Mehdi Medjaoui

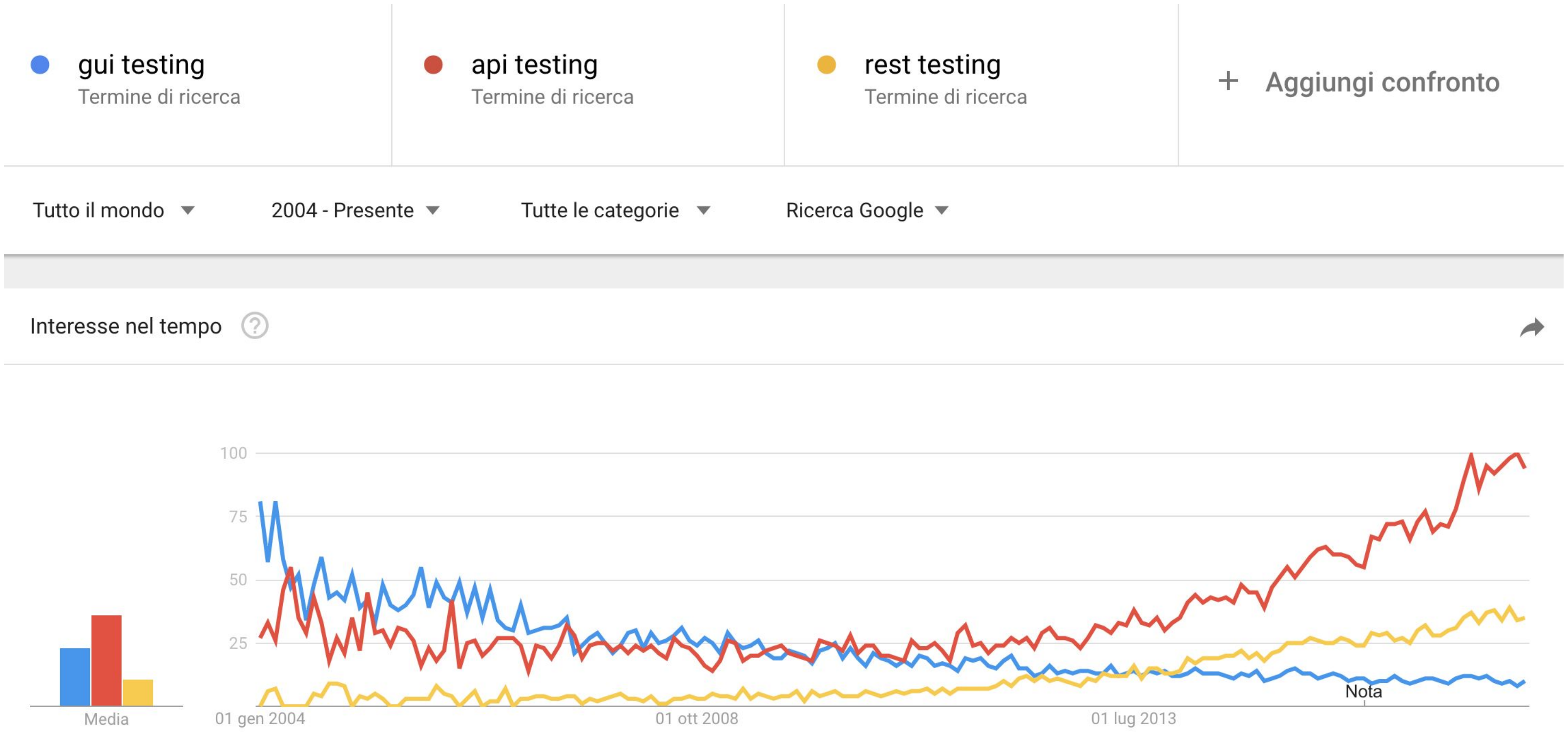
- Lead API Economist @APIAcademy
- Co-founder of OAuth.io
- Founder of APIdays Conferences
- Author of “Banking APIs : State of the market 2015-16-17” and “The state of API Documentation 2017” and the *API Industry Landscape*
- *Author on APIs on Readwrite.com*
- Author *of API weekly newsletter (100,000+ subscribers)*

API Testing

API Testing

1. Strategy and tactics
2. Types of API testing
   - Usability
   - Security
   - Automated testing
   - Performance

API Testing :trends



● gui testing
Termine di ricerca

● api testing
Termine di ricerca

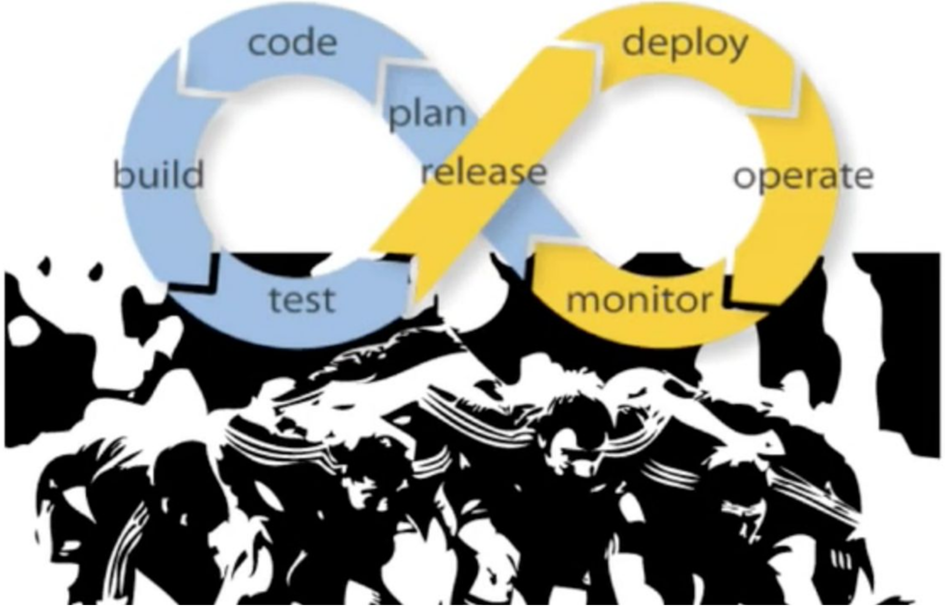
● rest testing
Termine di ricerca

+ Aggiungi confronto

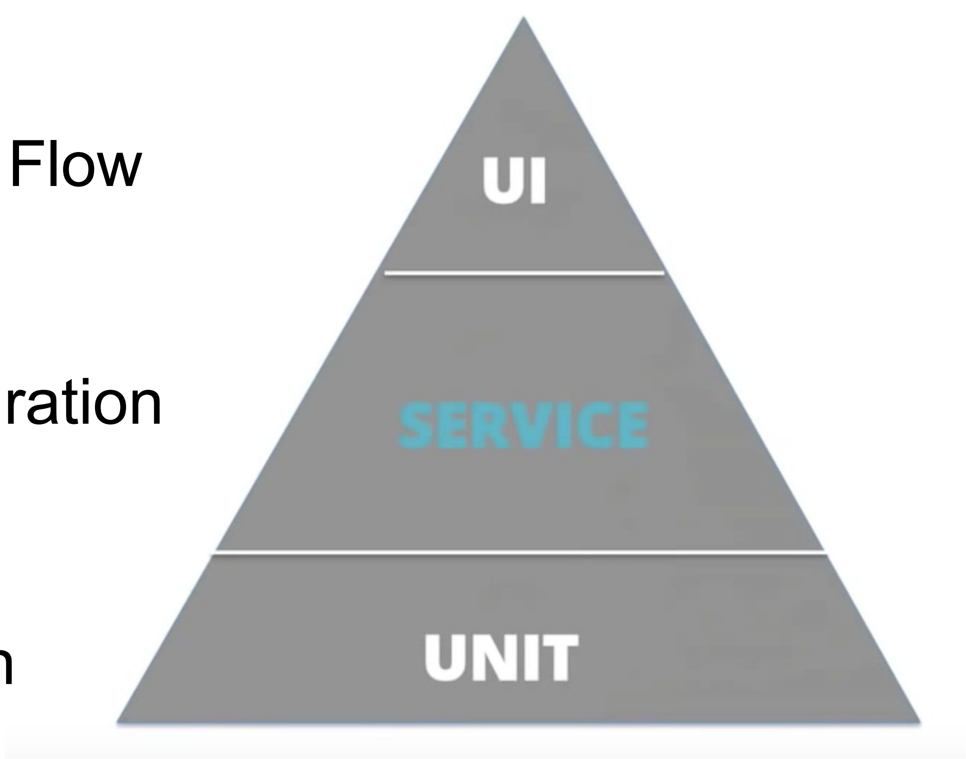
Tutto il mondo ▾ 2004 - Presente ▾ Tutte le categorie ▾ Ricerca Google ▾

Interesse nel tempo ?

API Testing : Agile practices



API Testing :Testing pyramid



Flow

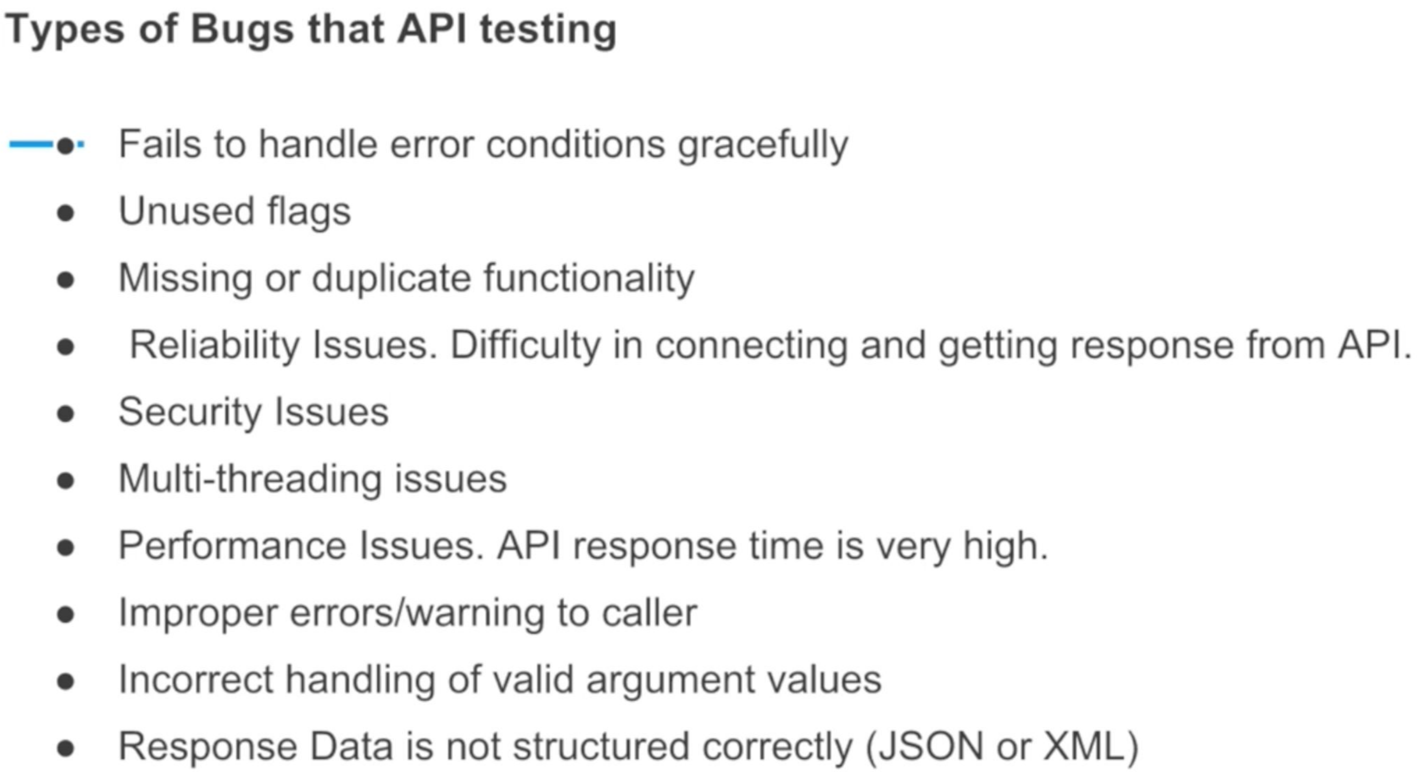
Integration

Function

API Testing : Types of testing

- Usability testing
- Security testing
- Automated testing
- Performance testing

API Testing : Lots of possible bugs



Types of Bugs that API testing

- Fails to handle error conditions gracefully
- Unused flags
- Missing or duplicate functionality
- Reliability Issues. Difficulty in connecting and getting response from API.
- Security Issues
- Multi-threading issues
- Performance Issues. API response time is very high.
- Improper errors/warning to caller
- Incorrect handling of valid argument values
- Response Data is not structured correctly (JSON or XML)

API Testing strategy and tactics

API Testing : Strategy and Objectives 1/2

- Prove implementation is working correctly as expected

- Ensure implementation is working according to requirements specification

- Prevent regressions in between releases

API Testing : Objectives 2/2

Humans versus machines : Automation

- Tools available to continuously run unit tests whenever a code file is saved on disk
- There are continuous integration systems to run unit and integration tests when code is committed to local branch or to central repo
- Test script can be scheduled to run every x minutes continuously to check for certain conditions 24x7

Example : monitoring checks for health checking purposes, e.g. "make a reservation to restaurant then cancel the booking" repeated every 3 minutes.

