Recent evolutions in the OAuth 2.0 and OpenID Connect landscape

Dr. Philippe De Ryck

https://Pragmatic Web Security.com

Dr. Philippe De Ryck

- Deep understanding of the web security landscape
- Google Developer Expert (not employed by Google)
- Course curator of the SecAppDev course (https://secappdev.org)

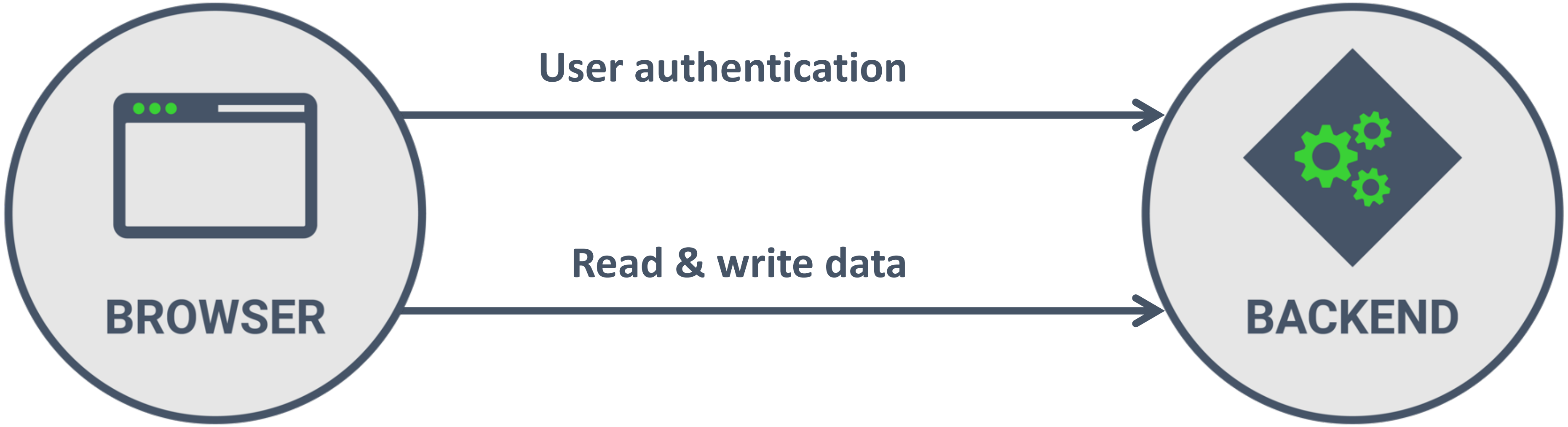
Pragmatic Web Security

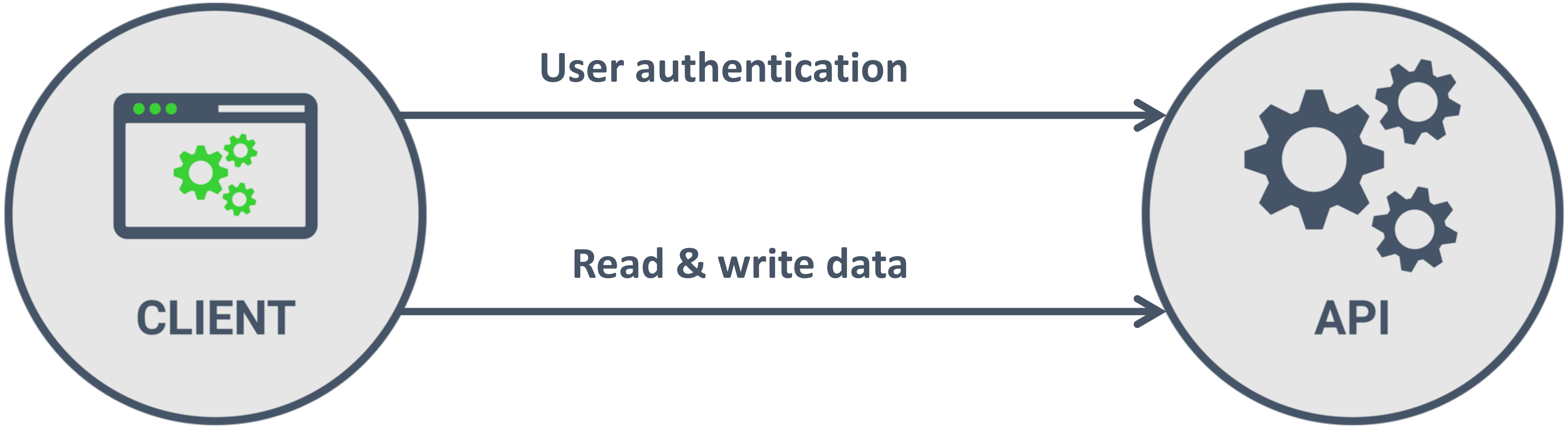
High-quality security training for developers and managers

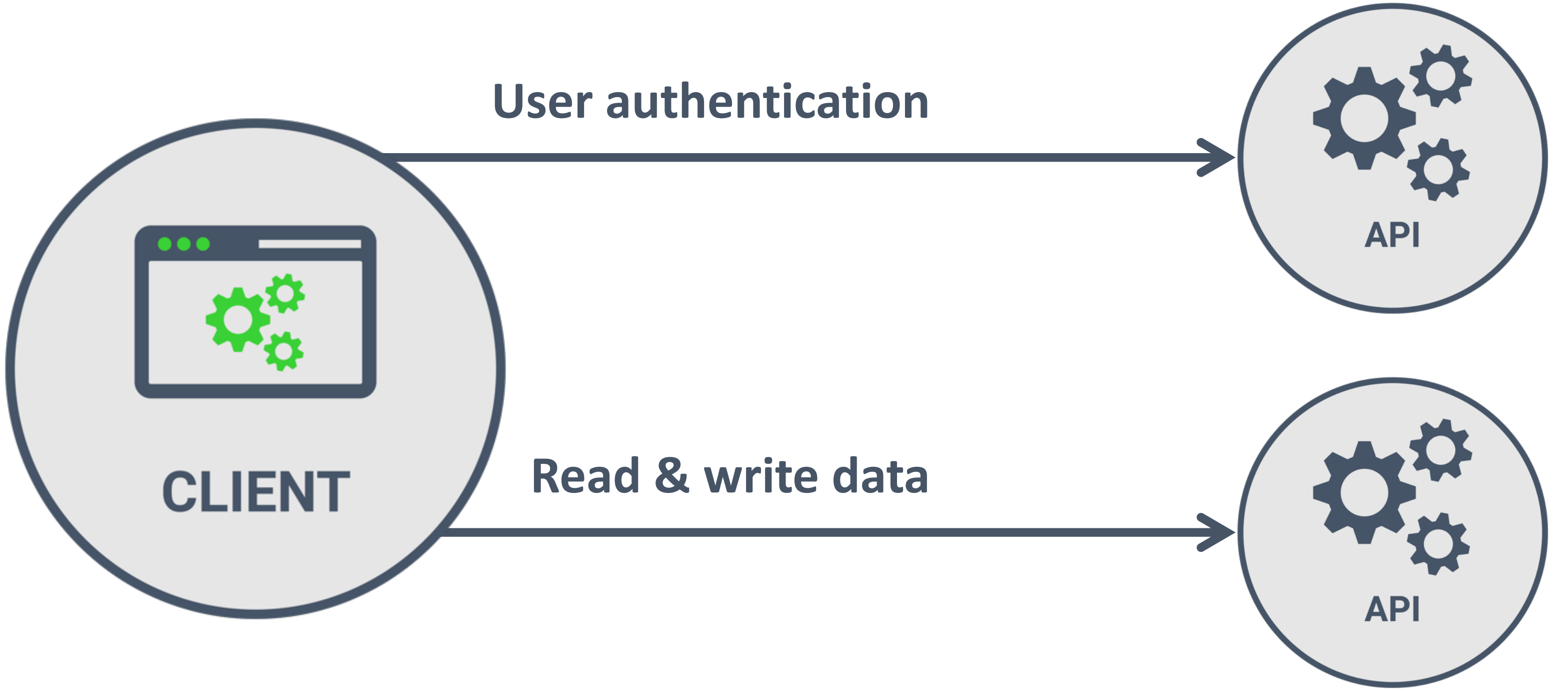
Custom courses covering web security, API security, Angular security, …

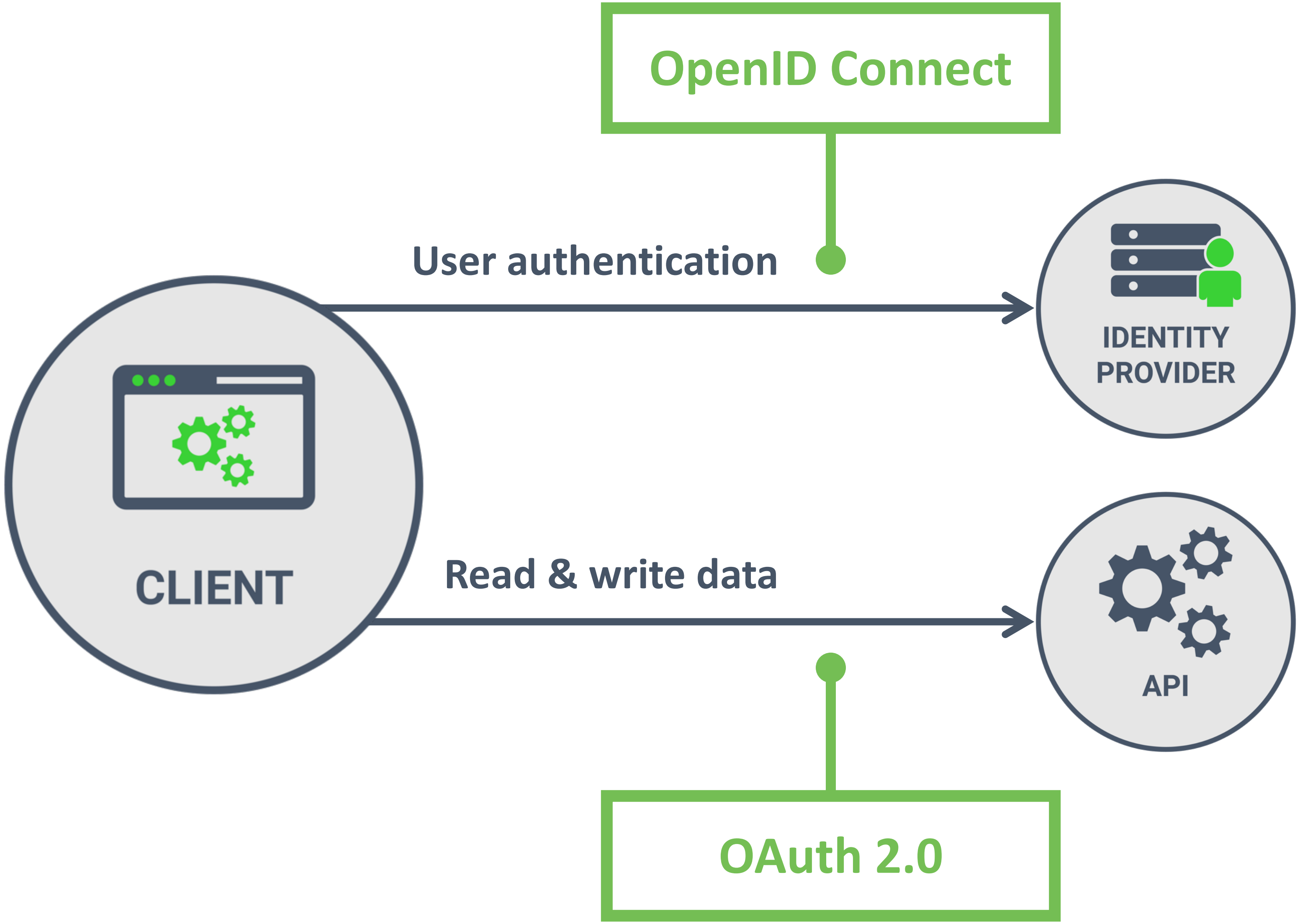
Consulting services on security, Oauth 2.0, OpenID Connect, …

