Oauth

INFO 7255

Use cases for security

- Washington Post/Boston Globe: paywall, tiered-based subscription
- Flash sales
  - Can I prevent bots from sweeping up all inventory?
  - Can my application hold up against excessive demand?
    - Digital Waiting Room
- Authenticated access
  - Quota and throttling
- Anonymous access
  - throttling
- Bots Access
  - Good bots versus bad bots

Security requirements

- Authorized access against API
  - Only users authorized to access resources are allowed
  - Users able to see/edit their own plans
  - Users may read other plans, but no change them
    - Users may have certain access to this endpoint but not to the other one
- Anonymous browsing may be allowed
  - This is prior to user authentication
- App may not exceed certain requests per day/month: quota
- Apps that are making excessive number of requests need to be throttled
  - Digital Waiting Room

High-level Approach

- Client includes an authorization header
  - The value of the header is a token
- API uses the authorization header value (token) for authorization and authentication
  - Client signs token
  - API verifies token

Key design questions

- What is the overall approach for securing APIs?
  - Bearer Tokens
- What is the token structure?
  - JWT
- How are token generated?
  - How are they signed?
    - By an Idp
- How are tokens verified?
  - Authenticate the signer of the token
- Security crypto: Asymmetric? RS256
- Security guarantees: Authentication, non-tampering

First approach: API keys for securing access by apps

- High level flow:
  - Each app is granted a key at build time by the server
  - app includes key in every request that goes to server
  - Implications on quota and throttling?

OAUTH 1.0

- Username & Password

Industry accepted approach: OAUTH 2.0

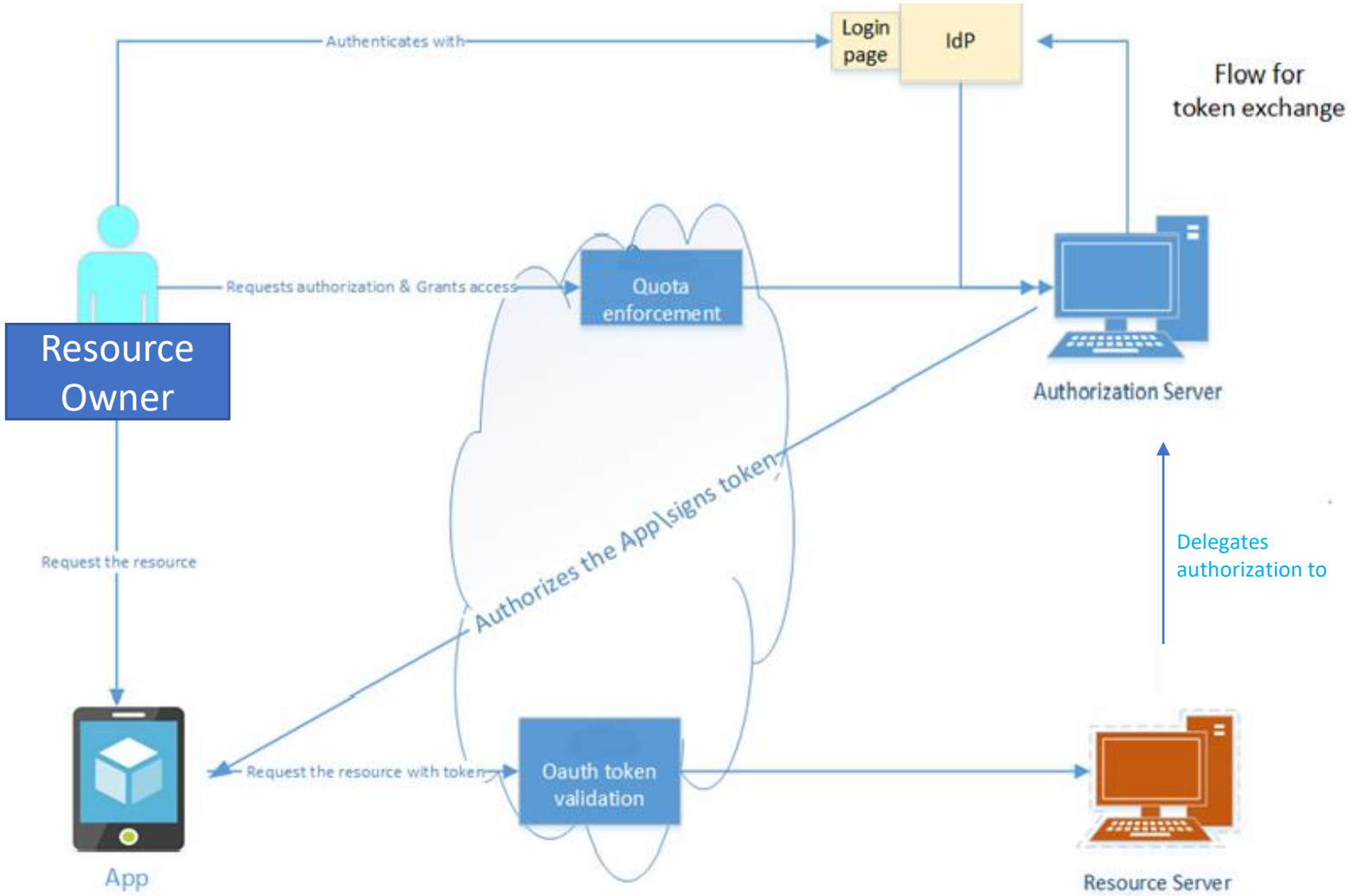
- User downloads an app
- User authenticates with an IDP/Auth server
- User consents to give app access to user’s data
- IDP generates token
- App includes the token in the API calls

