Common mistakes and misconceptions in Web Application Security using OAuth 2.0 and OpenID Connect

Nahid Farrokhi

@nahid_fa

nahidf

Agenda:

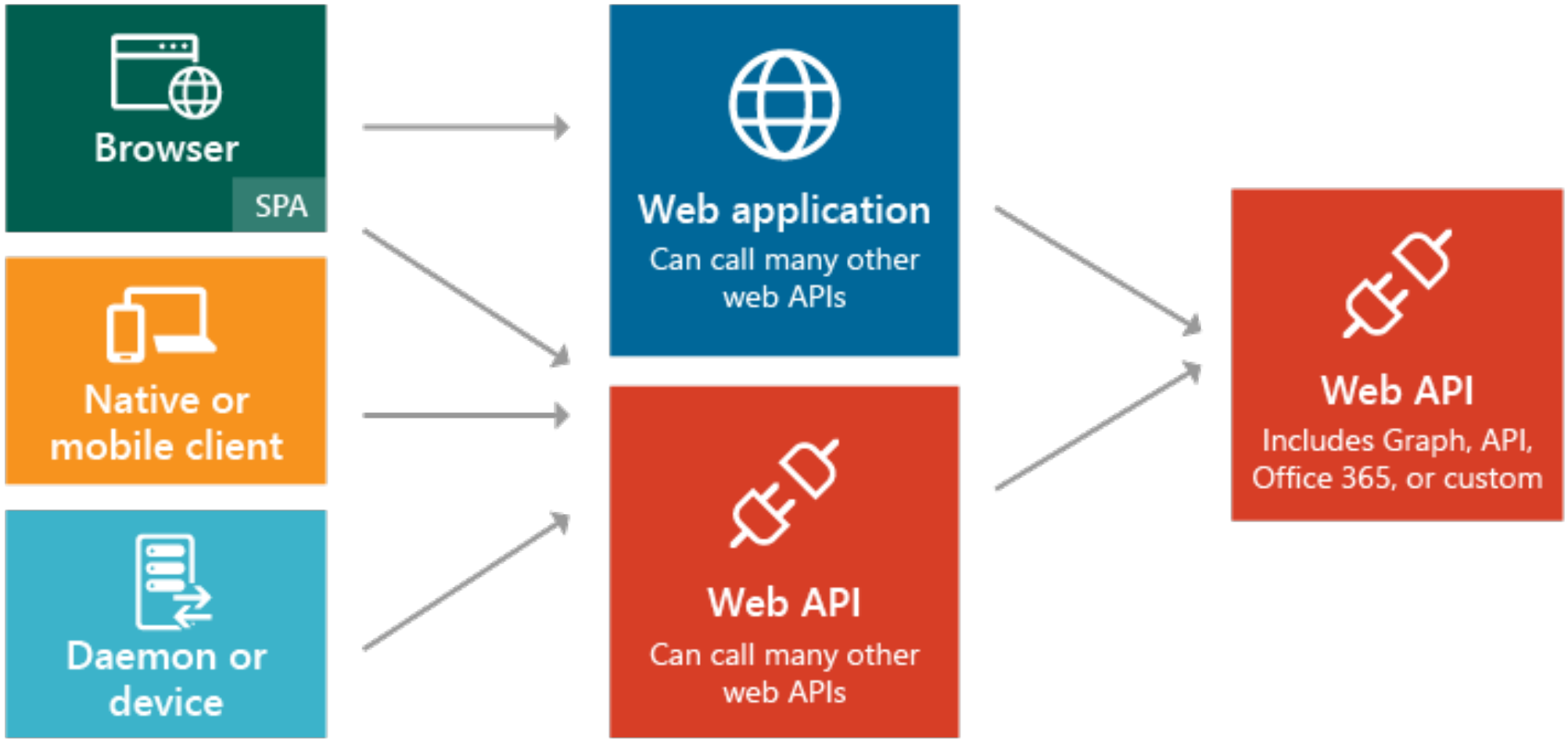
- Specs
- Best Practices
- Demo
- The future
- Question?

Authentication

Who you are

Authorization

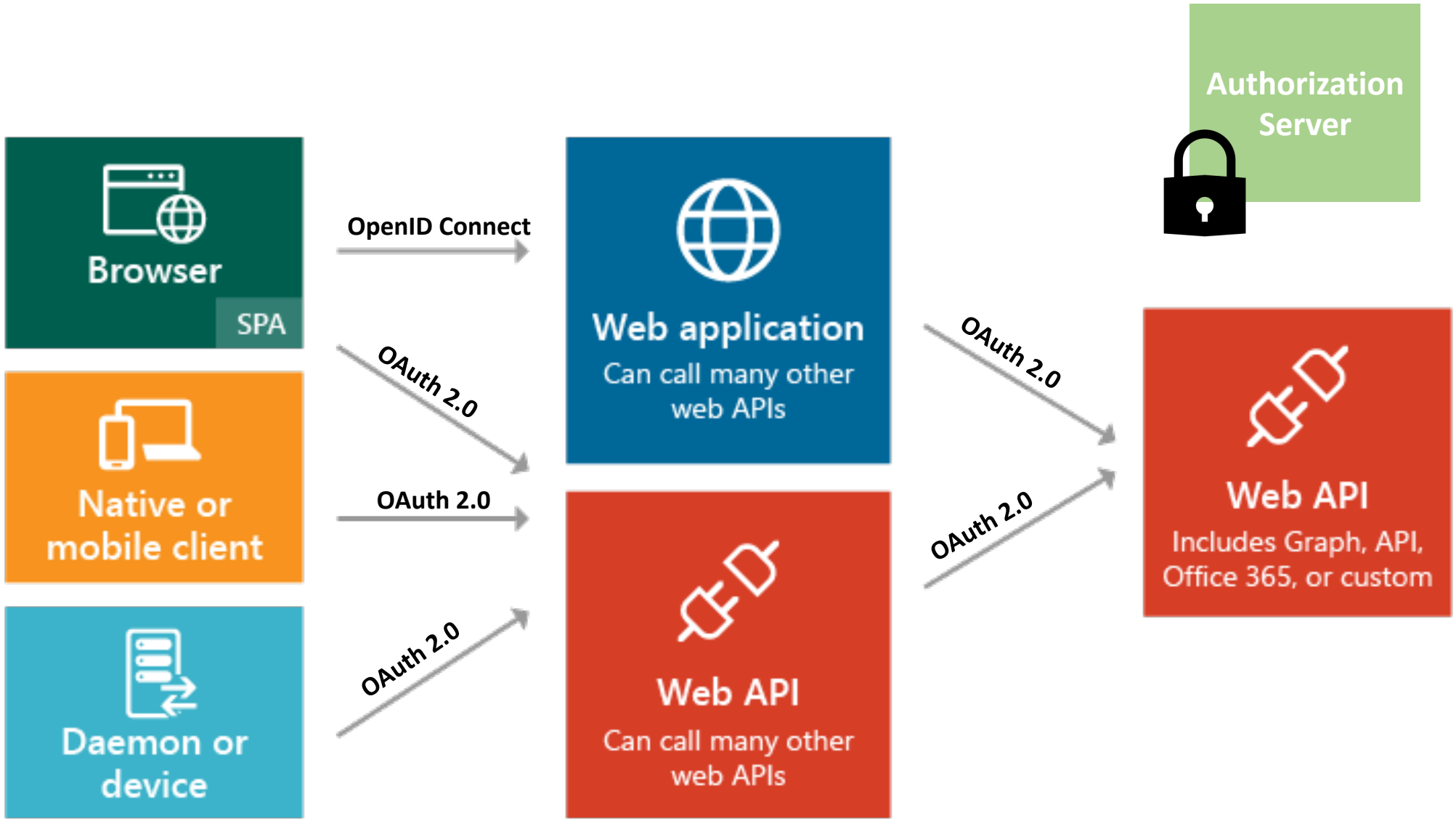
What you can do


Browser
SPA
Native or mobile client
Daemon or device
Web application
Can call many other web APIs
Web API
Can call many other web APIs
Web API
Includes Graph, API, Office 365, or custom

