***What is OAuth****?*

*OAuth is an open-standard authorization protocol or framework that provides applications the ability for “secure designated access.” This minimizes risk in a major way: In the event, your password remains safe.*

*OAuth doesn’t share password data but instead uses authorization tokens to prove an identity between consumers and service providers. OAuth is an authentication protocol that allows you to approve one application interacting with another on your behalf without giving away your password.*

***OAuth Examples***

*The simplest example of OAuth in action is one website saying “hey, do you want to log into our website with other website’s login?” In this scenario, the only thing the first website – let’s refer to that website as the consumer – wants to know is that the user is the same user on both websites and has logged in successfully to the service provider – which is the site the user initially logged into, not the consumer.*

*Facebook apps are a good OAuth use case example. Say you’re using an app on Facebook, and it asks you to share your profile and pictures. Facebook is, in this case, the service provider: it has your login data and your pictures. The app is the consumer, and as the user, you want to use the app to do something with your pictures. You specifically gave this app access to your pictures, which OAuth is managing in the background.*

*OAuth is about authorization and not authentication. OAuth doesn’t pass authentication data between consumers and service providers – but instead acts as an authorization token of sorts.*

***How OAuth Works -***

*There are 3 main players in an OAuth transaction: the user, the consumer, and the service provider.*

*Let us have a look at how OAuth authorizes access. Assume a user has already signed in to a website or service, and now the user wants to initiate a transaction that needs access to a third-party site or service. The authorization process follows the steps below:*

* *The application requests authorization to access a protected service provider.*
* *The user authorizes the request.*
* *The application provides proof of user authorization to the service provider in exchange for an access token.*
* *The user is redirected to the service provider to provide permission.*
* *Once approved by the user, the application obtains the access token.*
* *The application requests access to the protected resources from the service provider.*

*The greatest benefit of OAuth for a website, such as a news, community, or e-commerce site, is that authenticated website access can be extended to an unlimited number of additional users without those users creating new accounts requiring an email address and a new password. Open authorization reduces friction for both parties. Websites can scale, and users do not have to create yet another online account.*

*Further, there is more security. When a user joins a website by signing in with Facebook, for example, if that website becomes the target of a cyberattack, users who logged on with OAuth will not have their credentials exposed or stolen.*

*Ex - In example, Joe is the user, Bitly is the consumer, and Twitter is the service provided who controls Joe’s secure resource (his Twitter stream).*

***Step 1*** *– The User Shows Intent*

*Joe (User): “Hey, Bitly, I would like you to be able to post links directly to my Twitter stream.”*

*Bitly (Consumer): “Great! Let me go ask for permission.”*

***Step 2*** *– The Consumer Gets Permission*

*Bitly: “I have a user that would like me to post to his stream. Can I have a request token?”*

*Twitter (Service Provider): “Sure. Here’s a token and a secret.”*

*The secret is used to prevent request forgery. The consumer uses the secret to sign each request so that the service provider can verify it is actually coming from the consumer application.*

***Step 3*** *– The User Is Redirected to the Service Provider*

*Bitly: “OK, Joe. I’m sending you over to Twitter so you can approve. Take this token with you.”*

*Joe: “OK!”*

*- Bitly directs Joe to Twitter for authorization>*

*This is the scary part. If Bitly were super-shady Evil Co. it could pop up a window that looked like Twitter but was really phishing for your username and password. Always be sure to verify that the URL you’re directed to is actually the service provider (Twitter, in this case).*

***Step 4*** *– The User Gives Permission*

*Joe: “Twitter, I’d like to authorize this request token that Bitly gave me.”*

*Twitter: “OK, just to be sure, you want to authorize Bitly to do X, Y, and Z with your Twitter account?”*

*Joe: “Yes!”*

*Twitter: “OK, you can go back to Bitly and tell them they have permission to use their request token.”*

