CPSC 436C
Cloud Computing for Data Science

Cloud Security – Part 2

Maryam R.Aliabadi

mraiyata@cs.ubc.ca

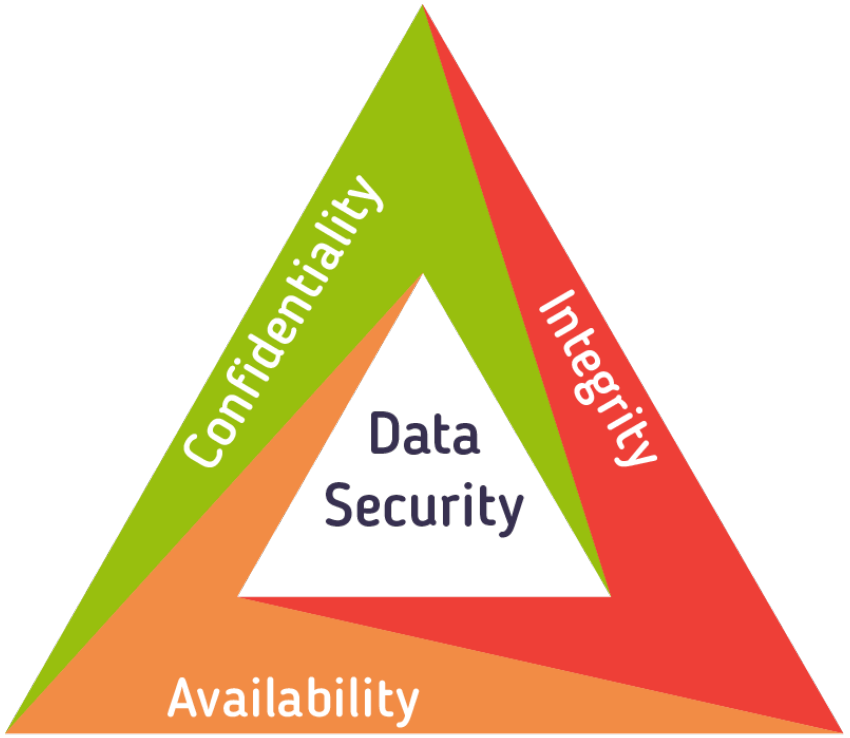
Spring 2024

Last Class‘s Review

- Guest speaker debrief
- Cloud security challenges
- Cloud attack surface
- Cloud top vulnerabilities

Data: The Main Asset In The Cloud

- Data is the most important component of the data center and Cloud
- Data is unique for the organization: everything else we saw above (building, software, hardware) can be replaced, rebought, rebuilt except data.
- Data should be protected from not only the risk of loss, but also from the risk of unauthorized access.



Traditional systems security Vs. Cloud Computing Security

Securing a house

Biggest user concerns
Securing perimeter
Checking for intruders
Securing assets

Securing a motel

Biggest user concern
Securing room against (the bad guy in next room | hotel owner)

Challenges for the attacker

How to find out **where** the target is located

How to be **co-located** with the target in the same (physical) machine

How to gather information about the target

Challenges for the Attacker

- Finding the victim's location
- Being the victim's co-resident
- Exploiting co-residence

Challenges for the Attacker

- Finding the victim's location
  - IP Address Geolocation
  - Network Tracing and Latency
- Being the victim's co-resident
- Exploiting co-residence

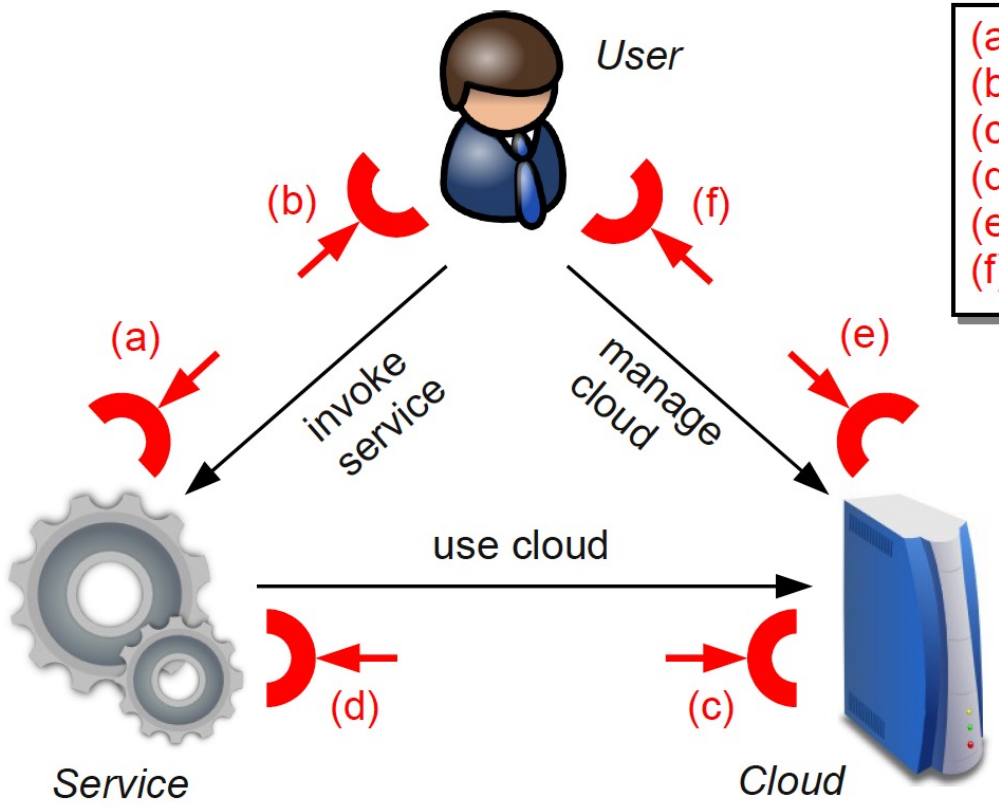
Challenges for the Attacker

- Finding the victim's location
  - IP Address Geolocation
  - Network Tracing and Latency
- Being the victim's co-resident
  - Choosing the same Region
  - Brute-force placement (launch many instances over a relatively long period of time)
  - Leveraging placement locality
- Exploiting co-residence

Challenges for the Attacker

- Finding the victim's location
  - IP Address Geolocation
  - Network Tracing and Latency
- Being the victim's co-resident
  - Choosing the same Region
  - Brute-force placement (launch many instances over a relatively long period of time)
  - Leveraging placement locality
- Exploiting co-residence
  - Cross-VM attacks can allow for information leakage
  - Side/Covert Channel attack such as
    - Time required to access a file or resource
    - Power consumption data of a computational process
    - Measuring cache usage

Analyzing Attack Surfaces in Clouds



(a) service-to-user
(b) user-to-service
(c) cloud-to-service
(d) service-to-cloud
(e) cloud-to-user
(f) user-to-cloud

Cloud attack surfaces can be modeled using a 3 entity model (user, service, cloud)

Figure from: Gruschka et al., Attack Surfaces: A Taxonomy for Attacks on Cloud Services.

OWASP Top Ten In The Cloud

R1. Accountability & Data Ownership (Loss of control)
R2. User Identity Federation
R3. Legal & Regulatory Compliance
R4. Business Continuity & Resiliency
R5. User Privacy & Secondary Usage of Data
R6. Service & Data Integration
R7. Multi-tenancy
R8. Incidence Analysis & Forensics
R9. Infrastructure Security
R10. Non-production Environment Exposure

Winter Term 2 UBC Instructor SEI Survey

CPSC 436C 201 - Topics in Computer Science

Students

https://go.blueja.io/nwc3zSiffkmYNTd4pci_vQ

https://go.blueja.io/nwc3zSiffkmYNTd4pci_vQ

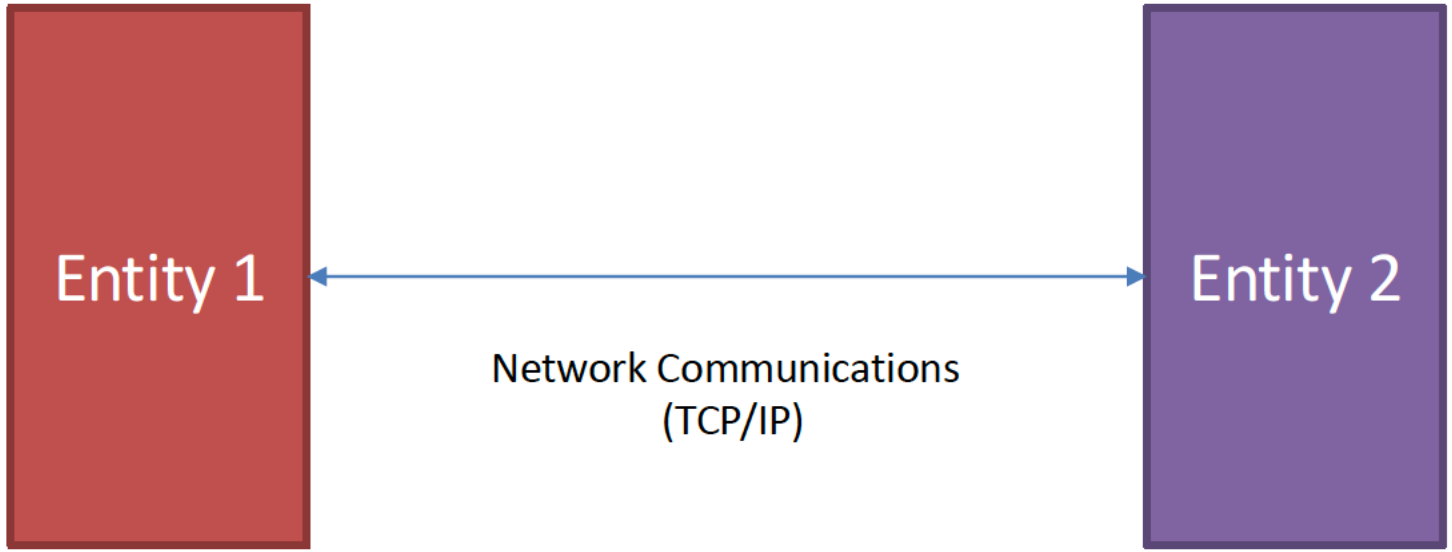
Cloud Platforms Comparison Survey

- https://docs.google.com/forms/d/1v9wPW_M3Hz3TvgpeRFAqkXjorZom3M2aUl7XOAROvtc/edit