Authorization --> OAuth 2.0

+

Authentication --> OpenID Connect (OIDC)

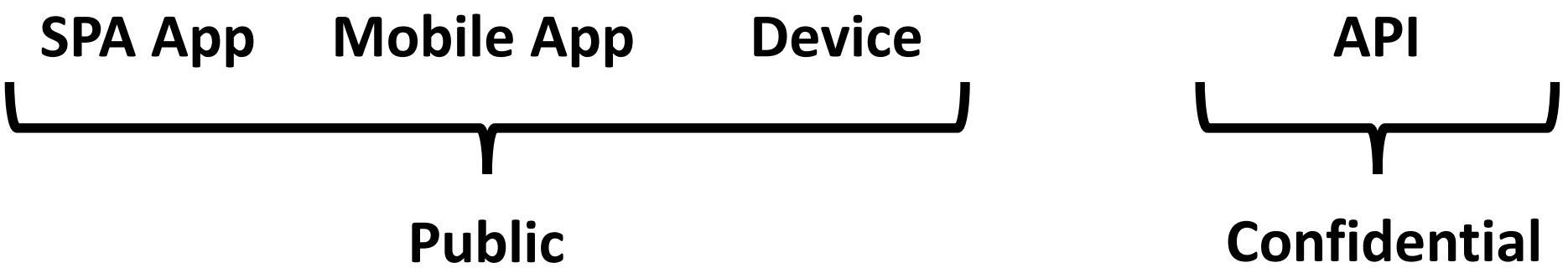

Authorization Server
Browser
SPA
Native or mobile client
Daemon or device
OpenID Connect
OAuth 2.0
OAuth 2.0
OAuth 2.0
Web application
Can call many other web APIs
Web API
Can call many other web APIs
OAuth 2.0
OAuth 2.0
Web API
Includes Graph, API, Office 365, or custom

OAuth 2.0 Terminology

- Roles
  - Resource Owner (RO) - User
  - Resource Server (RS) - API
  - Client - Application
  - Authorization Server(AS)
- Client Types
- Scopes & Consent
- Access Token
- Endpoints
- Grant Types
- Refresh Token

OAuth 2.0 Terminology

- Roles
- Client Types
  - Confidential
  - Public Clients
- Scopes & Consent
- Access Token
- Endpoints
- Grant Types
- Refresh Token


SPA App
Mobile App
Device
API
Public
Confidential

Devices are public client

OAuth 2.0 Terminology

- Roles
- Client Types
- Scopes & Consent
- Access Token
- Endpoints
- Grant Types
- Refresh Token

```
"scope":
[
    "order.write",
    ”order.delete",
    "order.read",
    "invoice.read "
]
```

OAuth 2.0 Scopes

Scopes alone in an access token are not enough to carry on authorization decisions

OAuth 2.0 Terminology

- Roles
- Client Types
- Scopes & Consent
- Access Token
- Endpoints
- Grant Types
- Refresh Token

Encoded PASTE A TOKEN HERE


```
eyJhbGciOiJSUzI1NiIsImtpZCI6IkVCRDAzM0M
3ODBGQUYyOEIzMDY2Q0I4Q0Y1RTUzMDFEIiwidH
lwIjoiYXQrand0In0.eyJuYmYiOjE2MDI1NjIxO
DIsImV4cCI6MTYwMjU2NTc4MiwiaXNzIjoiaHR0
cHM6Ly9sb2NhbGhvc3Q6NTAwMSIsImF1ZCI6WyJ
vcmRlciIsImludm9pY2UiLCJodHRwczovL2xvY2
FsaG9zdDo1MDAxL3Jlc291cmNlcyJdLCJjbGllb
nRfaWQiOiJqc2NsaWVudCIsInN1YiI6IjgxODcy
NyIsImF1dGhfdGltZSI6MTYwMjU2MjE4MSwiaWR
wIjoibG9jYWwiLCJqdGkiOiIyRENBOUFFNjA4QT
czMjY0MTZFRTAxMjFCRjlDRTMyNCIsInNpZCI6I
jc5N0JBODVDMTI4RUQyODdGNEI3M0UzOUJENzlF
MkI0IiwiaWF0IjoxNjAyNTYyMTgyLCJzY29wZSI
6WyJvcGVuaWQiLCJwcm9maWxlIiwib3JkZXIucm
VhZCIsIm9yZGVyLmRlbGV0ZSIsImludm9pY2Uuc
mVhZCJdLCJhbXIiOlsicHdkIl19.jrZLiegVnU4
rSKRZrC31yMWUHrY7eAZiLUScXIAZIXzijnFu6R
UtPUCuz0J5QLnB-
CaaLJsRLuVZIrdp924TABF4bR7zF79hleatB4gD
xDrpj_XHfuBV2EndsG705R50pyN1yFY6RMLHFzj
LQHC6mEMG8_LsV2EcEX7n3VGrs_UksmvPthnAfz
gR_vLARyYyQKlsNt9lGPx7dRydMEoBRWzV4yjQ4
S0BAORsUjivPYvbWqWw5lVMM6TgSD7XL7_nW4Va
35t43LTx_yG1GClOAcffj1xtlqsSLwsoE_mJw_D
Cw8m1TIw89vyjlYUDdxsYpcpZQzdHx7gY9GoUW1
7c2w
```


Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE


```
{
  "alg": "RS256",
  "kid": "EBD033C780FAF28B3066CB8CF5E5301D",
  "typ": "at+jwt"
}
```


PAYLOAD: DATA


```
{
  "nbf": 1602562182,
  "exp": 1602565782,
  "iss": "https://localhost:5001",
  "aud": [
    "order",
    "invoice",
    "https://localhost:5001/resources"
  ],
  "client_id": "jsclient",
  "sub": "818727",
  "auth_time": 1602562181,
  "idp": "local",
  "jti": "2DCA9AE608A7326416EE0121BF9CE324",
  "sid": "797BA85C128ED287F4B73E39BD79E2B4",
  "iat": 1602562182,
  "scope": [
    "openid",
    "profile",
    "order.read",
    "order.delete",
    "invoice.read"
  ],
  "amr": [
    "pwd"
  ]
}
```


VERIFY SIGNATURE

```
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  Public Key or Certificate. Enter it in plain text only if you want to verify a token
,
  Private Key. Enter it in plain text only if you want to generate a new token. The key never leaves your browser.
```

OAuth 2.0 Access Token

Use typed Json Web Token

JWT Token can be verified without doing a database lookup or round trip to Authorization Server

There is no way to invalidate a token until it expires.

