like content-based routing, SSL termination, and integration with AWS services, making it suitable for HTTP /HTTPS traffic.
- ALB can be registered as a target of NLB, allowing traffic to be forwarded from NLB to ALB without the need for active management of ALB IP address changes. This combination enables the benefits of NLB, including Private Link and zonal static IP addresses, along with the advanced request-based routing of ALB for load balancing traffic to applications.

Private Subnets and Security:

- Deploy domain microservices within private subnets of Virtual Private Cloud (VPC) for enhanced security and isolation from the public internet.
- Implement security groups and network ACLs to control inbound and outbound traffic to microservices.
- Consider using AWS Secrets Manager for secure storage and retrieval of sensitive credentials needed to access domain microservices.

ECS Services:

- Utilize Amazon Elastic Container Service (ECS) to manage and deploy your containerized microservices at scale.
- Implement auto-scaling policies based on metrics like CPU utilization or request count to ensure optimal performance and cost efficiency.

WAF:

- Implement Web Application Firewall (WAF) to protect API endpoints from common web exploits, such as SQL injection and cross-site scripting (XSS).
- Configure WAF rules and policies to safeguard API Gateway and domain microservices from security threats.
- Regularly update and fine-tune WAF configurations to stay protected against emerging threats and evolving attack patterns.

2.6 Internal Traffic between Services

VPC Prefix List:

- Use VPC Prefix List to define and manage CIDR blocks within a single VPC.
- Apply VPC Prefix List to network ACLs and security groups for granular traffic control within the VPC.
- Standardize the naming and organization of VPC Prefix Lists to ensure consistency and ease of management.
- Use VPC Prefix Lists for intra-VPC traffic filtering and segmentation, ensuring secure and controlled communication within the VPC.

Private Link:

- When requiring secure and private access to AWS services or third-party SaaS solutions over the AWS network, consider utilizing Private Link.
- Establish interface VPC endpoints to connect to services without exposing them to the public internet.
- Leverage Private Link for accessing services like AWS Lambda, S3, DynamoDB, and third-party solutions securely within VPCs.

Transit Gateway:

- When seeking to streamline network connectivity and routing between multiple VPCs and on-premises networks, consider opting for Transit Gateway.
- Design a hub-and-spoke network architecture with Transit Gateway as a central hub for interconnecting VPCs and on-premises environments.
- Follow recommended practices for configuring Transit Gateway attachments, route propagation, and route tables to ensure efficient and secure traffic routing.

2.7 Guidelines for API Keys

- Keys must be able to be rotated without downtime for any system. New keys should be able to be **added** to a system. The consumer is then given the opportunity to switch keys before the old key is **deleted**.
- Keys must not be shared between different consumers. It is a security risk and makes it very challenging to rotate keys.
- Consumers should not store keys in plain text or in source control. Use Secrets Manager or similar tool to manage key securely.

2.8 References

- https://datatracker.ietf.org/doc/html/rfc6749
- https://openid.net/specs/openid-connect-core-1_0.html
- Pearson Single Sign On Session Management library V2
- System to System Token Library
- Authentication & Authorization - Security Module
- https://docs.spring.io/spring-security/reference/servlet/architecture.html
- https://docs.aws.amazon.com/whitepapers/latest/building-scalable-secure-multi-vpc-network-infrastructure/transit-gateway.html
- https://aws.amazon.com/blogs/compute/protecting-your-api-using-amazon-api-gateway-and-aws-waf-part-i/
- https://aws.amazon.com/blogs/compute/protecting-your-api-using-amazon-api-gateway-and-aws-waf-part-2/