KONTROLA DOSTĘPU

Waldemar Korłub

Aplikacje i Usługi Internetowe
KASK ETI Politechnika Gdańska

Kontrola dostępu

- W aplikacja internetowych typowo występuje potrzeba rozróżnienia dostępnych funkcji i zasobów w zależności od użytkownika, np.:
  - Administrator sklepu może edytować produkty
  - Zalogowany użytkownik może składać zamówienia
  - Niezalogowany użytkownik może tylko przeglądać ofertę i dodawać produkty do koszyka
- Kluczowe aspekty kontroli dostępu:
  - Uwierzytelnianie
  - Autoryzacja

Uwierzytelnianie

- Weryfikacja tożsamości użytkownika
- Polega na sprawdzeniu informacji, którą z dużym prawdopodobieństwem zna tylko dany użytkownik:
  - Hasło
  - Token dostępowy wysłany w wiadomości SMS
  - Treść wygenerowana z użyciem klucza prywatnego (kryptografia asymetryczna)
- Uwierzytelnianie dwuetapowe/wieloetapowe
  - Wykorzystanie więcej niż jednej informacji
  - np. potwierdzanie przelewów w banku internetowym

Autoryzacja

- Nadanie uprawnień do wykonywania operacji i dostępu do zasobów
- Najczęściej w oparciu o ustaloną wcześniej tożsamość użytkownika
- Wykorzystuje często metadane przypisane użytkownikowi, np.:
  - Przynależność do grup
  - Przypisane role
  - Nadane uprawnienia

Kontrola dostępu oparta o role

- RBAC (ang. *role-based access control*)
- Dostęp do zasobu lub operacji wymaga posiadania określonej roli
- Każdemu użytkownikowi można przypisać wiele ról
  - …a jedną rolę wielu użytkownikom
- Role odzwierciedlają rzeczywiste funkcje użytkownika w organizacji/firmie
  - np. księgowa, klient, magazynier, administrator
- Nie przypisuje się uprawnień bezpośrednio do użytkowników

Kontrola dostępu oparta o role

- W najprostszym podejściu role są przypisywane bezpośrednio do użytkowników
  - Wystarczające w mniejszych projektach
- W aplikacjach korporacyjnych dużej skali często występują hierarchiczne modele uprawnień
  - np. kohorty → grupy → role
  - Umożliwiają odzwierciedlenie struktury organizacyjnej klienta

Użytkownicy

- Informacje o tożsamości użytkowników mogą być zgromadzone m.in.:
  - w bazie danych
  - na serwerze LDAP
  - w usłudze Active Directory
  - w pliku na dysku
  - u zewnętrznego dostawca tożsamości (np. Google, Facebook)

Użytkownicy

- Kontrola dostępu opiera się o bazę użytkowników aplikacji
- Baza musi zawierać informacje umożliwiające uwierzytelnienie użytkownika, np.:
  - Skrót hasła (hash)
  - Numer telefonu (token dostępowy z wiadomości SMS)
  - Klucz publiczny (kryptografia asymetryczna)
- Najczęściej spotykany sposób uwierzytelniania użytkowników wykorzystuje parę: login + hasło

Hasła użytkowników

- W idealnym świecie użytkownicy aplikacji…
  - …wybierają długie hasła
  - …nie stosują wyrażeń słownikowych
  - …używają dużych i małych liter, cyfr i innych znaków
  - …nigdy nie zapisują haseł na kartkach
    - np. pod klawiaturą, w portfelu, przyklejone do monitora
  - …nigdy nie używają tego samego hasła w różnych serwisach
  - …regularnie zmieniają swoje hasła

Hasła użytkowników

- W rzeczywistym świecie nie każdy użytkownik trzyma się powyższych zasad…
- …a wymuszanie zbyt restrykcyjnej polityki haseł może przynieść skutki odwrotne do oczekiwanych

- Autorzy aplikacji i administratorzy powinni kompensować błędy swoich użytkowników
  - Albo przynajmniej się starać

Przechowywanie haseł użytkowników

- Nie należy przechowywać haseł w postaci jawnej (ang. *plaintext*)
  - Włamanie do bazy → natychmiastowy dostęp do kont użytkowników
  - Szczególnie groźne dla użytkowników wykorzystujących to samo hasło w wielu serwisach
- Należy wykorzystywać funkcje hashujące (mieszające)
  - Skrót hasła nie powinien umożliwiać łatwego odtworzenia oryginalnego hasła, np. dla hasła p@ssw0rd:
  - `$2y$10$8Bc9/72wX4TRsMoZcCyF/.7f0s5gteQf4ioarhkzGcikb2iFT03/C`

API Access Control

Who is involved?

