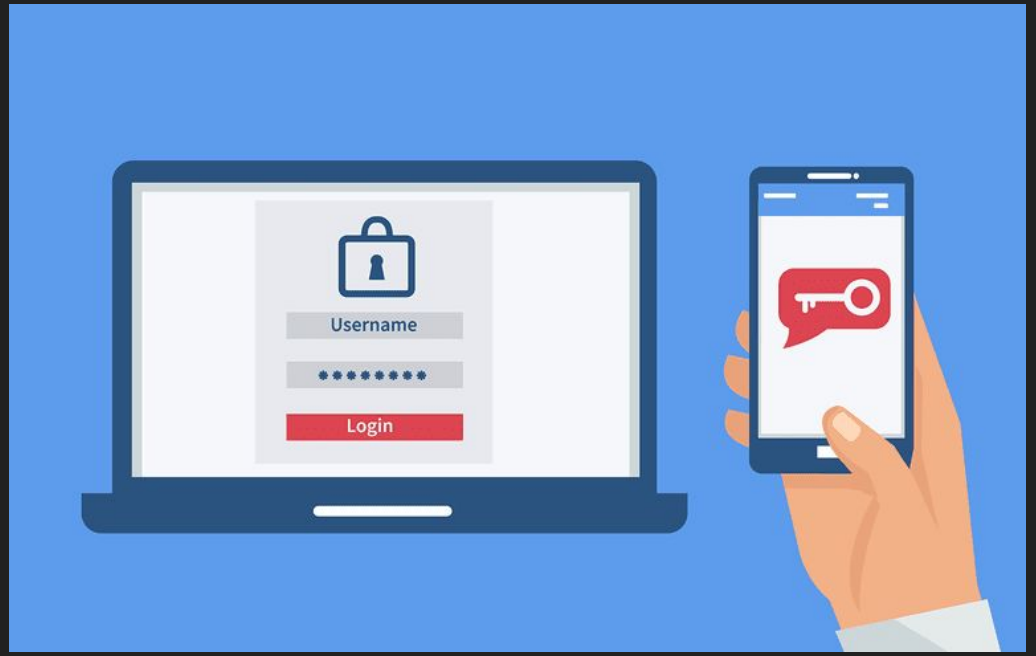
Authentication and Authorization

By Arslan and Brian

What is Authentication?

- The process of proving and determining whether or not a user is genuine and who the user claims to be
- Used in a variety of applications that require usernames and passwords to log in



Authentication vs Identification and Authorization

- **Identification** involves using something such as a username, ID card, etc. in order to uniquely identify a user
- **Authentication** involves proving that an identity actually belongs to the user
- **Authorization** involves providing permissions to a user according to the user's proven identity

Identify

Authenticate

Authorize

Authentication Factors

- Elements which are used in order to help authenticate a user
- Most commonly used are knowledge, possession, and inherence factors
- Single-factor authentication, Multi-factor authentication

Authentication Factors

- **Knowledge Factor**: Information that is known only to a specific user (Ex. Passwords, PIN numbers)
- **Possession Factor**: Items that are owned by a specific user (Ex. Software/Hardware tokens, ID card)
- **Inherence Factor**: What the user is (Ex. Biometrics) and the actions of the user (Ex. Keystroke dynamics)
- **Location Factor**: The user's physical location




Diving into Possession Factors

- There are two main types: Hardware and Software Tokens.

Diving into Possession Factors

Hardware Tokens

- **Connected Tokens**
  - Devices that are physically connected to the user's computer to be used later for authentication purposes.
  - Known to transmit data automatically
  - Different types include card readers, wireless tags and tokens in a form of USB key.

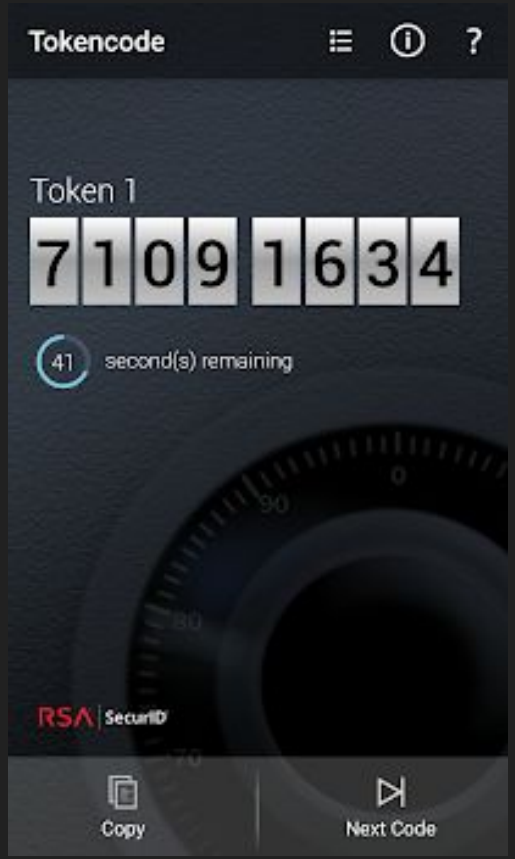
Diving into Possession Factors

- **Disconnected Tokens**
  - Have no connection to user's computer
  - Typically uses a built-in-screen to display the generated authenticated data, which is manually typed in by the user

Diving into Possession Factors

Software Tokens

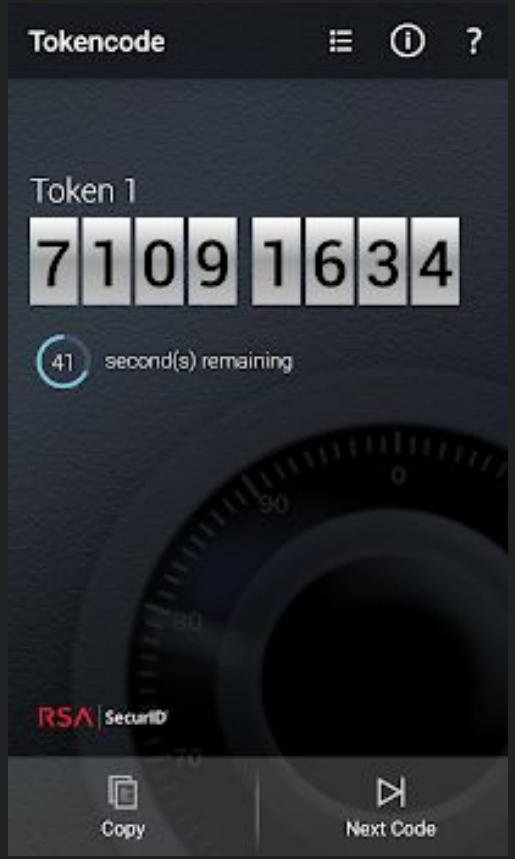
- Typically stored on a general-purpose electronic devices such as computers, mobile phones, etc.
- If needed, software tokens can be duplicated
- Conversely, credentials in the hardware tokens are stored on a dedicated hardware device and cannot be copied.