Usability API testing

Usability API Testing

- Humans based testing, cannot be automated
- Works better with people who did not design the API or don’t know it
- Should be run be API product manager (do you have one?)
- Feedback loop with developers
- Think about the whole developer journey (Login, documentation, authentication, sandbox code samples, etc)

API Security testing

API security testing

Think like a bad guy. Be strict and safe on what you accept. What is not intended need to be rejected. No surprises.

- For a given input, the API must provide the expected output
- Inputs must appear within a specific range for the most part, so values outside the range must be rejected
- Inputs of an incorrect type must be rejected
- Any input that is null (empty), when a null is unacceptable, must be rejected
- Inputs of an incorrect size must be rejected

Automated testing

API Testing : Questions?

- Do you have a current testing strategy being used?

- Are you familiar with testing automation? Are tests currently automated? Do you have the tools available to automate new tests as they are built?

- Test Driven Development (TDD) or Behaviour Driven Development (BDD) ?

- How stable / reliable are the non production versions of the target system being exposed? This will inform decisions around how much to mock the target systems?

- How mature is your Agile methodology in general?

API Testing : Target Mocking 1/2

Mocking Target systems may help automate testing and is recommended in following scenarios:

- When target APIs are not mature or reliable enough
  - Availability of target APIs - deployment, migrations, lifecycle-impedance
  - When target APIs are being developed at the same time as \proxies - create independent and parallel development streams for target APIs and proxies
  - There are network, systems, data stability or maturity issues

- When data is constantly changing or nature of data is such that it is not predictable to be asserted consistently using automated testing

API Testing : Target Mocking 2/2

- When it is not possible (or very difficult) to simulate certain scenarios for testing purposes
  - 5xx errors from target, timeouts, data collisions, conflicts

- When tests rely on previous data population, e.g. user change password, reset, duplicate email, forgot password cases

- Target API has bad response times, e.g. API that responds in 2 minutes. We need fast tests and should be relatively cheap to execute them.

Building mocks for targets systems should be seen as a long term strategy and the cost of keeping these mocks up to date and how much time is invested in building the mocks should be considered as part of the testing strategy.

API Testing : Unit Testing

- Unit testing the smallest testable part of an application
- The objective in unit testing is to isolate a unit and validate its correctness
- Unit tests are narrow in scope, they should be easy to write and execute
- Each Test case should be run independently (ideally)
- Developer focused testing

Advantages	Disadvantages
Find problems early	Combinatorial problem (3 for 1 line of code)
Facilitate change and refactoring	Testing code is as buggy as production code
Simplifies later integration testing	Non realistic and non useful tests
Documentation	Version control and changes
Design (for TDD)	Maintenance
Can be run locally	External code is mocked

API Testing : Integration Testing

- Done to demonstrate that different pieces of the system work together
- Integration tests do a more convincing job of demonstrating the system works
- For technical and non technical audience

Advantages	Disadvantages
Can be user behaviour oriented	Need to deploy
Higher value realistic tests for product teams	Slower tests because of network
Testable by non programmers	Can miss some non working parts
Easier to think and design	Harder to automate

API Testing : Integration test versus Unit Testing

- Look for the right balance and make them complementary
- Overlap can be a lot of maintenance

Performance testing

API Testing : Performance Testing 1/2

Find the capacity limit point of the whole system: response latency (TTFB, TTLB, average response time), number of successful transactions per second

- the relative rate of increase in response times grows exponentially
- even though we continue to increase the load on the system, the rate of successful responses stay more or less the same
- error rate starts to increase

API Testing : Performance Testing 2/2

The way we usually conduct the performance testing:

- Come up with a test plan, document this and agree with your Ops team
- Select a performance testing tool and implement test scripts that executes the test plan
- Extract reports from testing tool and record them in a child page of the test plan

If you have access to hardware analytics : check Memory, CPU, IO

Questions about API testing?

API security

API security

1.Context

2.API security principles

3.Real world use cases/examples

4.The new security stack

- OAuth
- Tokens
- JWT
- OpenID connect

5. Common API security attacks

6. DevSecOps practices

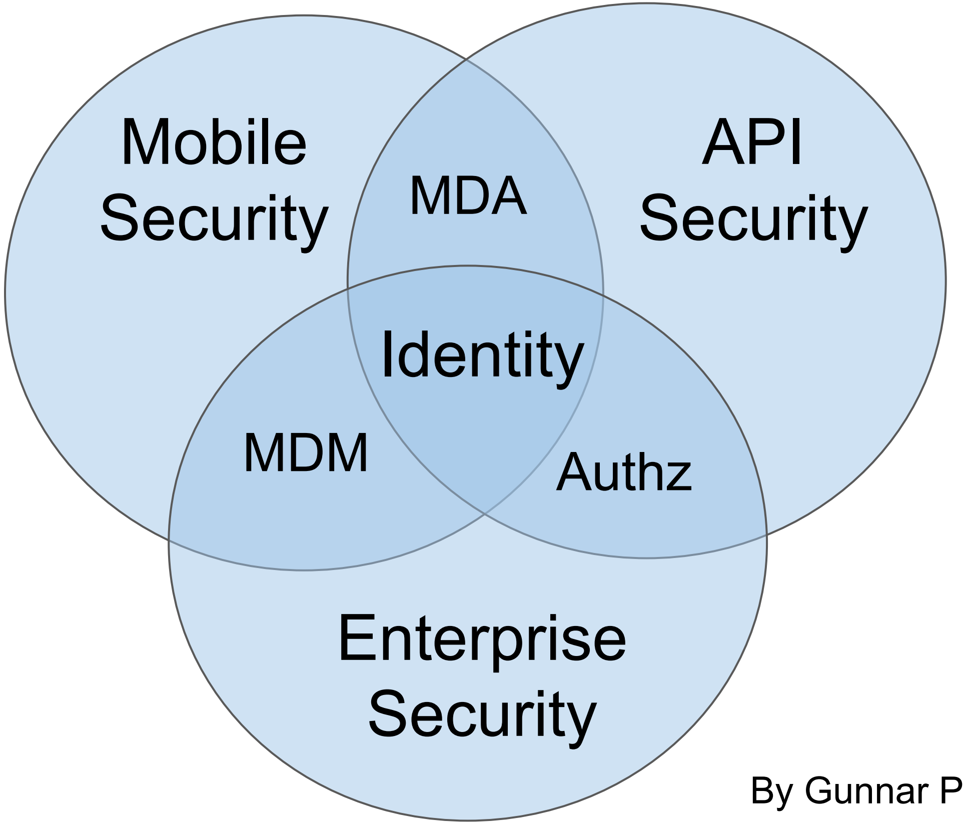
New security concerns

Mobile/IoT Security

API Security

Enterprise Security

Identity is at the core



By Gunnar Peterson

Platforms are valuable targets

- Core business value is accessible
- Critical mass
- Lots of attack factors and surface (endpoints)
- You need to be wrong only once
- High ROI for efforts invested

12 API security principles

High level security concepts (CIA)

Confidentially

- Limit access to the information. Data must be available for authorized users only, and protected from unintended recipients during transit, processing or at rest.

Integrity

- Information is trustworthy and accurate. Data is protected from intentional and unintentional alterations, modifications, or deletions.
- Important feature is reliable detection of those unwanted changes,

Availability

- Availability is a guarantee of reliable access to the information by authorized people.
- It also impose availability requirements to the infrastructure and application levels, combined with appropriate engineering processes in the organization.

By Yuri Sabach

Main API security principles

Economy of Mechanism

- Keep design and implementation of the system as simple as possible.
- Complex solutions are difficult to inspect and improve, they are more error-prone.
- From the security standpoint minimalism is a good thing.

Fail-safe defaults

- Access to any API endpoint/resource should be denied by default.
- Access granted only in case of specific permission.
- Protection scheme “when access should be granted” vs “when access should be restricted” #mistake

Complete Mediation

- Access to all resources of a system should always be validated.
- Every endpoint must be equipped with an authorization mechanism. This principle brings security considerations to a system-wide level.

Main API security principles

Open Design

- Security design should not be a secret and based on defined security standards and protocols.
- Security design or algorithm is separated from protection keys/passwords,
- Many people to review and contribute to this design without risk of being allowed to access the system.

Least Privilege

- Every user of the system should operate with minimal permissions required to the job.
- Limits the damage caused by an accident or error related to the specific user.

Psychological Acceptability

- Security implementation should protect a system but not hamper users of the system.
- Security architecture is well documented and easy to understand and use.

Main API security principles

Minimize Attack Surface Area

- Limit surface attack : Minimization of what can be exploited by malicious users, expose only what is needed
- Limit area damage : limit scope, rate limitations

Defense in Depth

- Multiple layers of control make it harder to exploit a system.
- SSH access to the server may require specific private key. You can limit SSH access to the server to several known IP addresses (white labelling)
- Reduces probability of unauthorized access to the protected resource.

Don’t trust services or *zero trust policy*

- Treat always 3rd party services as unsafe by default and implement all relevant security measures.
- 3rd party data needs to be validated and verified.

Main API security principles

Fail Securely

- All APIs fail often to process transactions due to incorrect input or other reasons.
- Any failure inside the system should not overcome security mechanisms.
- Implementation logic should deny access in case of failure. (fail-safe defaults principle)

Fix security issues correctly

- Once a security issue has been identified, fix it in a right way.
- Developers and security experts need to understand root cause of the issue, create a test for it, and fix with a minimal impact to the system.
- Once fix is done the system should be tested in all supported environments and on all platforms.