OAuth 2.0 Terminology

- Roles
- Client Types
- Scopes & Consent
- Access Token
- Endpoints
  - Authorization Endpoint
  - Token Endpoint
- Grant Types
- Refresh Token

Authorize Endpoint

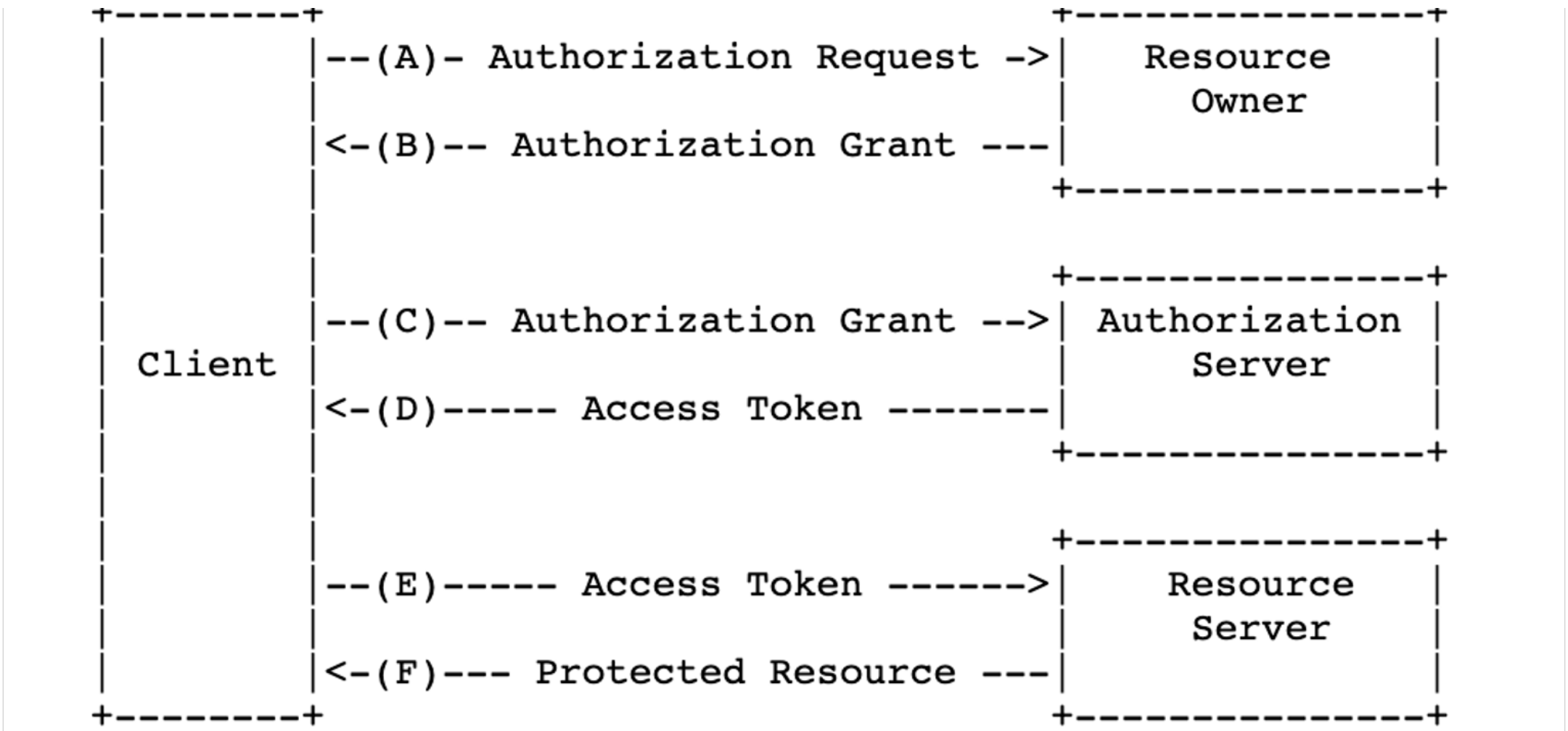
```
GET /connect/authorize?
client_id=client1&
scope=api1&
response_type=id_token token&
redirect_uri=https://myapp/callback&
state=abc&
nonce=xyz
```

Token Endpoint

```
POST /connect/token
CONTENT-TYPE application/x-www-form-urlencoded

client_id=client1&
client_secret=secret&
grant_type=authorization_code&
code=hdh922&
redirect_uri=https://myapp.com/callback
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store
Pragma: no-cache
{
  "access_token":"MTQ0NjJkZmQ5OTM2NDE1ZTZjNGZmZjI3",
  "token_type":"bearer",
  "expires_in":3600,
  "refresh_token":"IwOGYzYTlmM2YxOTQ5MGE3YmNmMDFk
NTVk",
  "scope":"create"
}
```

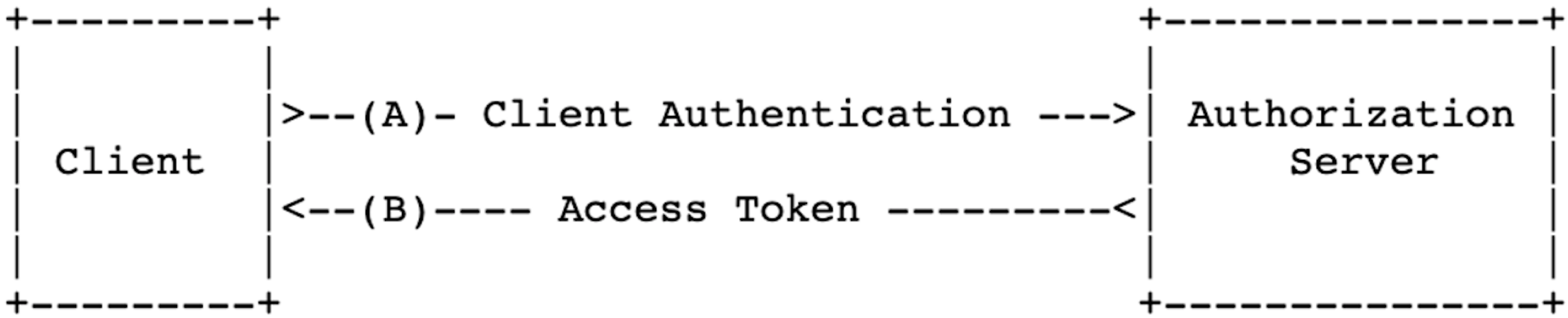
```
HTTP/1.1 400 Bad Request
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache
{
  "error": "invalid_request",
  "error_description": "Request was missing the 'redirect_uri'
parameter.",
  "error_uri": "See the full API docs at https://authorization-
server.com/docs/access_token"
}
```


Client
--(A)- Authorization Request ->
<-(B)-- Authorization Grant ---
Resource Owner
--(C)-- Authorization Grant -->
<-(D)----- Access Token -------
Authorization Server
--(E)----- Access Token ------>
<-(F)--- Protected Resource ---
Resource Server

OAuth 2.0 Terminology

- Roles
- Client Types
- Scopes & Consent
- Access Token
- Endpoints
- Grant Types
  - Client Credentials
  - *Authorization Code*
  - PKCE
  - *Password*
  - *Implicit*
- Refresh Token