Diving into Possession Factors

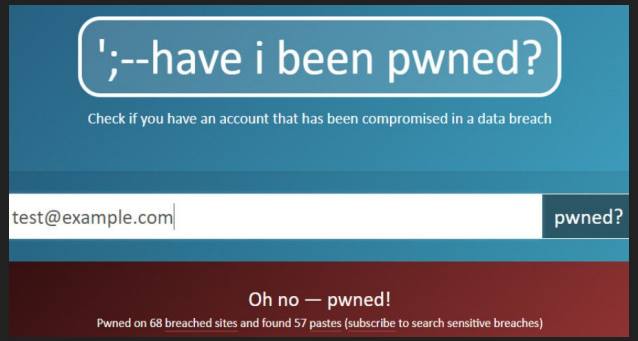
Software Tokens

- A certificate loaded onto the device and stored securely may serve this purpose
- Authentication resources generate a single-use PIN and send the soft token code through email or other formats which then shows up on user’s device for the user to authorize it.



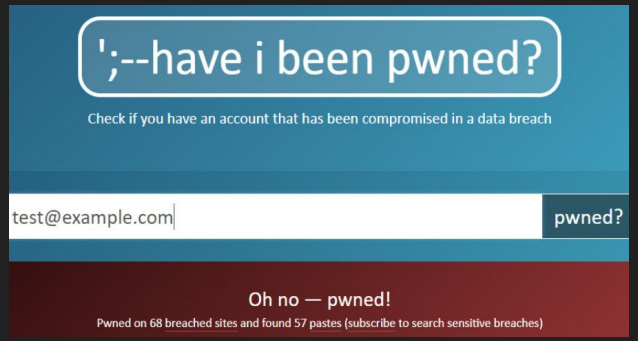
Single-Factor Authentication

- Most common form is the use of a password to authenticate an identity determined by the username
- Vulnerable to password leaks, password cracking



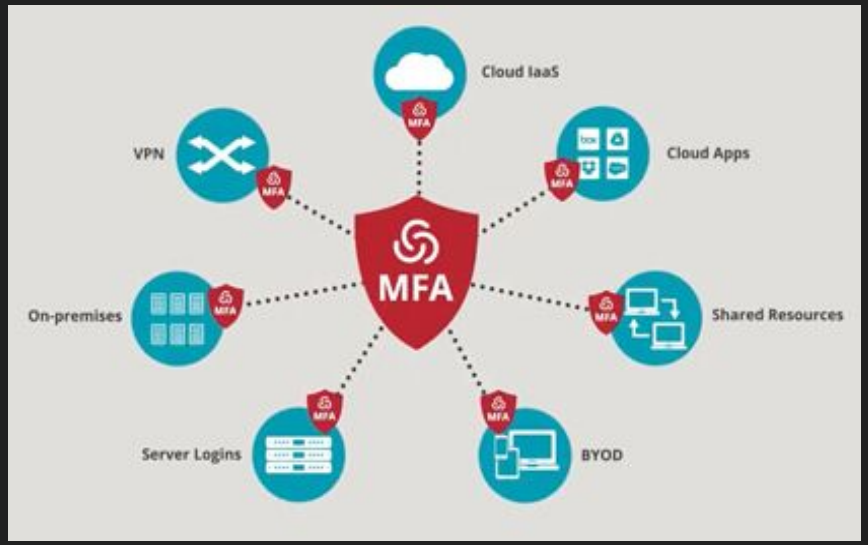
Single-Factor Authentication

- Using two or more elements all under the same factor is still single-factor authentication
- Ex. Requiring a password along with date of birth is single-factor authentication



Multi-Factor Authentication

- An authentication method where a user is granted access only after successfully presenting two or more different factors
- Two-factor authentication (2FA) is an example or subset of multi factor authentication



Two-Factor Authentication (2FA)

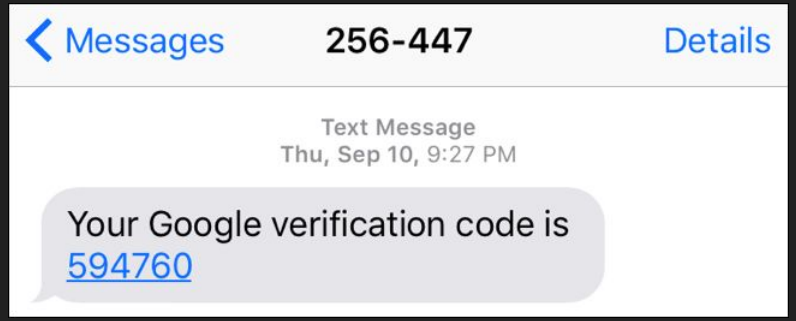
- Most common form is a combination of knowledge and possession factors
- Phones are widely used as the possession factor

Two-Factor Authentication (2FA)

- Ex. Debit card and PIN, password and Google Authenticator code
- Two-factor authentication mechanisms for online accounts include SMS, push-based, Time-based One-time Password

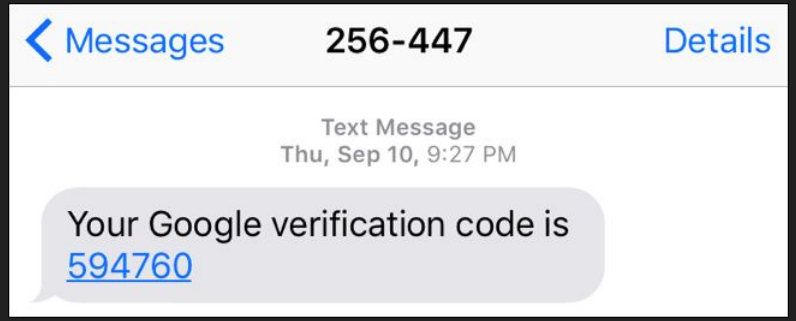
SMS Two-Factor Authentication

- Sends a short one-time code through SMS that needs to be input at some point during the login process
- Considered to be one of the most vulnerable forms of 2FA
- Vulnerable if lock-screen notifications are allowed



SMS Two-Factor Authentication

- SIM card can be taken from phone to view SMS 2FA messages elsewhere
- SS7 protocol vulnerability allows for SMS message interception
- Attacker tricks phone company to transfer phone number to new SIM card