- User
- User's client device – smartphone/tablet/laptop
- Authorization server
- Resource server

Key aspects of access control

- Acquisition
  - How to obtain the security information required for access control?
- Verification
  - How the obtained security information if verified?
- Authorization
  - How to establish user's privileges based on the provided security information?
- Preservation
  - How to keep the user logged in? How to store the required security information on user's device?
- Revocation
  - How to revoke the access to the service when mobile device is stolen or lost?

BASIC authentication

- HTTP BASIC
- Login and password attached to every request
- Acquisition:

```
login:password
```

BASE64

```
dXNlcjpwYXNzd29yZA==
```

```
Authorization: Basic dXNlcjpwYXNzd29yZA==
```

BASIC authentication

- Simplicity
- No separate authentication phase before calling actual operations in the API
- All requests handled the same way
- Stateless
- Easy horizontal scalling

Preservation and revocation

- Preservation: how to keep the user logged in?
  - Credentials have to be stored on the device
  - How to store the password?
    - Plaintext
    - Hashed
    - Encrypted
      - Symmetric
      - Asymmetric
- Revocation: How to revoke the access to the service when mobile device is stolen or lost?

BASIC + session

- If the password is attached to every request
  - …an attacker needs to catch only one of them to impersonate user
  - …the password needs to be stored by client application
- Another approach is to use BASIC only for the first request, which opens a *session*
  - Subsequent requests belong to the same session
  - Subsequent requests contain only the session id and not the password
  - Disadvantage: back-end application is no longer stateless

Access token

- Mobile app provides user's credentials to request an access token
- Once the token is obtained it is attached to subsequent requests
- Does it affect scalability compared to BASIC auth?
- Does it affect statelessness compared to BASIC auth?

- Access token is like a key to a lock

Preservation and revocation

- How to keep the user logged in?
  - Access tokens have to be stored on the device
  - How to store the access token?
    - Plaintext
    - Hashed
    - Encrypted
      - Symmetric
      - Asymmetric
- How to revoke the access to the service when mobile device is stolen or lost?

Access token + refresh token

- Mobile app provides user's credentials to request a refresh token
- Refresh token is used to obtain access token
- Access token is attached to subsequent requests until it expires

- Access token resembles a session on a website (but a stateless one!)
- Refresh token resembles a password on a website

Access token + refresh token

- How does it improve security?
  - Access tokens are short-lived (e.g. expiration date)
    - Stolen token is only usable for a short period of time
    - …which is an improvement on its own
  - Without refresh token the app would have to use user's credentials to obtain new access token now and then
    - Refresh token is easier to revoke
  - If an attacker can steal the access token he can steal the refresh token as well
    - …but he needs to sniff many more requests!
    - Mobile device can switch networks before an attacker even has a change to steal the refresh token

Preservation and revocation

- How to keep the user logged in?
  - Refresh token have to be stored on the device
  - How to store the refresh token?

- How to revoke the access to the service when mobile device is stolen or lost?

Bearer token

- Bearer token is a self-contained access token
- It contains information about bearer privileges
- Resource server does not need to query authorization server for privileges
  - Why is it important?

- Bearer token is like cash
  - If you find cash on the street you can pick it up take it to a shop and spend it
  - The clerk will not ask who you are or how you got the cash
  - The clerk will only check if the banknote is genuine or not

How to check if bearer token is genuine?

- Real-world banknotes contain:
  - Information about banknote issuer
  - Special imprints to confirm their genuineness
- Bearer tokens should be digitally signed by their issuer
  - e.g. with a private-key or a secret
- Signature guarantees token integrity
  - e.g. privileges cannot be modified
- Private-key based signature additionally guarantees:
  - Issuer identity
  - Undeniability

} both can be verified with the public key!

Bearer tokens

- Bearer tokens must have an expiration date
- Bearer tokens cannot be revoked other than by expiring
  - Why is that?
- Bearer tokens should be used in conjunction with refresh tokens

Preservation and revocation

- How to keep the user logged in?
  - Refresh token have to be stored on the device
  - How to store the refresh token?

- How to revoke the access to the service when mobile device is stolen or lost?

Bearer tokens

- Several popular formats
  - Simple Web Token – SWT
  - Security Assertion Markup Language – SAML Token
  - JSON Web Token – JWT (pronounced as “jot”)

JSON Web Tokens – BASE64 encoded

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ
9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZ
SI6ImdhbmRhbGYiLCJhZG1pbiI6dHJ1ZX0.
NUB0OfUgBGL2N2j9kqYUgu3pOL92zvMbX_g
DynvLYo4

JSON Web Tokens


```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Header:
algorithm type, token type

```
{
  "sub": "1234567890",
  "name": "gandalf",
  "admin": true
}
```

Body:
claims about permissions

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  "secret")
```

Signature:
hash-based message authentication code

How to pass tokens?