Today's Topic:

OAuth and OpenID Connect

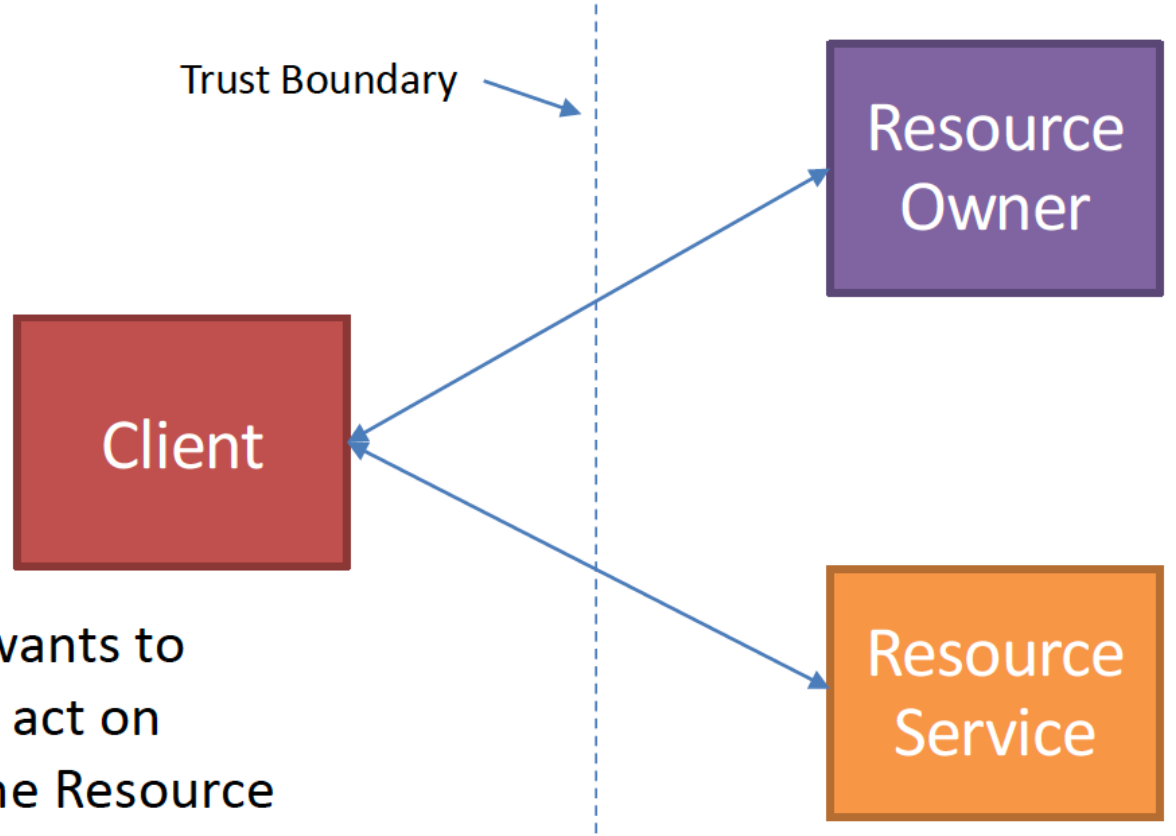
Entities on a Network



Some Basic Network Security Concepts

Security Concept	Description
Identity	Entities have unique identities.
Authentication (AuthN)	Entities can establish and prove their identities. Commonly implemented with public-private key pairs
Authorization (AuthZ)	How an entity responds to a request from another entity. Usually coupled with authentication.
Message Signing	Entities can verify that messages came from a particular authenticated entity. Implemented with cryptographic keys
Message Integrity	Detecting if the network message between entities has been altered. Implemented with message digests (hashes).
Message Privacy	Communications between entities can only be read by those entities. Implemented with encryption, shared secret keys
Message Singularity	Each message between entities is unique. Avoids accidental or malicious replays. Uses nonces, timestamps, etc.

The Authorization Problem



The **Resource Owner** wants to authorize the **Client** to act on **Resource Service** on the Resource Owner's behalf. How do you do delegate this authority?

Authorization and 3rd Party Services

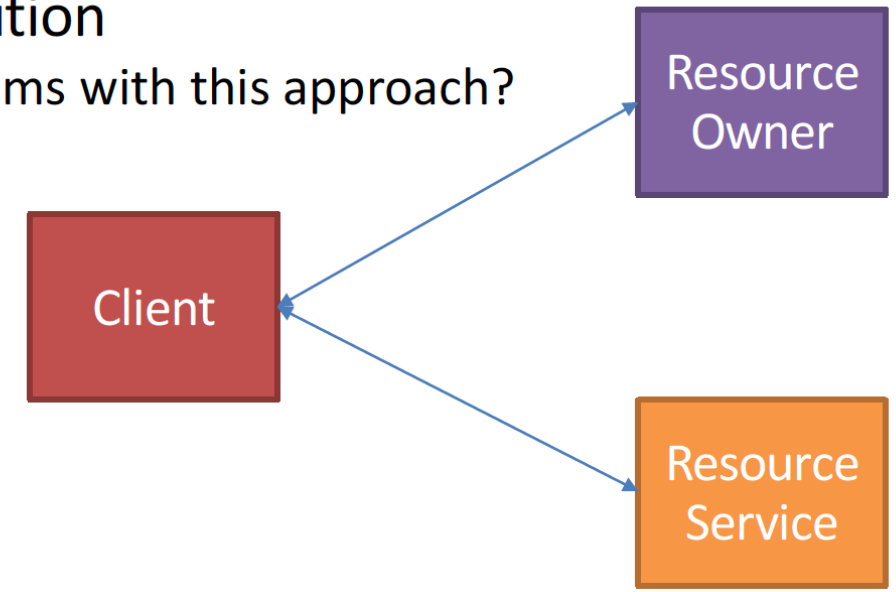
- This scenario has become very common driven by social networking, PaaS and SaaS, and mobile devices.
- Platforms and devices such as Facebook, Google, and Apple hold your personal data. Third party applications need to access some of this data.
- You decide which applications to authorize
  - "Facebook, it is ok for this application to access the names of my Facebook friends and other personal information."
  - "IPhone, it is OK for this app to know my location"

I am the Resource Owner.
My list of friends, personal information, and location are accessible through a Resource Service. Facebook and IPhone apps are Clients.

Problems Delegating Authority

- Straightforward Approach: Client requests an access-restricted resource by authenticating **using the resource owner's credentials**, like passwords
  - The Resource Owner shares its credentials with the third party Client.
  - The Client impersonates the Resource Owner.

- This is a really bad solution
  - What are some problems with this approach?



Problems with Credential Sharing

- Third-party applications gain overly broad access to the resource owner's protected resources.
  - No ability to restrict duration or access to a limited subset of resources.
- Resource owners cannot revoke access to a specific client without revoking access to all clients
  - Changing passwords.
- Compromise of the client results in compromise of the end-user's long term credentials and all of the data protected by that password.

Can you think about some solutions?

OAuth and OpenID

- OAuth and Open ID Connect are authorization and authentication protocols invented to solve different problems.

- They both use access tokens that contain scopes and claims. Usually, OAuth and Open ID Connect are applied in combination.

- OAuth solves an authorization problem, while Open ID Connect solves an authentication problem.

OAuth Example

- Download your Google contacts into LinkedIn
  - LinkedIn has a feature that imports your google contacts and invites them to connect with you through LinkedIn.
  - Back in day, LinkedIn would ask you to give them your google username and password.
  - They would use it to log in on your behalf, download your contacts and log out. How you make sure that they don't use any other information?

