RECENT EVOLUTIONS IN THE OAUTH 2.0 AND OPENID CONNECT LANDSCAPE

DR. PHILIPPE DE RYCK

https://Pragmatic Web Security.com
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, ...

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)

@PhilippeDeRyck
https://PragmaticWebSecurity.com
User authentication

BROWSER

Read & write data

BACKEND
User authentication

Read & write data
User authentication

CLIENT

Read & write data

API

API
User authentication

OpenID Connect

CLIENT

Read & write data

Identity Provider

Read data

API

OAuth 2.0

OAuth 2.0
User authentication

Results in an identity token and access token

Read & write data

Uses an access token

Read data

Uses an access token
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

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

1. Refresh token with client credentials
2. Access token & refresh token
3. Access resource
4. Protected resource
**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
The hackers were able to steal some of our Facebook and Twitter access tokens from our users.
The danger of bearer tokens

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

Access tokens are bearer tokens, allowing immediate abuse upon theft.
Binding tokens to TLS certificates

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_1ESsXE8o91tc05089jdN-dg2"
    }
}
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

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

Mobile applications cannot handle client credentials in a secure way.
THE OIDC HYBRID FLOW

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
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

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. Match code challenge to verifier
14. Access token / refresh token
15. Access resource
16. Protected resource
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

Access and refresh tokens are bearer tokens, allowing immediate abuse upon theft.
BINDING TOKENS TO TLS CERTIFICATES ON PUBLIC CLIENTS

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
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

1. Request access
2. Redirect for authentication
3. Authenticate yourself
4. Login credentials
5. Request client authorization
6. Authorize client
7. Redirect with access token and identity token
8. access token and identity token
9. Authenticate user with identity token
10. Access resource
11. Protected resource

USER

IDENTITY PROVIDER

BROWSER

CLIENT

RESOURCE SERVER
**The OAuth Implicit Flow**

1. Request access
2. Redirect for authentication
3. Authenticate yourself
4. Login credentials
5. Request client authorization
6. Authorize client
7. Redirect with access token and identity token
8. Access token and identity token
9. Authenticate user with identity token
10. Access resource
11. Protected resource

Access token in URL and browser history
THE OIDC HYBRID FLOW WITH PKCE

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

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

Web applications cannot store a refresh token in a secure location.

Re-running the flow allows the re-use of the user's session.
THE OIDC HYBRID FLOW WITH PKCE

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 / 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.
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

OAuth 2.0 Security Best Current Practice

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

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

https://cheatsheets.pragmaticwebsecurity.com/
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