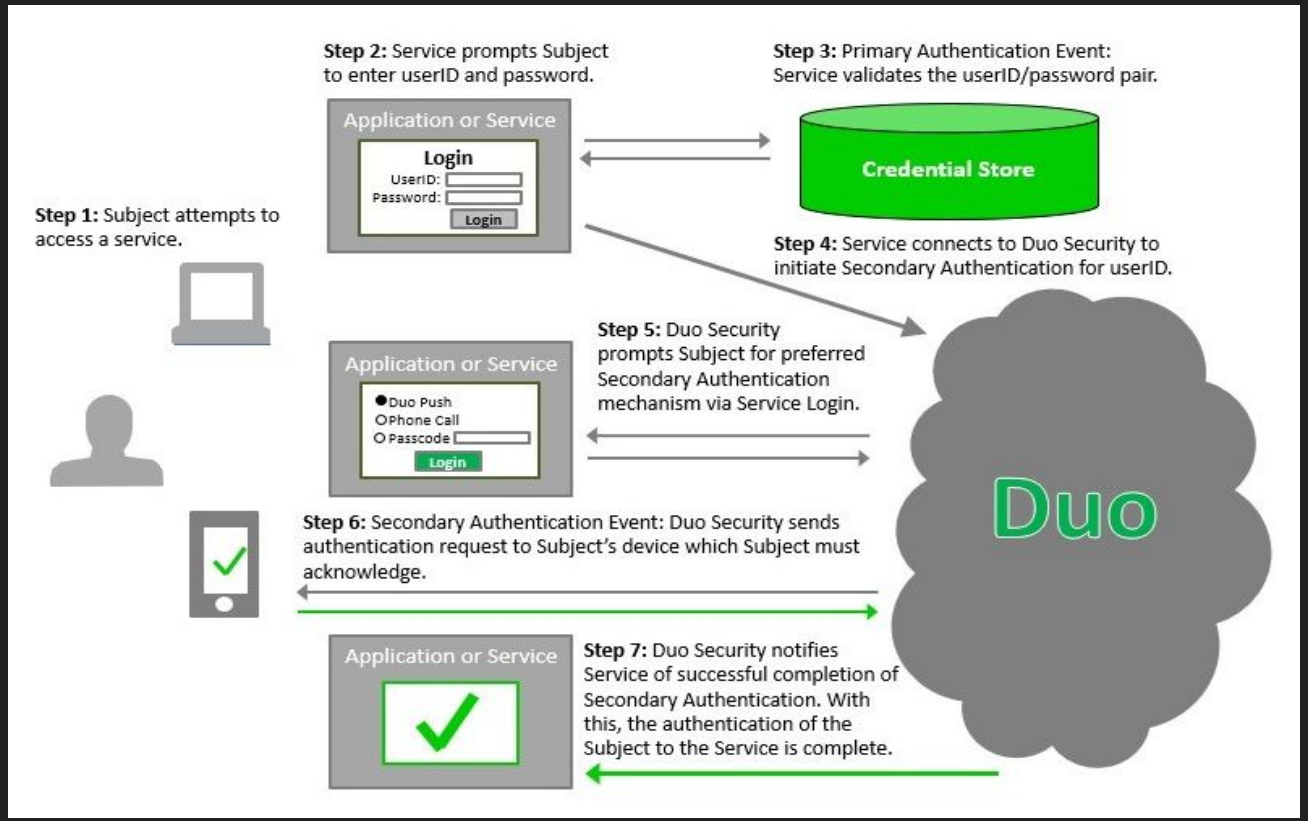
Push-based Two-Factor Authentication

- Whenever a user logs into an online account, an access request is sent to the associated phone and a push notification appears
- Requires a connection to the internet

Push-based Two-Factor Authentication

- Requests encrypted using asymmetric key pairs (private key stored on phone, public key on server)
- Server uses public key to verify signature from the phone's response

Push-based Two-Factor Authentication



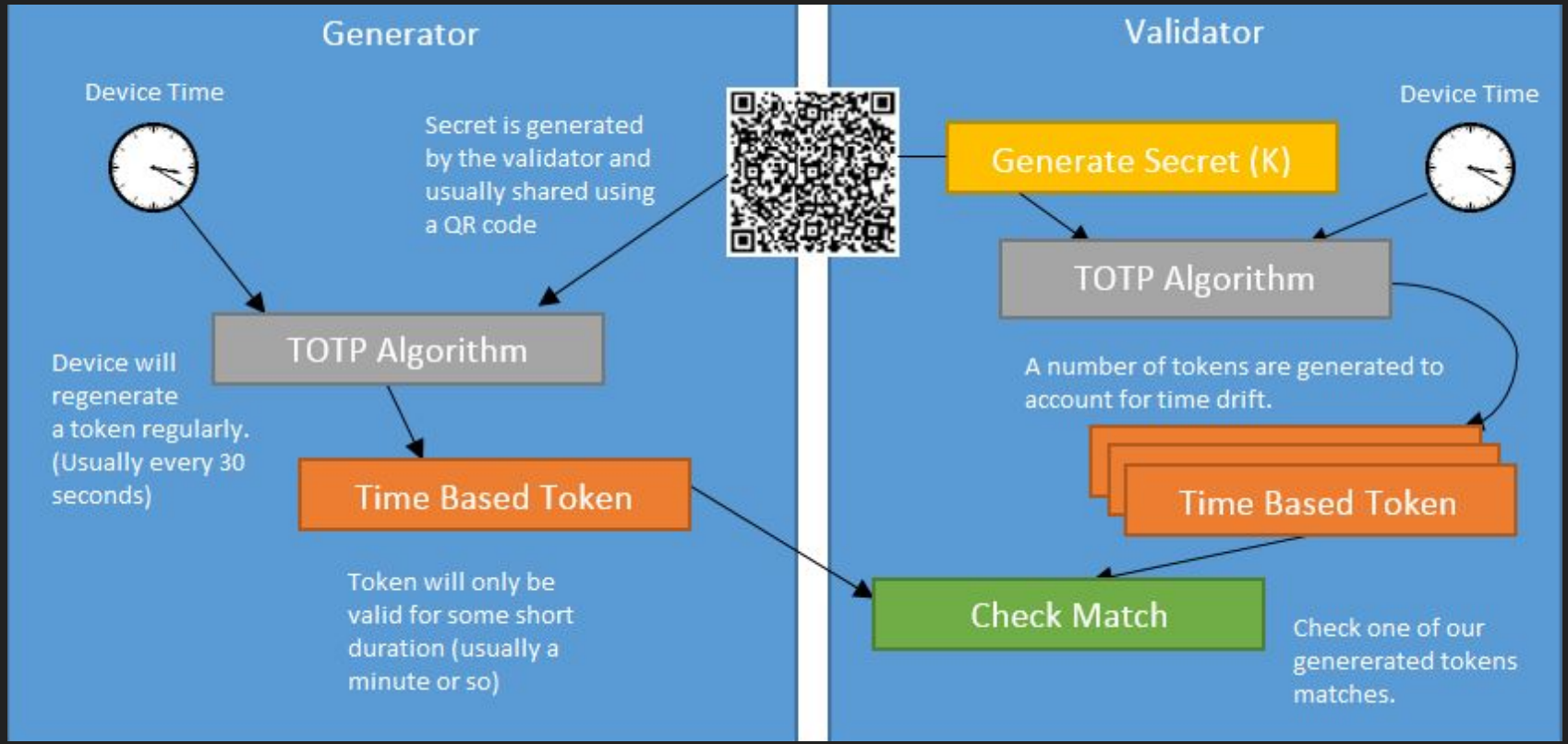
Time-based One-Time Password (TOTP) 2FA

- A secret key is shared between the phone application and server (typically with a QR code)
- Algorithm creates a one-time password based on the key and current time at regular intervals

Time-based One-Time Password (TOTP) 2FA

- Both the phone and server generate passwords that are synchronized by time
- Derived from the HMAC-based One-time Password (HOTP) algorithm

Time-based One-Time Password (TOTP) 2FA



TOTP and the HMAC-based OTP Algorithms (HOTP)

- HMAC: Hash-based Message Authentication Code, generated through using a hash function on a secret key and an arbitrary value
- hash(key, value)
- Hash function HMAC-SHA1 is often used for this purpose