Real world use cases

Example with passwords

- Reduce passwords by using SSO (better to manage users roles and permissions)
- Demand 2FA
- Ask your employees to use a password manager
- Don’t reuse passwords
- Ask for secure passwords (passphrases)
- No password, keys or tokens in code

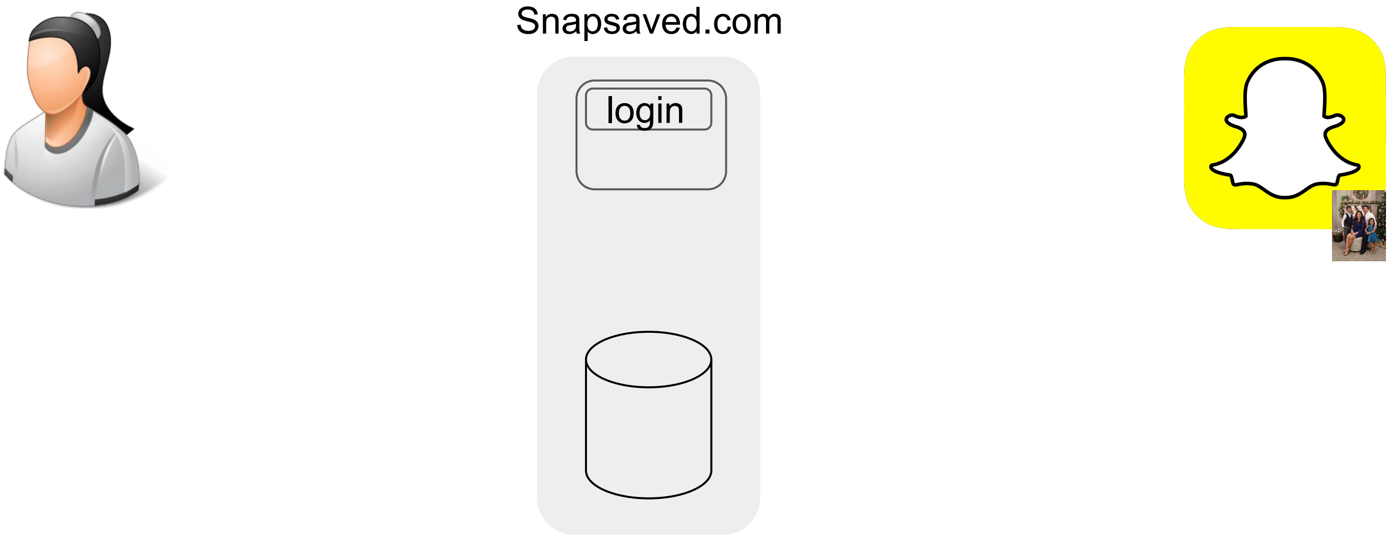
Example with Apple hack #fappening

- Movies stars attacked on their Itunes account
- No rate limiting on a endpoint, no throttling
- Hackers guessed Itunes account and attacked it with “dictionary attack”
- With enough time they guessed passwords on some accounts
- Downloaded all their Itunes data and share them on social medias

Actually it was mostly a phishing attack to people looking for photos on the web

Example with Snapchat hack

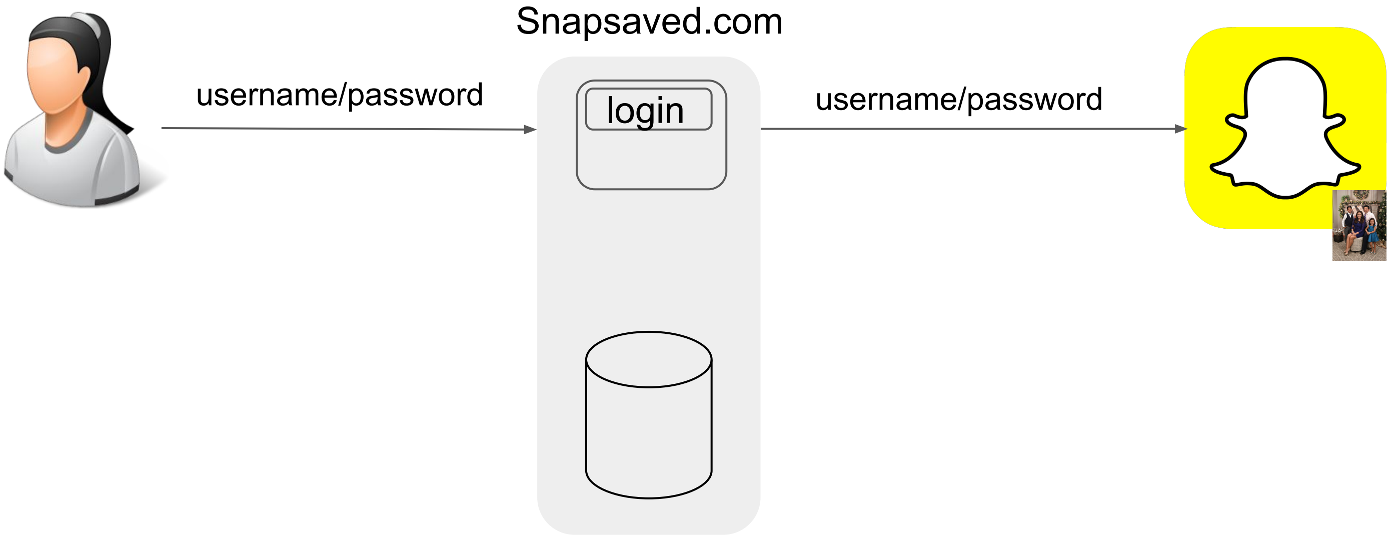
Example with Snapchat hack



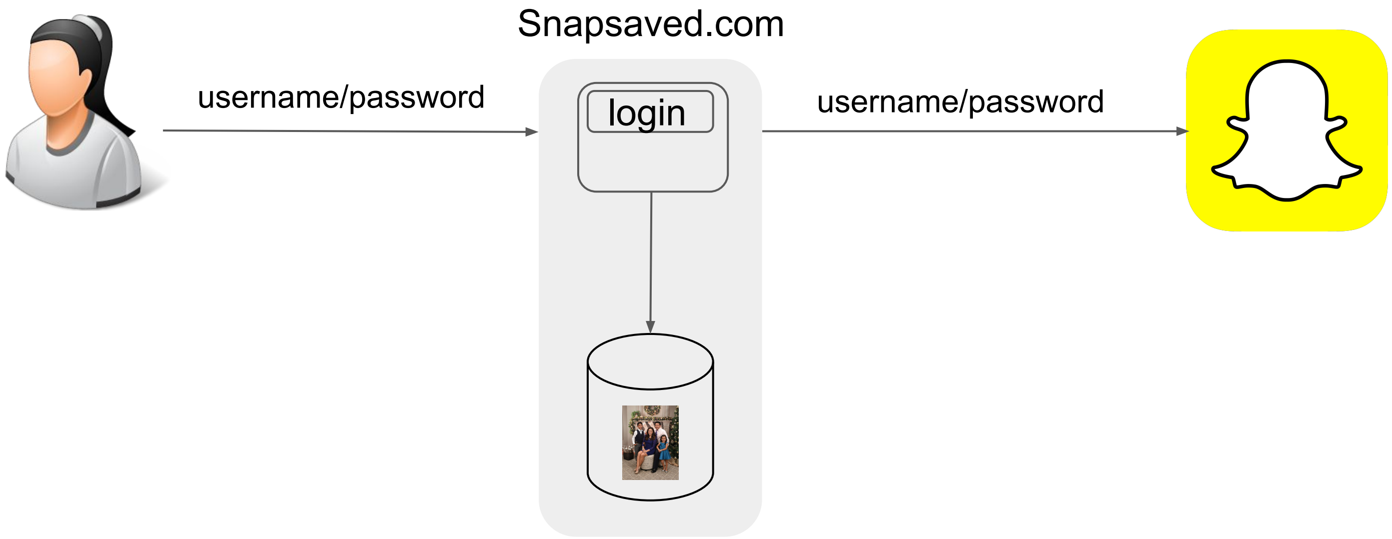
Snapsaved.com

login

Example with Snapchat hack



Example with Snapchat hack



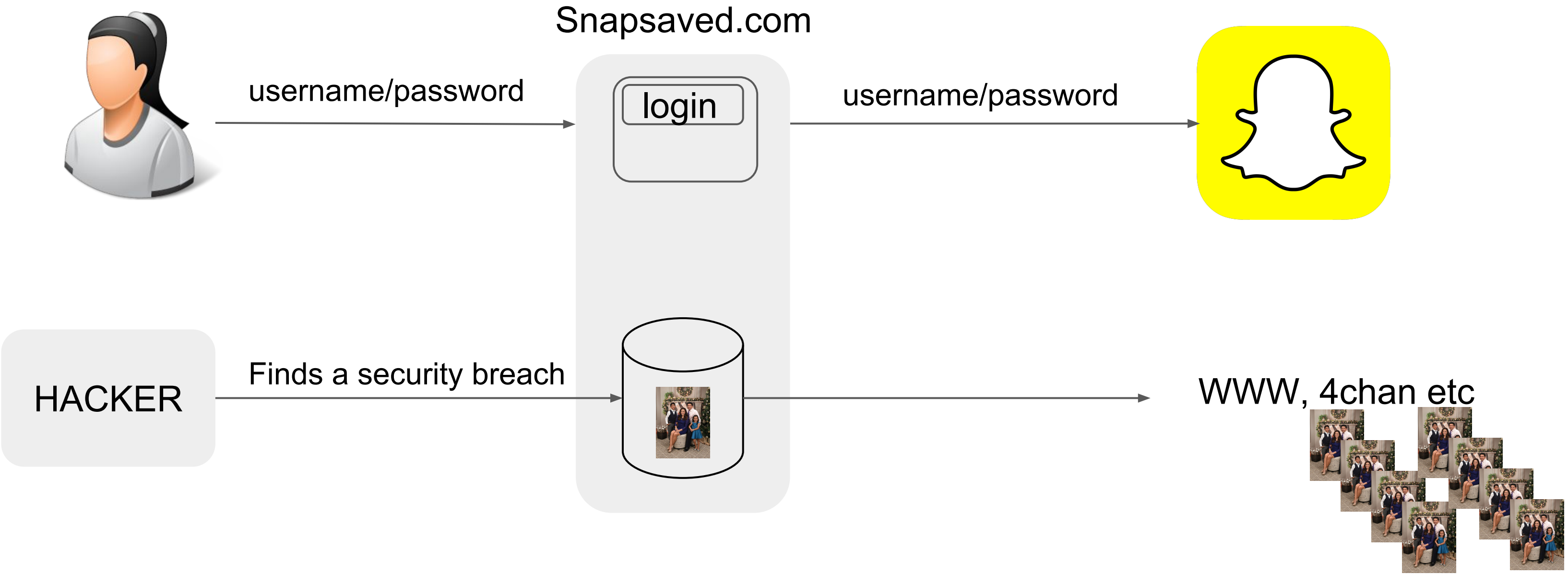
Snapsaved.com

username/password

login

username/password

Example with Snapchat hack



The new security stack
OAuth/OpenID connect

The problem of API keys

- Revocable, non-expiring (bearer access tokens)
- Symmetric Keys
- Passwords

New standards for a new stack

Authentication	Fido U2F
Provisioning	SCIM
Identities	JSON IDentity Suite
Federation	OpenID Connect
Delegated Access	OAuth2.0
Authorization	ALFA

By Travis spencer

SCIM (System for Cross-domain Identity Management)

> *“The SCIM standard was created to simplify user management in the cloud by defining a schema for representing users and groups and a REST API for all the necessary CRUD operations.”*

- Defines RESTFul API to manage users and groups
- Specifies core user and groups

JSON Identity suite

- Suite of JSON-based identity protocols

Token (JWT)
Keys (JWK)
Algorithms (JWA)
Encryption (JWE)
Signatures (JWS)

- Bearer Token spec explains how to use w/ OAuth
- Being defined in IETF

OAuth

OAuth

- OAuth 2.0 is a framework
- OAuth is the base of others specs (OpenID connect, UMA etc)
- For delegate access
- No password sharing
- Revocation of access

OAuth

- HTTPs

- Get you tokens instead of a secret

- Use the tokens to let software gain access to resources (Web APIs) without revealing the secret

OAuth parties

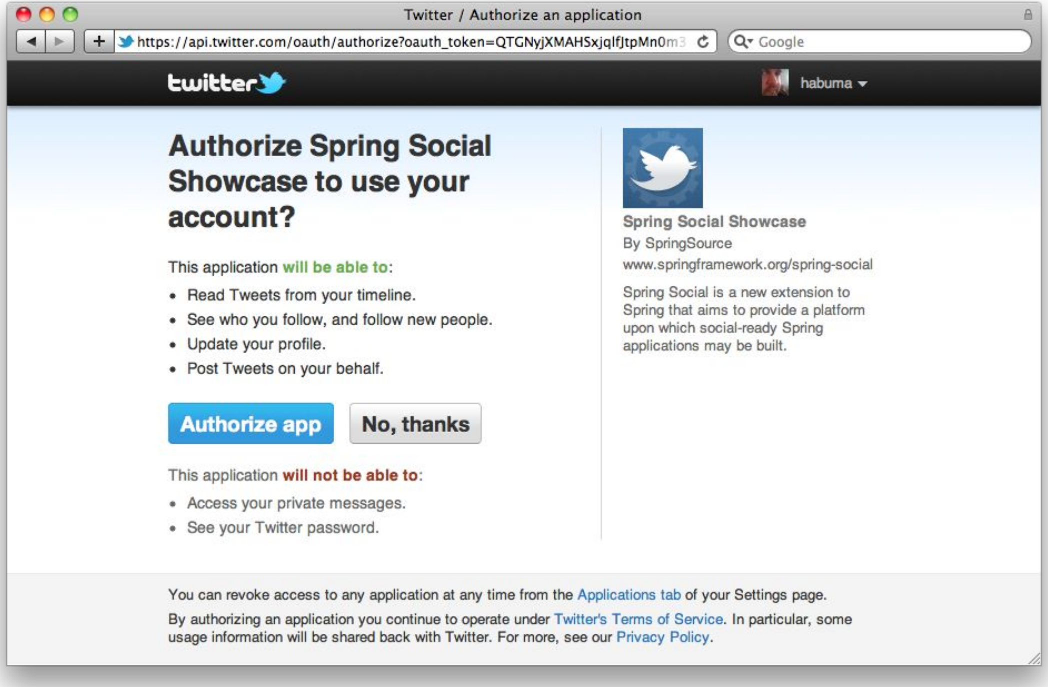
Resource owner

Authorization server

Client

Resource server

OAuth : Request authenticate , consent



Twitter / Authorize an application
https://api.twitter.com/oauth/authorize?oauth_token=QTGNyjXMAHSxjqlfJtpMn0m3
Google
twitter
habuma

Authorize Spring Social Showcase to use your account?