Public versus private app

- Public apps are those that cannot secure their credentials: single page application, mobile apps

- Private apps are those that can secure their credentials: any app running behind a firewall

oAUTH 2.0 Overview and Actors



Token Validation by Resource Server

- 1. Validate the structure of a JWT
- 2. Create an “allow list” that contains valid values for iss claim
- 3. Base64decode JWT header, payload
- 4. Retrieve alg and kid from Header
- 5. Retrieve iss from payload
- 6. Compare the value of iss to that stored in the “allow list”

5. If iss value in allow list, use JWKS_URI to retrieve public key. Otherwise, signature invalid

6. Verify signature

7. Validate any other claims such as scope, aud, exp, etc.

Overview

- RFC OAUTH 2.0:https://tools.ietf.org/html/rfc6749
- JWT https://tools.ietf.org/html/rfc7519
- Example: https://dev.fitbit.com/docs/oauth2/

Oauth provider (Authorization Server)

- /register
- /Authorize
  - unsecure
  - Authorization code grant flow
    - Returns both access token and refresh token
    - Use for secure clients
  - Authorization code grant flow with PKCE
    - Use for unsecured client
  - implicit grant flow
    - Returns only access token
    - Use for unsecured client
- /Token
  - Secure
  - Exchange authorization code for a token
  - generate a new token from a refresh token

/register

- Input:
  - Client_type = confidential (private) or public
  - redirect_URI: https://

- Output:
  - client_ID, client_secret if client is confidential
  - Client_id for public

/Authorize

- The authorization endpoint must support "get"

- The supported query parameters are:
  - response_type
    REQUIRED. Value MUST be either "code“ or "token“
  - client_id
    REQUIRED. The client identifier obtained from the registration
  - redirect_uri
    Required. As described in Section 3.1.2.
    https
  - scope
    Required.
  - state
    Required

Authorization grant code flow example:

GET /authorize?response_type=code&client_id=s6BhdRkqt3&state=xyz
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb &scope=read
HTTP/1.1