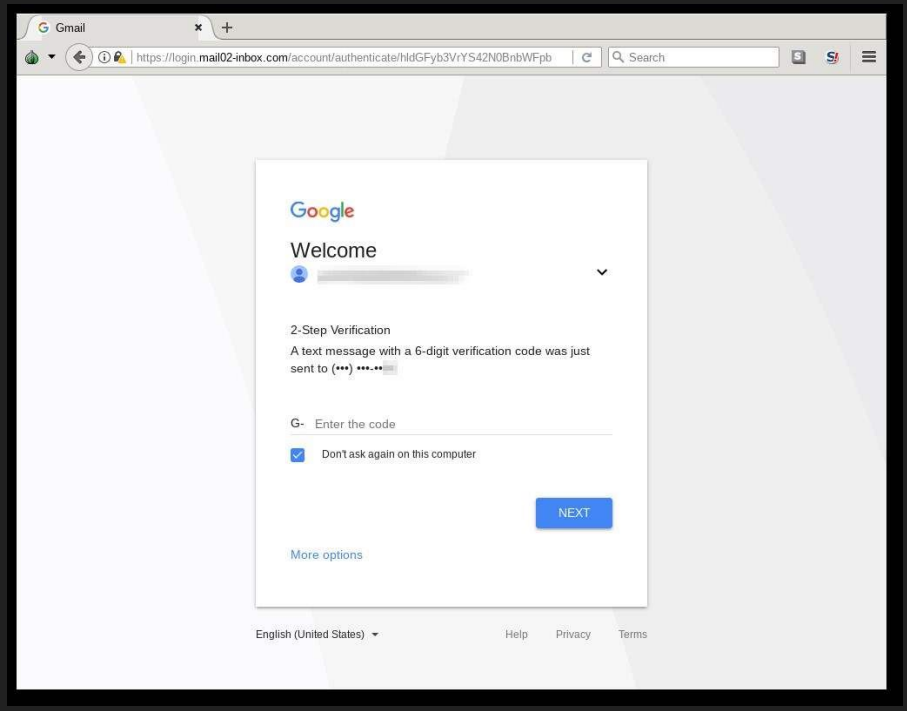
TOTP and the HMAC-based OTP Algorithms (HOTP)

- HOTP algorithm first generates an HMAC with a “counter” as the value, then truncates it (see picture below for truncate algorithm, DT) to generate a more user-friendly one-time password
- truncate(hash(key, counter))
- Take P, convert to decimal, take at least the last 6 digits of this number
- TOTP algorithm replaces the “counter” in HOTP with the current Unix time

```
DT(String) // String = String[0]...String[19]
 Let OffsetBits be the low-order 4 bits of String[19]
 Offset = StToNum(OffsetBits) // 0 <= OffSet <= 15
 Let P = String[OffSet]...String[OffSet+3]
 Return the Last 31 bits of P
```

Two-Factor Authentication Vulnerabilities

- Tokens that act as the possession factor can still be phished or stolen through social engineering
- Ex. Phishing site pretends to be Google login, asks for credentials and Google Authenticator token, automate submission



Two-Factor Authentication Vulnerabilities

- Compromised device (backdoor, spyware, etc.)
- Physically stealing the device holding the token

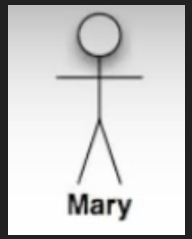
Authorization with OAuth 2.0 (Open Authorization)

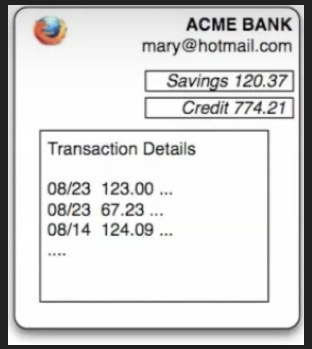
- Open standard for delegated authorization
- Designed to work in tandem with HTTP
- Basic purpose is to allow for utilization of a user's account information through a third-party
- Does not expose the user's credentials and limits the information given to third-party

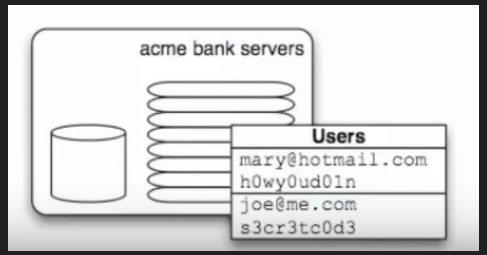
A Brief History of OAuth

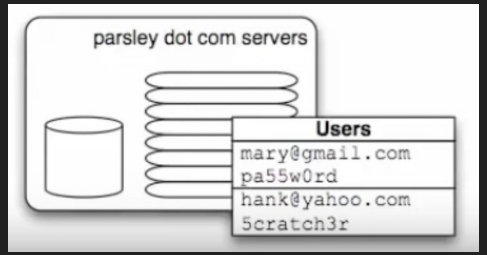
- First released as OAuth 1.0 in 2007
- At first, conceived as an authentication method for the Twitter application program interface
- OAuth 2.0 became widely popular and is now used by lots of third-party applications
- SInce then, multiple new features for OAuth 2.0 have been implemented, such as new flows, simplified signatures and short lived tokens with long lived authorizations