- OAuth was designed to solve that problem.
  - When using OAuth, LinkedIn redirects you to Google to log in
  - Google pops up a message: "is it okay to share your contact with LinkedIn?"
  - If you agree, LinkedIn is authorized to query Google for your contacts (and nothing more)

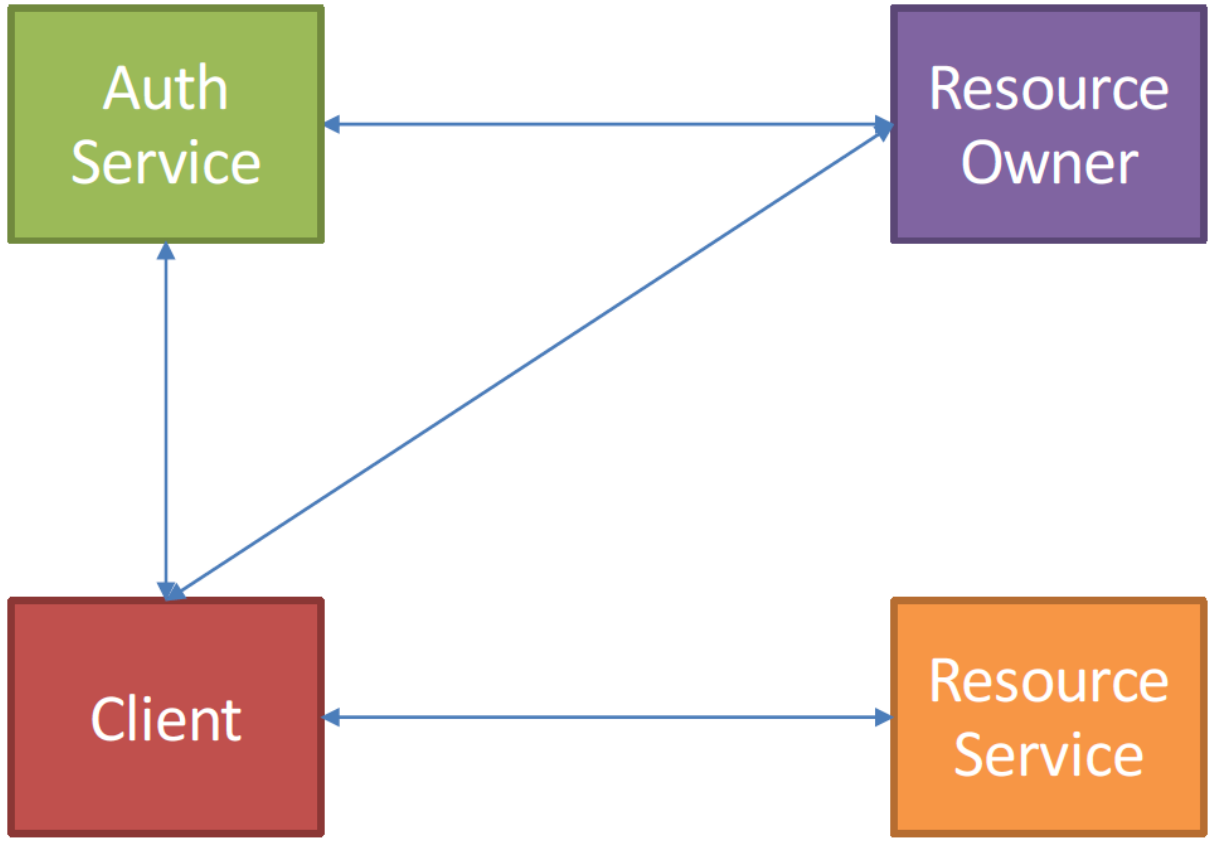
How does OAuth work?

- OAuth is an **authorization** protocol used to protect resources. by **verifying the identity** of an end-user and **granting permissions to a third party.**
- The OAuth specifications => https://tools.ietf.org/html/rfc6749
- This verification results in a token. The third party can use this token to access resources on the user's behalf.
- Tokens have a scope. The scope is used to verify whether a resource is accessible to a user, or not. An access token is like the key card you get when you check into a hotel. The receptionist verifies your identity, and based on your reservation, you receive a key card that allows you to access your room, and maybe the spa. When you present the access token to the door, it validates your credentials and allows you to enter.

OAuth 2.0

OAuth2 solves this problem by introducing a **mutually trusted*** Authorization Service

*There are rigorous ways, like key exchanges, for establishing mutual trust.



OAuth2 Main Concepts

- OAuth2 introduces an authorization layer
  - Separates the role of the client from that of the resource owner.
- In OAuth2, the client is issued a different set of credentials than those of the resource owner.
  - OAuth2 access tokens rather than passwords
- An OAuth2 access token has a specific scope, lifetime, and other access attributes.
- Access tokens are issued to third-party clients by an Authorization Server with the approval of the Resource Owner.
- The Client uses the access token to access the protected resources hosted by the Resource Server.

Client Registration: Trusting the Client

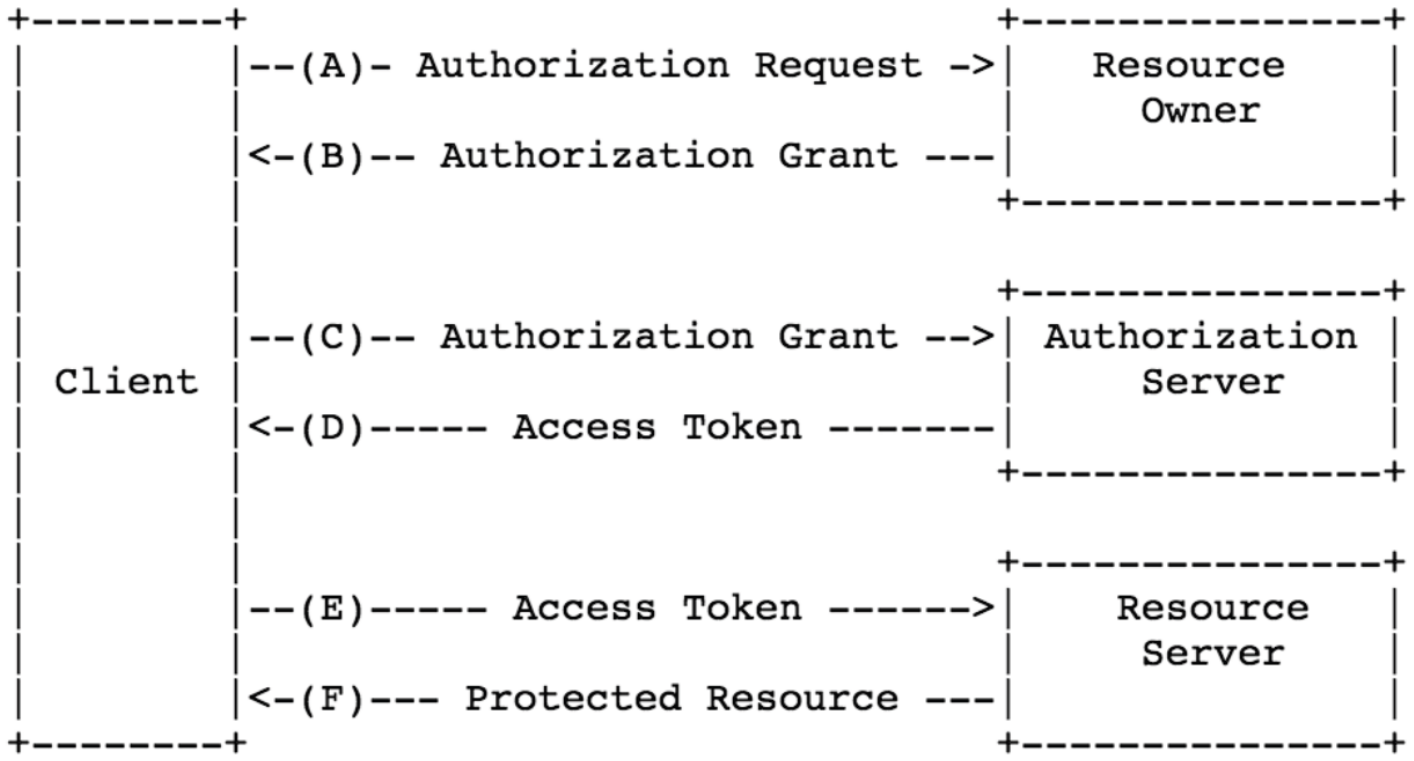
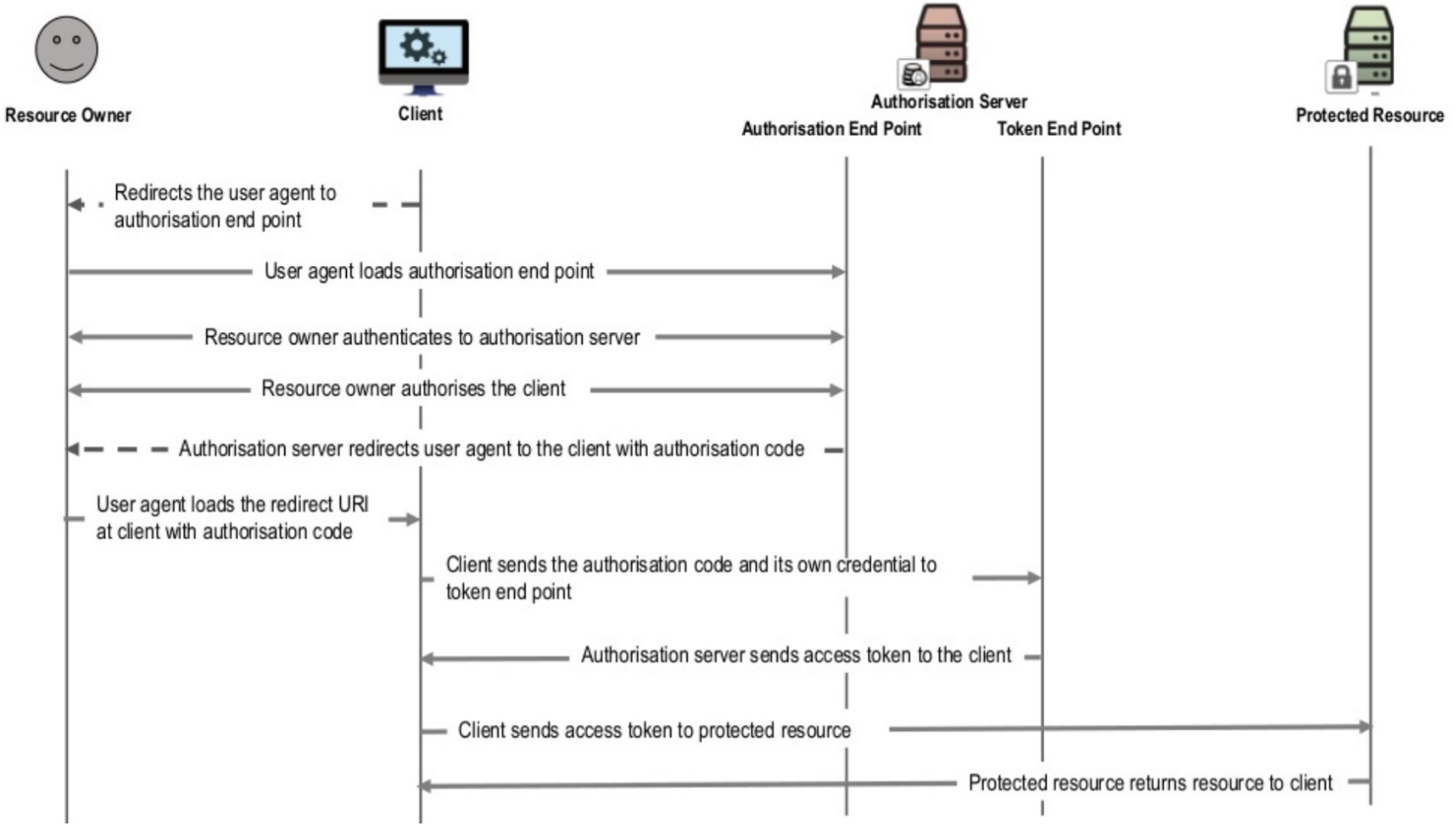