This application **will be able to**:

- Read Tweets from your timeline.
- See who you follow, and follow new people.
- Update your profile.
- Post Tweets on your behalf.

Authorize app | No, thanks

This application **will not be able to**:

- Access your private messages.
- See your Twitter password.

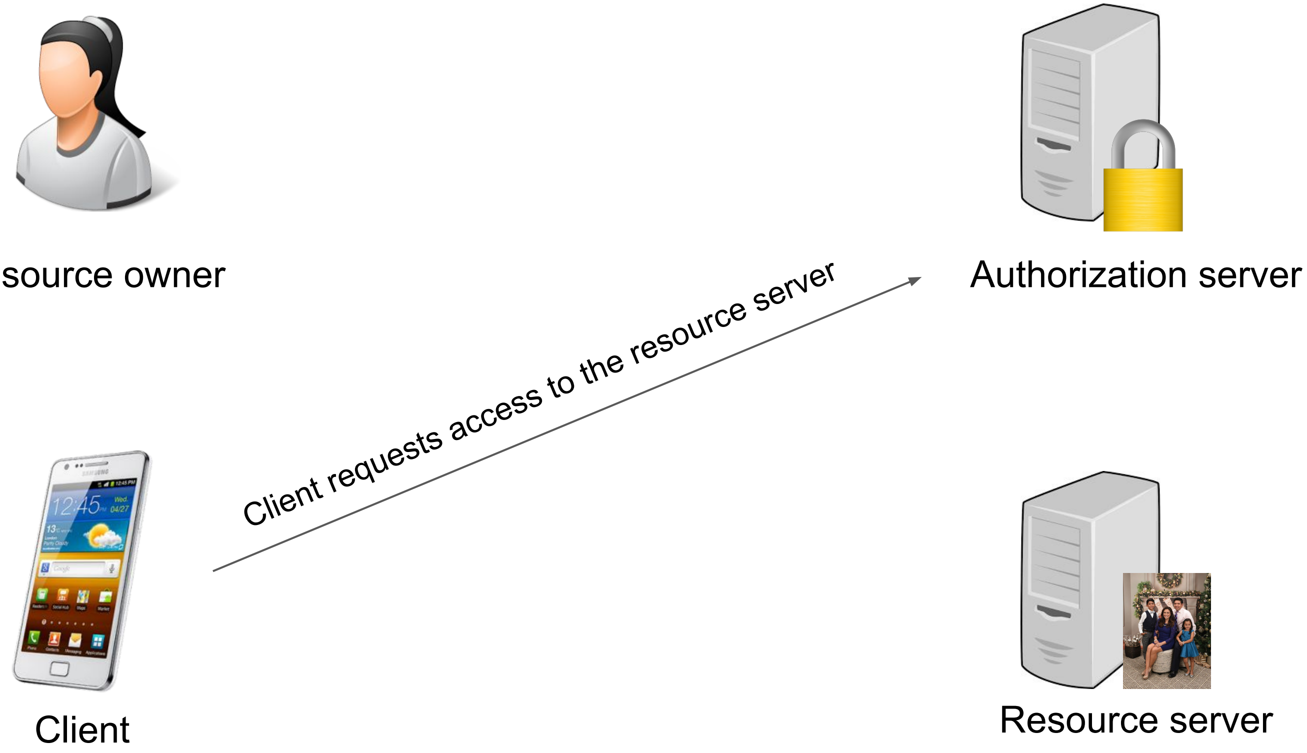
Spring Social Showcase
By SpringSource
www.springframework.org/spring-social

Spring Social is a new extension to Spring that aims to provide a platform upon which social-ready Spring applications may be built.

You can revoke access to any application at any time from the Applications tab of your Settings page.

By authorizing an application you continue to operate under Twitter's Terms of Service. In particular, some usage information will be shared back with Twitter. For more, see our Privacy Policy.