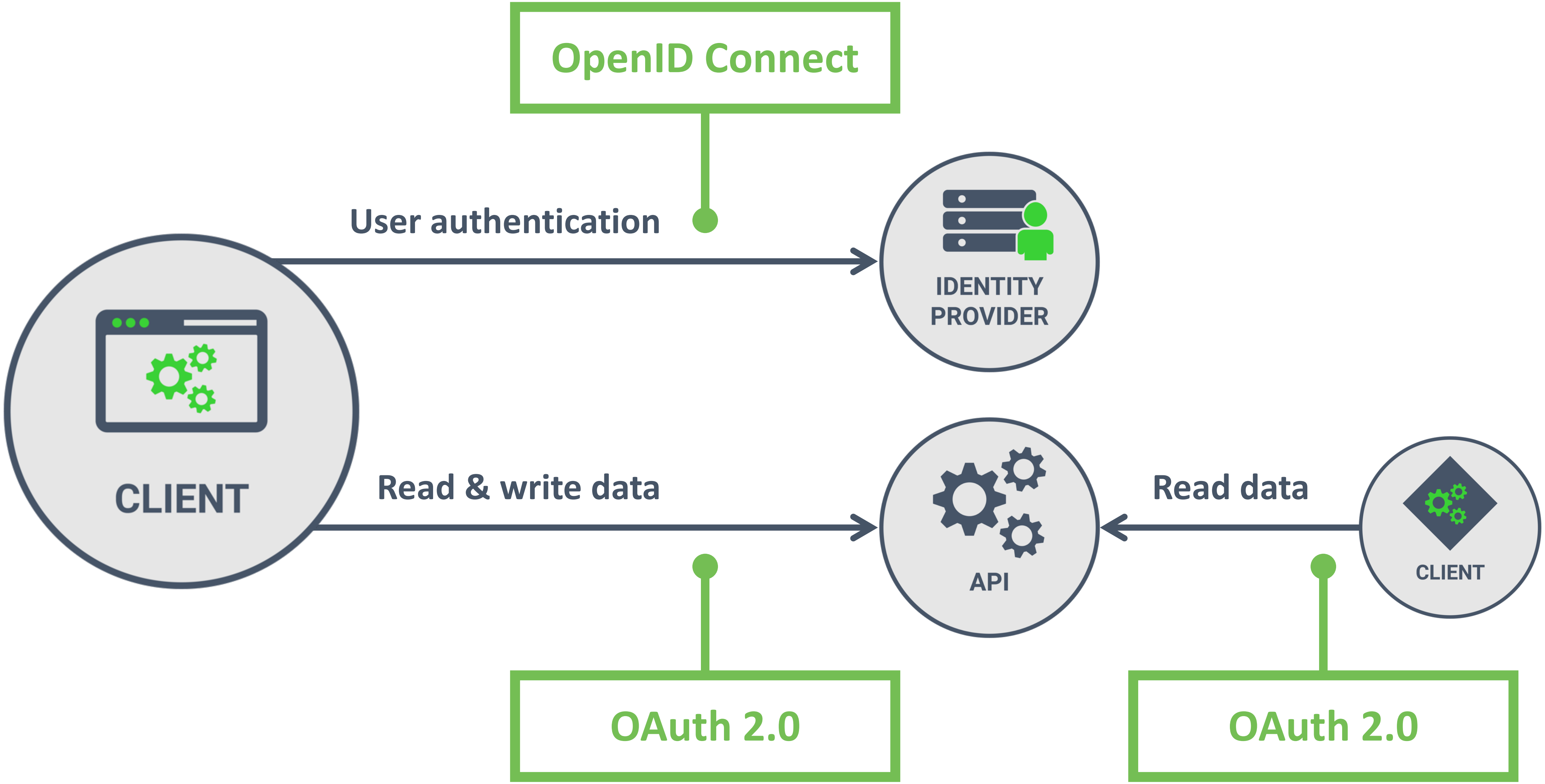
@PhilippeDeRyck
https://PragmaticWebSecurity.com


User authentication
Read & write data
BROWSER
BACKEND


User authentication
Read & write data
CLIENT
API


User authentication
CLIENT
Read & write data
API
API


OpenID Connect
User authentication
IDENTITY PROVIDER
CLIENT
Read & write data
API
OAuth 2.0



OpenID Connect

User authentication

IDENTITY PROVIDER

CLIENT

Read & write data

API

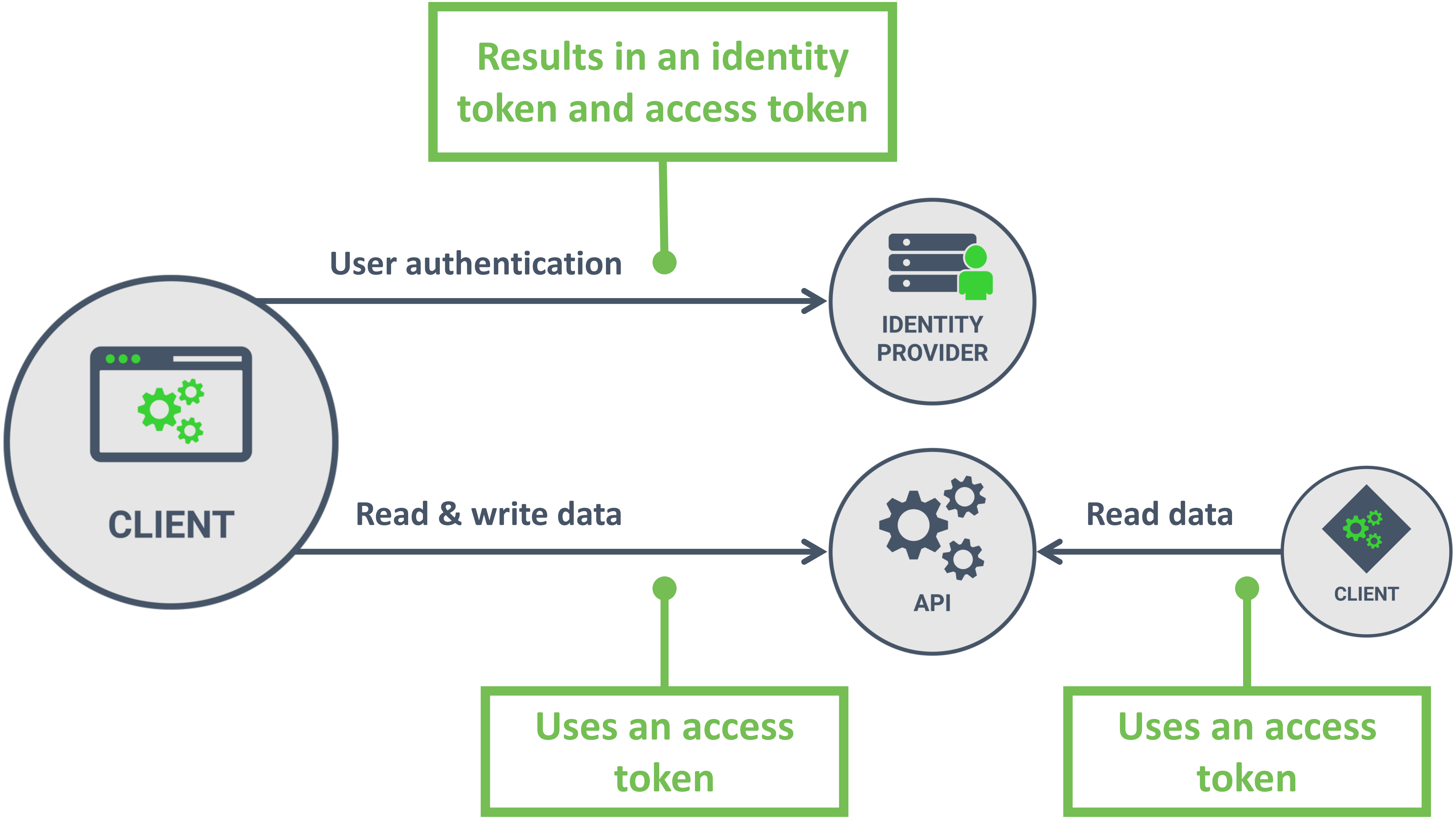
Read data

CLIENT

OAuth 2.0

OAuth 2.0

@PhilippeDeRyck


Results in an identity token and access token
User authentication
IDENTITY PROVIDER
CLIENT
Read & write data
API
Read data
CLIENT
Uses an access token
Uses an access token