Figure 1: Abstract Protocol Flow

https://tools.ietf.org/html/rfc6749

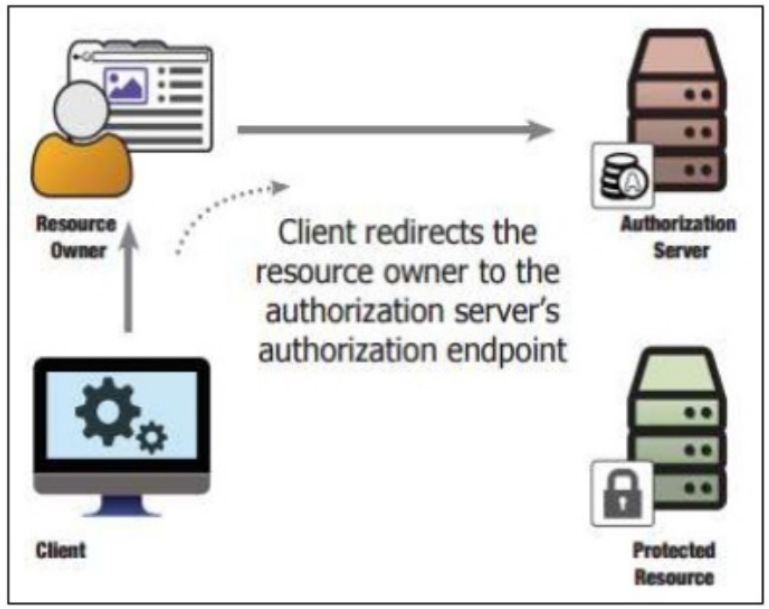
OAuth Architecture



OAuth Architecture

It starts with:

OAuth Architecture




```
HTTP/1.1 302
Content-Length: 0
Date: Sun, 30 Apr 2017 17:47:27 GMT
Location: https://unibet.okta.com/
oauth2/v1/authorize?
client_id=P9BxWcYwdNnGFmIt8Oiz&
redirect_uri=http://localhost:8080&
response_type=code&scope=openid&
state=2qdOs6PJAcE1E6qwg81R
Set-Cookie: JSESSIONID=646430;path=/;HttpOnly
```

OAuth Architecture

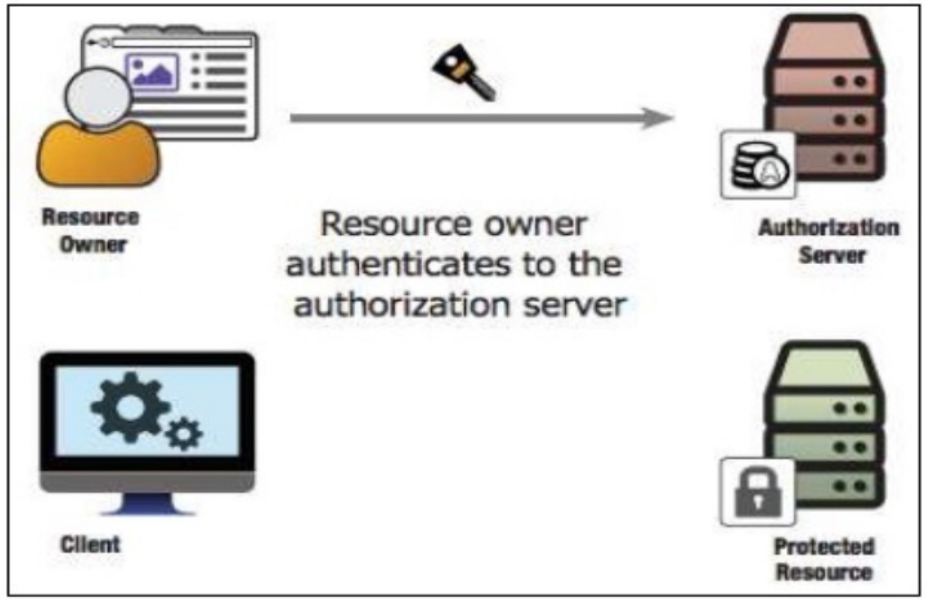
This redirect to the browser causes the browser to send an HTTP GET to the authorisation server.

```
/authorize/authorize?client_id=P9BxWcYwdNnGFmI
t8Oiz&redirect_uri=http://localhost:8080&respo
nse_type=code&scope=openid&state=2qdOs6PJAcE1E
6qwg81R
```

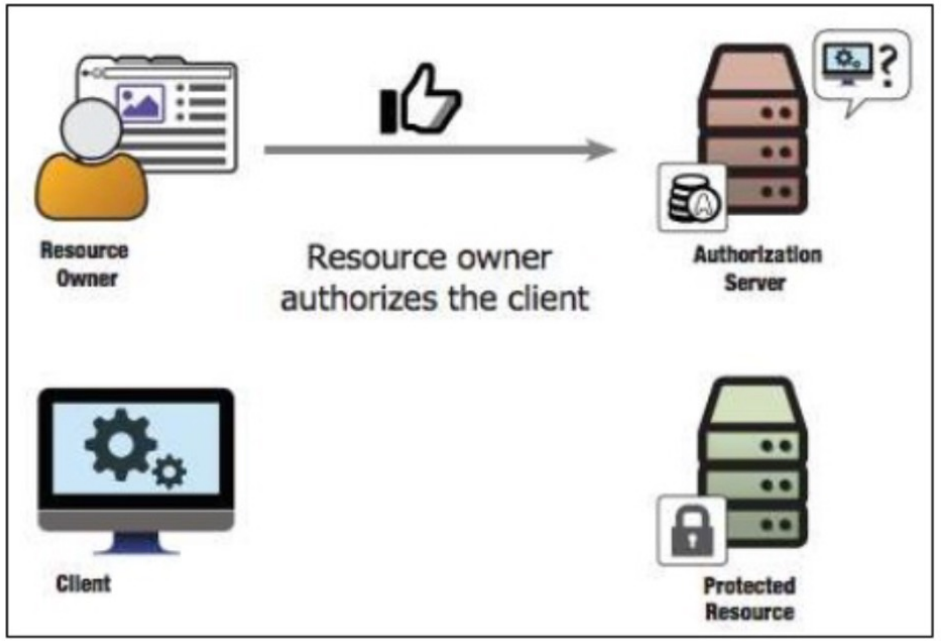
OAuth Architecture

Now the authorisation server will usually require the user to authenticate.

This step is essential in determining **who the resource owner is and what rights they're allowed to delegate to the client**.



OAuth Architecture



User (resource owner) authorizes the client application.