Host: server.example.com
•The response should have:
•HTTP status code should be set to 302:
•the redirect URI as the value of the location header.
•the query parameter code and its value, state and its value appended to the redirect URI.
•The code is required at all times. The state is required only if it has been present in the request.
•example:
•HTTP/1.1 302 Found
•Location: and that we should specify the location and the location it should contain it&state=xyz

error

- HTTP/1.1 302 Found
- Location: https://client.example.com/cb?error=access_denied&state=xyz
- REQUIRED. A single ASCII [USASCII] error code from the following:
- invalid_request
- The request is missing a required parameter, includes an
- invalid parameter value, includes a parameter more than
- once, or is otherwise malformed.
- unauthorized_
- The client is not authorized to request an authorization
- code using this method.
- access_denied
- The resource owner or authorization server denied therequest.
- unsupported_response_type
- The authorization server does not support obtaining an
- authorization code using this method.
- invalid_scope
- The requested scope is invalid, unknown, or malformed.
- server_error
- The authorization server encountered an unexpected condition that prevented it from fulfilling the reques
- temporarily_unavailable
- The authorization server is currently unable to handle the request due to a temporary overloading or maintenanceof the server. (This error code is needed because a 503
- Service Unavailable HTTP status code cannot be returned to the client via an HTTP redirect.)

error_description

- OPTIONAL. Human-readable ASCII [USASCII] text providing additional information, used to assist the client developer in

understanding the error that occurred. Values for the "error_description" parameter MUST NOT include

characters outside the set %x20-21 / %x23-5B / %x5D-7E.

- error_uri
- OPTIONAL. A URI identifying a human-readable web page with information about the error, used to provide the client developer with additional information about the error. Values for the "error_uri" parameter MUST conform to the URI-reference syntax and thus MUST NOT include characters outside the set %x21 / %x23-5B / %x5D-7E.
- state
- REQUIRED if a "state" parameter was present in the clientauthorization request. The exact value received from the client.

Implicit grant flow

- GET /authorize?response_type=token&client_id=s6BhdRkqt3&state=xyz&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb&state=xyz&scope=read
- Host: server.example.com
- the authorization server issues an access token and delivers it to the client by adding
- the following parameters to the fragment component of the redirectionURI:
- HTTP/1.1 302 Found
- Location: https://example.com/cb#access_token=2YotnFZFEjr1zCsicMWpAA&state=xyz&token_type=Bearer&expires_in=3600
- access_token
- REQUIRED. The access token issued by the authorization server.
- token_type
- REQUIRED. The value should be set to bearer
- expires_in
- RECOMMENDED. The lifetime in seconds of the access token
- scope
- REQUIRED, if identical to the scope requested by the client;
- otherwise, REQUIRED. The scope of the access token as described by Section 3.3.
- state
- REQUIRED if the "state" parameter was present in the client

authorization request. The exact value received from the client.

HTTP/1.1 302 Found
Location: https://client.example.com/cb#error=access_denied&state=xy

- error
- REQUIRED. A single ASCII [USASCII] error code from the following:
- invalid_request
- The request is missing a required parameter, includes an invalid parameter value, includes a parameter more than

once, or is otherwise malformed.

- unauthorized_client
- The client is not authorized to request an access token using this method.
- access_denied
- The resource owner or authorization server denied the request.
- unsupported_response_type
- The authorization server does not support obtaining an access token using this method.
- invalid_scope
- The requested scope is invalid, unknown, or malformed.
- server_error
- The authorization server encountered an unexpected condition that prevented it from fulfilling the request.
- (This error code is needed because a 500 Internal Server Error HTTP status code cannot be returned to the client
- via an HTTP redirect.)
- temporarily_unavailable
- The authorization server is currently unable to handle the request due to a temporary overloading or maintenance
- of the server. (This error code is needed because a 503 Service Unavailable HTTP status code cannot be returned
- to the client via an HTTP redirect.)
- Values for the "error" parameter MUST NOT include characters outside the set %x20-21 / %x23-5B / %x5D-7E.
- error_description
- OPTIONAL. Human-readable ASCII [USASCII] text providing additional information, used to assist the client developer in
- understanding the error that occurred. Values for the "error_description" parameter MUST NOT include
- characters outside the set %x20-21 / %x23-5B / %x5D-7E.
- error_uri
- OPTIONAL. A URI identifying a human-readable web page with
- information about the error, used to provide the client developer with additional information about the error.
- Values for the "error_uri" parameter MUST conform to the URI-reference syntax and thus MUST NOT include characters
- outside the set %x21 / %x23-5B / %x5D-7E.
- state
- REQUIRED if a "state" parameter was present in the client authorization request. The exact value received from the
- client.