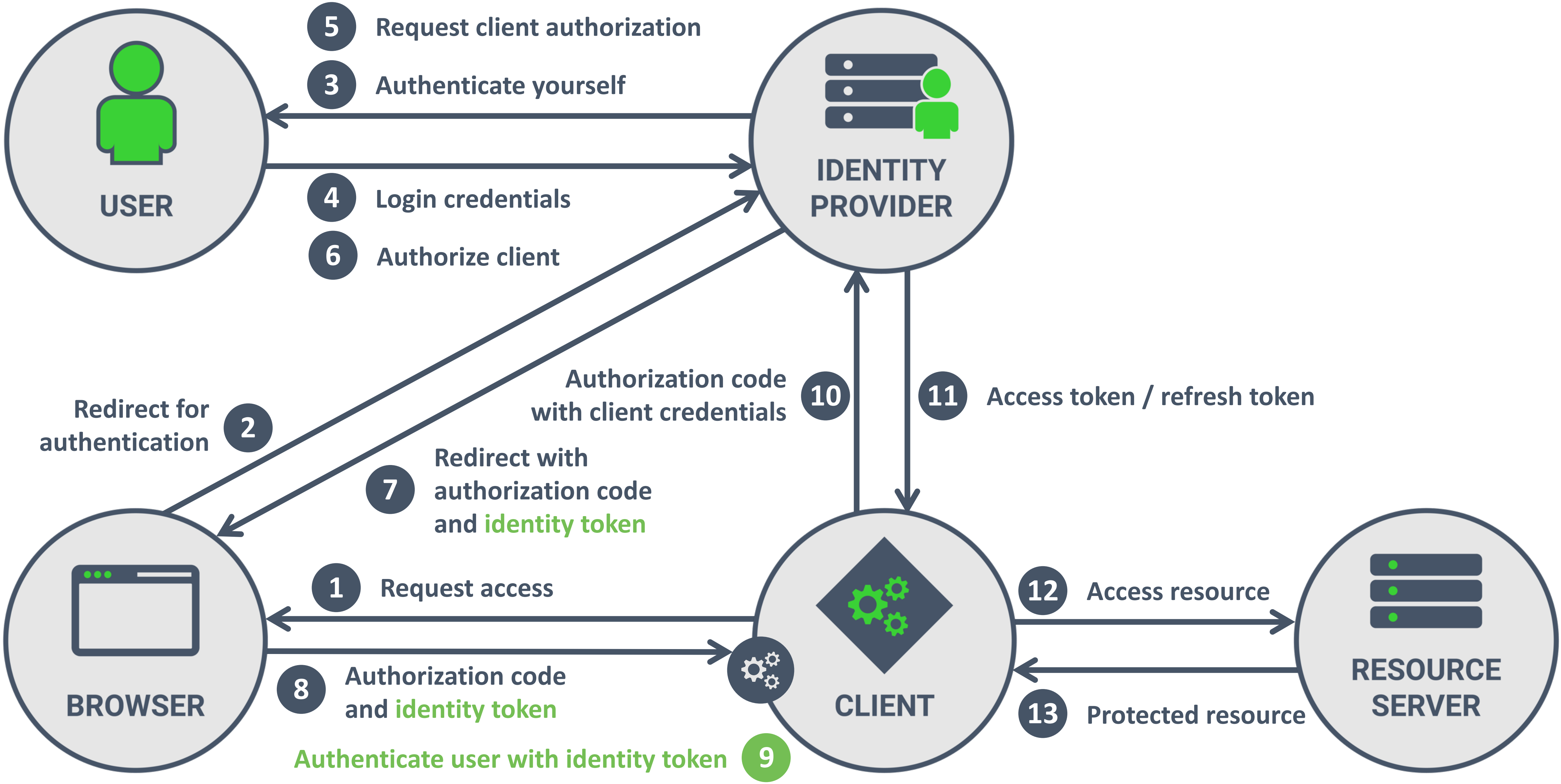
OAuth 2.0 and OpenID Connect

OpenID Connect provides user authentication

OAuth 2.0 allows a client to access resources on behalf of the user

Modern applications use a combination of both protocols

THE OIDC HYBRID FLOW



USER

IDENTITY PROVIDER

BROWSER

CLIENT

RESOURCE SERVER

1 Request access

2 Redirect for authentication

3 Authenticate yourself

4 Login credentials

5 Request client authorization

6 Authorize client

7 Redirect with authorization code and identity token

8 Authorization code and identity token

9 Authenticate user with identity token

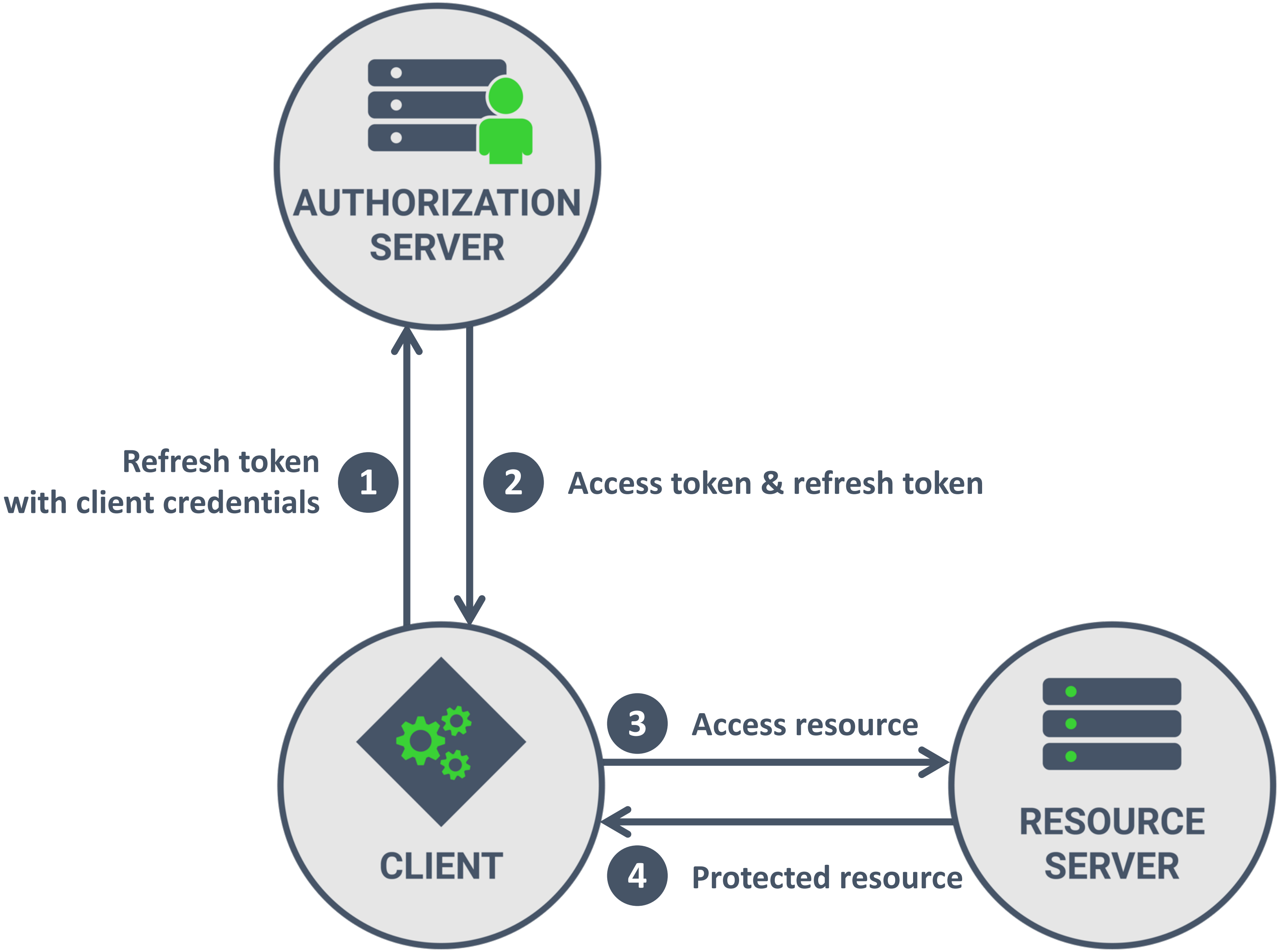
10 Authorization code with client credentials

11 Access token / refresh token

12 Access resource

13 Protected resource

THE REFRESH TOKEN FLOW

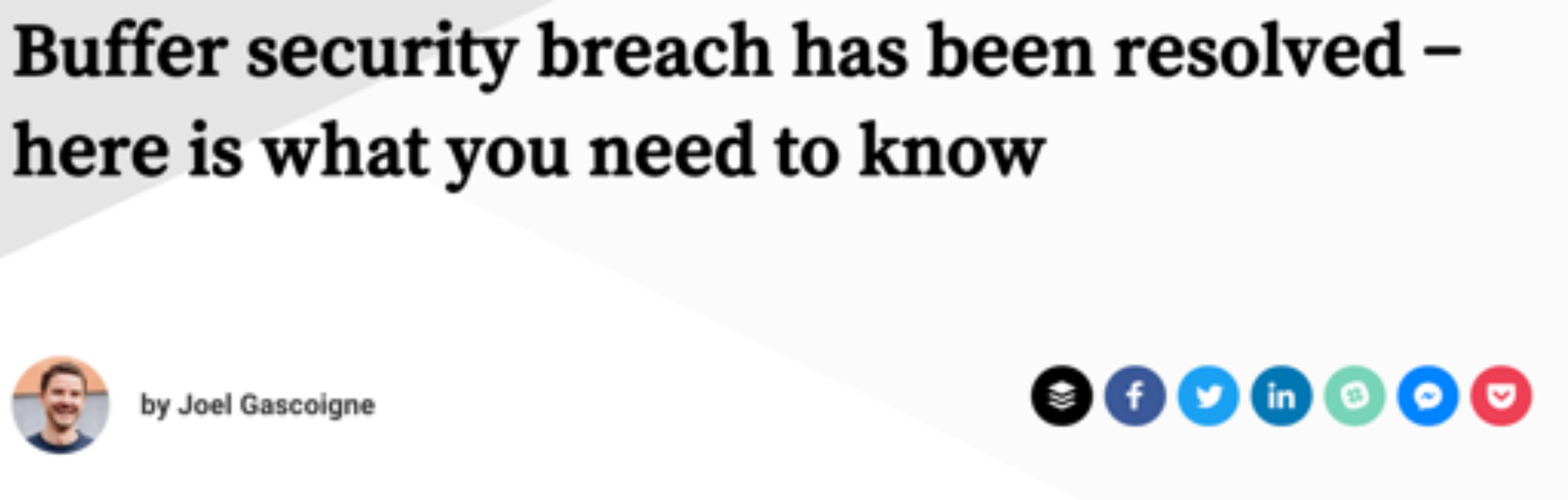


THE OIDC HYBRID FLOW

- Clients are backend applications running in a "secure" environment

- The hybrid flow returns an identity token, access token and refresh token
  - Identity tokens are issued through the frontchannel, along with an authorization code
  - The authorization code can be exchanged for an access token and refresh token
  - Using the authorization code requires client authentication

- Refresh tokens allow the client to obtain a new access token
  - Using a refresh token requires client authentication