OAuth Architecture

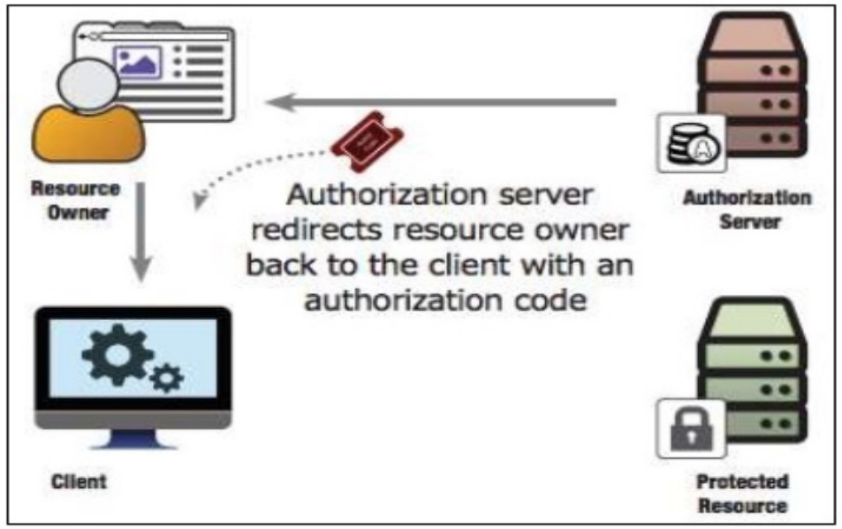
Next, the authorisation server redirects the user back to the client application.

This takes the form of an HTTP redirect to the client's **redirect_uri**.

```
http://localhost:8080/?
code=VzYHKvGXwqxBMq9qn8Pw&
state=moBRz4CFpviCE4e5h0HZ
```

Authorisation code

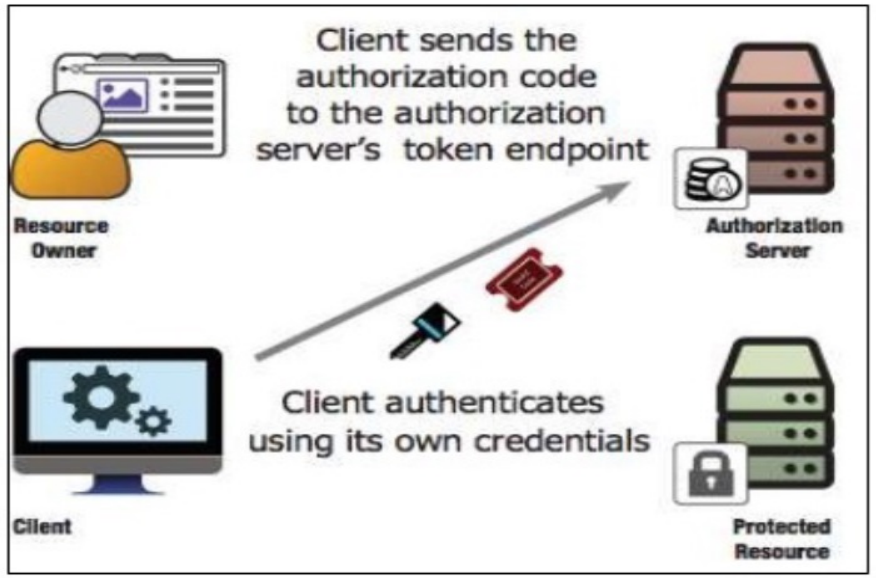
OAuth Architecture



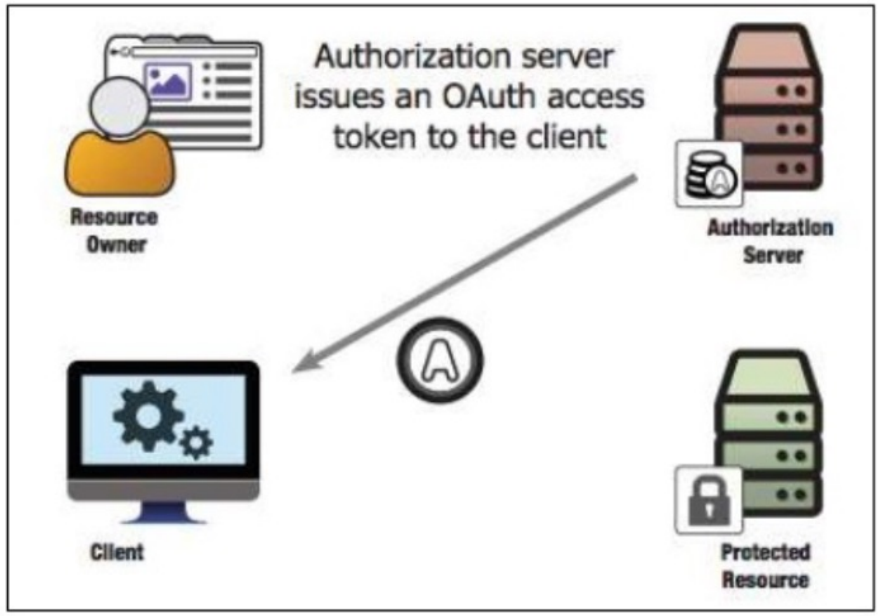
The authorisation code is sent back to the client.
This code is a one-time use credential. It represents the result of the user's authorisation decision.

OAuth Architecture

The client now sends the code and it's own credential (client id and client secret) to the authorisation server on its token end point.



OAuth Architecture



Clients receives an access token

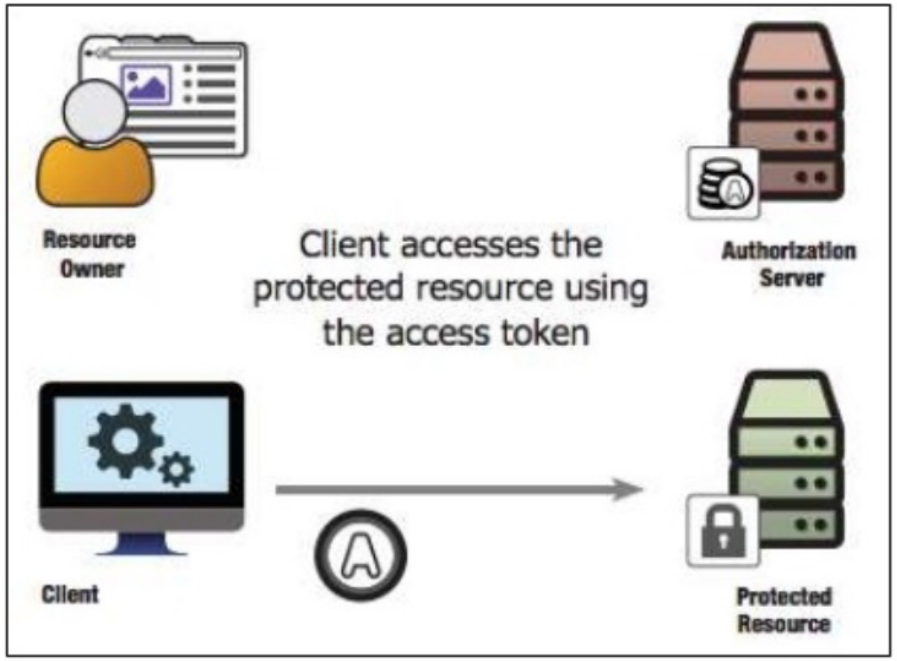
Access Token-Example

This access token is returned in the HTTP response as a JSON object:

```
{
      "access_token": "tdgsgsdfq232wASDq232a",
      "token_type": "Bearer"
}
```

The client can now parse the token response and get the access token value from it to be used at the protected resource.

OAuth Architecture



The client uses the access token to do things

OAuth's components

- Access Token
- Scope
- Refresh Token
- Authorisation grant

Access Token

- ✓ An OAuth access token is an artifact issued by the authorisation server to a client that indicates the rights that the client has been delegated.
- ✓ OAuth does not define a format or content for the token itself, but it always represents the combination of the client's requested access, the resource owner that authorised the client, and the rights conferred during that authorisation.
- ✓ OAuth tokens are opaque to the client, which means that the client has no need (and often no ability) to look at the token itself. The client's job is to carry the token, requesting it from the authorisation server and presenting it to the protected resource.

Scope

- ✓ An OAuth scope is a representation of a set of rights at a protected resource.
- ✓ Scopes are represented by strings in the OAuth protocol, and they can be combined into a set by using a space-separated list.
- ✓ The scope value can't contain the space character.
- ✓ Scopes are an important mechanism for limiting the access granted to a client. Scopes are defined by the protected resource, based on the API that it's offering.

Refresh Token

- ✓ An OAuth refresh token is similar in concept to the access token, in that it's issued to the client by the authorisation server and the client doesn't know or care what's inside the token.
- ✓ But this token is never sent to the protected resource. Instead, the client uses the refresh token to request new access tokens without involving the resource owner.
- ✓ In OAuth, an access token could stop working for a client at any point. When the access token expires, client can use the refresh token to request a new access token.