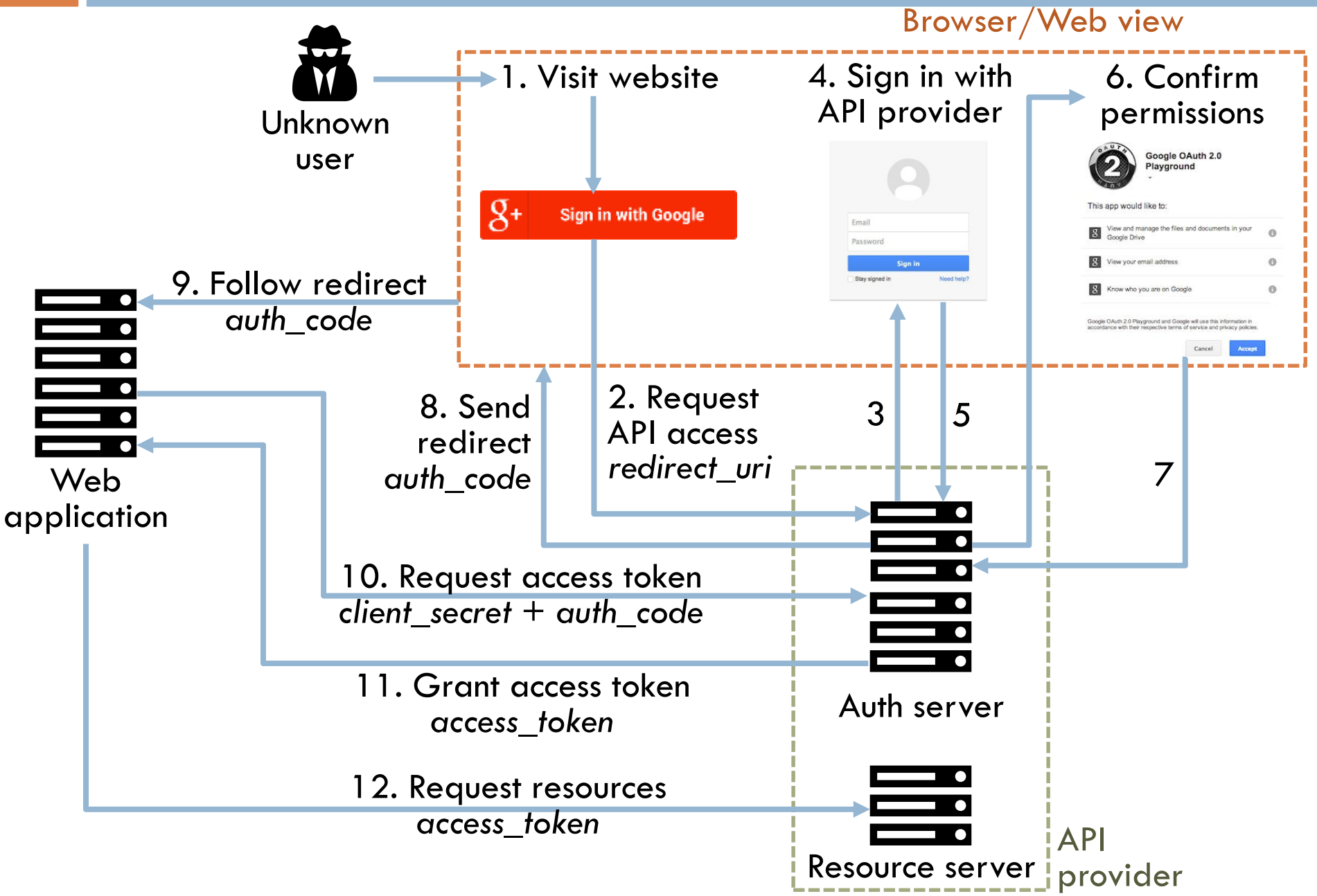
- By value
- By reference
  - a *reference* pointing to a *token*
    - e.g. GUID
    - It can be signed as well
  - The client app does not receive the actual token – it only receives a reference
  - Dereferencing happens at the API firewall
  - The identity data never leaves provider's network