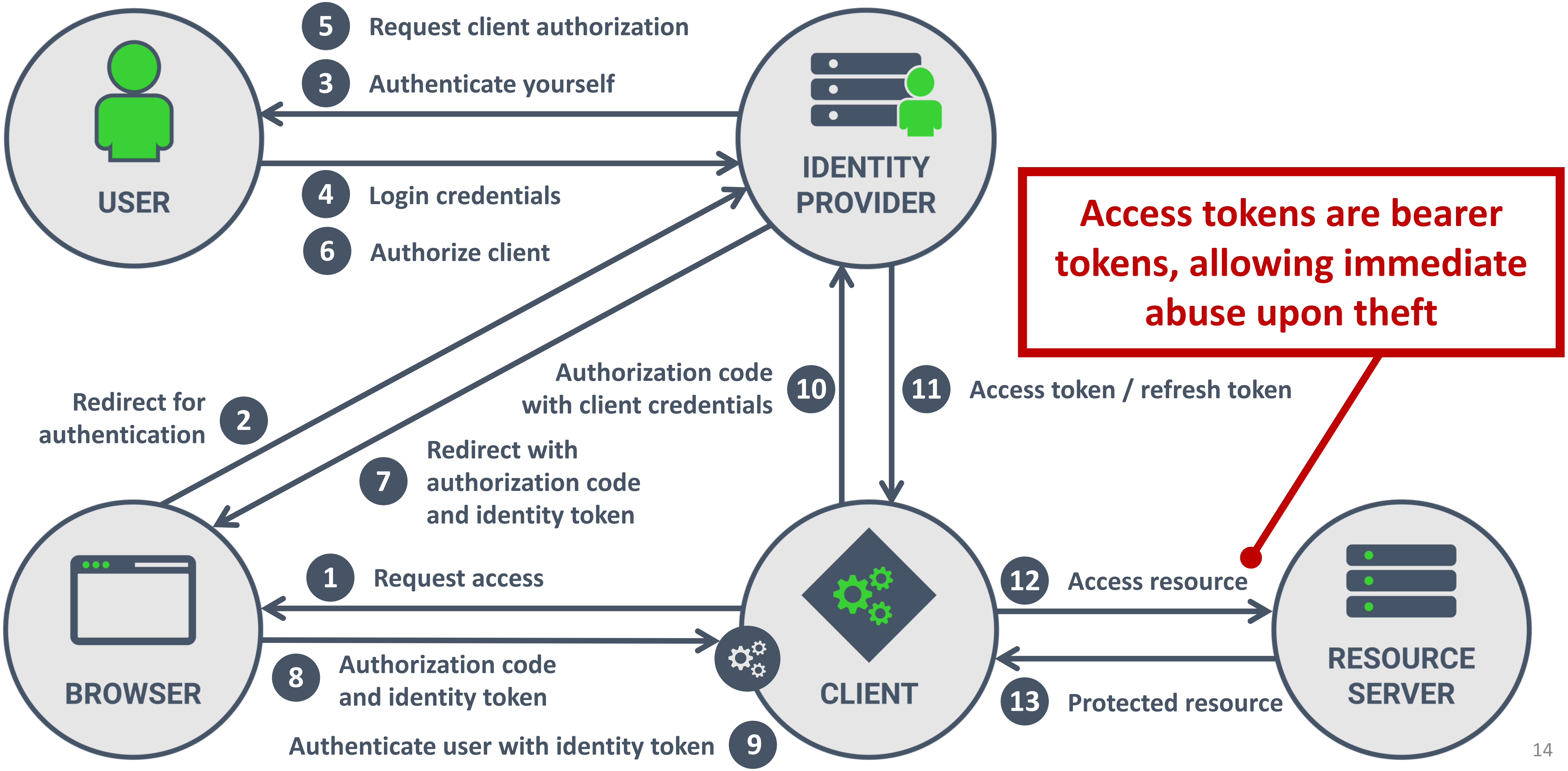


Buffer security breach has been resolved – here is what you need to know

by Joel Gascoigne

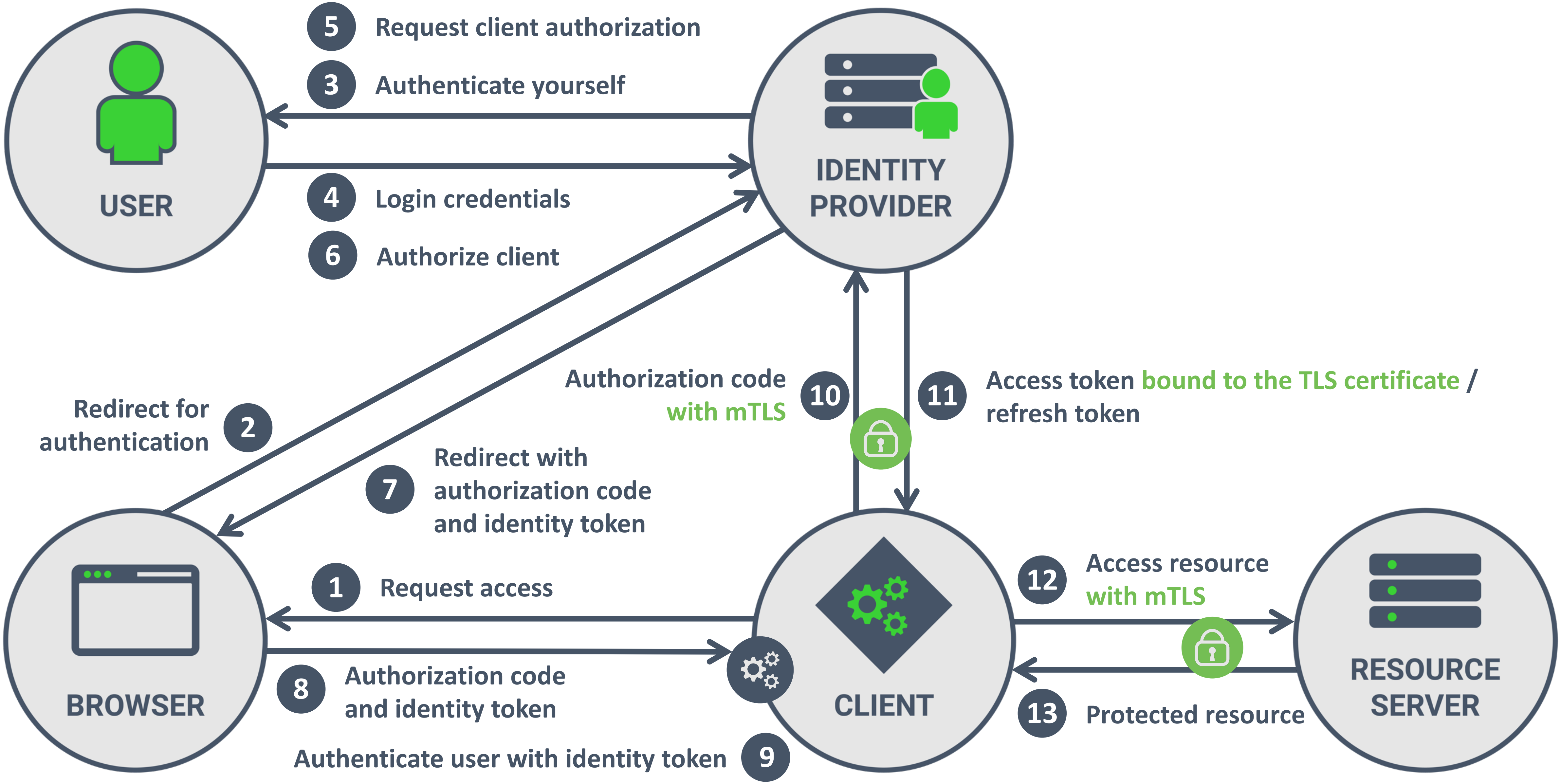
> “The hackers were able to steal some of our Facebook and Twitter access tokens from our users.”

THE DANGER OF BEARER TOKENS


5 Request client authorization
3 Authenticate yourself
USER
IDENTITY PROVIDER
4 Login credentials
6 Authorize client
Redirect for authentication 2
Authorization code with client credentials 10
11 Access token / refresh token
7 Redirect with authorization code and identity token
1 Request access
BROWSER
CLIENT
8 Authorization code and identity token
Authenticate user with identity token 9
12 Access resource
13 Protected resource
RESOURCE SERVER

Access tokens are bearer tokens, allowing immediate abuse upon theft

BINDING TOKENS TO TLS CERTIFICATES



USER

IDENTITY PROVIDER

BROWSER

CLIENT

RESOURCE SERVER

1 Request access

2 Redirect for authentication

3 Authenticate yourself

4 Login credentials

5 Request client authorization

6 Authorize client

7 Redirect with authorization code and identity token

8 Authorization code and identity token

9 Authenticate user with identity token

10 Authorization code with mTLS

11 Access token bound to the TLS certificate / refresh token

12 Access resource with mTLS

13 Protected resource

```
{

   "sub": "jdoe@example.com",
   "aud": "https://api.example.com",
   "azp": "RandomClientID",
   "iss": "https://authorizationserver.example.com/",
   "exp": 1419356238,
   "iat": 1419350238,
   "scope": "read write",
   "jti": "405b4d4e-8501-4e1a-a138-ed8455cd1d47",
   "cnf":{
     "x5t#S256": "bwcK0esc3ACC3DB2Y5_lESsXE8o9ltc05O89jdN-dg2"
   }
}
```

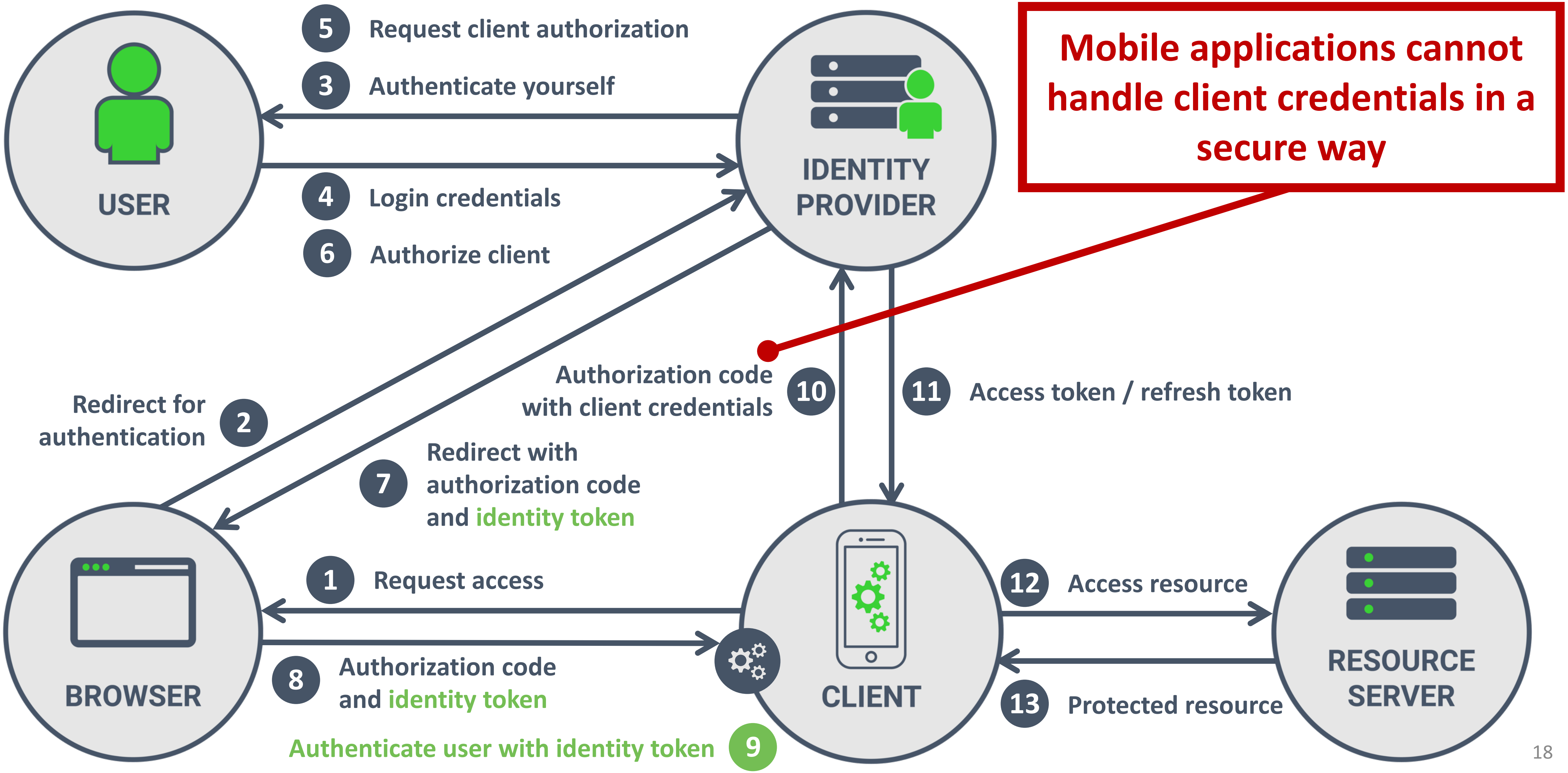
PROOF-OF-POSSESSION FOR ACCESS TOKENS

Many confidential clients still rely on bearer access tokens

The confidential client can authenticate with a TLS certificate

The TLS certificate can be used to enable token binding

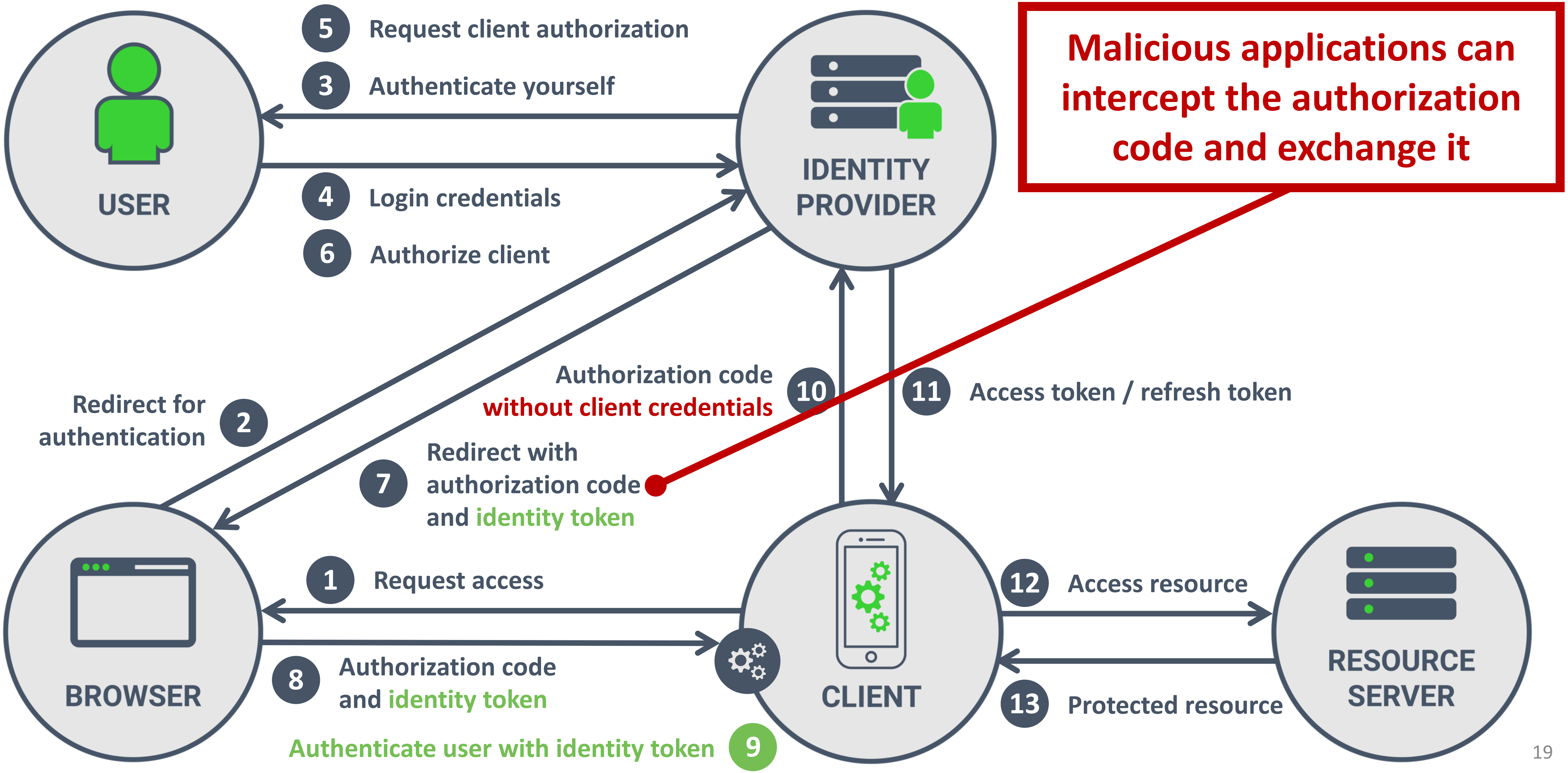
The OIDC Hybrid flow



Mobile applications cannot handle client credentials in a secure way

5 Request client authorization
3 Authenticate yourself
USER
IDENTITY PROVIDER
4 Login credentials
6 Authorize client
Redirect for authentication 2
Authorization code with client credentials 10
11 Access token / refresh token
7 Redirect with authorization code and identity token
BROWSER
1 Request access
8 Authorization code and identity token
Authenticate user with identity token 9
CLIENT
12 Access resource
13 Protected resource
RESOURCE SERVER