Example Scenario Regarding Authorization

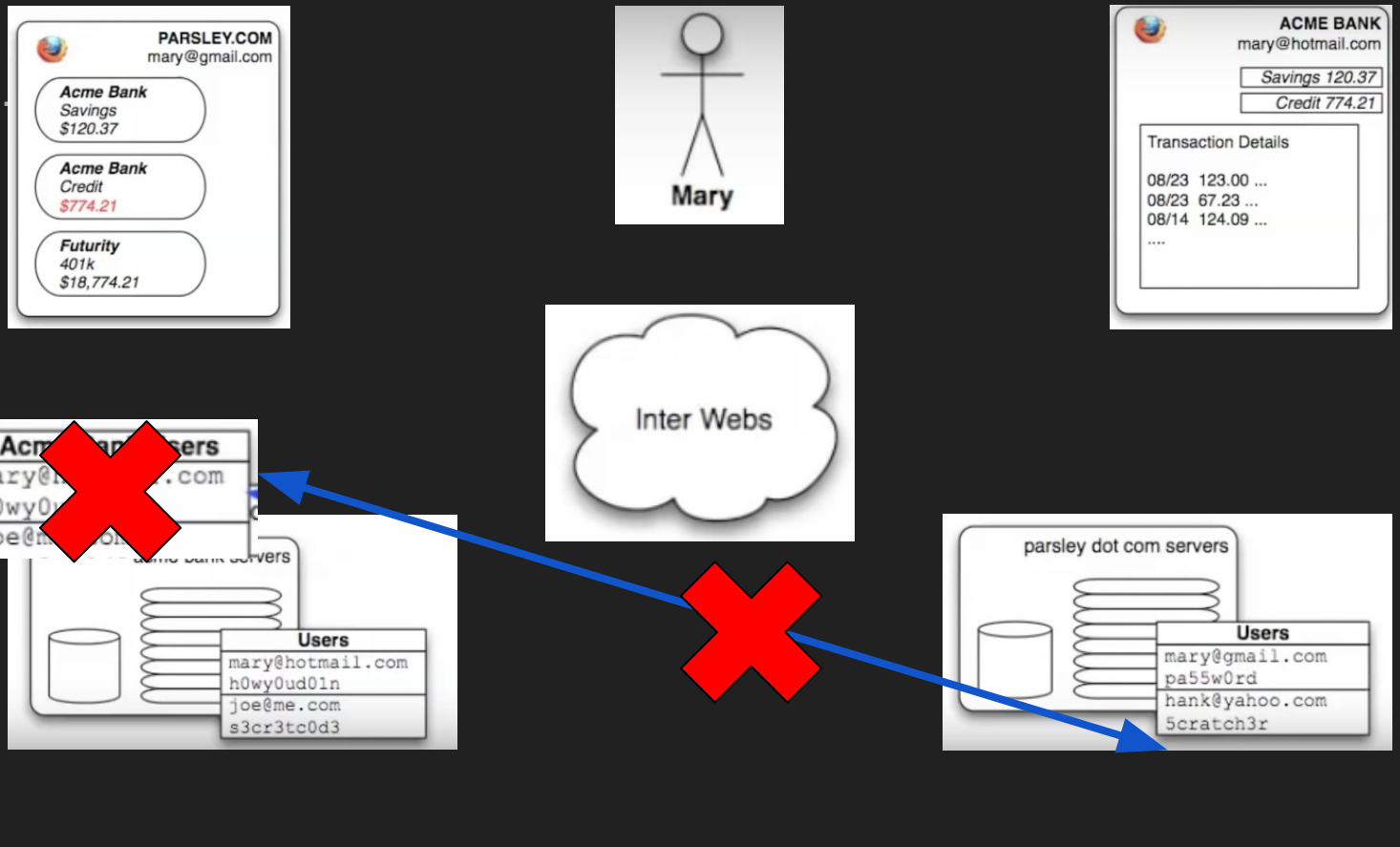




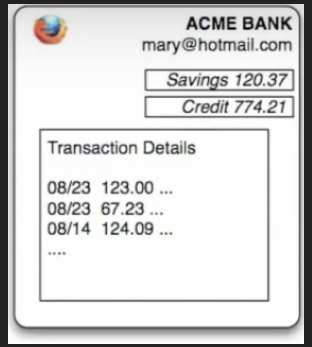


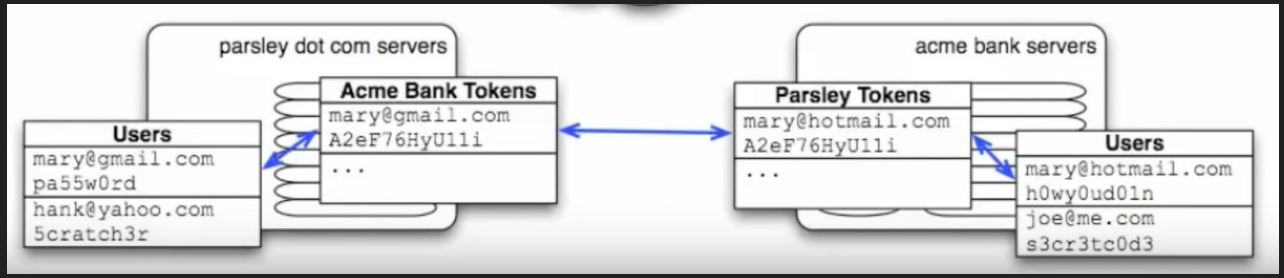


Example Scenario Regarding Authorization



Example Scenario Regarding Authorization

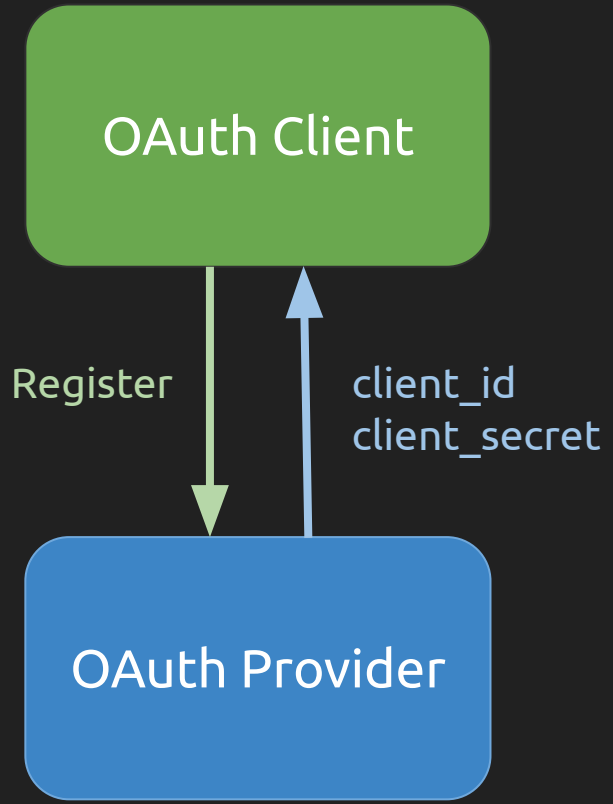




How Does OAuth 2.0 Work?

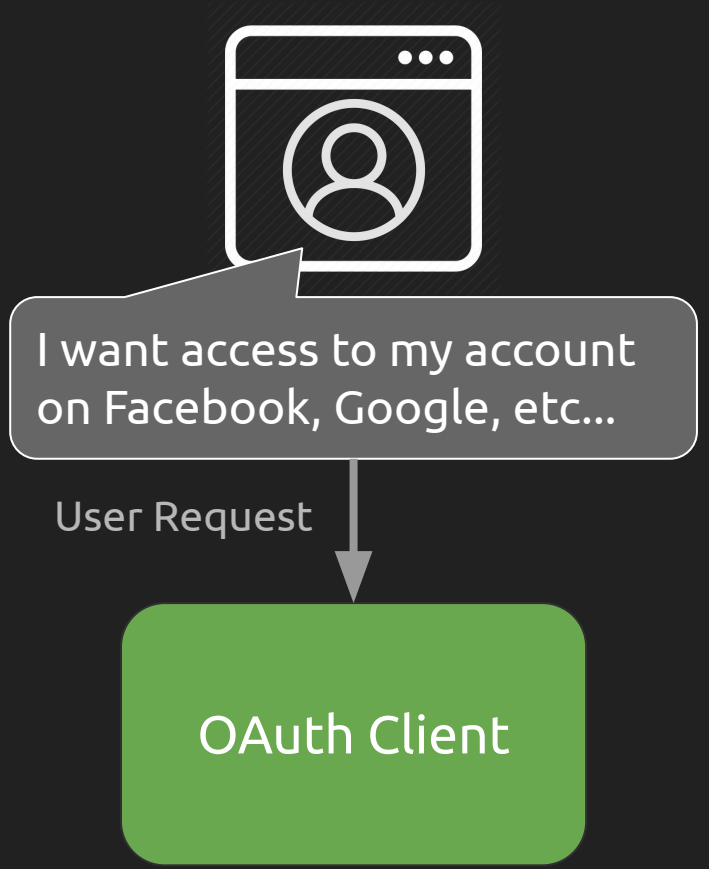
1. Developer manually registers his/her website as an OAuth client on the OAuth provider's website. Client receives the following for identification:

- client_id (publicly known)
- client_secret (only shared between client and provider)



How Does OAuth 2.0 Work?