/token

- The token request endpoint must support post with Content-Type: application/x-www-form-urlencoded
- the token request endpoint must authenticate the client making the request
- the token request end point must support basic authentication
- the token endpoint must ensure that the authorization code was issued to this client_ID
- the token endpoint must ensure that the authorization code is valid.
- the token endpoint must ensure that the authorization code is used ONLY once.
- authorization code must expire in 10s of seconds
- The token endpoint must set Cache-Control: no-store, Pragma: no-cache headers
- The token request endpoint supports the following parameters:
  - grant_type with value set to authorization_code, client_credentials, password, or refresh_token
  - code with its value set to the authorization code
  - redirect_URI with its value set to the redirect URI that was provided in the request for the authorization code
  - client_id; this value is required if the client is not authenticating with the authorization server
- The return payload must include the following:
  access_token, token_type, expires_in refresh_token, and any other key value pairs.

Exchange an authorization code for a token

- POST /token HTTP/1.1
  Host: server.example.com
  Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
  Content-Type: application/x-www-form-urlencoded
- grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSbIA
  &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
- HTTP/1.1 200 OK
  Content-Type: application/json;charset=UTF-8
  Cache-Control: no-store
  Pragma: no-cache

{
"access_token":"2YotnFZFEjr1zCsicMWpAA", "token_type":“Bearer", "expires_in":3600,
"refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
"example_parameter":"example_value"
}

Refreshing the access token

- POST /token HTTP/1.1
  Host: server.example.com
  Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
  Content-Type: application/x-www-form-urlencoded

- grant_type=refresh_token&refresh_token=tGzv3JOkF0XG5Qx2TlKWIA

Methodology for securing rest API

- Client app registers with Oauth/Authorization Server
- Client app request a token
- oAuth provider generates an access token to client APP
- Client app includes access token in every HTTP request using Authorization header
- Client app sets the Authorization header to Bearer {access token}
- The rest API validates the access token
  - What does it need to validate the token?

Token Validation by Resource Server

- 1. Validate the structure of a JWT
- 2. Create an “allow list” that contains valid values for iss claims
- 3. Base64decode JWT header, payload
- 4. Retrieve alg and kid from Header
- 5. Retrieve iss from payload
- 6. Compare the value of iss to that stored in the “allowed list”