OAuth flow example



Resource owner

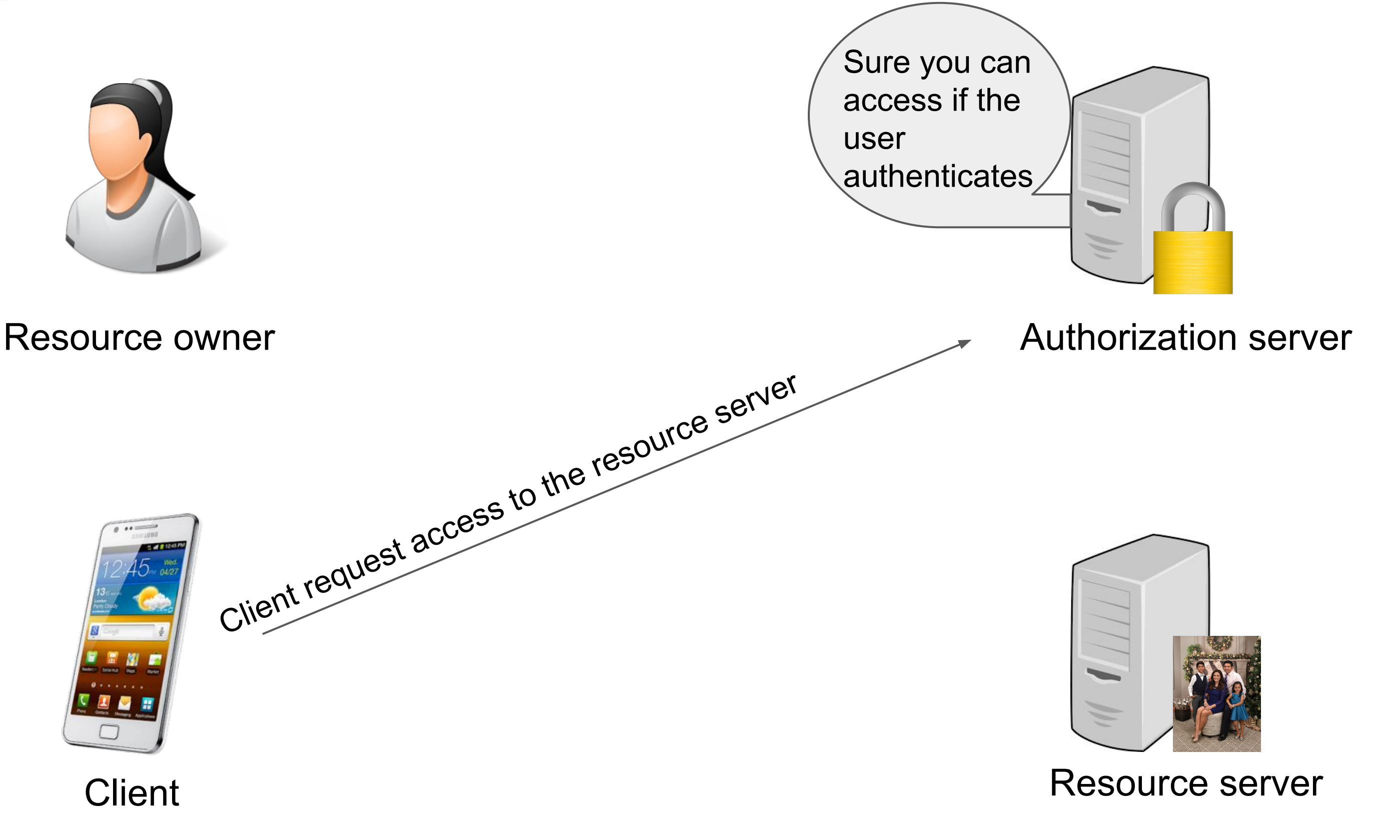
Authorization server

Client requests access to the resource server

Client

Resource server

OAuth flow example



Resource owner

Sure you can access if the user authenticates

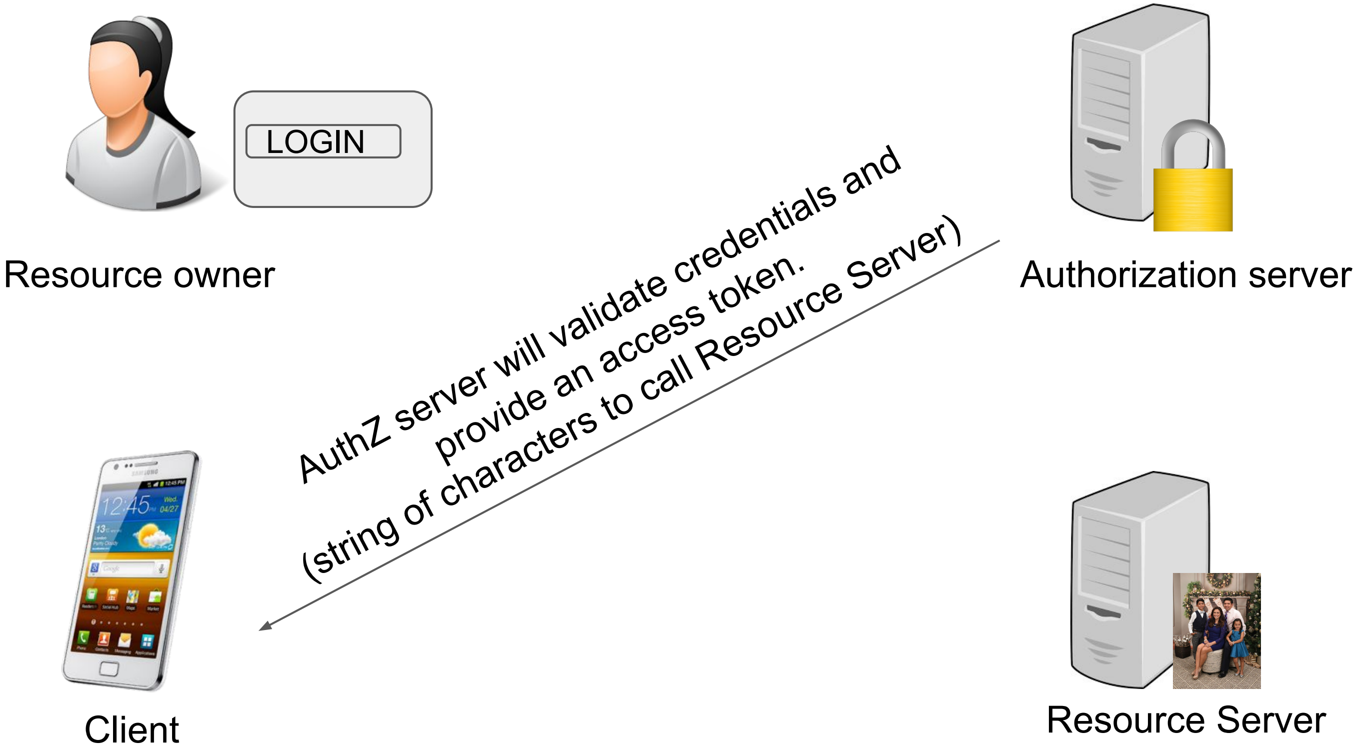
Authorization server

Client request access to the resource server

Client

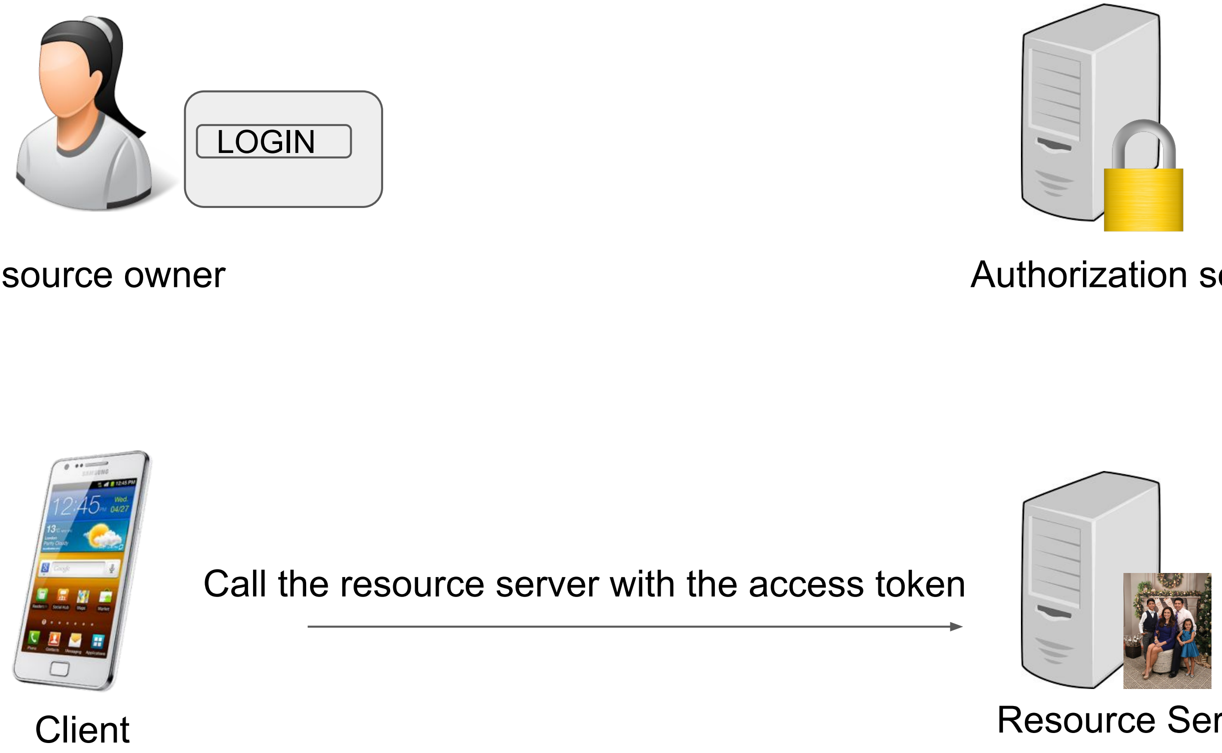
Resource server

OAuth flow example



AuthZ server will validate credentials and provide an access token. (string of characters to call Resource Server)

OAuth flow example



LOGIN

Resource owner

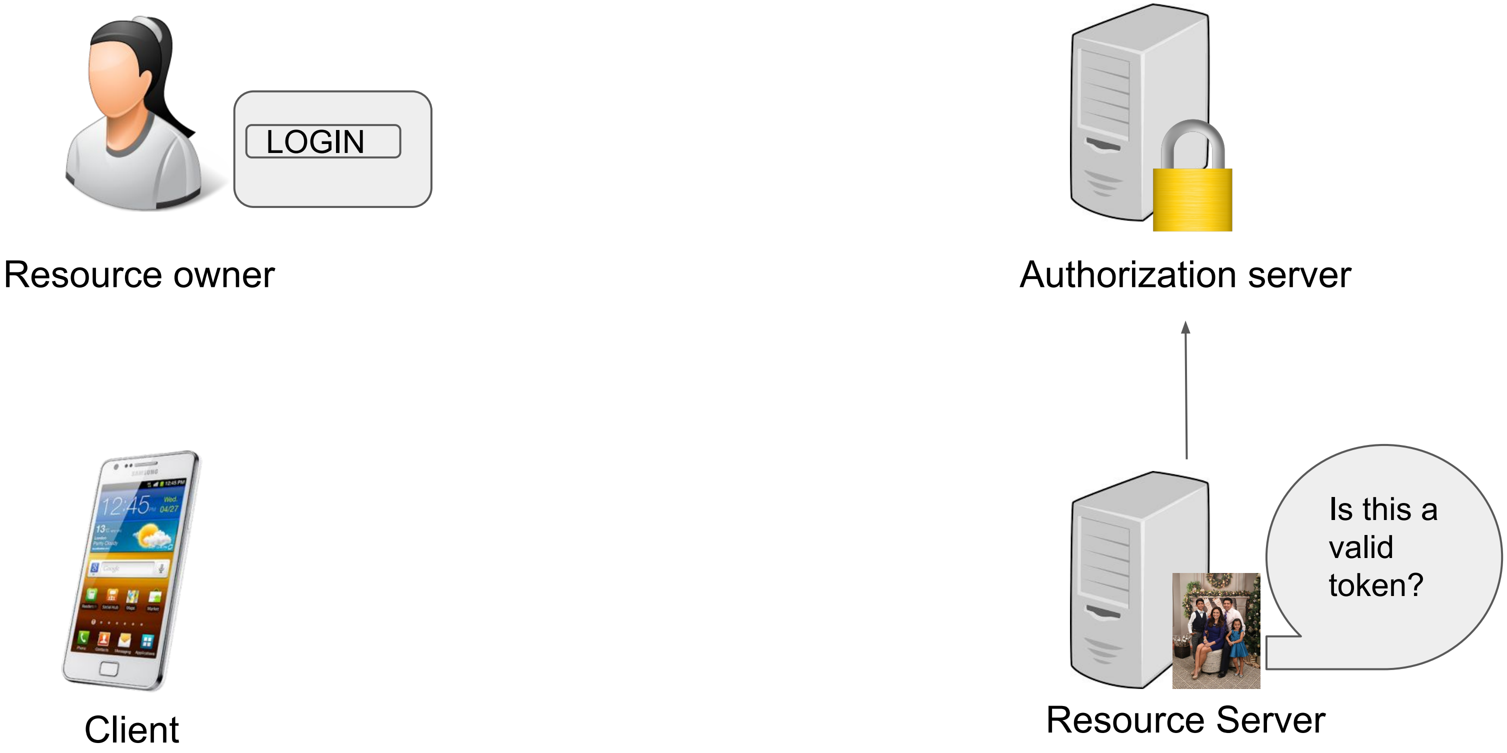
Authorization server

Call the resource server with the access token

Client

Resource Server

OAuth flow example



LOGIN

Resource owner

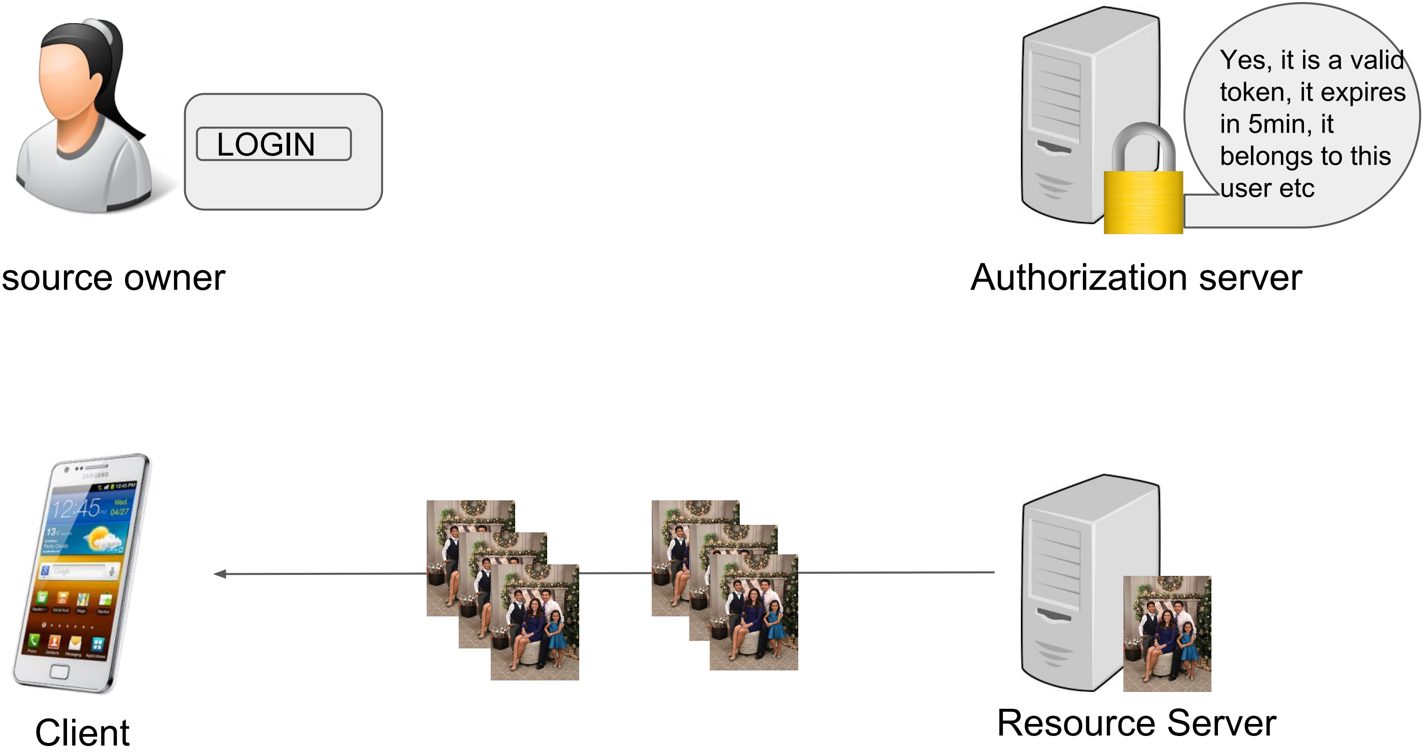
Authorization server

Client

Is this a valid token?

Resource Server

OAuth flow example



LOGIN

Resource owner

Yes, it is a valid token, it expires in 5min, it belongs to this user etc

Authorization server

Client

Resource Server

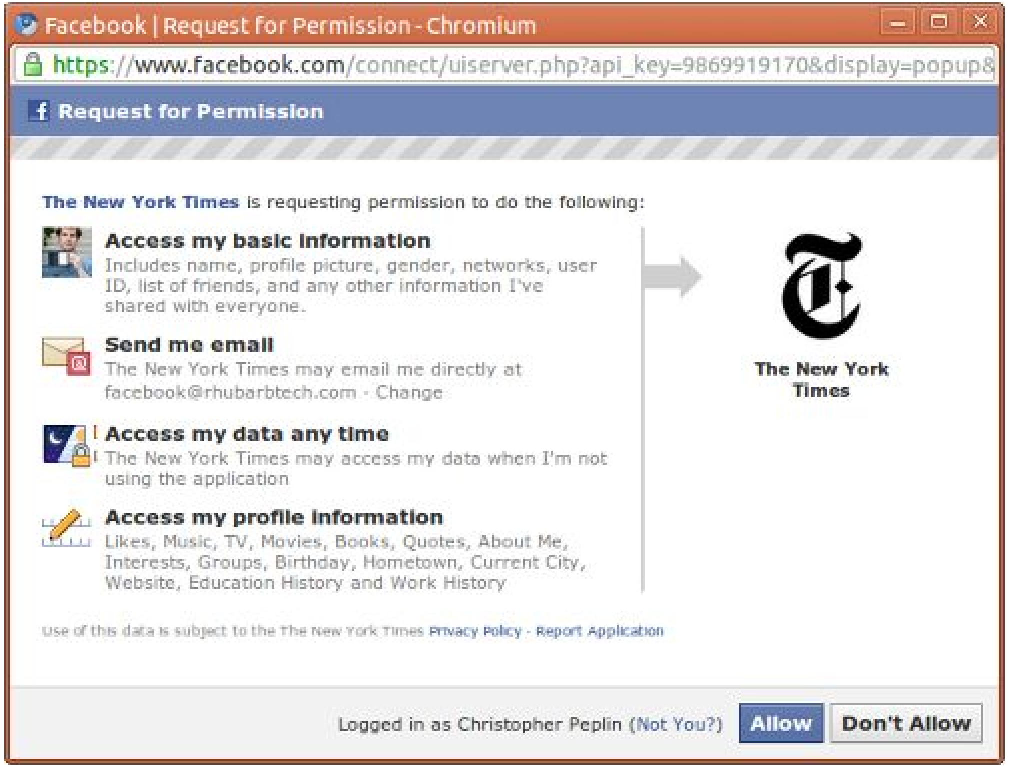
OAuth

The client knows nothing really about the user.