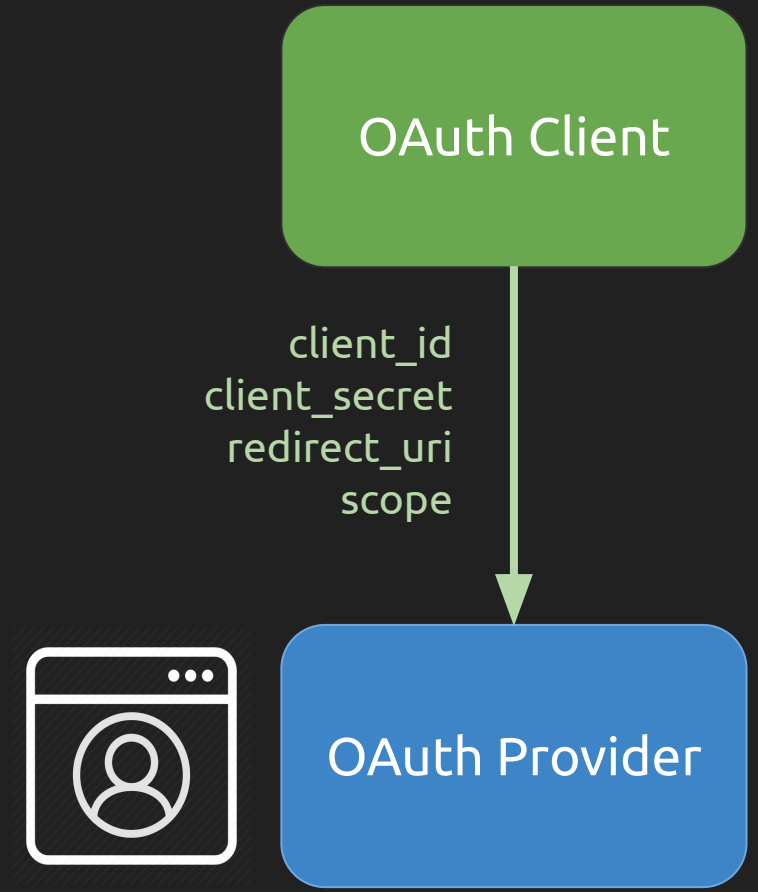
2. User's browser is on the client's website. User makes a request to the client side through the browser for delegated access to a provider's resources.



How Does OAuth 2.0 Work?

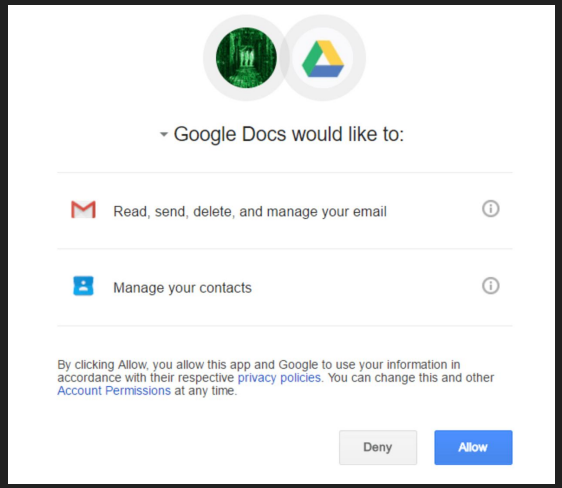
2. OAuth client specifies user's request in the scope parameter. User's browser is redirected by the client to provider's website. Client sends to provider:

- client_id
- client_secret
- redirect_uri (to return to client website)
- scope (specifies what is being requested).

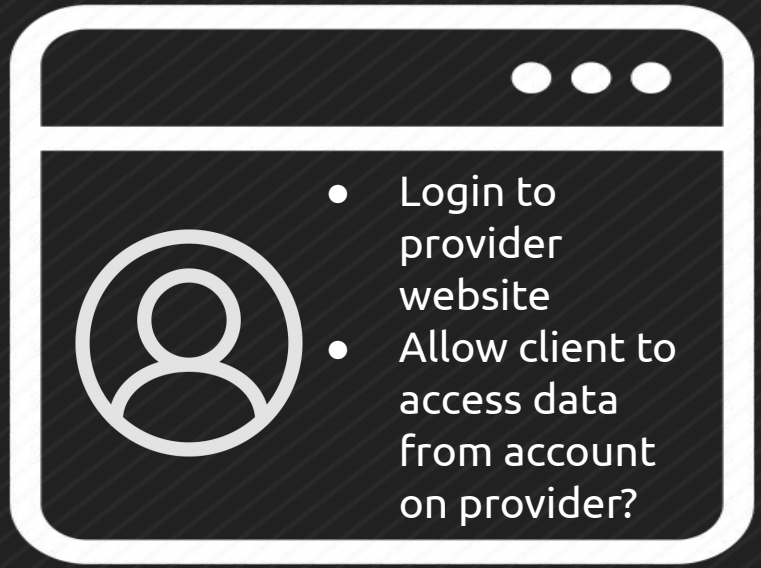


How Does OAuth 2.0 Work?

3. On the provider's website, the user is asked to log in and confirm that the client is allowed access to the user's account info from the provider.

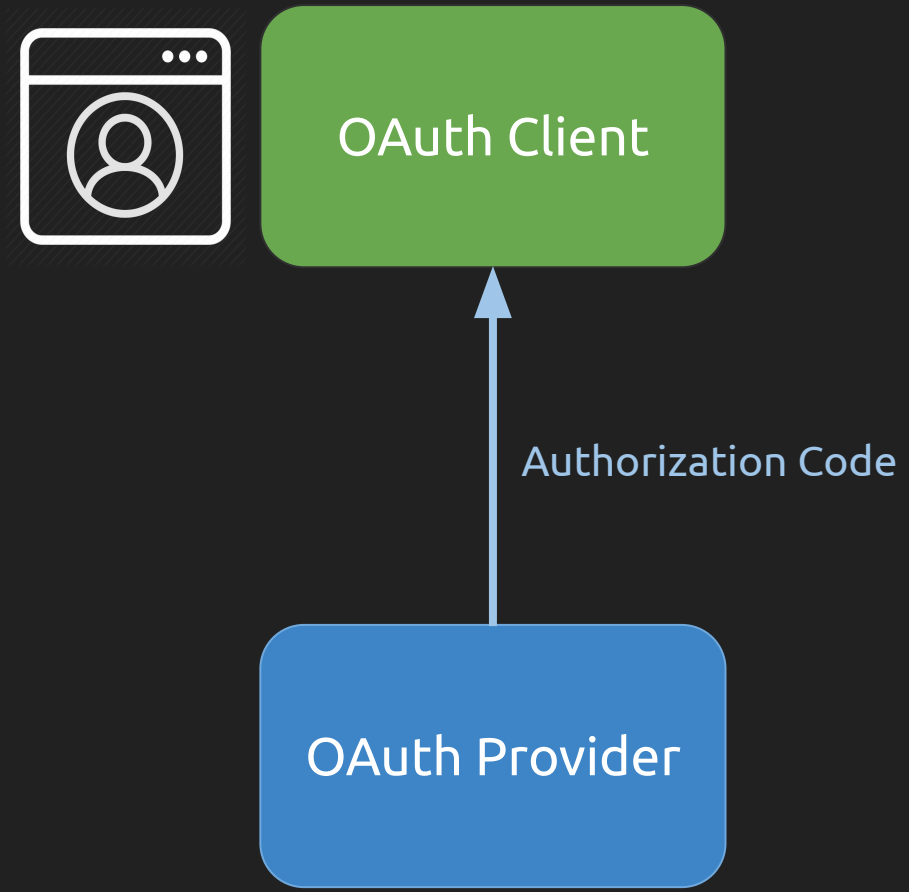


Provider's Website



How Does OAuth 2.0 Work?