OAuth 2.0 Client Credential Grant



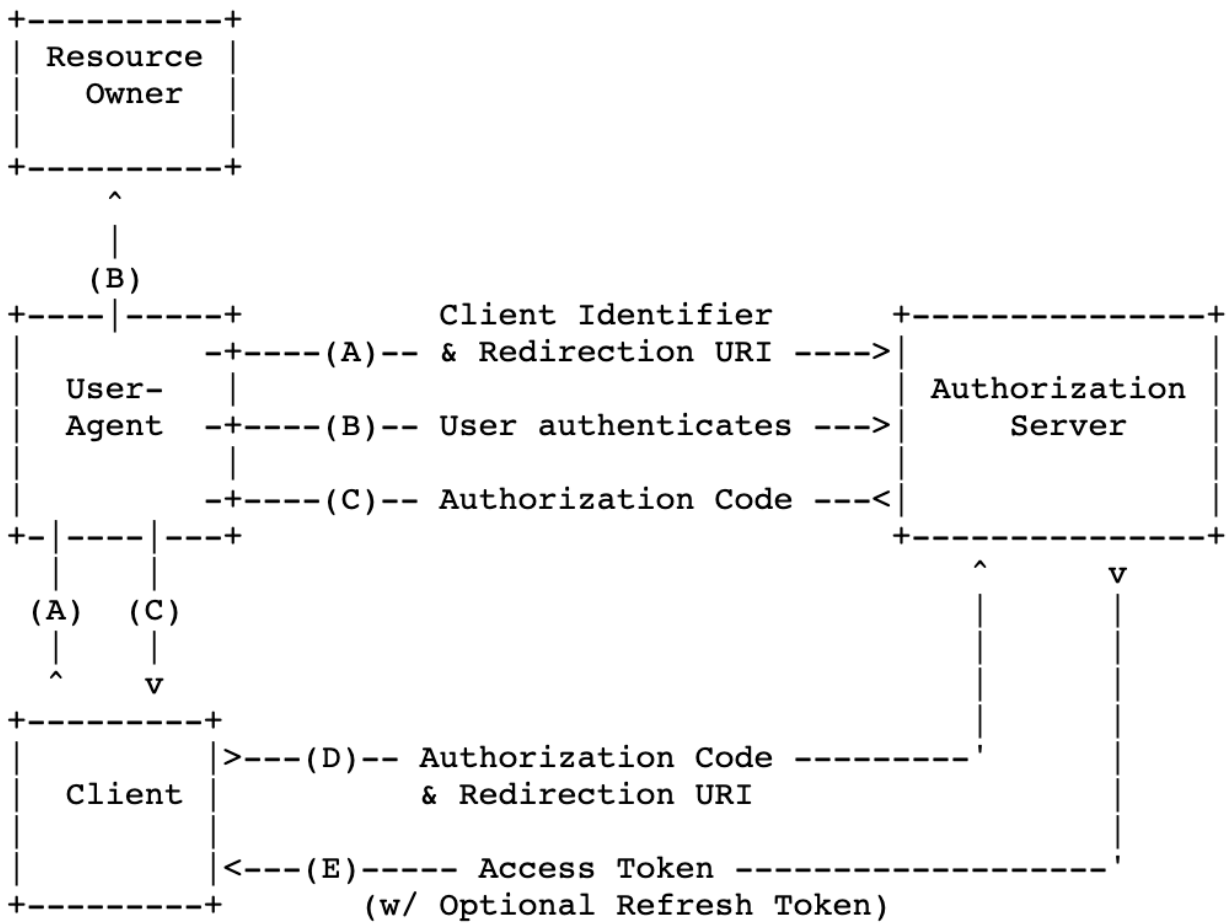

```
POST /token HTTP/1.1
Host: authorization-server.com

grant_type=client_credentials
&client_id=xxxxxxxxxx
&client_secret=xxxxxxxxxx
```

```
curl -H "Authorization: Bearer RsT5OjbzRn430zqMLgV3Ia" \
https://api.authorization-server.com/invoice/me
```

Use Json Web Token

Use HTTPS

OAuth 2.0 Authorization Code Grant



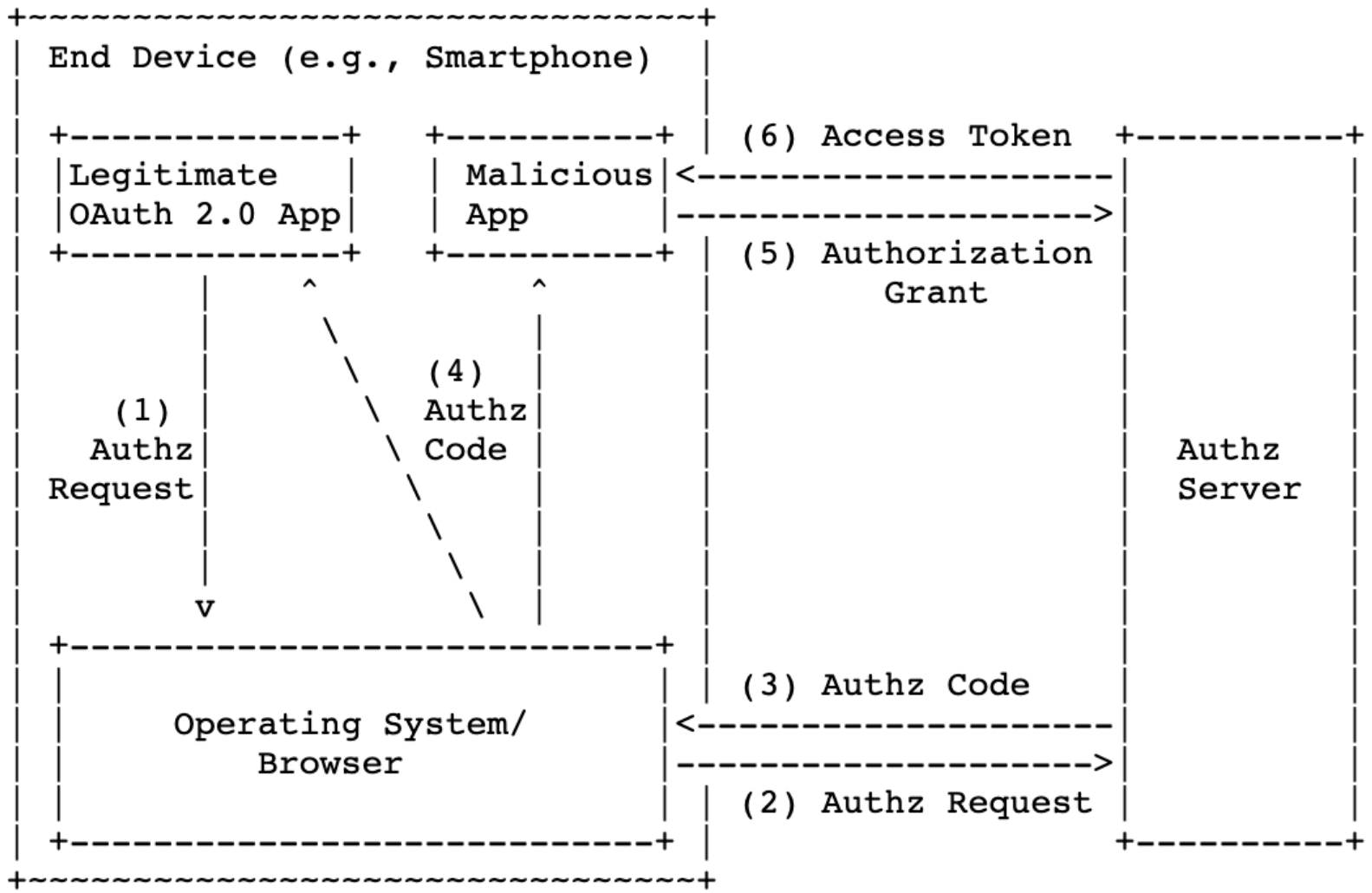
https://authorization-server.com/auth
?response_type=code &client_id= xxxxxxxxxx
&redirect_uri=https://example-app.com/redirect
&scope=create+delete
&state=xcoiv98y2kd22vusuye3kch

https://example-app.com/redirect
?code=g0ZGZmNjVmOWIjNTk2NTk4ZTYyZGI3
&state=xcoiv98y2kd22vusuye3kch

POST /oauth/token HTTP/1.1
Host: authorization-server.com

grant_type=authorization_code
&code=xxxxxxxxxx
&redirect_uri=https://example-app.com/redirect
&client_id=xxxxxxxxxx
&client_secret=xxxxxxxxxx