OAuth 2.0

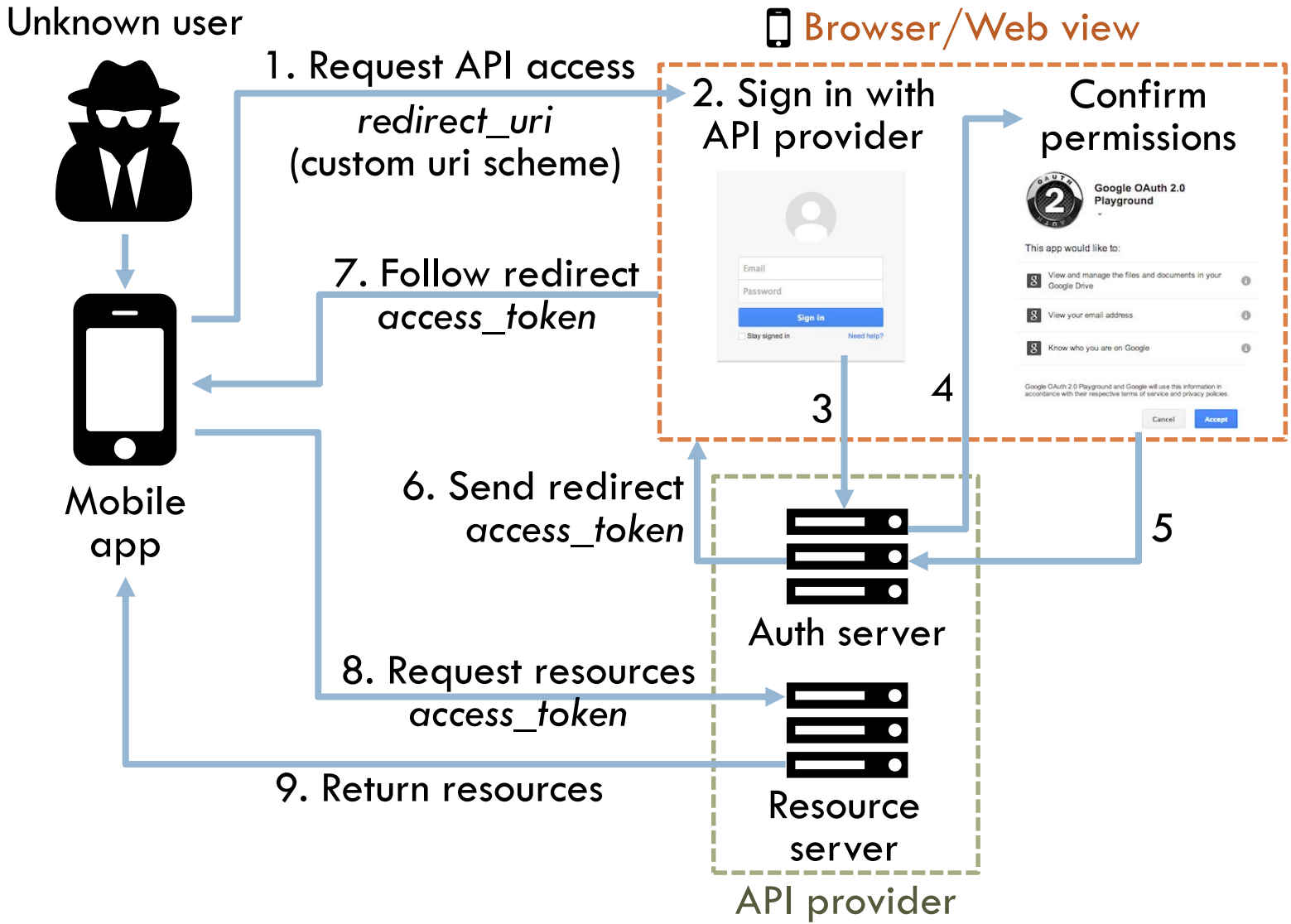
- It is not about **auth**entication
- Nor is it about **auth**orization

- So what is it really about?