OAuth Scopes

- Scopes are a kind of permissions
- Extend token usefulness
- Can be shown in UI for consent
- No standardized specs for scopes

OAuth : Request authenticate , consent



Facebook | Request for Permission - Chromium
https://www.facebook.com/connect/uiserver.php?api_key=9869919170&display=popup&
Request for Permission

The New York Times is requesting permission to do the following:

Access my basic information
Includes name, profile picture, gender, networks, user ID, list of friends, and any other information I've shared with everyone.

Send me email
The New York Times may email me directly at facebook@rhubarbtech.com · Change

Access my data any time
The New York Times may access my data when I'm not using the application

Access my profile information
Likes, Music, TV, Movies, Books, Quotes, About Me, Interests, Groups, Birthday, Hometown, Current City, Website, Education History and Work History

The New York Times

Use of this data is subject to the The New York Times Privacy Policy · Report Application

Logged in as Christopher Peplin (Not You?) Allow Don't Allow

Tokens

Token

- Issued by authorization server
- Credential for resource server
- Store/send token instead of secret
- Single sign on (SSO) to anybody that trust Authz Server

OAuth tokens :

- Contains claims
- Signed
- Expire
- JWT format (for Open ID)

Access and Refresh Tokens

- Access tokens are like a session.

Use them to secure API calls

- Refresh tokens are like a password.

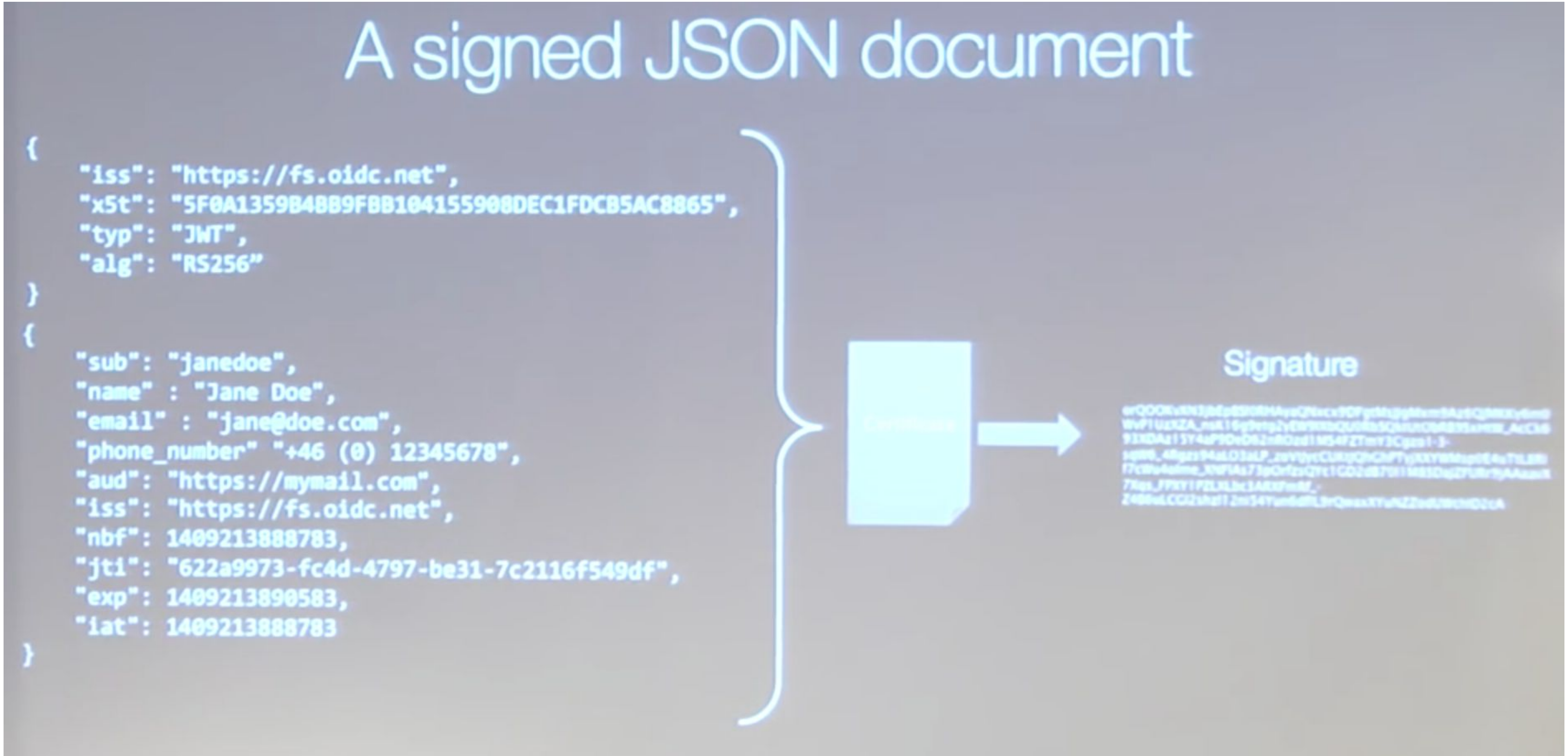
Use them to have newer access tokens

Bearer and HoK tokens

Bearer tokens : like a key

Holder of Key tokens : ties the token to the one who the token has been issued for

JWT tokens



A signed JSON document

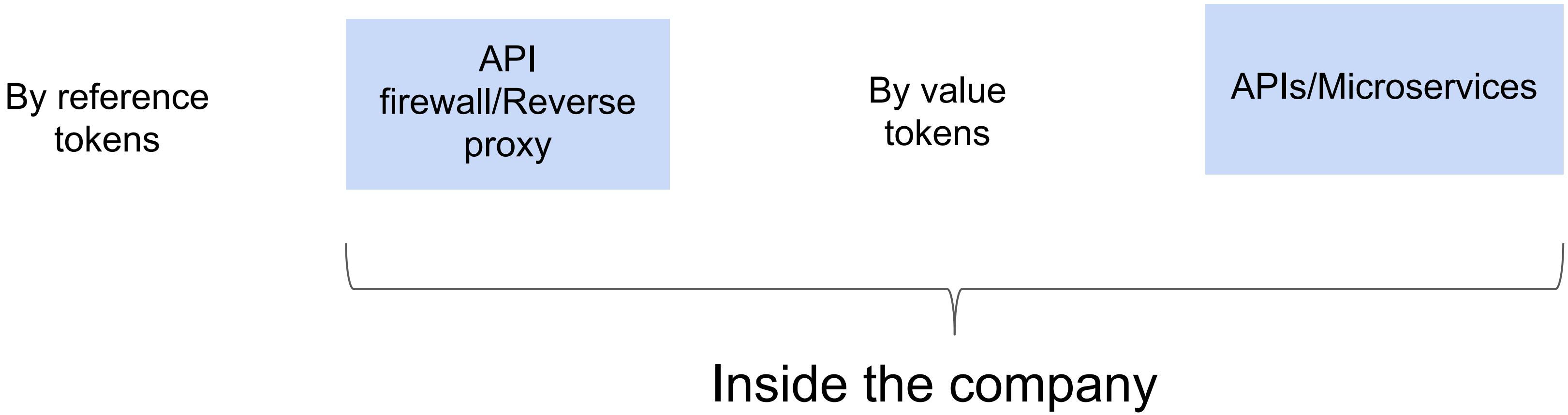
```
{
    "iss": "https://fs.oidc.net",
    "x5t": "5F0A1359B4BB9FBB104155908DEC1FDCB5AC8865",
    "typ": "JWT",
    "alg": "RS256”
}
{
    "sub": "janedoe",
    "name" : "Jane Doe",
    "email" : "jane@doe.com",
    "phone_number" "+46 (0) 12345678",
    "aud": "https://mymail.com",
    "iss": "https://fs.oidc.net",
    "nbf": 1409213888783,
    "jti": "622a9973-fc4d-4797-be31-7c2116f549df",
    "exp": 1409213890583,
    "iat": 1409213888783
}
```

By value and by reference tokens

By value tokens : Contains all the necessary data that we want. JWT tokens are *by value tokens*. To be shared between trusted parties

By reference tokens : random string point to the data, as the receiver can dereference it and get the whole data, like a *pointer*. It means nothing *outside*. Totally opaque.

By value and by reference tokens



By reference tokens

API firewall/Reverse proxy

By value tokens

APIs/Microservices

Inside the company

OAuth

- Not for Authentication
- Not for federation
- Not directly for Authorization
- OAuth is for delegation