THE OIDC HYBRID FLOW



USER

IDENTITY PROVIDER

BROWSER

CLIENT

RESOURCE SERVER

1 Request access

2 Redirect for authentication

3 Authenticate yourself

4 Login credentials

5 Request client authorization

6 Authorize client

7 Redirect with authorization code and identity token

8 Authorization code and identity token

9 Authenticate user with identity token

10 Authorization code without client credentials

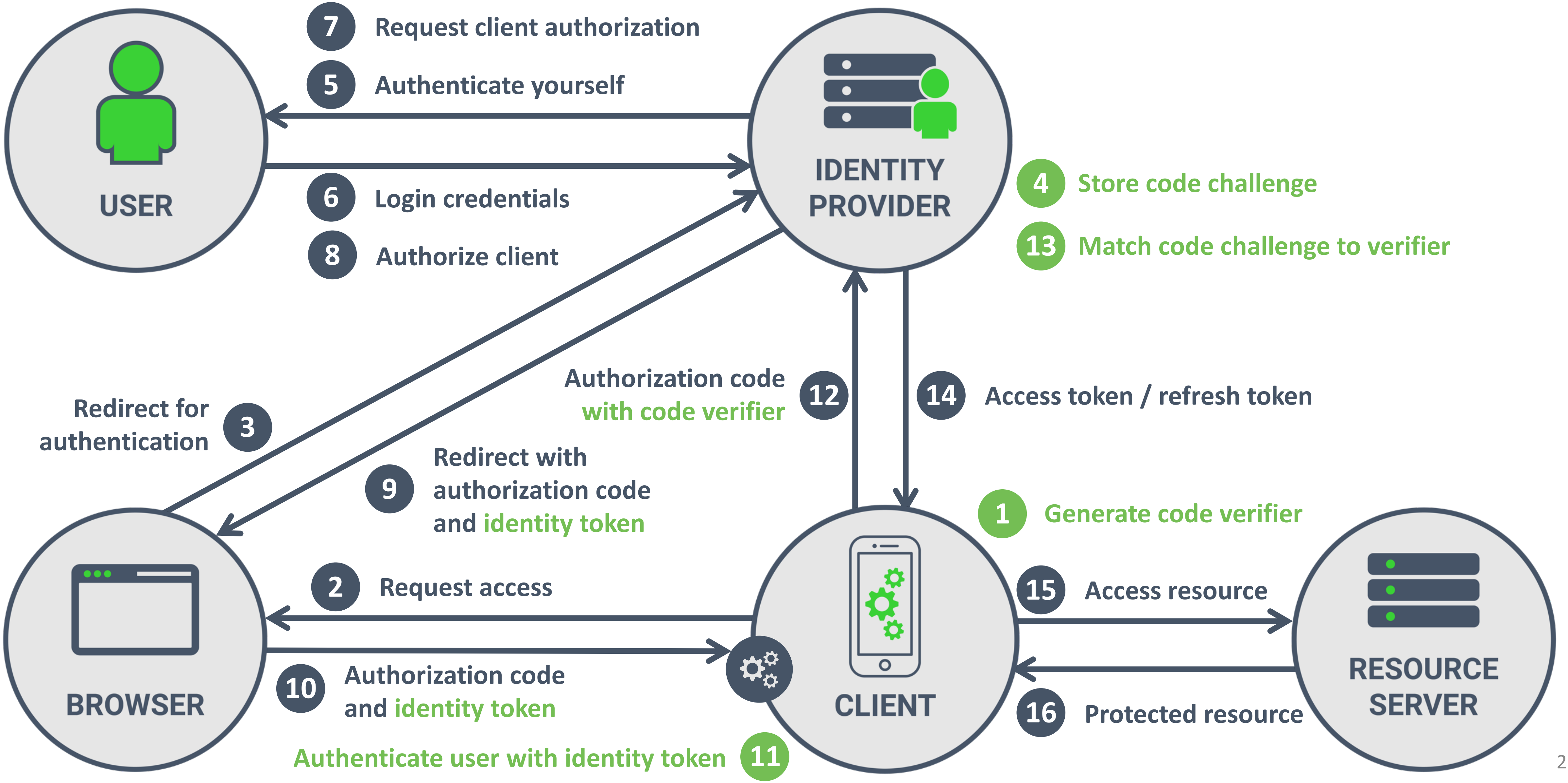
11 Access token / refresh token

12 Access resource

13 Protected resource

Malicious applications can intercept the authorization code and exchange it

The OIDC Hybrid flow with PKCE


USER
IDENTITY PROVIDER
BROWSER
CLIENT
RESOURCE SERVER
7 Request client authorization
5 Authenticate yourself
6 Login credentials
8 Authorize client
4 Store code challenge
13 Match code challenge to verifier
Redirect for authentication 3
Authorization code with code verifier 12
14 Access token / refresh token
9 Redirect with authorization code and identity token
1 Generate code verifier
2 Request access
15 Access resource
10 Authorization code and identity token
16 Protected resource
Authenticate user with identity token 11

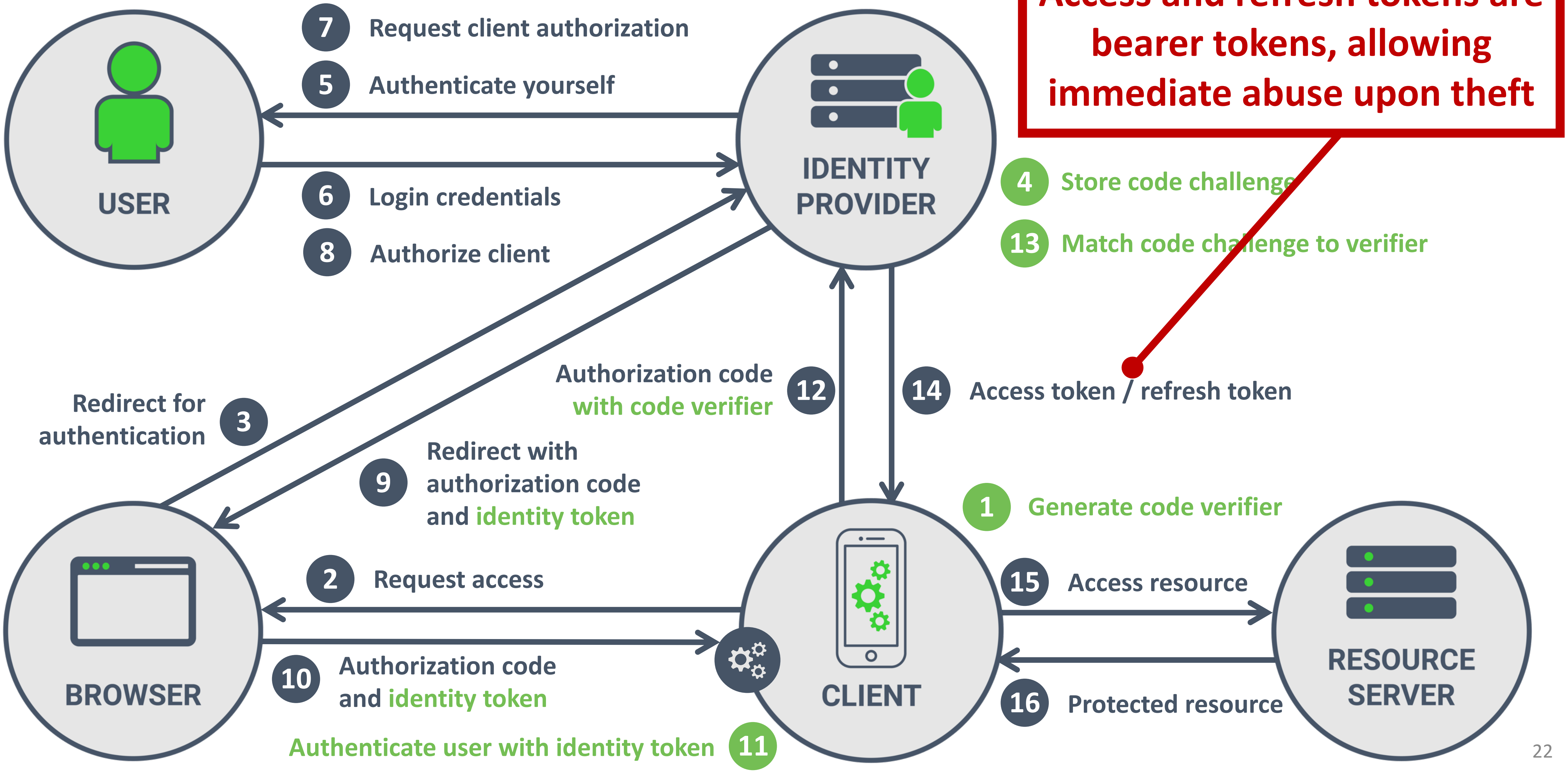
The OIDC Hybrid flow with PKCE

- Mobile applications are public clients
  - The lack of client authentication exposes the authorization code to attacks

- The Proof-Key-for-Code-Exchange addition keeps the authorization code secure
  - PKCE essentially acts as a one-time password for each individual client
  - Prevents the abuse of a stolen authorization code

- Mobile applications can use refresh tokens if they store them securely
  - Refresh tokens do not require authentication, so are bearer tokens
  - Only good place to store is in the OS's secure application storage