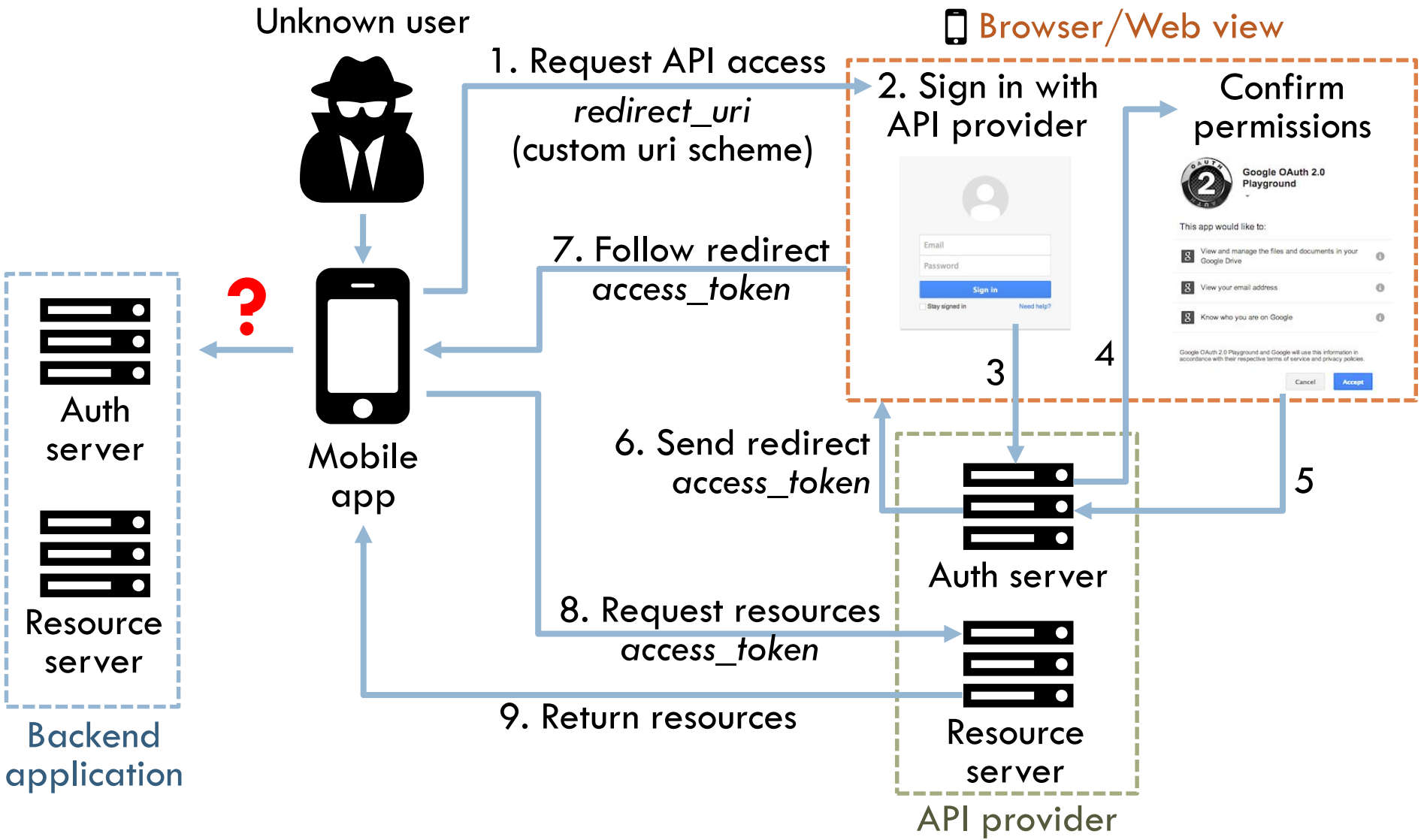
OAuth – authorization code grant type



OAuth – implicit grant type



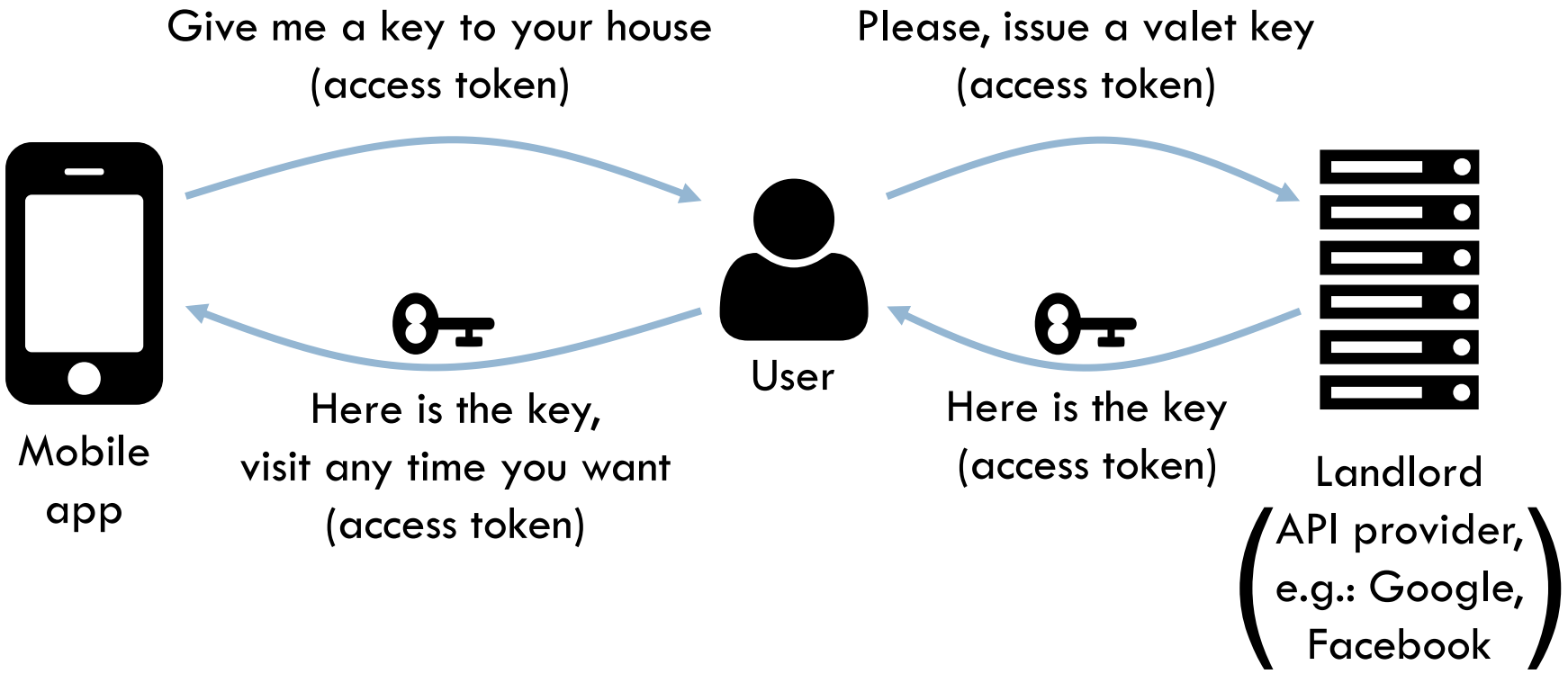
OAuth – implicit grant type



OAuth

- What is it about?
- Mobile app did not receive any information about user's identity during OAuth flow

Pseudo-Authentication with OAuth