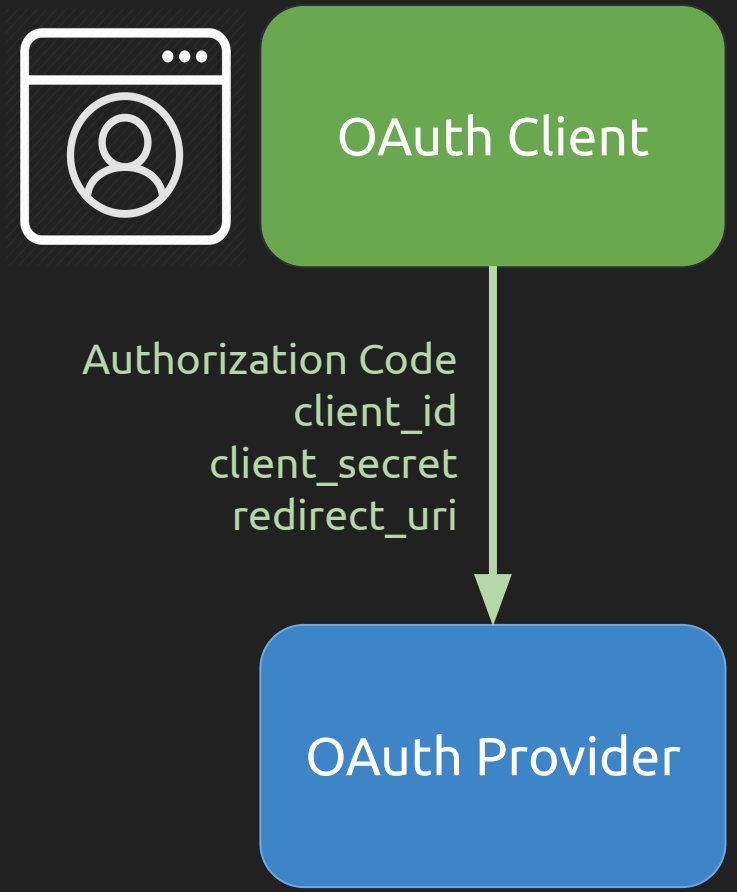
4. After user allows or denies access, the user's browser is sent back to redirect_uri (usually back to the client's website) by the provider. If permission is granted, an authorization code is sent back from provider to client.



How Does OAuth 2.0 Work?

5. Client requests an access_token and a refresh_token from provider by sending the following:

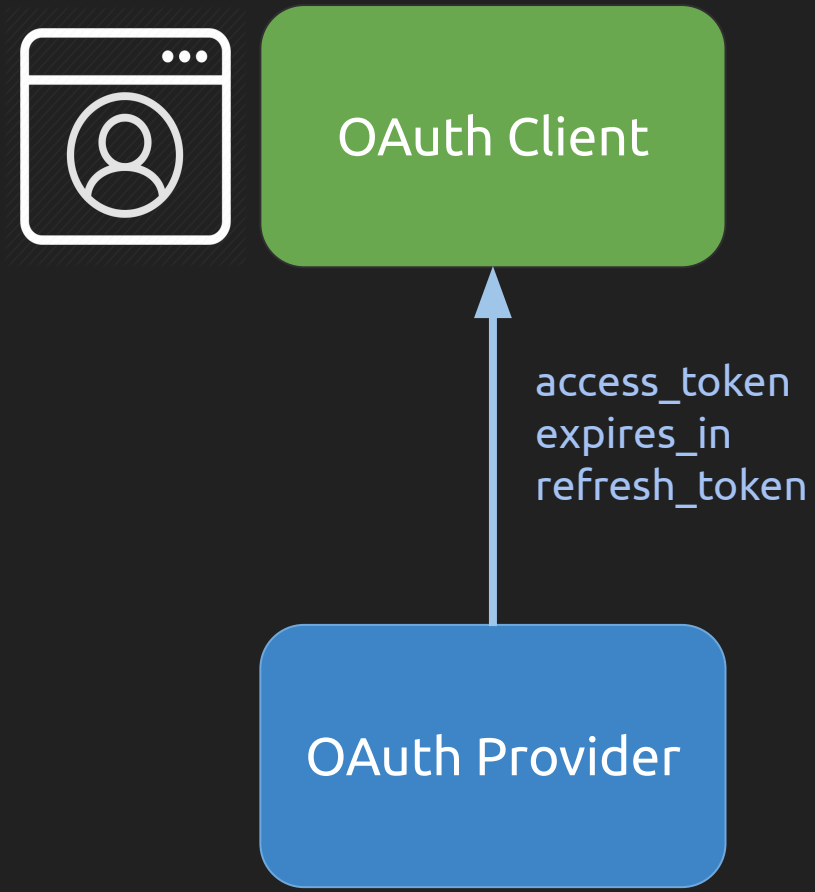
- authorization code (string)
- client_id
- client_secret
- redirect_uri



How Does OAuth 2.0 Work?

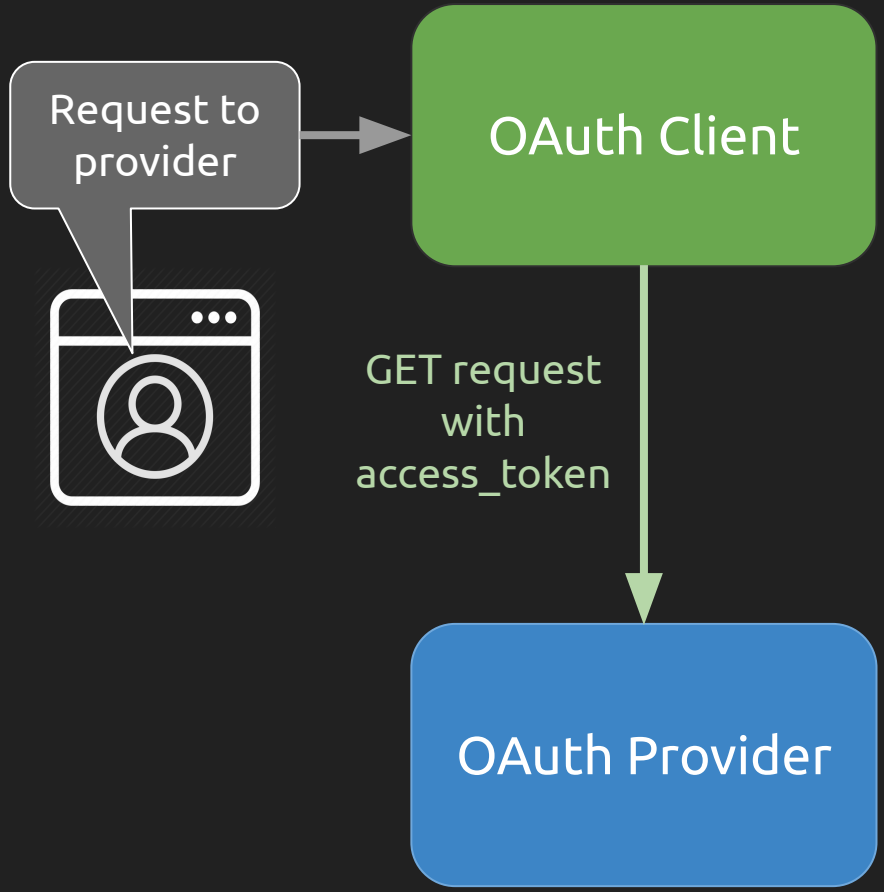
6. Provider sends the following to client through the redirect_uri:

- access_token
- expires_in (time that access_token is valid for)
- refresh_token (used to gain a new access_token)



How Does OAuth 2.0 Work?

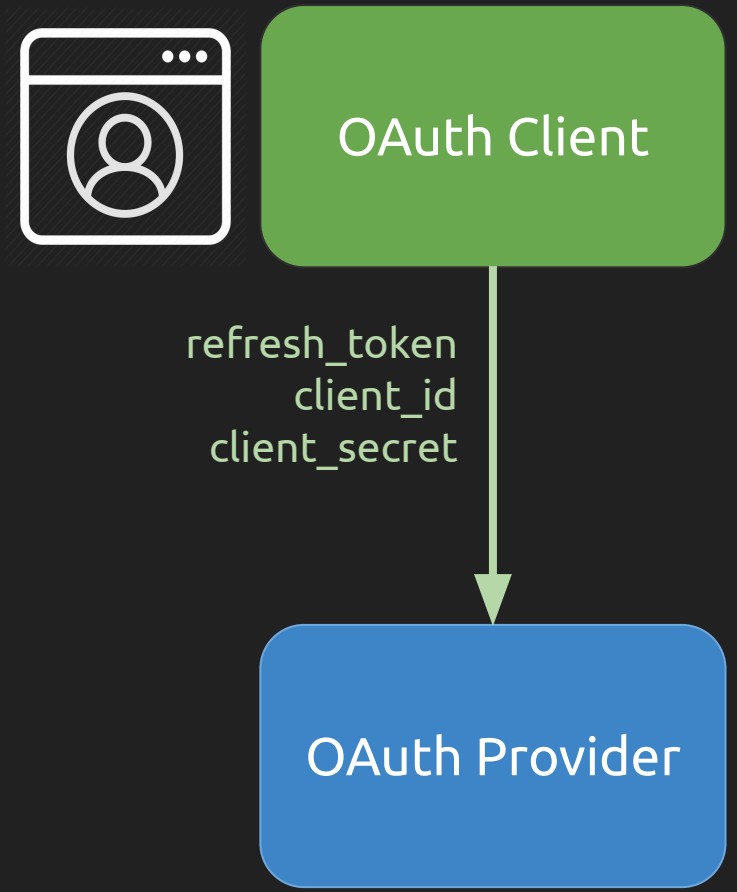
7. User on client website can now make requests to access data from provider. Client is authorized to get the data from provider using the access_token.



How Does OAuth 2.0 Work?

8. After access_token expires, the client can send the following to obtain a new access_token:

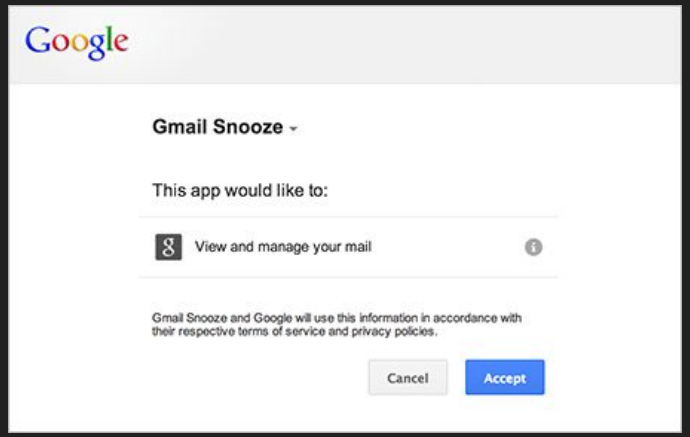
- refresh_token
- client_id
- client_secret



OAuth 2.0 Demo

OAuth 2.0 Vulnerabilities

- Users can be tricked into providing malicious third-party APIs with permission to access account info through phishing
- Ex. In 2017, State of Minnesota lost around $90,000 from users agreeing to share Gmail account information using OAuth 2.0 with the malicious “Google Apps” service



OAuth 2.0 Vulnerabilities

- Open redirect vulnerability
- Redirect is allowed to any URI specified in parameters, no validation
- Attacker can intercept and change the redirect_uri parameter to point to a malicious destination

OAuth 2.0 Vulnerabilities

- OAuth server sends responses, which can contain authentication codes and access_tokens, to the malicious destination
- Solution is to validate URIs in parameters by testing them against a list of allowed URIs

Sources

- https://blogs.getcertifiedgetahead.com/identification-authentication-authorization/
- https://medium.com/@renansdias/the-5-factors-of-authentication-bcb79d354c13
- https://blog.centrify.com/sfa-mfa-difference/
- https://venturebeat.com/2017/09/24/a-guide-to-common-types-of-two-factor-authentication/
- https://www.nick-horne.com/2017/03/10/time-based-one-time-password-algorithm-explained-totp/
- https://help.duo.com/s/article/3252?language=en_US
- https://rosettacode.org/wiki/Time-based_One-time_Password_Algorithm
- https://tools.ietf.org/html/rfc4226#section-5
- https://medium.freecodecamp.org/how-time-based-one-time-passwords-work-and-why-you-should-use-them-in-your-app-fdd2b9ed43c3
- https://www.cbtnuggets.com/blog/2016/02/part-1-authentication-for-the-modern-web/?_ga=2.51939629.522477182.1554136843-1936675294.1554136843
- https://www.youtube.com/watch?v=tFYrq3d54Dc
- https://mashable.com/article/hackers-beat-two-factor-authentication-2fa-phishing/#odwo6HsIPOqG
- https://en.wikipedia.org/wiki/Authentication
- https://en.wikipedia.org/wiki/OAuth
- https://www.cbtnuggets.com/blog/2016/03/part-3-lets-play-oauth-2-playground/?_ga=2.113748611.522477182.1554136843-1936675294.1554136843
- https://web.archive.org/web/20170508194157/http://www.bbc.co.uk/news/technology-39845545
- https://oauth.net/advisories/2014-1-covert-redirect/
- https://tools.ietf.org/html/rfc6819#section-4.1.5