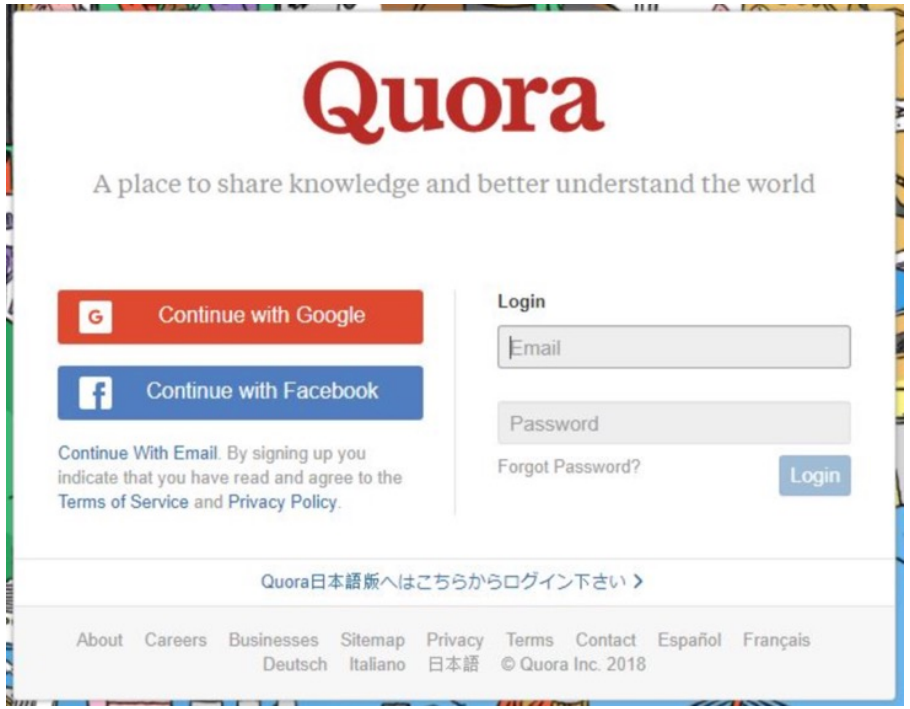
Authorization grant

- ✓ An authorisation grant is a way by which an OAuth client is given access to a protected resource using the OAuth protocol, and if successful it ultimately results in the client getting a token.
- ✓ The authorisation grant is the method for getting a token.

Activity

- Consider the use case of **Quora.com**. You are authorizing Google or Facebook to allow Quora to access your profile info. What are the following actors in this scenario:

1. Resource Owner
2. Client Application
3. Resource Server
4. Protected resource



Activity

- Consider the use case of **Quora.com**. You are authorizing Google or Facebook to allow Quora to access your profile info. What are the following actors in this scenario:

1. Resource Owner

The resource owner is the user who possesses the account and access to their profile information on Google or Facebook.

2. Client Application

Quora is the client application or service that seeks access to the user's profile information.

3. Resource Server

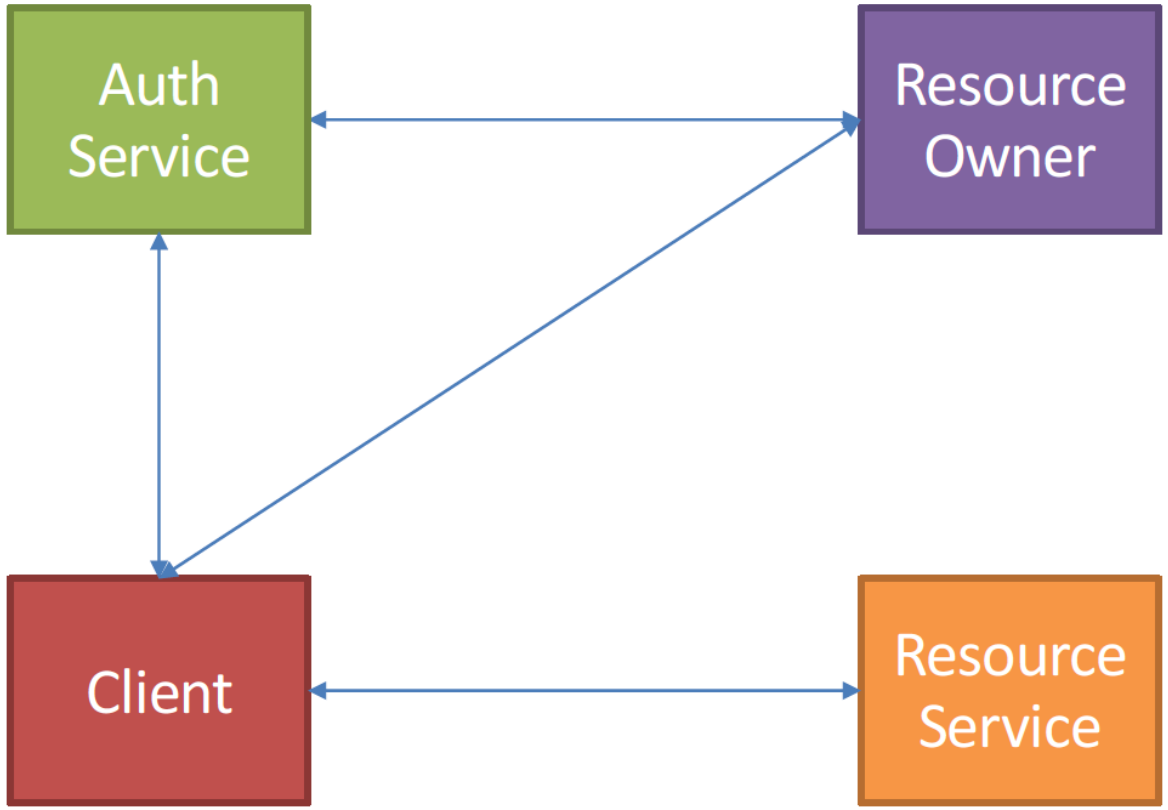
Google or Facebook acts as the resource server in this context. It hosts and protects the user's profile information.

4. Protected resource

The user's profile information stored on Google or Facebook servers is the protected user data

Assumptions in OAuth2

What are some ways to attack OAuth2? How can OAuth2 defend against these attacks?

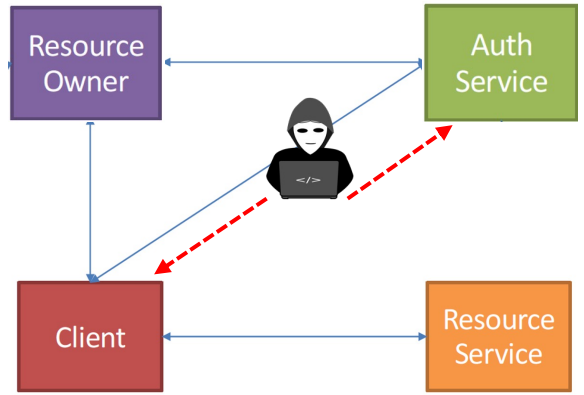


OAuth Vulnerabilities and Countermeasures

- A1- Extracting Credentials Or Tokens
- A2- Impersonating Authorization Server
- A3- Manufacturing Tokens
- A4- Client Session Hijacking

A1- Extracting Credentials Or Tokens

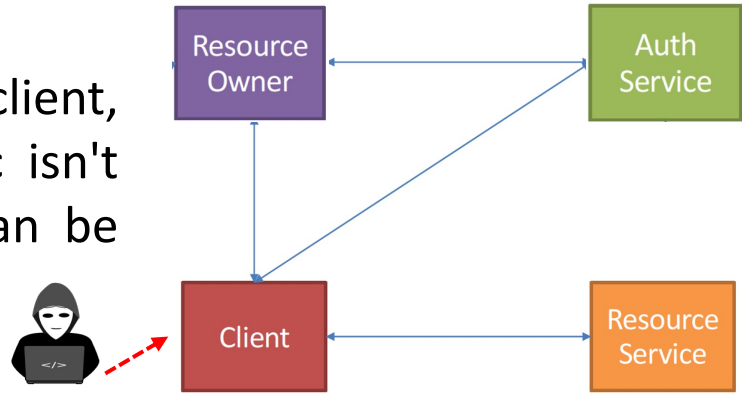
- Man-in-the-Middle (MITM) Attacks:
  - Attackers intercept communication between the client, authorization server, and resource server. If the traffic isn't encrypted (e.g., using HTTPS), tokens or credentials can be easily captured.



- How?
  - Packet Sniffing Tools like Wireshark or tcpdump can capture and analyze packets in transit, potentially exposing tokens or sensitive information if the communication isn’t encrypted.

A1- Extracting Credentials Or Tokens

- Man-in-the-Middle (MITM) Attacks:
  - Attackers intercept communication between the client, authorization server, and resource server. If the traffic isn't encrypted (e.g., using HTTPS), tokens or credentials can be easily captured.



- How?
  - Packet Sniffing Tools like Wireshark or tcpdump can capture and analyze packets in transit, potentially exposing tokens or sensitive information if the communication isn’t encrypted.

- Cross-Site Scripting (XSS):
  - If an application using OAuth is vulnerable to XSS attacks, malicious scripts injected into the application can capture tokens or credentials from the user's browser.

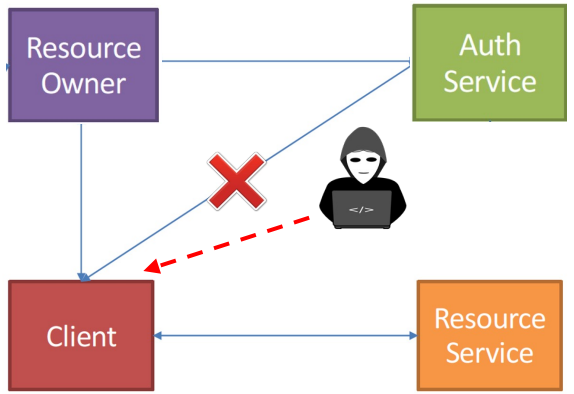
A1 Countermeasures

- Transport Layer Security (TLS):
  - Ensure communication channels are encrypted using HTTPS to protect traffic from eavesdropping.