*Twitter marks the request token as “good-to-go,” so when the consumer requests access, it will be accepted (so long as it’s signed using their shared secret).*

***Step 5*** *– The Consumer Obtains an Access Token*

*Bitly: “Twitter, can I exchange this request token for an access token?”*

*Twitter: “Sure. Here’s your access token and secret.”*

***Step 6*** *– The Consumer Accesses the Protected Resource*

*Bitly: “I’d like to post this link to Joe’s stream. Here’s my access token!”*

*Twitter: “Done!”*

*In our scenario, Joe never had to share his Twitter credentials with Bitly. He simply delegated access using OAuth in a secure manner. At any time, Joe can login to Twitter and review the access he has granted and revoke tokens for specific applications without affecting others. OAuth also allows for granular permission levels. You can give Bitly the right to post to your Twitter account, but restrict LinkedIn to read-only access.*

*OAuth 2.0 –*

*OAuth 2.0 is an authorization protocol and NOT an authentication protocol. As such, it is designed primarily as a means of granting access to a set of resources, for example, remote APIs or user data.*

*OAuth 2.0 uses Access Tokens. An Access Token is a piece of data that represents the authorization to access resources on behalf of the end-user. OAuth 2.0 doesn’t define a specific format for Access Tokens. However, in some contexts, the JSON Web Token (JWT) format is often used. This enables token issuers to include data in the token itself. Also, for security reasons, Access Tokens may have an expiration date.*

*OAuth2.0 Roles*

*These define the essential components of an OAuth 2.0 system.*

* *Resource Owner: The user or system that owns the protected resources and can grant access to them.*
* *Client: The client is the system that requires access to the protected resources. To access resources, the Client must hold the appropriate Access Token.*
* *Authorization Server: This server receives requests from the Client for Access Tokens and issues them upon successful authentication and consent by the Resource Owner. The authorization server exposes two endpoints: the Authorization endpoint, which handles the interactive authentication and consent of the user, and the Token endpoint, which is involved in a machine-to-machine interaction.*
* *Resource Server: A server that protects the user’s resources and receives access requests from the Client. It accepts and validates an Access Token from the Client and returns the appropriate resources to it.*

*OAuth 2.0 Scopes*

*Scopes are an important concept in OAuth 2.0. They are used to specify exactly the reason for which access to resources may be granted. Acceptable scope values, and which resources they relate to, are dependent on the Resource Server.*

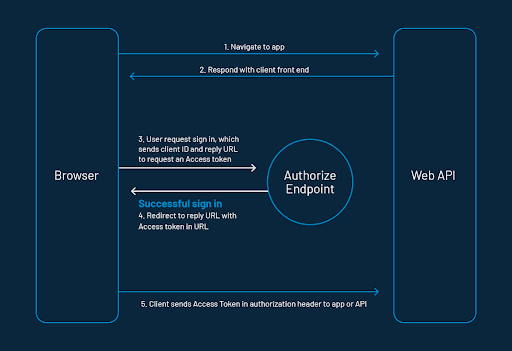
*OAuth 2.0 Access Tokens and Authorization Code*

*The OAuth 2 Authorization server may not directly return an Access Token after the Resource Owner has authorized access. Instead, and for better security, an Authorization Code may be returned, which is then exchanged for an Access Token. In addition, the Authorization server may also issue a Refresh Token with the Access Token. Unlike Access Tokens, Refresh Tokens normally have long expiry times and may be exchanged for new Access Tokens when the latter expires. Because Refresh Tokens have these properties, they have to be stored securely by clients.*

*How Does OAuth 2.0 Work?*

*At the most basic level, before OAuth 2.0 can be used, the Client must acquire its own credentials, a \_client id \_ and client secret, from the Authorization Server in order to identify and authenticate itself when requesting an Access Token. Using OAuth 2.0, access requests are initiated by the Client, e.g., a mobile app, website, smart TV app, desktop application, etc. The token request, exchange, and response follow this general flow:*