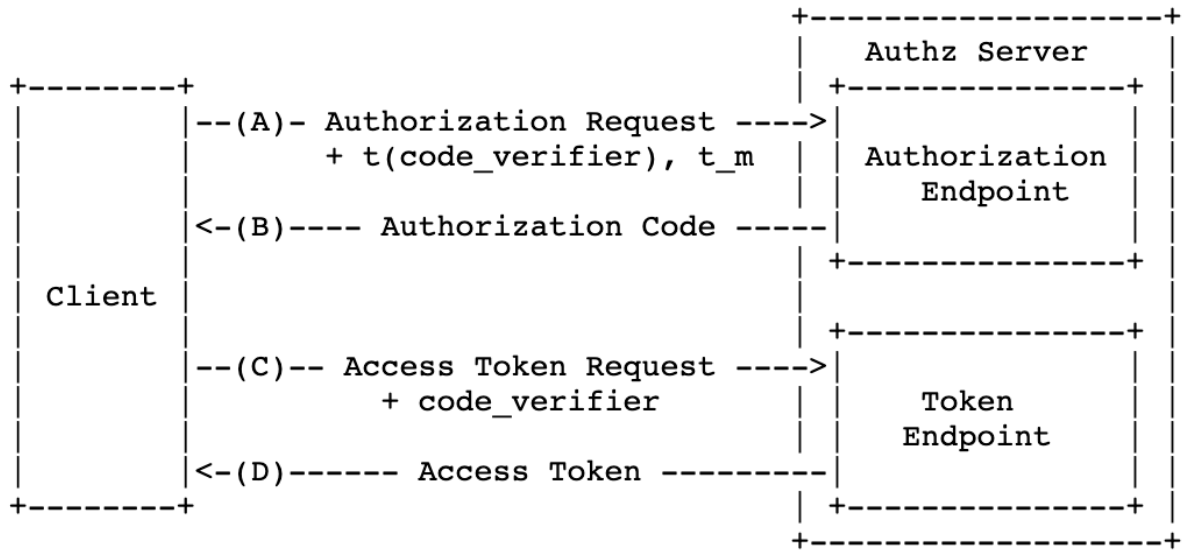
Authorization Code Interception Attack



Authorization Code Injection Attack

Do not use Authorization Code Grant, use PKCE instead

OAuth 2.0 Proof Key for Code Exchange



Provider + /oauth/redirect?

client_id={client_id} &
redirect_uri={Callback URL} &
scope={Scope} &
response_type=code &
state={random long string} &
code_challenge={code challenge} &
code_challenge_method=SHA256

POST Provider + /oauth/access_token
Request body:

```
{
  client_id:{client_id},
  client_secret:{client_secret},
  redirect_uri:{redirect_uri},
  response_type:token,
  Code:{code}
  code_verifier: {code verifier}
}
```

Use Authorization Code + PKCE grant for public and also confidential clients

Redirect URIs must be compared using exact string matching

Return URL:

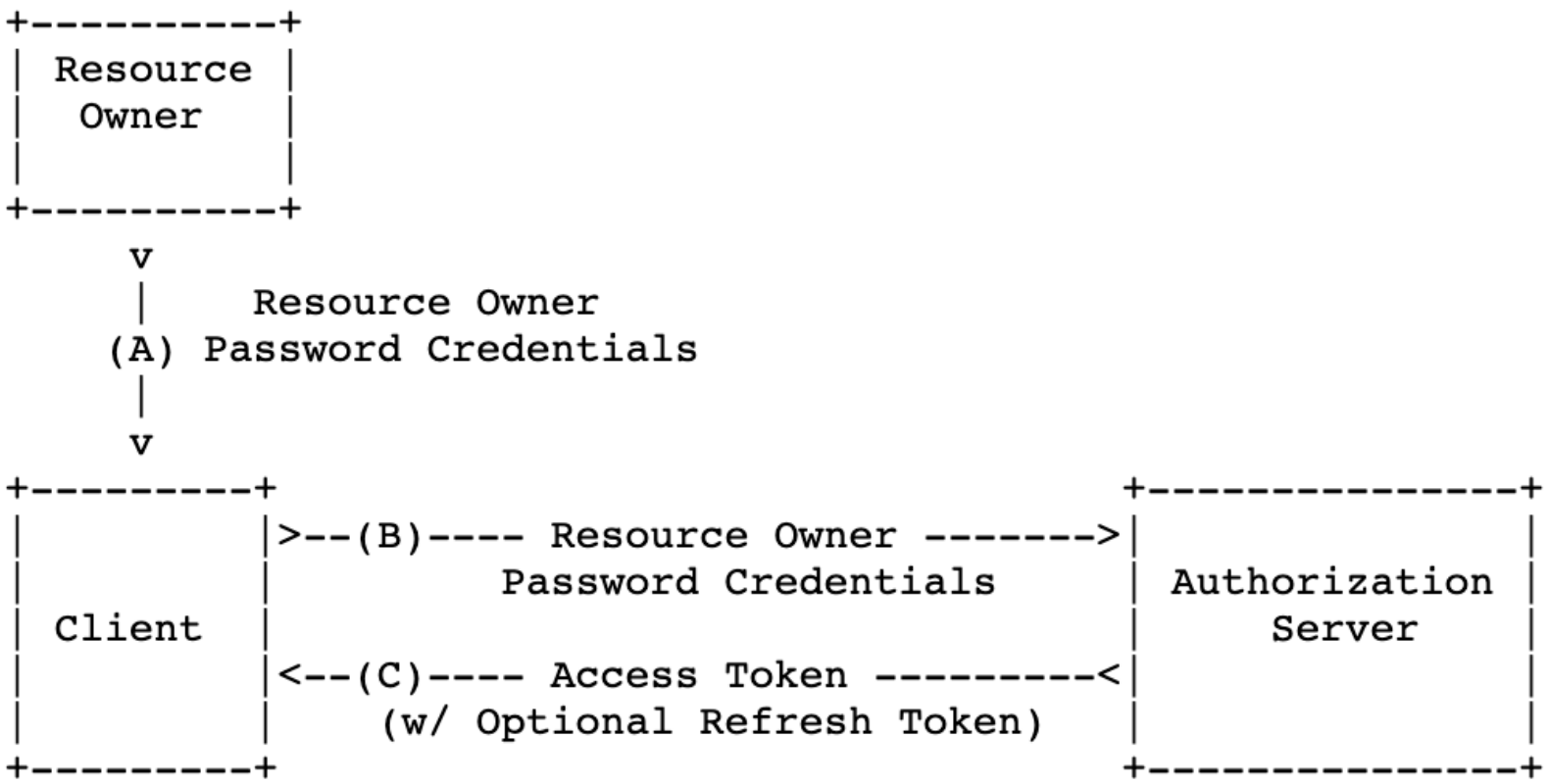
Registered: https://*.somesite.example/*