OAuth

- OAuth is about *delegation of access*
  - Doing things on others behalf
  - Delegation doesn't equal authorization!

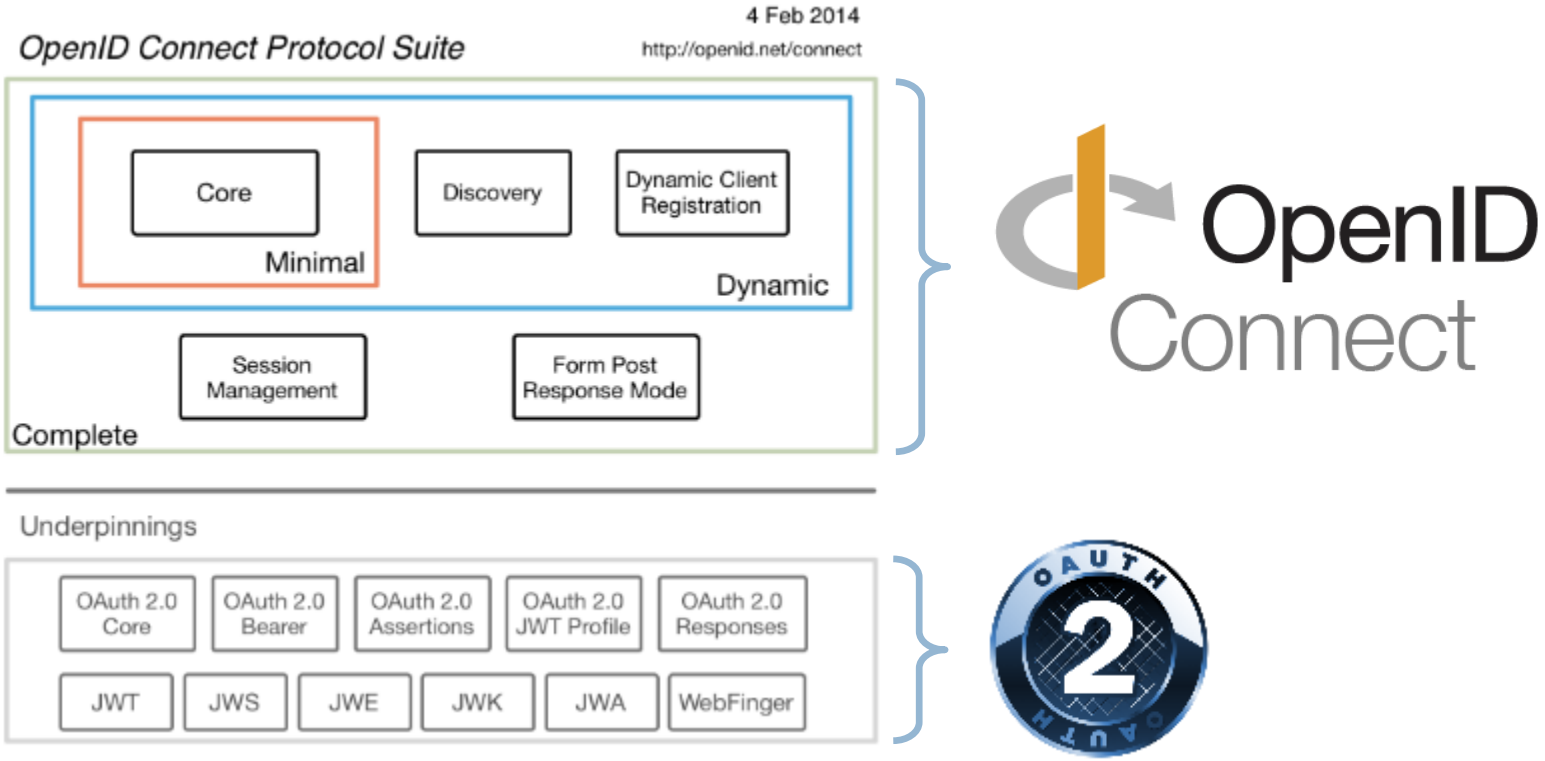
- If you use OAuth for authentication or authorization you're doing it wrong
  - Doing something wrong in regard to security is a straight way to a disaster