* *The Client requests authorization (authorization request) from the Authorization server, supplying the client id and secret to as identification; it also provides the scopes and an endpoint URI (redirect URI) to send the Access Token or the Authorization Code to.*
* *The Authorization server authenticates the Client and verifies that the requested scopes are permitted.*
* *The Resource owner interacts with the Authorization server to grant access.*
* *The Authorization server redirects back to the Client with either an Authorization Code or Access Token, depending on the grant type.*
* *With the Access Token, the Client requests access to the resource from the Resource server.*





*Grant Types in OAuth 2.0*

*In OAuth 2.0, grants are the set of steps a Client has to perform to get resource access authorization. The authorization framework provides several grant types to address different scenarios.*

* *Authorization Code grant: The Authorization server returns a single-use Authorization Code to the Client, which is then exchanged for an Access Token. This is the best option for traditional web apps where the exchange can securely happen on the server side.*
* *Implicit Grant: A simplified flow where the Access Token is returned directly to the Client. In the Implicit flow, the authorization server may return the Access Token as a parameter in the callback URI or as a response to a form post.*
* *Authorization Code Grant with Proof Key for Code Exchange (PKCE): This is similar to the Authorization Code grant, but with additional steps that make it more secure for mobile/native apps and SPAs.*
* *Resource Owner Credentials Grant Type: This grant requires the Client first to acquire the resource owner’s credentials, which are passed to the Authorization server. It is, therefore, limited to Clients that are completely trusted. It has the advantage that no redirect to the Authorization server is involved, so it is applicable in the use cases where a redirect is infeasible.*
* *Client Credentials Grant Type: Used for non-interactive applications e.g., automated processes, microservices, etc. In this case, the application is authenticated per se by using its client id and secret.*
* *Device Authorization Flow: A grant that enables use by apps on input-constrained devices, such as smart TVs.*
* *Refresh Token Grant: The flow that involves the exchange of a Refresh Token for a new Access Token.*

*Differences between OAuth 1.0 and OAuth 2.0 –*

*OAuth 2.0 is a complete rewrite of OAuth 1.0 and it’s not backward compatible with OAuth 1.0.*

* *Better support for non-browser applications - OAuth 1.0 has been designed focusing on the interactions of inbound and outbound messages in web client applications. Therefore, it is inefficient for non-browser clients. OAuth 2.0 has addressed this issue by introducing more authorization flows for different client needs that do not use web UIs.*
* *Reduced complexity in signing requests - OAuth 1.0 needs to generate a signature on every API call to the server resource and that should be matched with the signature generated at the receiving endpoint. OAuth 2.0 do not need to generate signatures. It uses TLS/SSL (HTTPS) for communication.*
* *The separation of roles - Handling resource requests and handling user authorization can be decoupled in OAuth 2.0. It has clearly defined the roles involved in communication which are client, resource owner, resource server, and authorization server.*
* *The short-lived access token and the refresh token - In OAuth 1.0, access tokens can be stored for a year or more. But in OAuth 2.0, access tokens can contain an expiration time, which improves the security and reduces the chances of illegal access. And it offers a refresh token which can be used to get a new access token at the access token expiration without reauthorizing.*

*SAML vs. OAuth*

*SAML (Security Assertion Markup Language) is an alternative federated authentication standard that many enterprises use for Single-Sign On (SSO). SAML enables enterprises to monitor who has access to corporate resources.*

*There are many differences between SAML and OAuth. SAML uses XML to pass messages, and OAuth uses JSON. OAuth provides a simpler mobile experience, while SAML is geared towards enterprise security. That last point is a key differentiator: OAuth uses API calls extensively, which is why mobile applications, modern web applications, game consoles, and Internet of Things (IoT) devices find OAuth a better experience for the user. SAML, on the other hand, drops a session cookie in a browser that allows a user to access certain web pages – great for short-lived work days, but not so great when have to log into your thermostat every day.*