Valid: https://app1.somesite.example/redirect

Attack: https://attacker.example/.somesite.example

Don’t use Referrer HTTP header

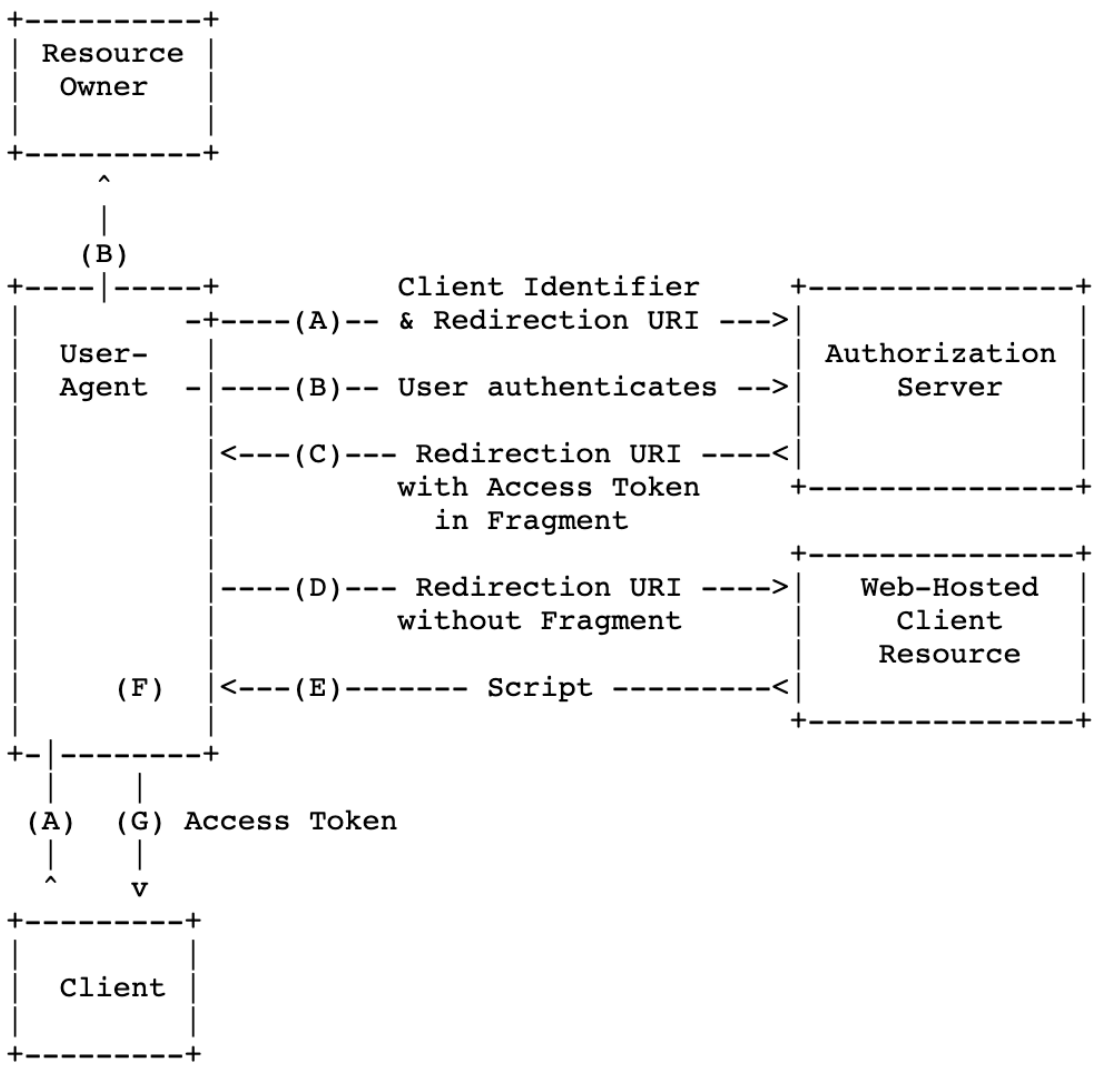
Password Grant




```
POST /oauth/token HTTP/1.1
Host: authorization-server.com

grant_type=password
&username=user@example.com
&password=123Password
&client_id=xxxxxxxxxx
&client_secret=xxxxxxxxxx
```

Do not use Password Grant, use PKCE instead.

Implicit Grant



https://authorization-server.com/auth
?response_type=token & client_id= xxxxxxxxxx
&redirect_uri=https://example-app.com/redirect
&scope=create+delete
&state=xcoiv98y2kd22vusuye3kch

https://example-app.com/redirect
#access_token=g0ZGZmNj4mOWIjNTk2Pw1Tk4ZTYyZGI3 &token_type=Bearer &expires_in=600
&state=xcoVv98y2kd44vuqwye3kcq

Do not use Implicit Grant, use PKCE instead.

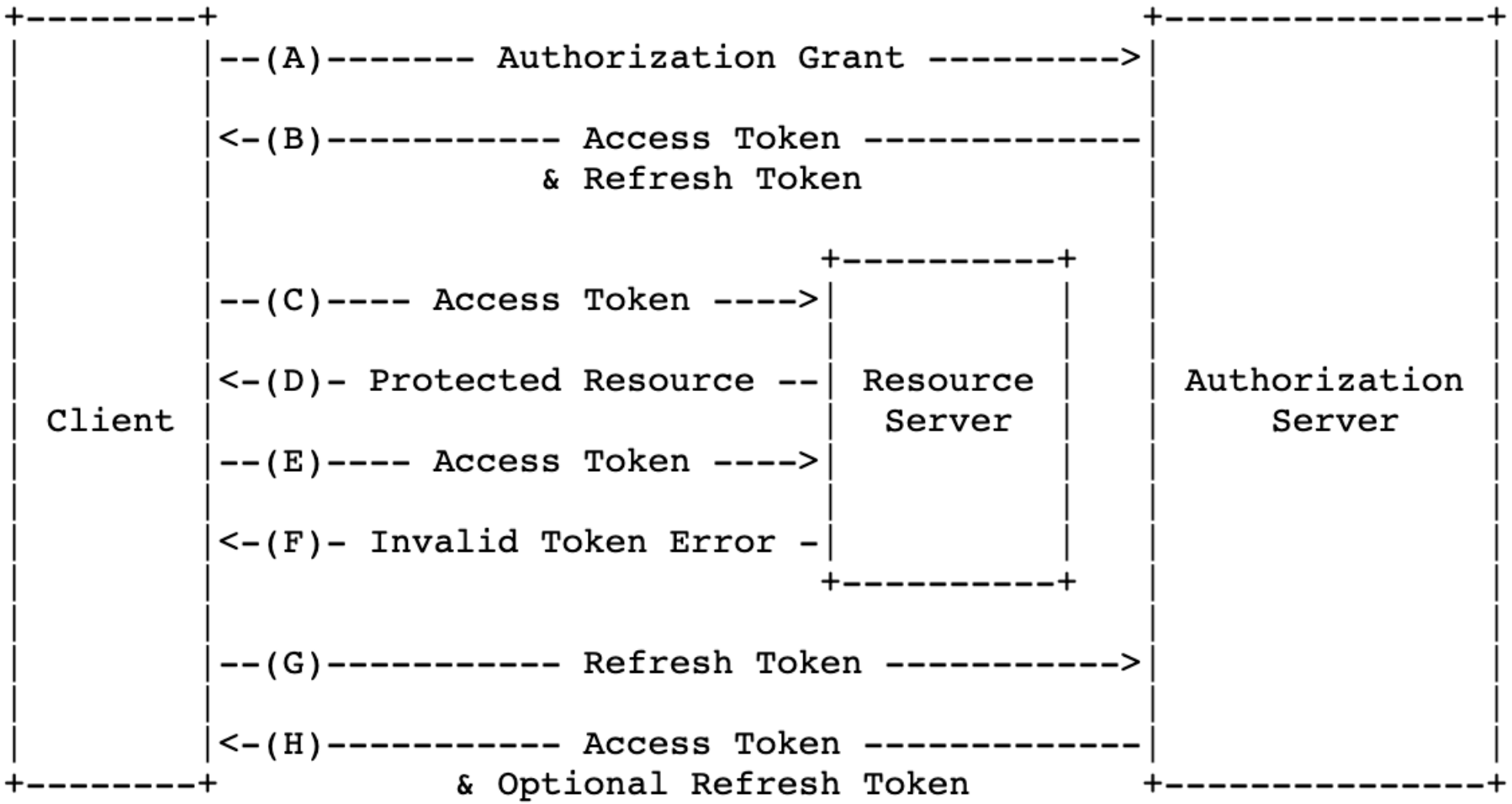
OAuth 2.0 Terminology

- Roles
- Client Types
- Scopes & Consent
- Access Token
- Endpoints
- Grant Types
  - ~~*Password*~~
  - ~~*Implicit*~~
  - Client Credentials
  - ~~*Authorization Code*~~
  - PKCE
- Refresh Token