Pseudo-Authentication with OAuth

- Received access token can be used to invoke operations on user's behalf
- Mobile app can request identity information on user's behalf!
  - But what kind of data should be requested so that:
    - It can be trusted,
    - It can be securely passed to the backend server,
    - It can be verified on the backend server?

OpenID Connect

- Released in February 2014
  - Already deployed by Google, Microsoft, PayPal…
- Identity layer on top of OAuth 2.0



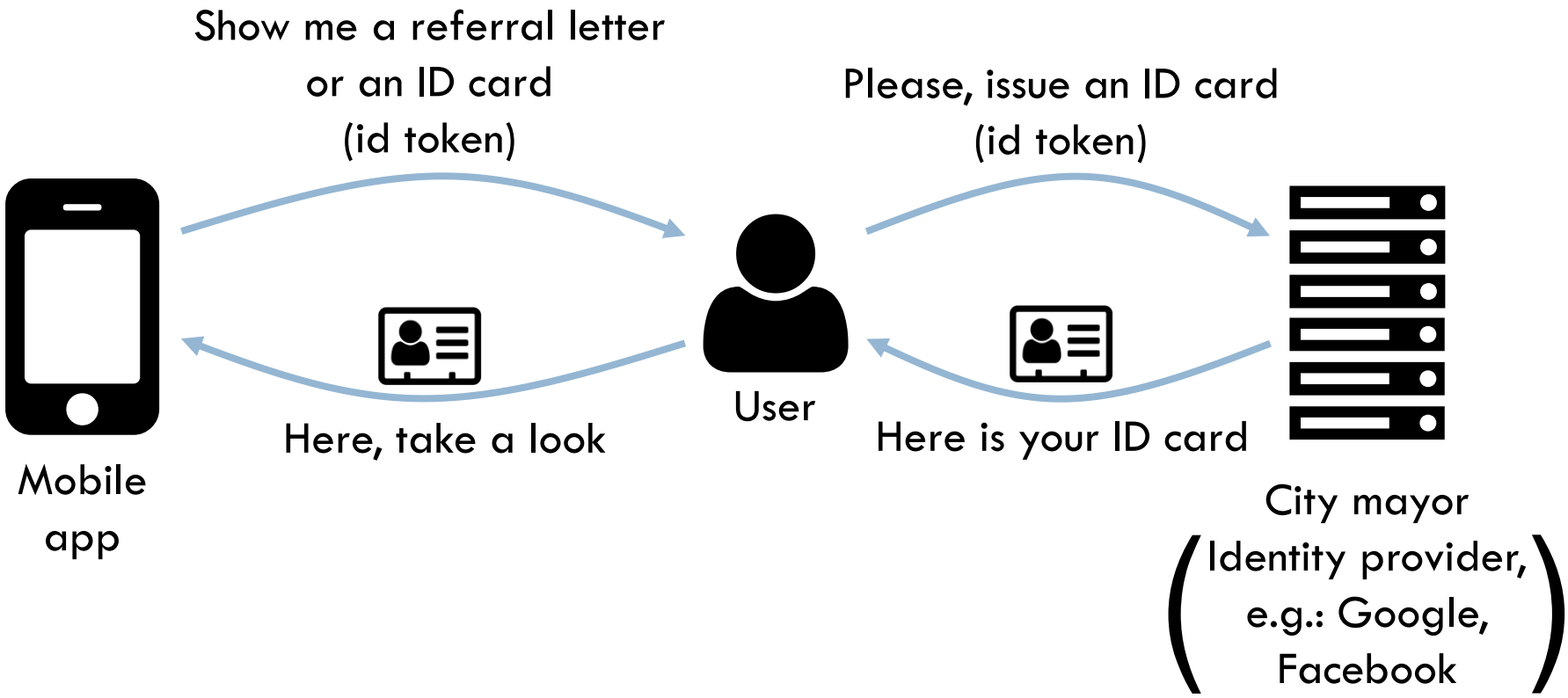
OpenID Connect: ID token

- JWT token
- Claims:
  - iss – issuer
  - sub – identifier of subject (user)
  - aud – audience for ID Token
  - iat – time token was issued
  - exp – expiration time
  - nonce – mitigates replay attacks
  - at_hash – access token hash
  - azp – authorized party