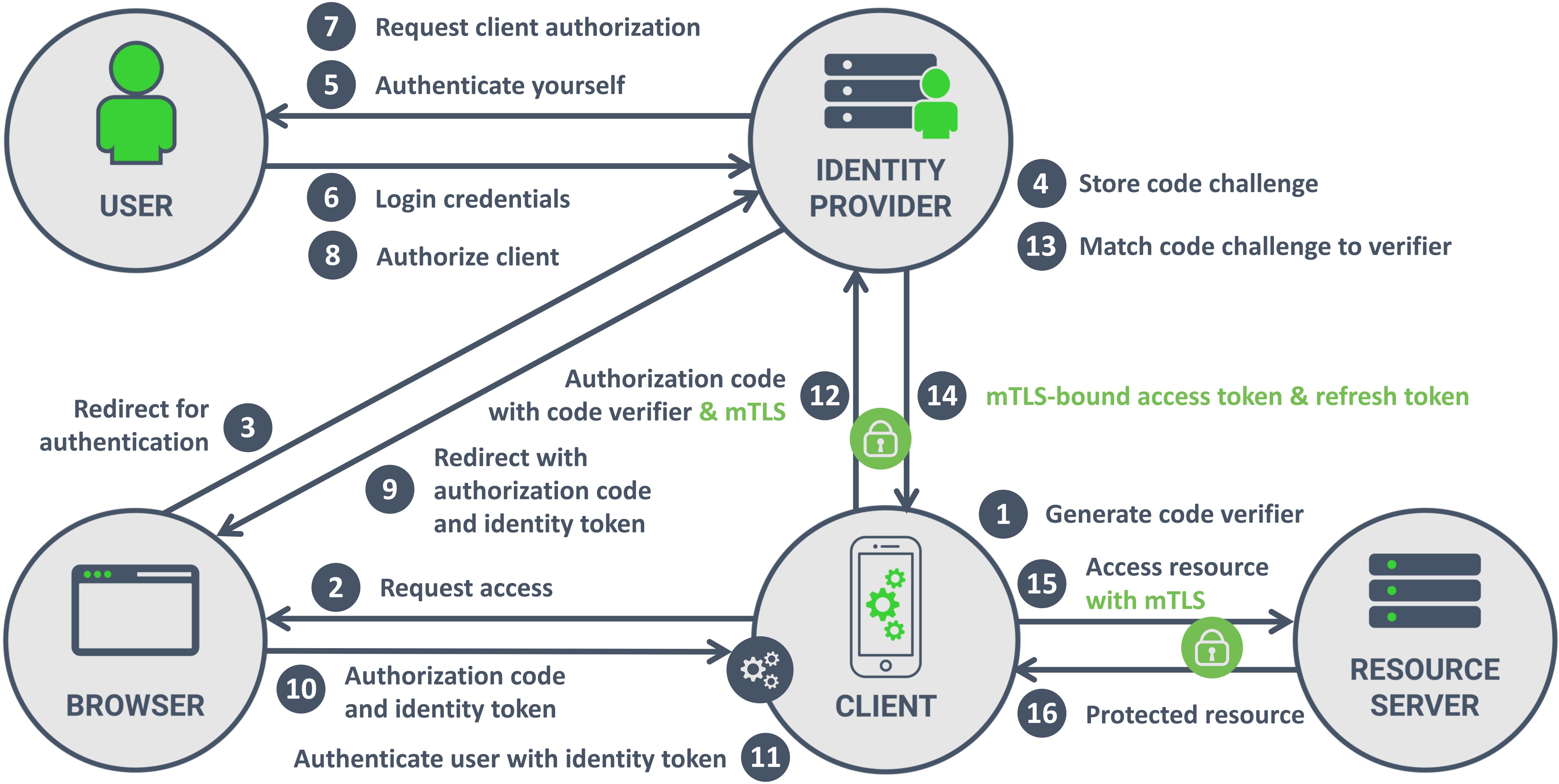
THE DANGER OF BEARER TOKENS



USER
IDENTITY PROVIDER
BROWSER
CLIENT
RESOURCE SERVER
7 Request client authorization
5 Authenticate yourself
6 Login credentials
8 Authorize client
4 Store code challenge
13 Match code challenge to verifier
Redirect for authentication 3
Authorization code with code verifier 12
14 Access token / refresh token
9 Redirect with authorization code and identity token
1 Generate code verifier
2 Request access
10 Authorization code and identity token
Authenticate user with identity token 11
15 Access resource
16 Protected resource

Access and refresh tokens are bearer tokens, allowing immediate abuse upon theft

Binding tokens to TLS certificates on public clients


USER
IDENTITY PROVIDER
BROWSER
CLIENT
RESOURCE SERVER
1 Generate code verifier
2 Request access
3 Redirect for authentication
4 Store code challenge
5 Authenticate yourself
6 Login credentials
7 Request client authorization
8 Authorize client
9 Redirect with authorization code and identity token
10 Authorization code and identity token
11 Authenticate user with identity token
12 Authorization code with code verifier & mTLS
13 Match code challenge to verifier
14 mTLS-bound access token & refresh token
15 Access resource with mTLS
16 Protected resource