- Secure Token Storage:
  - Employ best practices for storing tokens on the client-side, avoiding storing them in vulnerable locations like local storage or cookies susceptible to XSS attacks.

A2- Impersonating Authorization Server

- DNS Spoofing or MITM Attacks:
  - Attackers manipulate DNS records or intercept traffic to redirect users to a malicious server posing as the legitimate Authorization Server.



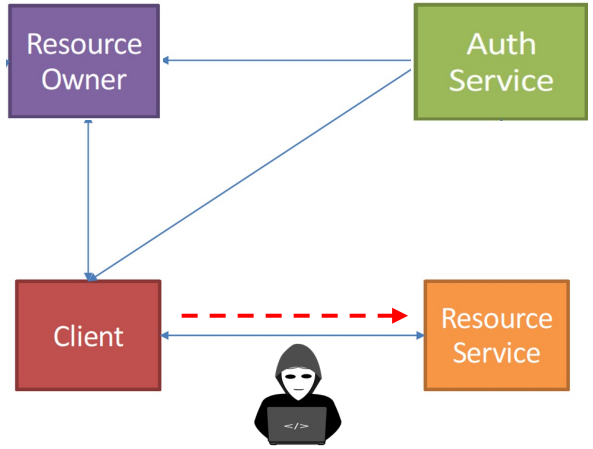
- Phishing Attacks:
  - Users might be tricked into interacting with a fake authorization server that closely resembles the legitimate one, thereby disclosing their credentials or tokens to the attacker.

A2 Countermeasure

- TLS (Transport Layer Security) Server Authentication
  - TLS Server Authentication is a fundamental security measure that ensures the authenticity of a server to the client during the establishment of a secure connection.

A3- Manufacturing Tokens

- Token Forgery:
  - Crafting or forging tokens involves creating tokens that appear legitimate to the Resource Server but are unauthorized or not issued by the legitimate Authorization Server. This can be done by tampering with token parameters or attempting to mimic valid tokens.

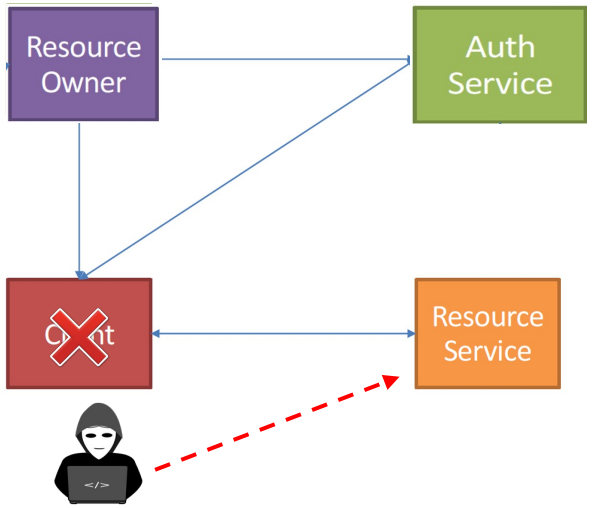


A3 Countermeasures

- Consider encrypting tokens or using techniques like signed JWT (JSON Web Token)
  - Tamper-Resistant: The signature ensures that the token hasn't been altered since it was issued. If any part of the token is modified, the signature verification will fail.

A4- Client Session Hijacking

- If the client's tokens (such as authorization codes, access tokens, or refresh tokens) are compromised, attackers could use these tokens to impersonate the client in OAuth transactions.



A4 Countermeasures

- Use of State parameters to ensure continuity of client session
  - Binding Requests and Responses: When the Authorization Server processes the authorization request, it includes the same state value in the redirect back to the client after the user grants authorization. The client compares this state value received in the callback with the one it had initially generated and stored.

OpenID Connect

An OAuth2-Based Authentication Protocol

http://openid.net/connect/

Why OpenID Connect?

- Authentication as a Service
  - Don't run your own authentication service
  - Use a trusted service instead
  - Authentication mechanisms and details handled by the service.

- Why? The trusted Identity Provider (IdP) absorbs lots of headaches
  - Best practices and implementations for securing user accounts and information.
  - Avoids the need to provide separate identity management for every application
  - Handles federated identities.
  - Handles advanced authentication mechanisms such as two-factor authentication

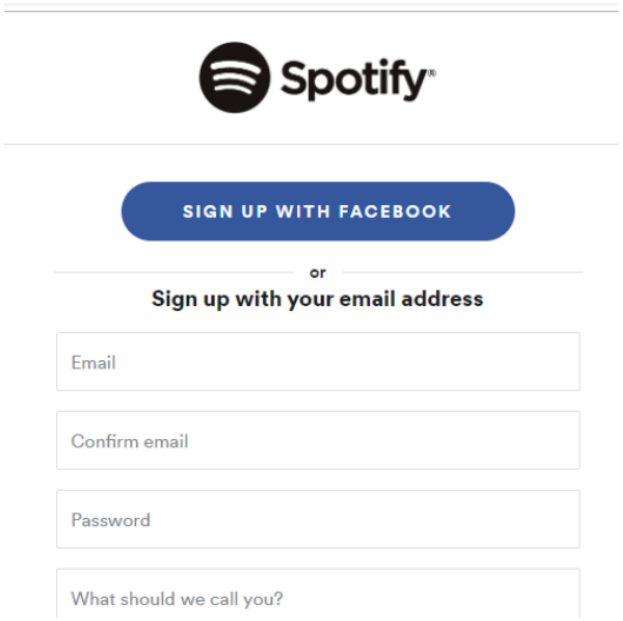
- Examples
  - Keycloak: Open source software for running your own IdP. We use this for Apache Airavata.
  - Google, Microsoft, Salesforce, Paypal, Yahoo (whoops...)

OpenID Example

- Log into Spotify using your OpenID: There are two ways of creating a Spotify account. You can choose to fill out a form and submit your e-mail, name, password, etc. Or, you can log in through Facebook.

Logging into Spotify with your Facebook account is a good example of how OpenID could be applied:
1. You log into Facebook.
2. Facebook sends your name and e-mail to Spotify.
3. Spotify uses those details to identify you.

It would be a nightmare for Facebook to track every user's permissions for every application that uses it for logging in. So OpenID communicates *identity*, not permissions. Information about the user's permissions may be added by the Spotify authorization server.



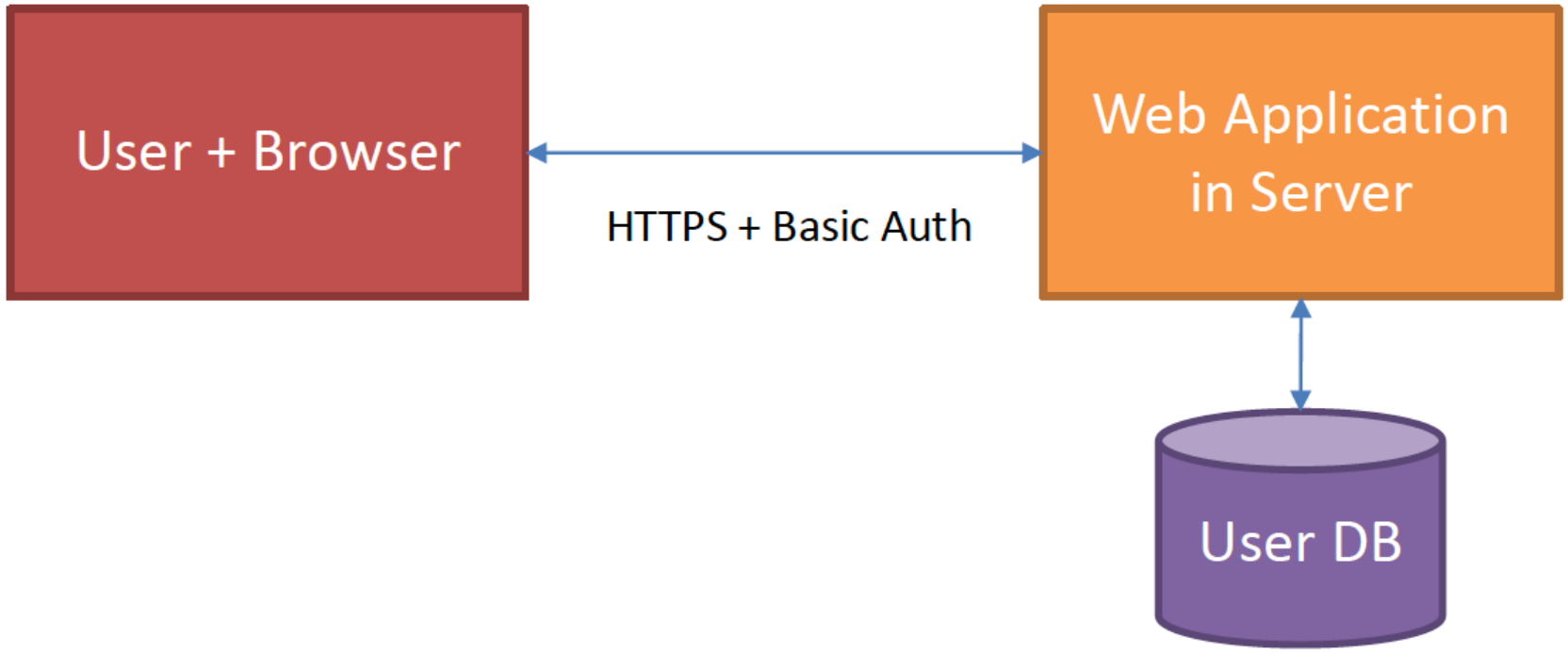
How does OpenID work?