5. If iss value in allow list, use JWKS to retrieve public key. Otherwise, signature invalid

6. Verify signature

7. Validate any other claims such as scope, aud, exp, etc.

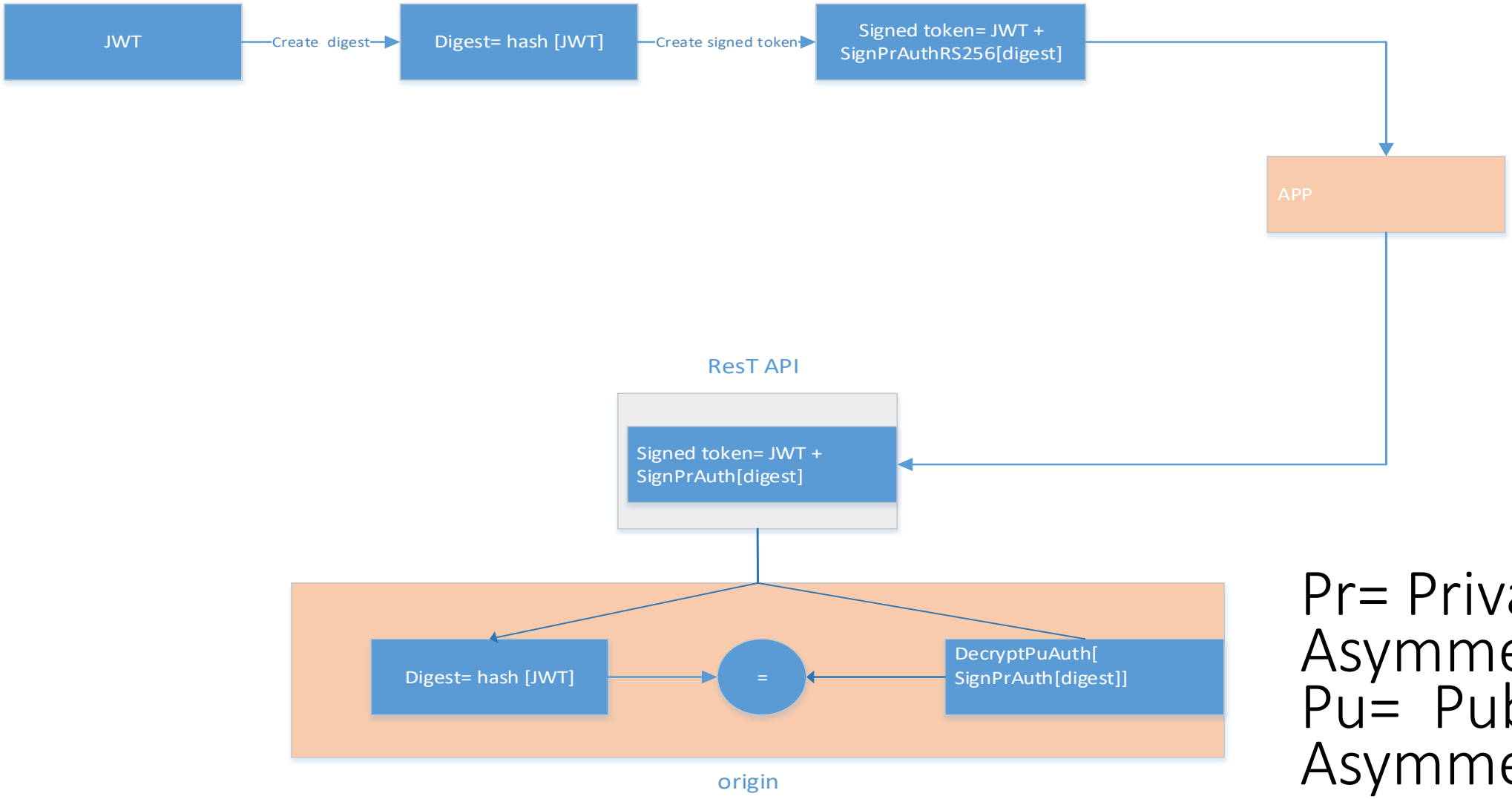
JWT example

```
{
  "typ": "JWT"
}



{
  "app": "TEST",
  "acc": "7888-a9a0-4de2-be72-57775575",
  "iss": "yyy",
“scope”:[”read,write”]
  "exp": 1561939073,
  "jti": "jhhhjhg-6cab-lkjjll-8512-kjkkjk“,
“aud”:”/plan/{id}”
}
RSASHA256( base64UrlEncode(header) + "." + base64UrlEncode(payload)
```

Signature Verification using RS 256



Pr= Private key of Asymmetric key
Pu= Public key of Asymmetric key

Key Distribution

- When using RS 256:
  - Generate a public/private key pair
  - Signer uses the private key to sign the token
  - Rest API uses the public key to verify the signature
  - Rest API must have access to the public key
    - JWK : https://tools.ietf.org/html/rfc7517

Key Rotation (Private) and Distribution (Public)

- Using Kid
- {
- "alg": “RS256",
- "typ": "JWT",
- "kid":"2",
- ~~"jku":https://myjwks ;;; Not recommended to include this~~
- }

References for token signing

- https://connect2id.com/products/nimbus-jose-jwt/examples/jwt-with-rsa-signature
- https://en.wikipedia.org/wiki/JSON_Web_Token
- https://tools.ietf.org/html/rfc7519
- https://developers.google.com/oauthplayground/
- https://developers.google.com/identity/protocols/oauth2/openid-connect
- https://console.developers.google.com/apis/credentials?project=vital-invention-306022
- https://accounts.google.com/.well-known/openid-configuration
- JWT.io
- https://developers.google.com/identity/protocols/oauth2/openid-connect