OAuth 2.0 Terminology

- Roles
- Client Types
- Scopes & Consent
- Access Token
- Endpoints
- Grant Types
- Refresh Token

Refresh Token



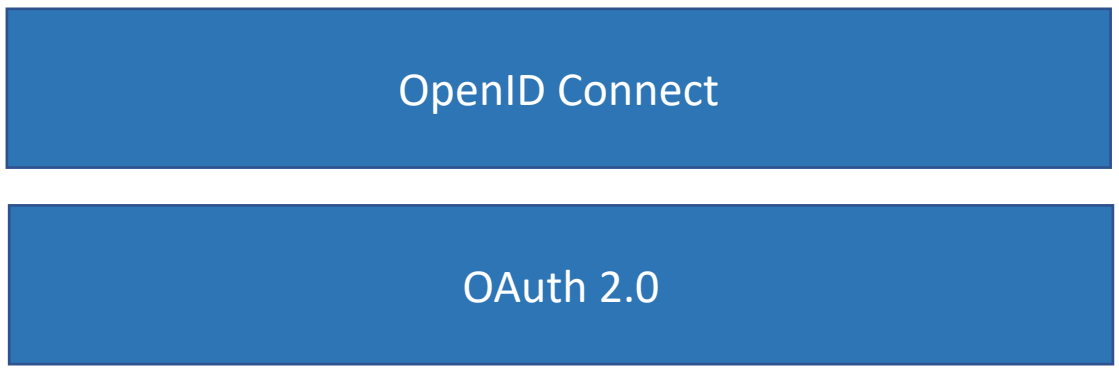
Refresh tokens must either be sender-constrained or one-time use

- Delegated authorization -> Authorization

- SPA app login -> Authentication
- Mobile app login -> Authentication
- Single Sign On -> Authentication

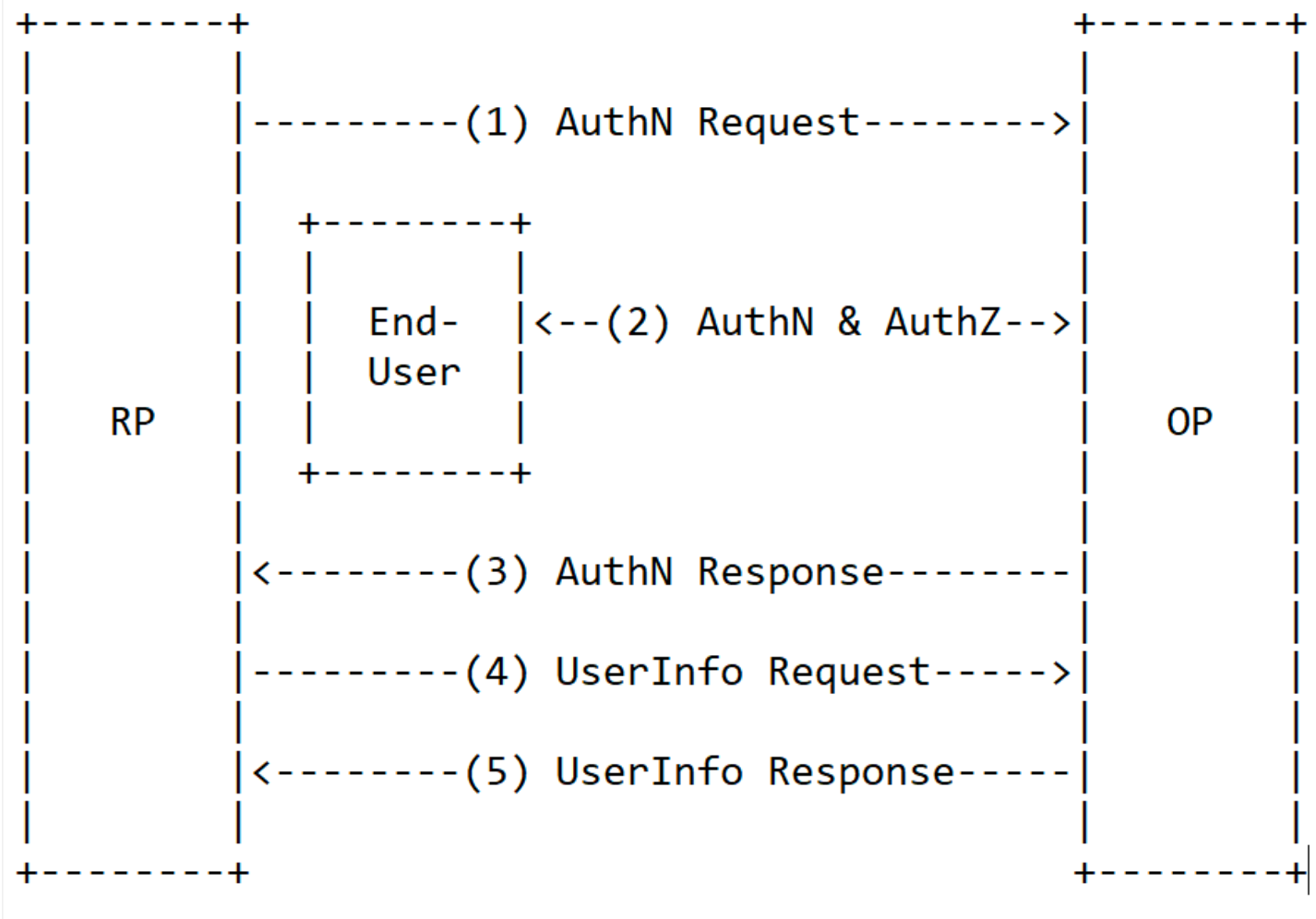
OAuth 2.0 is not for Authentication

Authentication --> OpenID Connect (OIDC)



OpenID Connect Terminology

- Roles
  - Relying Party(RP) - Client
  - OpenID Provider(OP) - AS
- Client Types
- Scopes & Consent
- ID Token
- Endpoints
- Grant Types
- Refresh Token


RP
(1) AuthN Request
End-User
(2) AuthN & AuthZ
OP
(3) AuthN Response
(4) UserInfo Request
(5) UserInfo Response

OpenID Connect Terminology

- Roles
- Client Types
- Scopes & Consent
- ID Token
- Endpoints
- Grant Types
- Refresh Token

```
scope=openid profile email phone
```

OpenID Connect Terminology

- Roles
- Client Types
- Scopes & Consent
- ID Token
- Endpoints
- Grant Types
- Refresh Token


```
{
   "iss": "https://server.example.com",
   "sub": "24400320",
   "aud": "s6BhdRkqt3",
   "nonce": "n-0S6_WzA2Mj",
   "exp": 1311281970,
   "iat": 1311280970,
   "auth_time": 1311280969,
   "acr": "urn:mace:incommon:iap:silver"
}
```