OpenID Connect: ID token

- ID token is kind of like my University ID card
  - iss – issuer: **Gdańsk University of Technology**
  - sub – identifier of subject (user): **Waldemar Korłub**
  - aud – audience for ID Token: **GUT staff members**
  - iat – time token was issued: **2015-10-01**
  - exp – expiration time: **2018-09-30**
  - nonce – mitigates replay attacks
  - at_hash – access token hash
  - azp – authorized presenter: **Waldemar Korłub**
    - Needed when the party requesting the ID token is not the same as the audience of the ID token

Legitimate authentication with an external identity provider



Authentication with a backend server

- By using OpenID Connect mobile app obtains user's ID token (issued by the Identity Provider)
- How to transfer user's identity to a backend server (our own server-side application)?

- Do **not** accept plain user IDs (the "sub" claim) on the backend server!
  - Malicious client app can send arbitrary user IDs to impersonate users

Authentication with a backend server

- Mobile app should send ID token to the backend server using safe HTTPS connection
- Backend server needs to ensure that the following criteria are satisfied:
  - The ID token was issued and signed by the expected issuer
    - Verify ID token integrity at the same time
  - The audience of the ID token contains the backend server
  - The expiration time (the "exp" claim) has not passed
- If criteria are satisfied the "sub" claim denotes user's identity

Authorization on a backend server

- By using OpenID Connect the backend server can obtain user's identity
  - This solves the problem of authentication (and authentication only)
- What about authorization of access to backend server's resources?
  - Which resources the user is allowed to read?
  - What operations the user is allowed to perform on those resources?
  - OpenID Connect does not answer these questions
    - …and it was never intended to do that!

Authorization on a backend server

- We need to use a separate mechanism for authorization, e.g.
  - Check user's permissions on every request
    - Based on the user identity provided by OpenID Connect
    - Same as we did with HTTP BASIC or access tokens issued by our own backend server
  - Issue our own bearer token (e.g. JWT) for the client device containing permissions granted to the authenticated user to resources on our server

OpenID Connect: ID token

- Acquisition
  - Obtain ID token using OpenID Connect protocol on top of OAuth
- Verification
  - Check criteria listed on the previous slide
- Preservation
  - Use ID tokens together with refresh tokens
- Revocation
  - ID tokens – revocation by expiration
  - Refresh tokens – revocation by invalidation

OAuth 2.0

- Other flows:
  - password flow
    - For trusted client applications (1st party)
    - Username/password is gathered by client application
      - No need for redirection to an external website
  - refresh_token flow
    - For obtaining new access/bearer token when the previous one expires

Kontrola dostępu w Angularze

Kontrola dostępu w Angularze

- Przykładowa aplikacja
  - OAuth 2.0 password flow (auth.service.ts)
  - Interceptor (auth-interceptor.ts)

Kontrola dostępu w Angularze

- Kontrola dostępu musi obowiązkowo odbywać się po stronie back-endu
  - Jeśli użytkownik nie ma uprawnień do zasobu, serwer zwraca odpowiedzi 401/403
- Kontrolę dostępu typowo uwzględnia się też po stronie front-endu
  - Zabezpieczenie dostępu do wybranych widoków bez konieczności odpytywania serwera

Strażnicy (ang. *guards*)

- Obiekty kontrolujące możliwość aktywacji widoków, np.:

```
@Injectable()
export class AuthenticatedGuard implements CanActivate {
  constructor(private identityService: IdentityService,
              private router: Router) {
  }

  canActivate(route: ActivatedRouteSnapshot,
              state: RouterStateSnapshot): boolean {

    if (this.identityService.getUserInfo() === null) {
      this.router.navigate(['welcome']);
      return false;
    }

    return true;
  }
}
```

Strażnicy (ang. *guards*)

- Klasy strażników należy zadeklarować w tablicy *providers* w obrębie adnotacji @NgModule
- Następnie można dołączyć je do reguł routingu:

```
const routes: Routes = [
  {
    path: 'sign-in’,
    component: SignInComponent
  },
  {
    path: 'dashboard’,
    component: DashboardComponent,
    canActivate: [AuthenticatedGuard]
  },
];
```

Pytania?