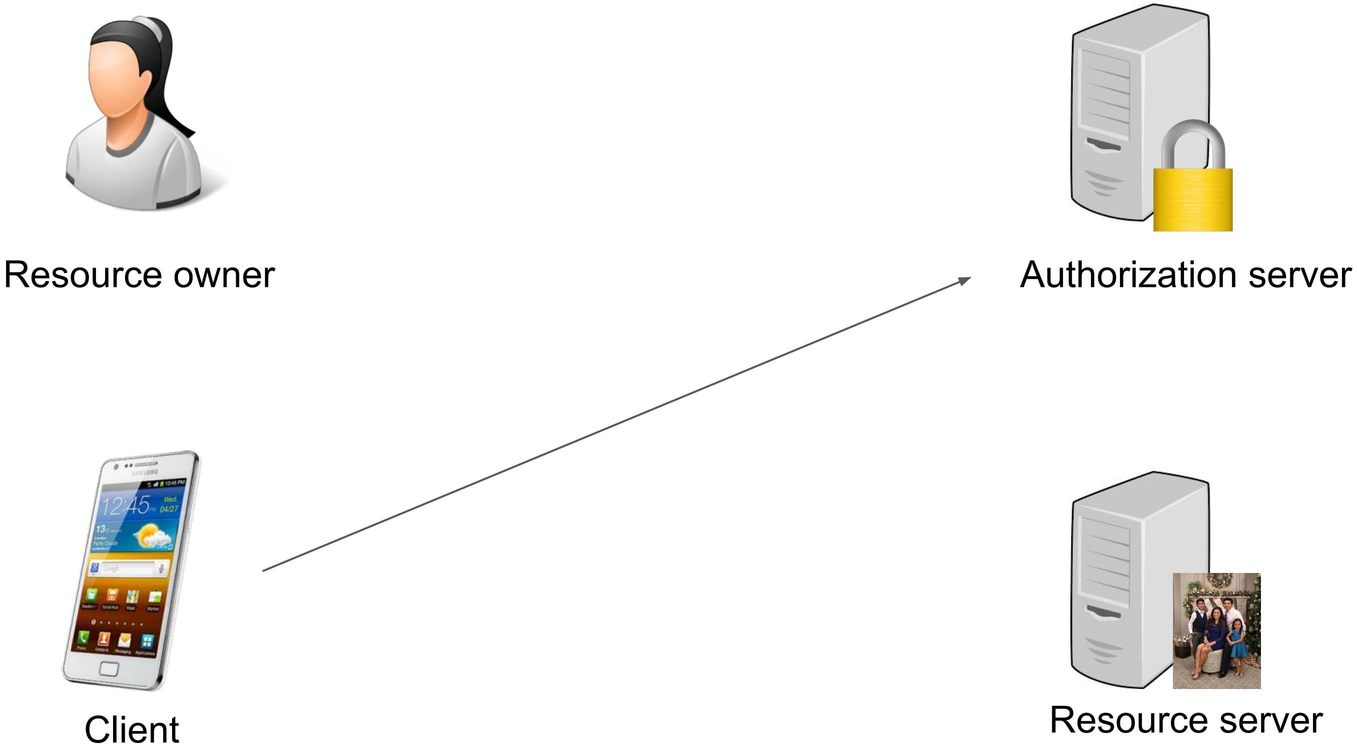
OpenID connect

OpenID connect

- Federation Based on OAuth2.0
- Made for mobile
- Non backward compatible

- Client and API receive tokens
- User info endpoint to get user data

OpenID connect



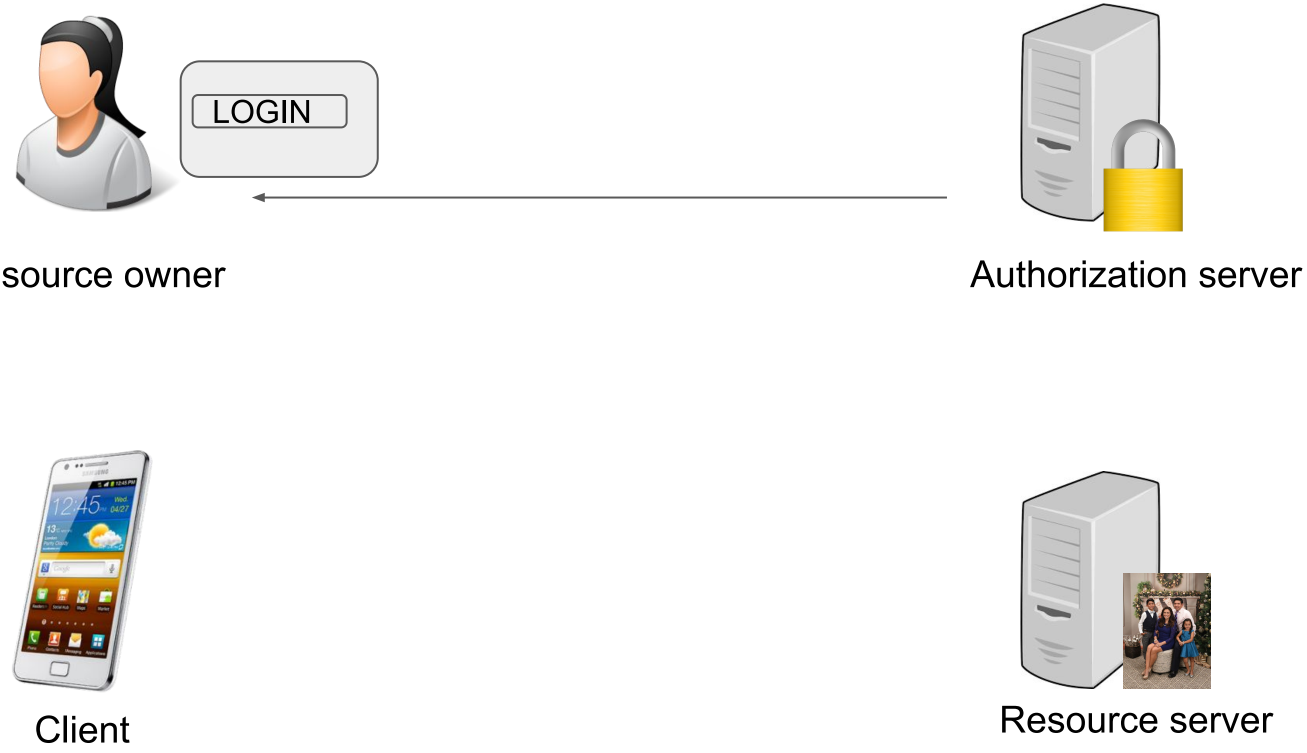
Resource owner

Authorization server

Client

Resource server

OpenID connect



LOGIN

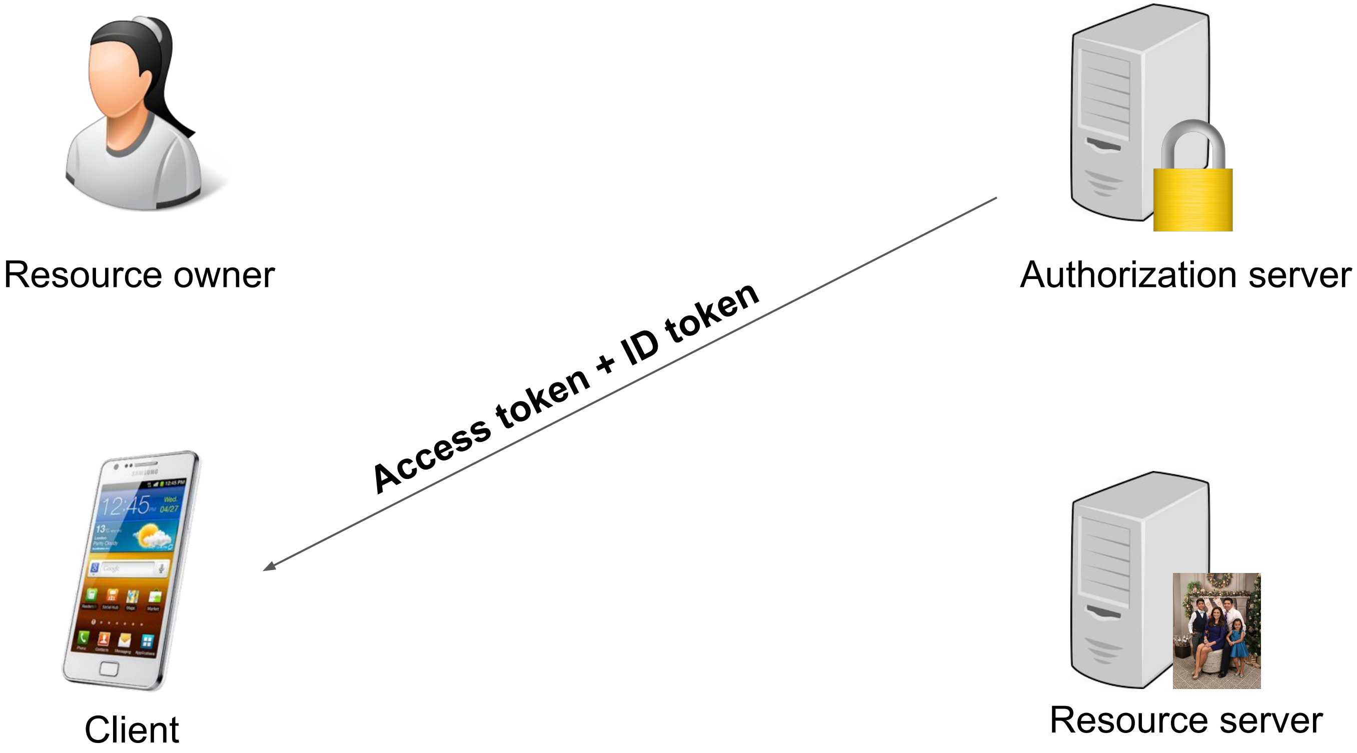
Resource owner

Authorization server

Client

Resource server

OpenID connect



Resource owner

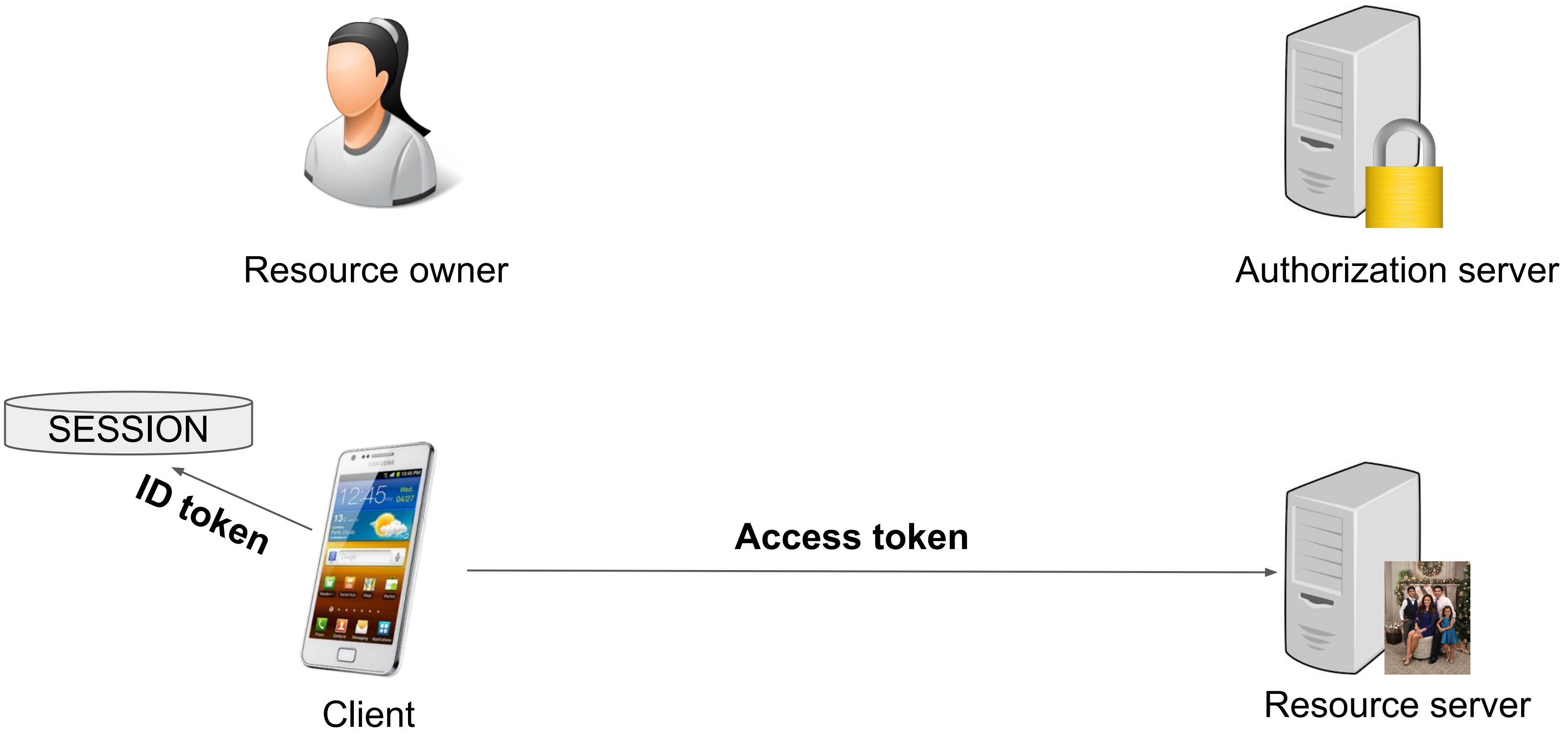
Authorization server

Access token + ID token

Client

Resource server

OpenID connect



Resource owner

Authorization server

SESSION

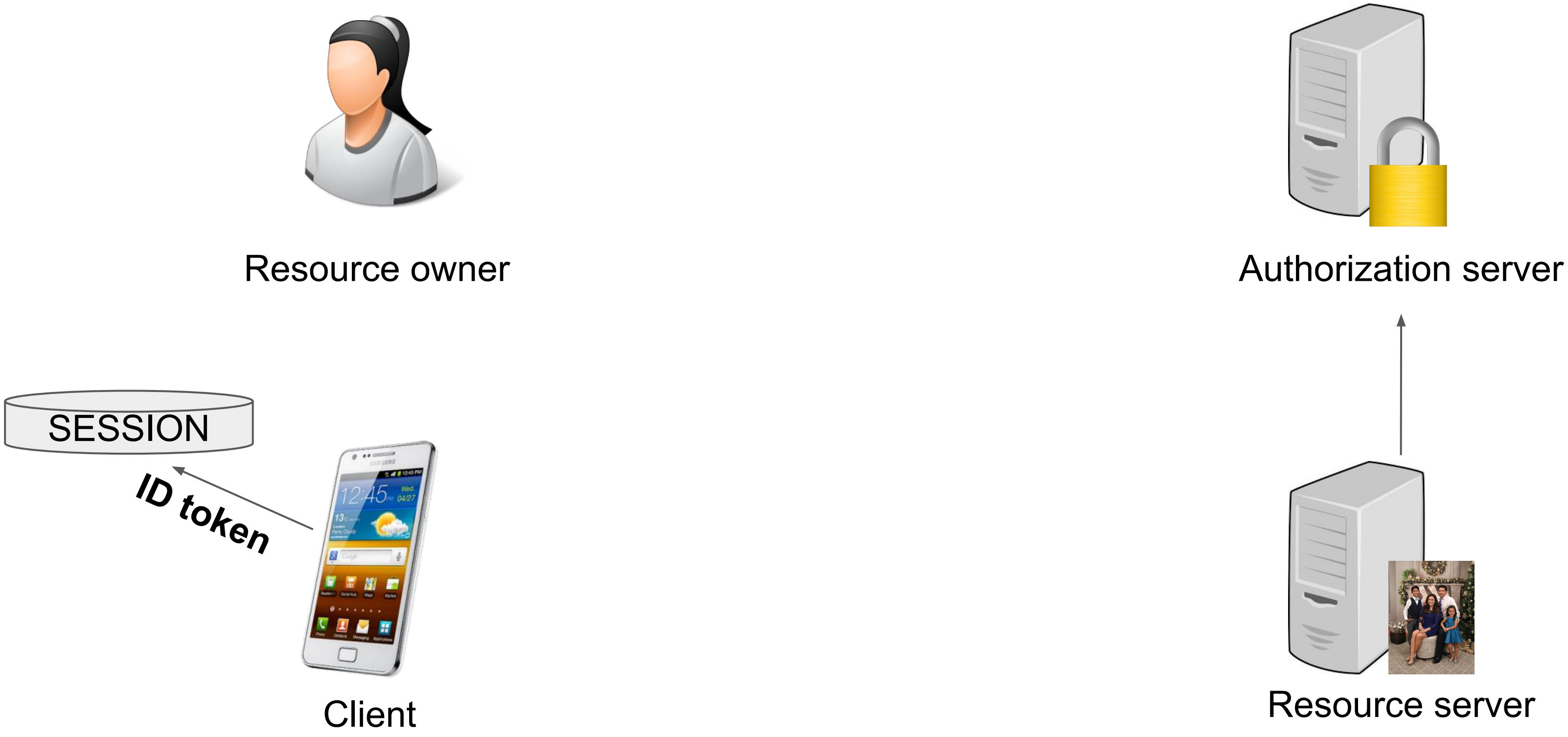
ID token

Client

Access token

Resource server

OpenID connect



Resource owner

Authorization server

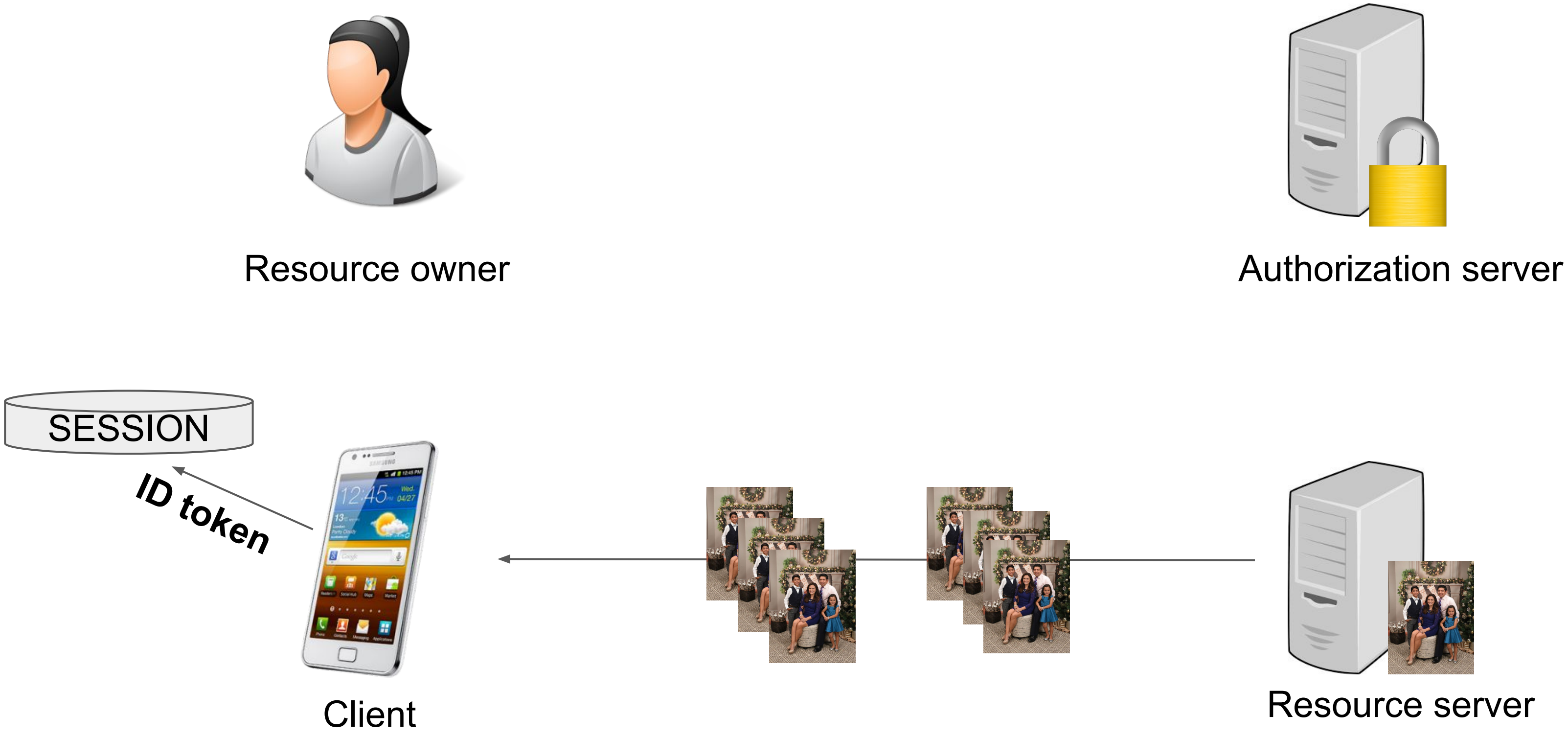
SESSION

ID token

Client

Resource server

OpenID connect



Resource owner

Authorization server

SESSION

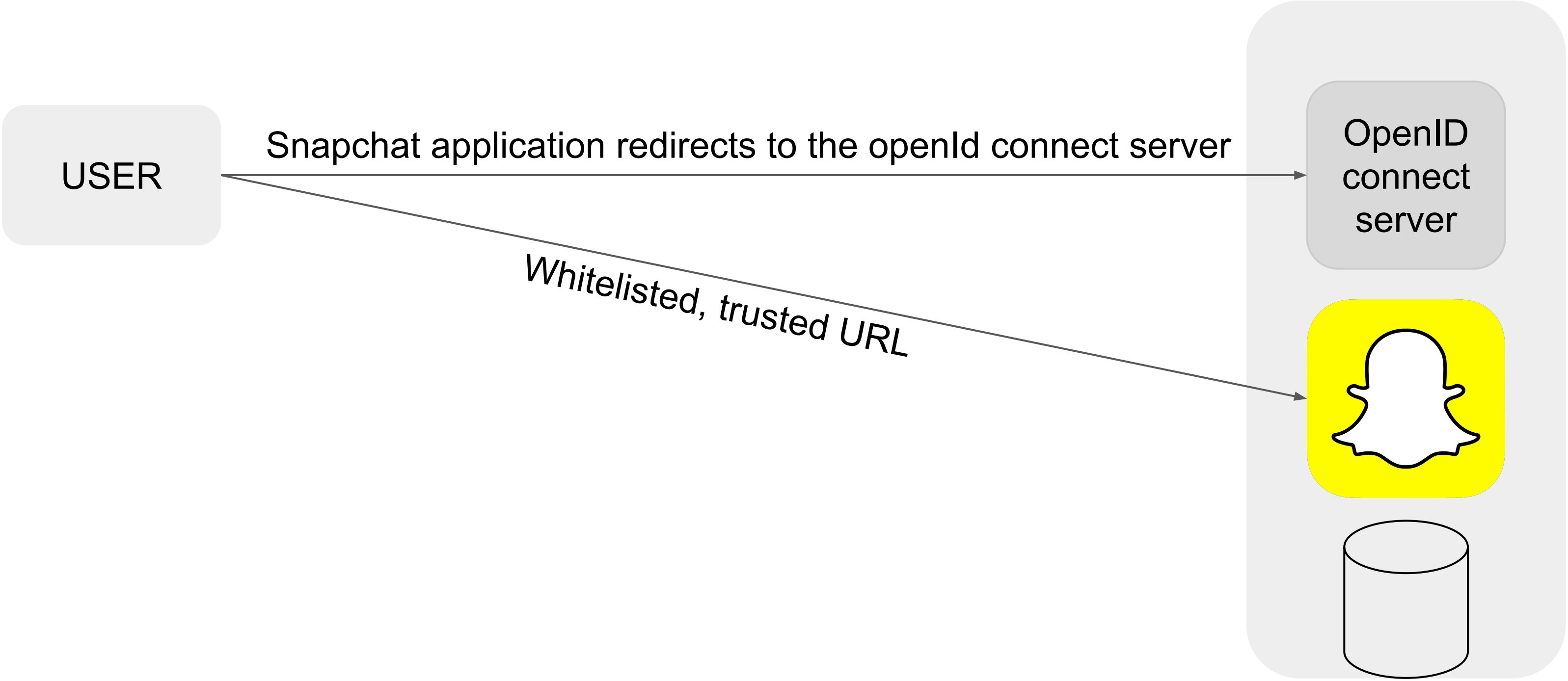
ID token

Client

Resource server

Example with Apple hack

Example with Snapchat hack



Common API security attacks

Classic Attacks : Cross-site scripting (XSS)

“When an attacker gets a user’s browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A Cross-site Scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting”

Web Application Security Consortium

Classic Attacks : Dependancy Vulnerabilities

- Many APIs are build on frameworks.
- Attackers often target these frameworks as they are not monitored enough by developers.

Classic Attacks : DDOS

API-focused DDoS attacks, where attackers identify which API calls will create the most work inside an application, and then send excessive requests to that API

A single request at the edge can fan out into thousands of requests for the middle tier and backend microservices. If an attacker can identify API calls that have this effect, then it may be possible to use this fan out architecture against the overall service. If the resulting computations are expensive enough, then certain middle tier services could stop working. Depending on the criticality of these services, this could result in an overal service outage,” Netflix Tech Blog

- SERVELESS APIS : COSTS ARE SCALABLE !

Classic Attacks : Information leakage

- API keys, secrets and passwords can be found in code, open source code publicly available on the web. (AWS keys for example)

Classic Attacks : MITM