- OpenID is a protocol used for **decentralized authentication**. *Authentication* is about **identity,** that is, establishing that the user is, in fact, the person who he or she claims to be. Decentralizing that means this service is unaware of the existence of any resources or applications that need to be protected. That's the key difference between OAuth and OpenID.

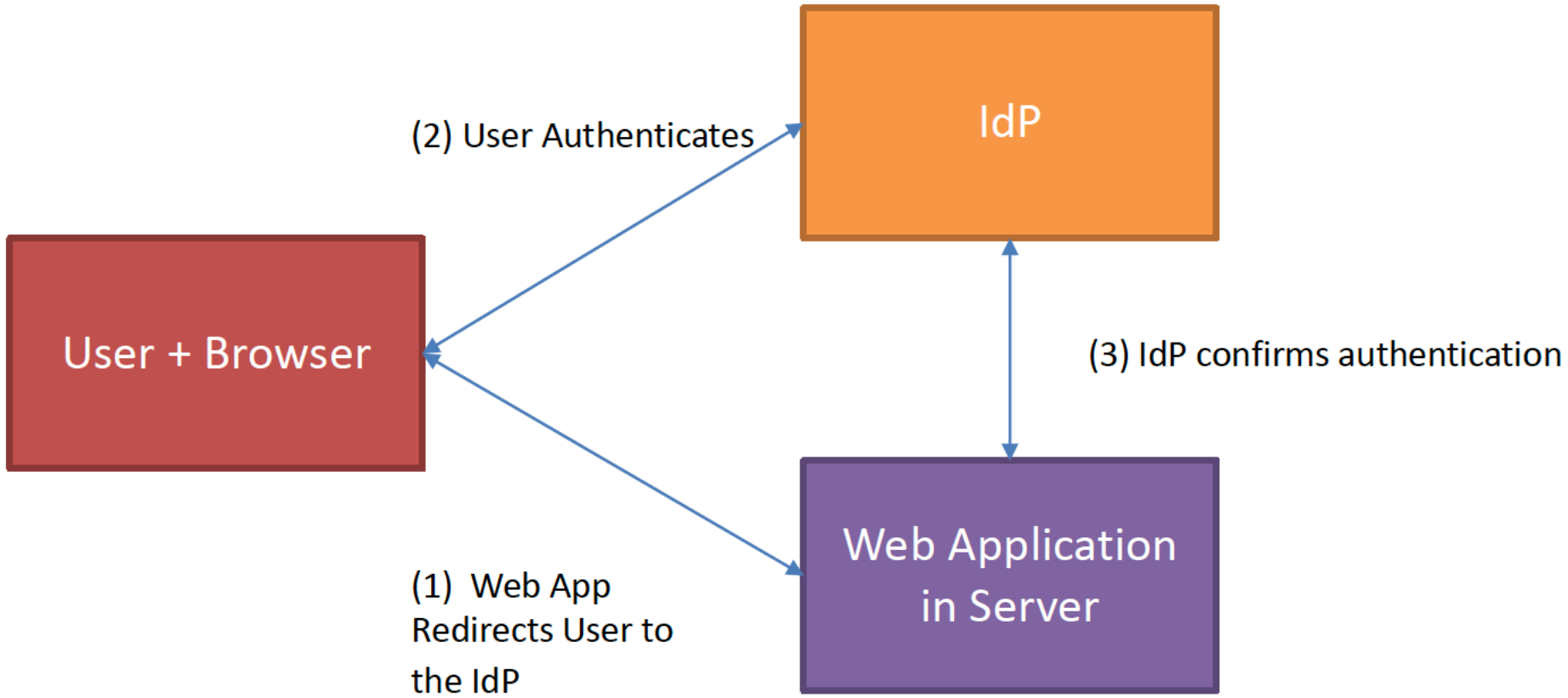
- The OpenID specifications => https://openid.net/specs/openid-connect-core-1_0.html

- Back to the hotel scenario, the receptionist asks for your passport. She looks at it and trusts the country that gave you the document. The country verified your identity. The verification resulted in a document that contains your details. You can use this ID in multiple situations. For example, to board an airplane or get into a bar. Your passport doesn't say anything about permissions. It doesn't say if he or she may, or may not fly and it's the airline who authorizes the permission.

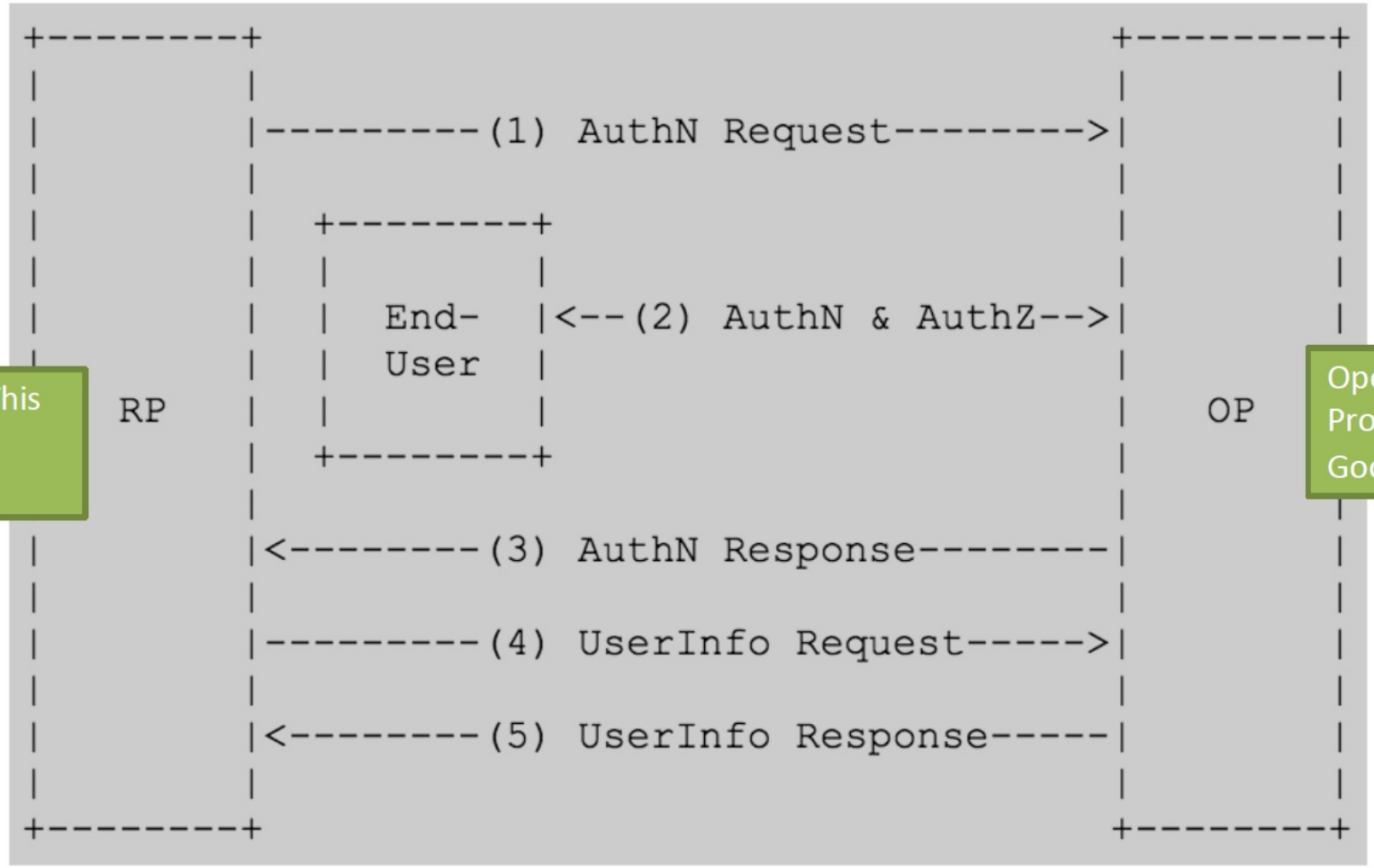
Direct Authentication



Authentication as a Service



Basic OIDC Flow



Relying Party. This is the OAuth2 Client.

OpenID Connect Provider (i.e., Google)

Basic OIDC Flow

- The Relying Party (RP) sends a request to the OpenID Provider (OP).
  - This is the science gateway
- The OP authenticates the End-User and obtains authorization.
- The OP responds with an ID Token and usually an Access Token.
  - Verifies to the client that the user authenticated correctly.
  - The ID Token is specific to OIDC and is its primary extension of OAuth2
- The RP can send a request with the Access Token to the UserInfo Endpoint.
- The UserInfo Endpoint returns Claims about the End-User.

We can make use of the returned Access Tokens for other authorization decisions.

Recap

- OAuth is an authorization protocol
- Open ID Connect is an Authentication protocol
- Combination is used identity and access management in cloud
- OAuth vulnerabilities and countermeasures

Up Next:

Final Exam
Apr 19th at 3:30pm