***Authentication vs. Authorization***

What's the difference between ***authentication*** and ***authorization***? **Authentication** ***confirms*** that users are who they say they are. **Authorization** gives those users ***permission*** to access a resource.

While authentication and authorization might sound similar, they are ***distinct*** security processes in the world of ***identity*** and access management (IAM).

## What Is Authentication?

Authentication is the act of validating that users are whom they claim to be. This is the first step in any security process.

An authentication process is complete with:

* **Passwords - u**sernames and passwordsare the most common authentication factors. If a user enters the correct data, the system assumes the identity is valid and grants access.
* One-time pins - grant access for only one session or transaction.
* **Authentication apps - g**enerate security codes via an outside party that grants access.
* Biometrics - **a** user presents a fingerprint or eye scan to gain access to the system.

In some instances, systems require the successful verification of more than one factor before granting access. This multi-factor authentication (MFA) requirement is often deployed to increase security beyond what passwords alone can provide.

## What Is Authorization?

Authorization in system security is the process of giving the user permission to access a specific resource or function. This term is often used interchangeably with access control or client privilege.

Giving someone permission to download a particular file on a server or providing individual users with administrative access to an application are good examples of authorization.

In secure environments, authorization must always follow authentication. Users should first prove that their identities are genuine before an organization’s administrators grant them access to the requested resources.

## Authentication vs. Authorization

Despite the similar-sounding terms, authentication and authorization are ***separate*** steps in the login process. Understanding the difference between the two is key to successfully implementing an IAM solution.

Let's use an analogy to outline the differences.

Consider a person walking up to a locked door to provide care to a pet while the family is away on vacation. That person needs:

* **Authentication**, in the form of a key. The lock on the door only grants access to someone with the correct key in much the same way that a system only grants access to users who have the correct credentials.
* **Authorization,** in the form of permissions. Once inside, the person has the authorization to access the kitchen and open the cupboard that holds the pet food. The person may not have permission to go into the bedroom for a quick nap.

Authentication and authorization work together in this example. A pet sitter has the right to enter the house (authentication), and once there, they have access to certain areas (authorization).

Systems implement these concepts in the same way, so it’s crucial that IAM administrators understand how to ***utilize*** both:

* **Authentication - L**et every staff member access your workplace systems if they provide the right credentials in ***response*** to your chosen authentication requirements.
* **Authorization -** Grant permission to department-specific files, and reserve access to confidential data, such as financial information, as needed. Ensure that employees have access to the files they need to do their jobs.

The OpenID Connect (OIDC) protocol is built on the OAuth 2.0 protocol and helps authenticate users and ***convey*** information about them. The OAuth 2.0 protocol controls authorization to access a protected resource, like your web app, native app, or API service.

The OAuth 2.0 protocol provides API security through scoped access tokens. OAuth 2.0 enables you to delegate authorization, while OIDC enables you to ***retrieve*** and store authentication information about your end users. OIDC ***extends*** OAuth 2.0 by providing user authentication and single sign-on (SSO) functionality.

SSO - an authentication method that enables users to securely authenticate with multiple applications and websites by using just one set of credentials.

**The OAuth 2.0 spec has four important roles:**

* **authorization server**: The server that issues the access token.
* **resource owner**: Normally your application's end user that grants permission to access the resource server with an access token.
* **client**: The application that requests the access token and then passes it to the resource server.
* **resource server**: Accepts the access token and must ***verify*** that it's valid. In this case, this is your application.

**Other important terms:**

* **OAuth 2.0 grant**: The authorization given (or granted) to the client by the user. Examples of grants are **authorization code** and **client credentials**.
* **access token**: The token issued by the authorization server in exchange for the grant.
* **refresh token**: An optional token that is exchanged for a new access token if the access token has expired.

**The usual OAuth 2.0 grant flow looks like this:**

1. Client requests authorization from the resource owner (usually the user).
2. If the user gives authorization, the client passes the authorization grant to the authorization server.
3. If the grant is valid, the authorization server returns an access token, possibly alongside a refresh and/or ID token.
4. The client now uses that access token to access the resource server.

OpenID Connect is an authentication standard built on top of OAuth 2.0. It adds an additional token called an ID token. OpenID Connect also standardizes areas that OAuth 2.0 leaves up to choice, such as scopes, endpoint discovery, and dynamic registration of clients.

Although OpenID Connect is built on top of OAuth 2.0, the OpenID Connect specification uses slightly different terms for the roles in the flows:

* **OpenID provider**: The authorization server that issues the ID token.
* **end user**: The end user's information that is contained in the ID token.
* **relying party**: The client application that requests the ID token.
* **ID token**: The token issued by the OpenID Provider and contains information about the end user in the form of claims.
* **claim**: The claim is a piece of information about the end user.

The high-level flow looks the same for both OpenID Connect and regular OAuth 2.0 flows. The primary difference is that an OpenID Connect flow results in an ID token, in addition to any access or refresh tokens.