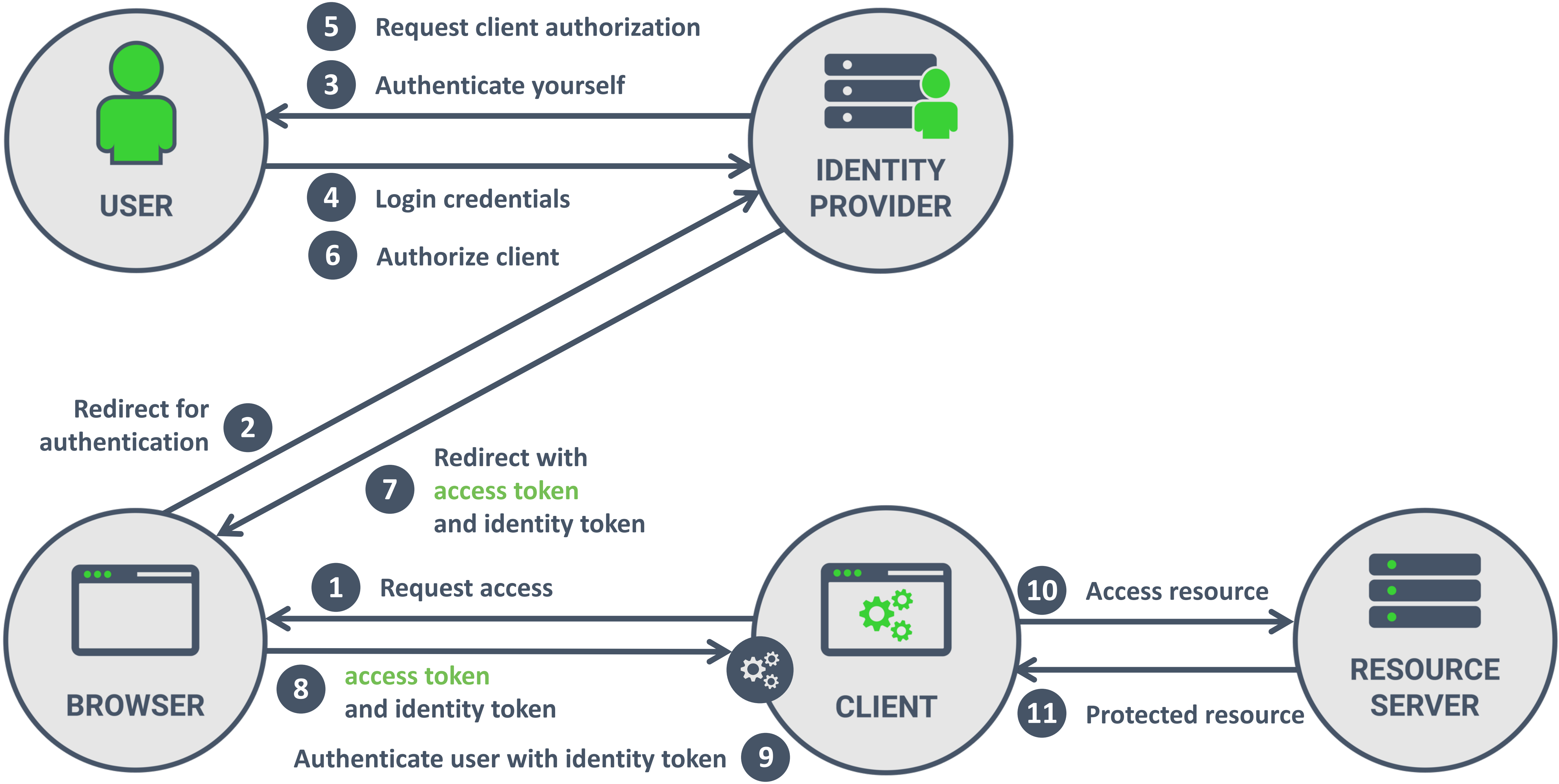
Proof-of-possession in mobile clients

Each client instance generates its own certificate

The client uses the self-signed certificate during TLS connections

The authorization server ties the tokens to the client certificate

The OIDC Implicit flow



5 Request client authorization

3 Authenticate yourself

4 Login credentials

6 Authorize client

Redirect for authentication 2

7 Redirect with access token and identity token

1 Request access

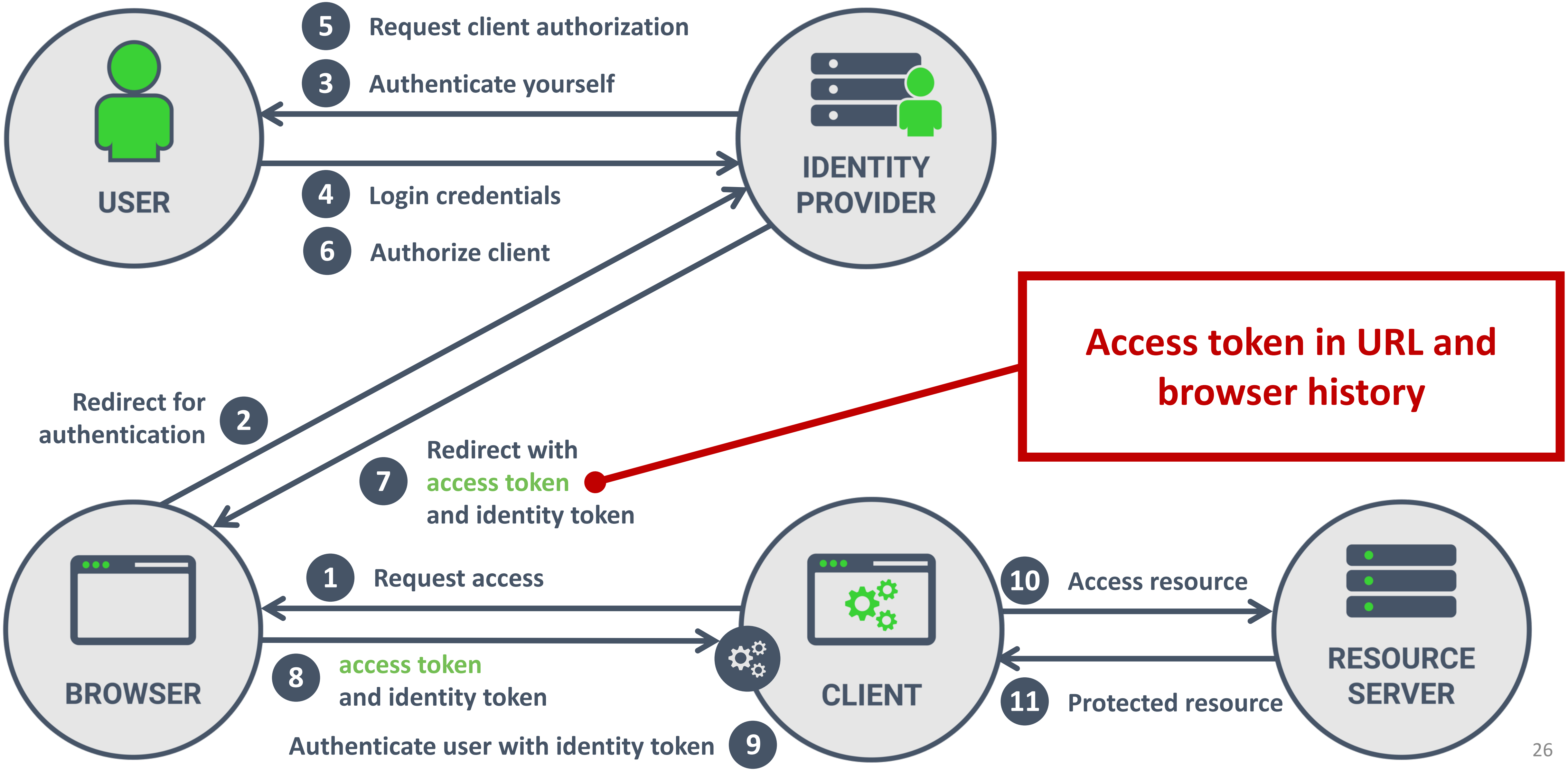
8 access token and identity token

Authenticate user with identity token 9

10 Access resource

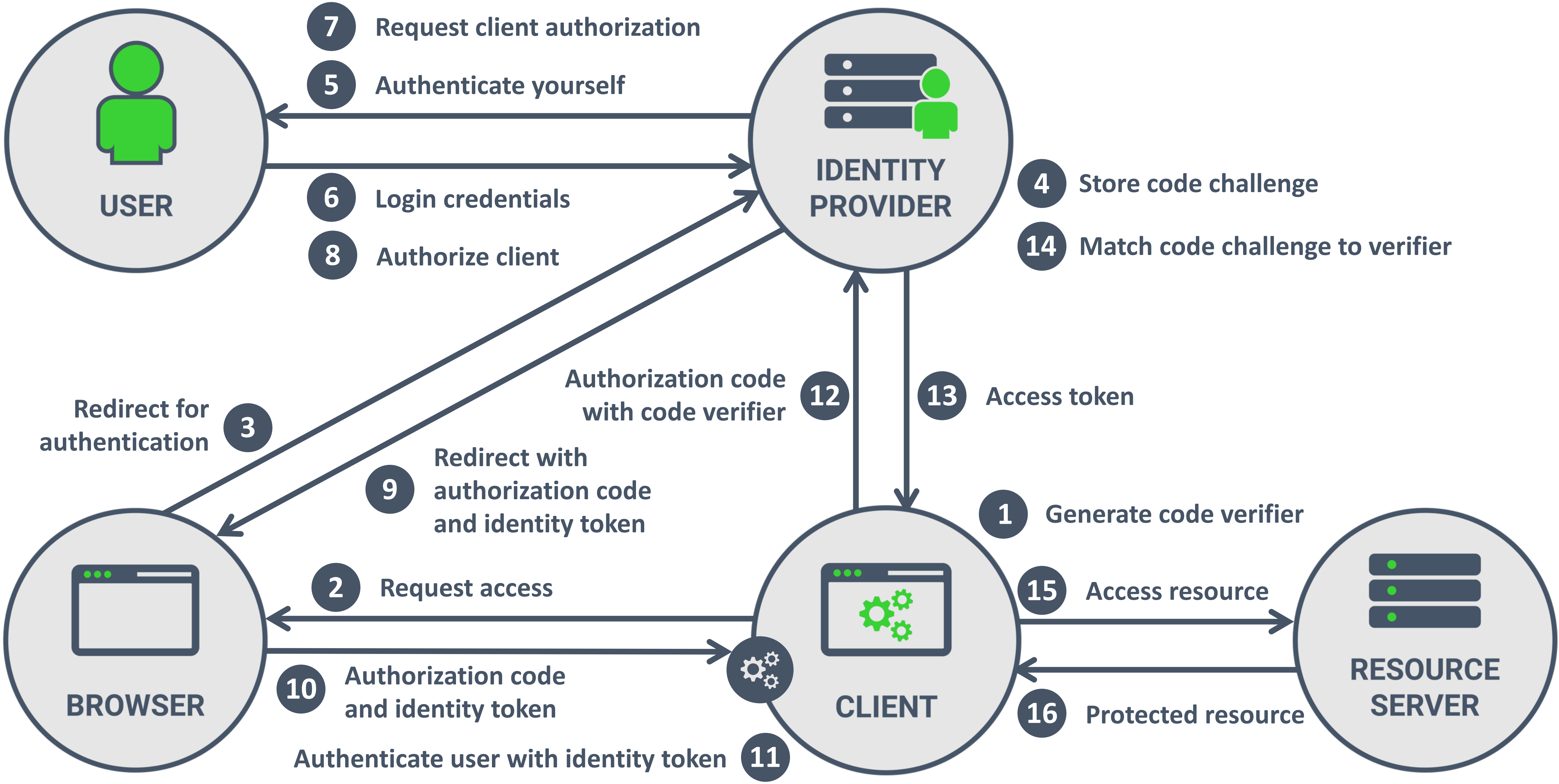
11 Protected resource

The OIDC Implicit flow


5 Request client authorization
3 Authenticate yourself
USER
IDENTITY PROVIDER
4 Login credentials
6 Authorize client
Redirect for authentication 2
7 Redirect with access token and identity token
1 Request access
BROWSER
CLIENT
8 access token and identity token
Authenticate user with identity token 9
10 Access resource
11 Protected resource
RESOURCE SERVER

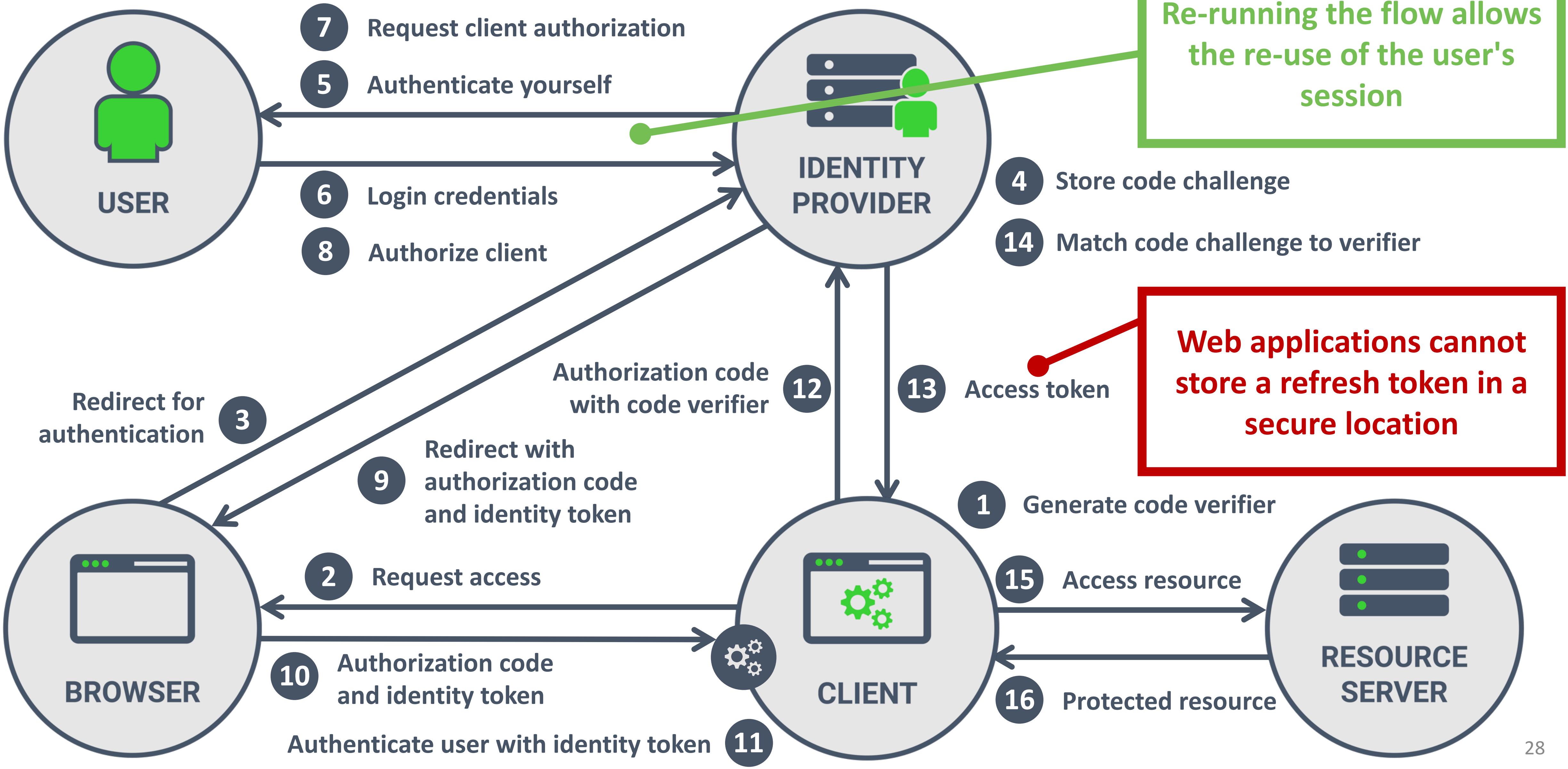
Access token in URL and browser history

THE OIDC HYBRID FLOW WITH PKCE



USER
IDENTITY PROVIDER
BROWSER
CLIENT
RESOURCE SERVER
1 Generate code verifier
2 Request access
3 Redirect for authentication
4 Store code challenge
5 Authenticate yourself
6 Login credentials
7 Request client authorization
8 Authorize client
9 Redirect with authorization code and identity token
10 Authorization code and identity token
11 Authenticate user with identity token
12 Authorization code with code verifier
13 Access token
14 Match code challenge to verifier
15 Access resource
16 Protected resource

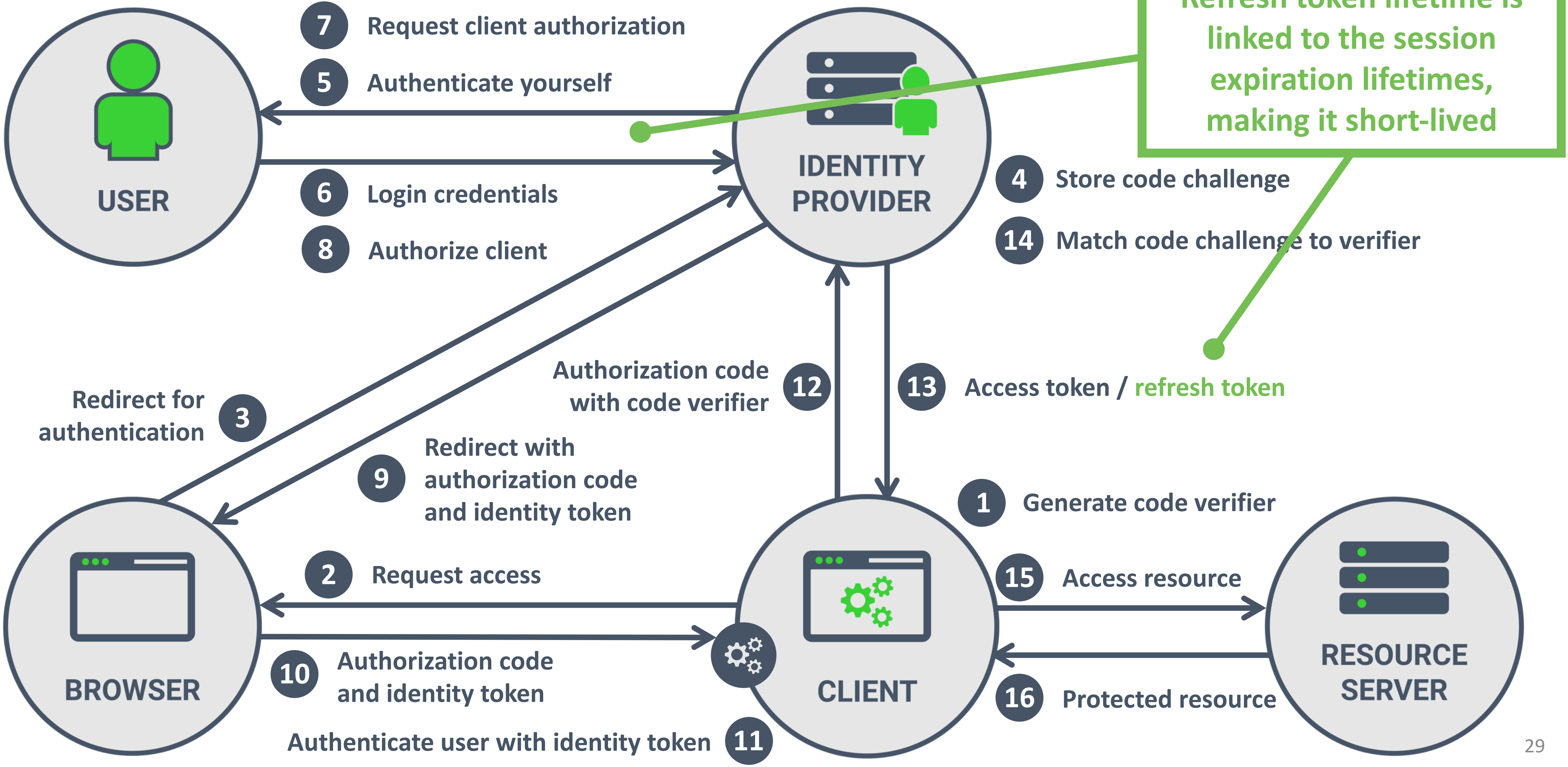
The OIDC Hybrid flow with PKCE


USER
IDENTITY PROVIDER
BROWSER
CLIENT
RESOURCE SERVER
7 Request client authorization
5 Authenticate yourself
6 Login credentials
8 Authorize client
4 Store code challenge
14 Match code challenge to verifier
Redirect for authentication 3
Authorization code with code verifier 12
13 Access token
9 Redirect with authorization code and identity token
1 Generate code verifier
2 Request access
10 Authorization code and identity token
Authenticate user with identity token 11
15 Access resource
16 Protected resource