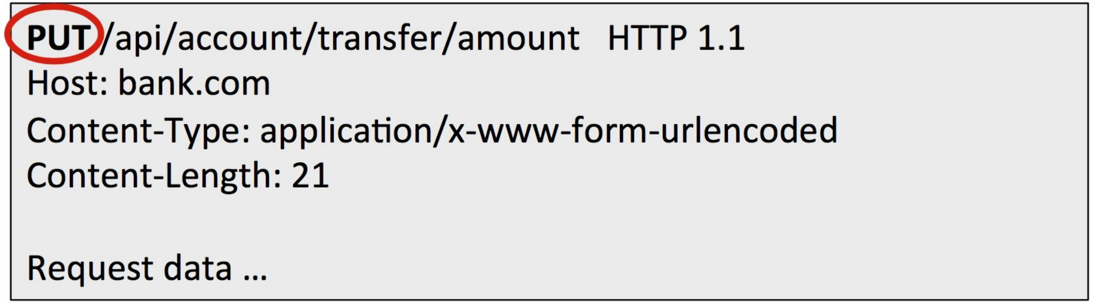
- Possible when there is no verification that the requester is the legitimate API caller authorized to make the API request
- Works when the the message received has been verified as untampered with and sent by the API provider.

Classic Attacks : HTTP verbs/URI tampering

- ManipulaIon of hidden fields in HTML form inputs (i.e. manipulate informaIon such as product costs that may be inherently used by the applicaIon in billing)

- URI itself may be modified to alter values in the abempt to access informaIon not pertaining to the original request (i.e. account numbers, profiles, session idenIfiers, etc.)

- Alter methods in the HTTP header to compromise informaIon via update and delete methods



```
PUT /api/account/transfer/amount   HTTP 1.1
Host: bank.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 21

Request data ...
```

Classic Attacks : SQL Injections

- Malicious dynamic query is made via an API call to test whether itcan be received and processed.
- Focuses on information leakage with unexpected queries to the server (example with credit card)

Classic Attacks : Unauthorized access

- Occurs when insufficient authentication and authorization protocols are not in place
- User roles and permissions are not enough managed
- An attacker can abuse permissions and rate limiting (apple)

Classic Attacks : Unauthorized access

- Occurs when insufficient authentication and authorization protocols are not in place
- User roles and permissions are not enough managed
- An attacker can abuse permissions and rate limiting (apple)

Applying DevSecOps for API security

Application level

- Ensure detection and blocking of applicative attacks as SQL injections, Cross-site scripting and information leakage.
- XML/JSON firewall
- Provide XML/JSON schema validation

Network level

- Encrypt at the message level (confidentialty)
- TLS (transport Layer security) at the transport level (confidentiality)
- Implement procedures to ensure that data has not be tapered in transit (Integrity check)
- Control of the rate of API traffic to avoid DDoS (Availability)
- Log all API transactions for security forensics (Audit)
- Ensurel ogging of transactions include legal proof of the execution of the API call

System level

- Verify the identity of users and clients applications
- Use tokens
- Implement 2FA
- Once users are Auth, check permissions to access API resources

Questions?