Re-running the flow allows the re-use of the user's session

Web applications cannot store a refresh token in a secure location

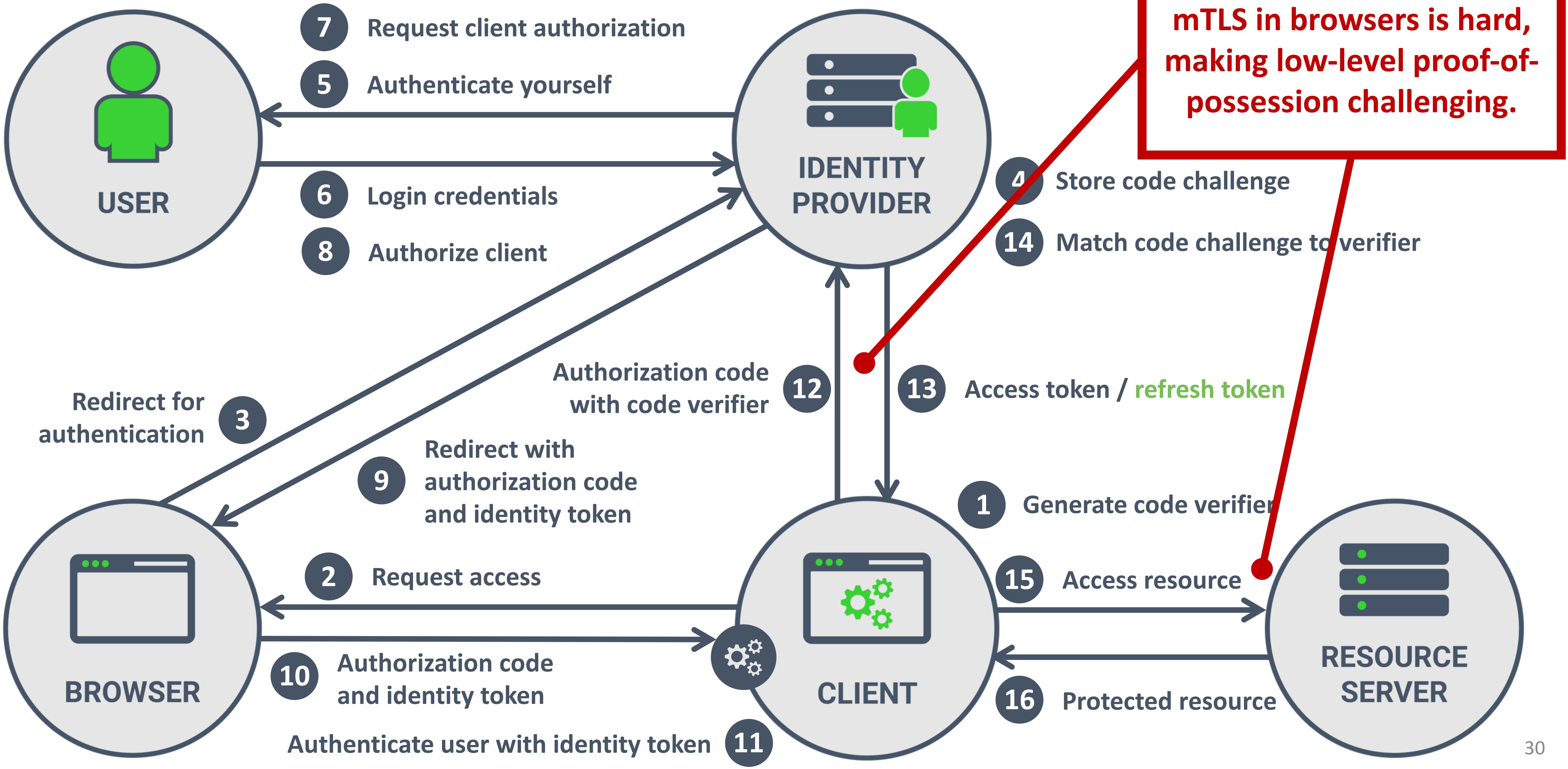
The OIDC Hybrid flow with PKCE



USER
IDENTITY PROVIDER
BROWSER
CLIENT
RESOURCE SERVER
1 Generate code verifier
2 Request access
Redirect for authentication 3
4 Store code challenge
5 Authenticate yourself
6 Login credentials
7 Request client authorization
8 Authorize client
9 Redirect with authorization code and identity token
10 Authorization code and identity token
Authenticate user with identity token 11
Authorization code with code verifier 12
13 Access token / refresh token
14 Match code challenge to verifier
15 Access resource
16 Protected resource

Refresh token lifetime is linked to the session expiration lifetimes, making it short-lived

THE OIDC HYBRID FLOW WITH PKCE



USER
IDENTITY PROVIDER
BROWSER
CLIENT
RESOURCE SERVER
7 Request client authorization
5 Authenticate yourself
6 Login credentials
8 Authorize client
4 Store code challenge
14 Match code challenge to verifier
Redirect for authentication 3
Authorization code with code verifier 12
13 Access token / refresh token
9 Redirect with authorization code and identity token
1 Generate code verifier
2 Request access
15 Access resource
10 Authorization code and identity token
16 Protected resource
Authenticate user with identity token 11

mTLS in browsers is hard, making low-level proof-of-possession challenging.

Web security is hard

The Hybrid flow with PKCE is recommended (Implicit flow is still OK)

Refresh tokens cannot be used, unless they are short-lived

PoP tokens for web applications require application-level code

REFERENCES

Proof Key for Code Exchange by OAuth Public Clients

https://tools.ietf.org/html/rfc7636

OAuth 2.0 Security Best Current Practice

https://tools.ietf.org/html/draft-ietf-oauth-security-topics-13

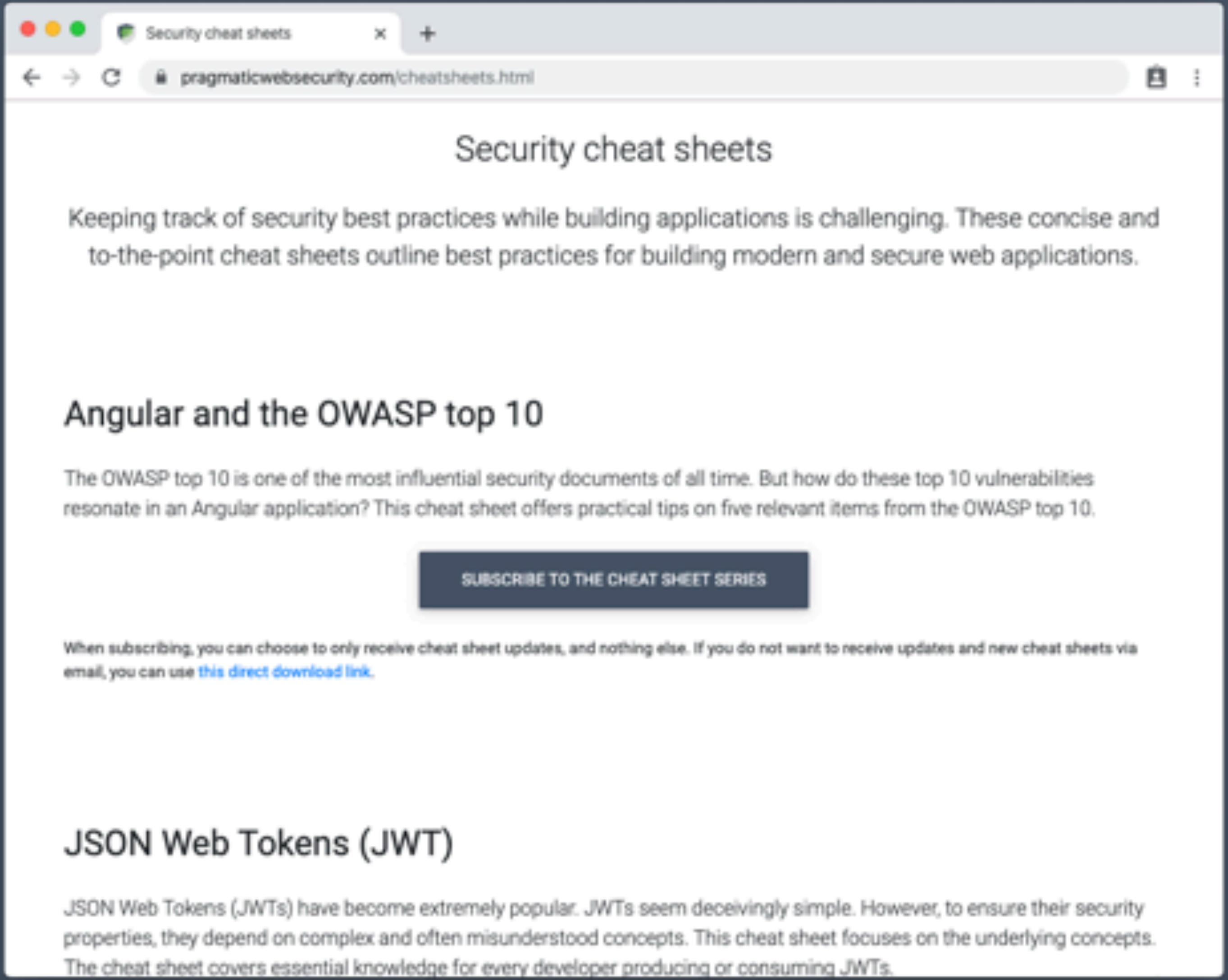
OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens

https://tools.ietf.org/html/draft-ietf-oauth-mtls-17

OAuth 2.0 Demonstration of Proof-of-Possession at the Application Layer

https://tools.ietf.org/html/draft-fett-oauth-dpop-00

Free security cheat sheets for modern applications



Security cheat sheets

Keeping track of security best practices while building applications is challenging. These concise and to-the-point cheat sheets outline best practices for building modern and secure web applications.

Angular and the OWASP top 10

The OWASP top 10 is one of the most influential security documents of all time. But how do these top 10 vulnerabilities resonate in an Angular application? This cheat sheet offers practical tips on five relevant items from the OWASP top 10.

When subscribing, you can choose to only receive cheat sheet updates, and nothing else. If you do not want to receive updates and new cheat sheets via email, you can use this direct download link.

JSON Web Tokens (JWT)

JSON Web Tokens (JWTs) have become extremely popular. JWTs seem deceivingly simple. However, to ensure their security properties, they depend on complex and often misunderstood concepts. This cheat sheet focuses on the underlying concepts. The cheat sheet covers essential knowledge for every developer producing or consuming JWTs.

https://cheatsheets.pragmaticwebsecurity.com/

SecAppDev

March 9th – 13th, 2020
Leuven, Belgium

A week-long course on Secure Application Development

Taught by experts from around the world

38 in-depth lectures and 3 one-day workshops

https://secappdev.org

A yearly initiative from the SecAppDev.org non-profit, since 2005

THANK YOU!

Follow me on Twitter to stay up to date on web security best practices

@PhilippeDeRyck