Encoded PASTE A TOKEN HERE


```
eyJhbGciOiJSUzI1NiIsImtpZCI6IkVCRDAzM0M
3ODBGQUYyOEIzMDY2Q0I4Q0Y1RTUzMDFEIiwidH
lwIjoiSldUIn0.eyJuYmYiOjE2MDI1NjI0NDAsI
mV4cCI6MTYwMjU2Mjc0MCwiaXNzIjoiaHR0cHM6
Ly9sb2NhbGhvc3Q6NTAwMSIsImF1ZCI6Im12Y2N
saWVudCIsIm5vbmNlIjoiNjM3MzgxNTkyNDAzOD
gzODU5Lk5qVTVZV00zTlRBdFpHUXhNaTAwWlRne
UxXRmlNelF0TlRjM05UWmhZMk5tWVRabU9HUTBa
RE0xT1RBdE1qVmlZeTAwTWpjNUxUazRZV1F0TlR
Sa05qUTFOelppT0RGayIsImlhdCI6MTYwMjU2Mj
Q0MCwiYXRfaGFzaCI6IkNJOTN1WkZ3QVB0dWxyQ
TJSYXNrQ1EiLCJzX2hhc2giOiJ0NTZTLUg0bjEw
YWhmYzViaHQ1Z1RRIiwic2lkIjoiNzk3QkE4NUM
xMjhFRDI4N0Y0QjczRTM5QkQ3OUUyQjQiLCJzdW
IiOiI4MTg3MjciLCJhdXRoX3RpbWUiOjE2MDI1N
jIxODEsImlkcCI6ImxvY2FsIiwiYW1yIjpbInB3
ZCJdfQ.rCK5dG2QcjbNk9RW_aDkW7b5DR36FVDK
0s8kk4P9vwk52edJ0Z9naHu8__DQzNRu1DmaPyB
rjjeBFSThgihTRb73Sqwxd8NVIbaZzRcOWdQMXQ
Mmug7p5mUlHC_mAok6givmkxSyPk5T2T5QuTEB9
rVIEKBlOchVfn5BEqJRkYwMHCqAznPzGpyjL52C
eSdn6V7kR0h03BdQna-
1ZAbrl88SbVmyKKug5VS3D96sqJaSiRK2JekQaw
BnOrTrsdrCsLQupZ_v6K284vvtxzuQenb_sr8Iu
eXuev13gHD1DZjFbzf3Bn5rNBaEii9OJntxb_jK
qrEiGJpjKoU09itOvA
```


Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE


```
{
  "alg": "RS256",
  "kid": "EBD033C780FAF28B3066CB8CF5E5301D",
  "typ": "JWT"
}
```


PAYLOAD: DATA


```
{
  "nbf": 1602562440,
  "exp": 1602562740,
  "iss": "https://localhost:5001",
  "aud": "mvcclient",
  "nonce":
"637381592403883859.NjU5YWM3NTAtZGQxMi00ZTgyLWFiMzQtNTc
3NTZhY2NmYTZmOGQ0ZDM1OTAtMjViYy00Mjc5LTk4YWQtNTRkNjQ1Nz
ZiODFk",
  "iat": 1602562440,
  "at_hash": "CI93uZFwAPtulrA2RaskCQ",
  "s_hash": "t56S-H4n10ahfc5bht5gTQ",
  "sid": "797BA85C128ED287F4B73E39BD79E2B4",
  "sub": "818727",
  "auth_time": 1602562181,
  "idp": "local",
  "amr": [
    "pwd"
  ]
}
```


VERIFY SIGNATURE

```
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  Public Key or Certificate. Enter it in plain text only if you want to verify a token,
  Private Key. Enter it in plain text only if you want to generate a new token. The key never leaves your browser.
)
```

OpenID Connect Terminology

- Roles
- Client Types
- Scopes & Consent
- ID Token
- Endpoints
  - Discovery Endpoint
  - User Info
- Grant Types
- Refresh Token

```
/.well-known/openid-configuration
```

User Info Endpoint

```
GET /connect/userinfo Authorization: Bearer <access_token>
```

```
HTTP/1.1 200 OK Content-Type: application/json
{
   "sub": ”656756",
   "name": ”Nahid farrokhi",
   "given_name": ”Nahid",
   "family_name": ”Farrokhi",
   "role": [ "user", "admin" ]
}
```

OpenID Connect Terminology

- Roles
- Client Types
- Scopes & Consent
- ID Token
- Endpoints
- Grant Types
  - ~~Authorization Code~~
  - PKCE
  - Hybrid
  - ~~Implicit~~
- Refresh Token

OIDC Hybrid Grant

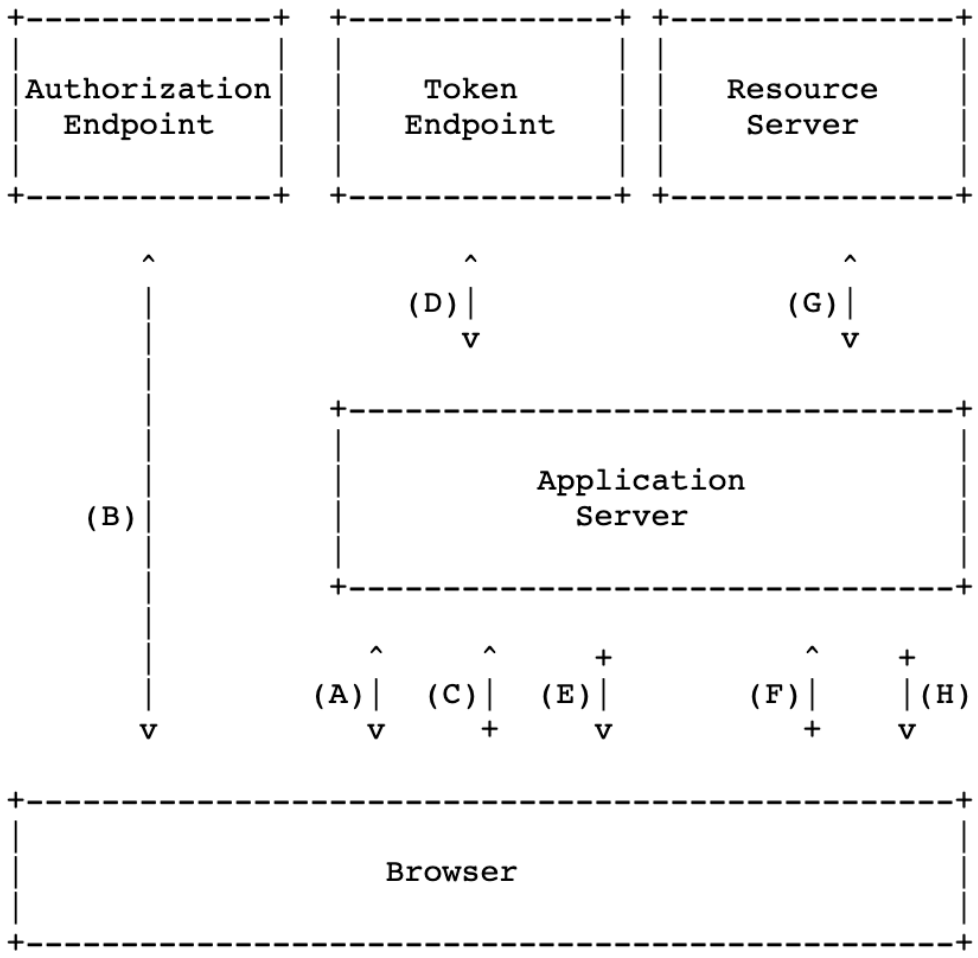

```
{
 "at_hash": "JrZY9MtYVEIIJUx-DDBmww",
 "c_hash": "S5UOXRNNyYsI6Z0G3xxdpw",
 "sub": "admin",
 "aud": [
  "nczbg5m5xxt6tP4UMZwB6PtQoQoa"
 ],
 "azp": "nczbg5m5xxt6tP4UMZwB6PtQoQoa",
 "iss": "https://localhost:9443/oauth2/token",
 "exp": 1510831512,
 "nonce": "asd",
 "iat": 1510831508
}
```

Use Authorization Code + PKCE grant for all clients

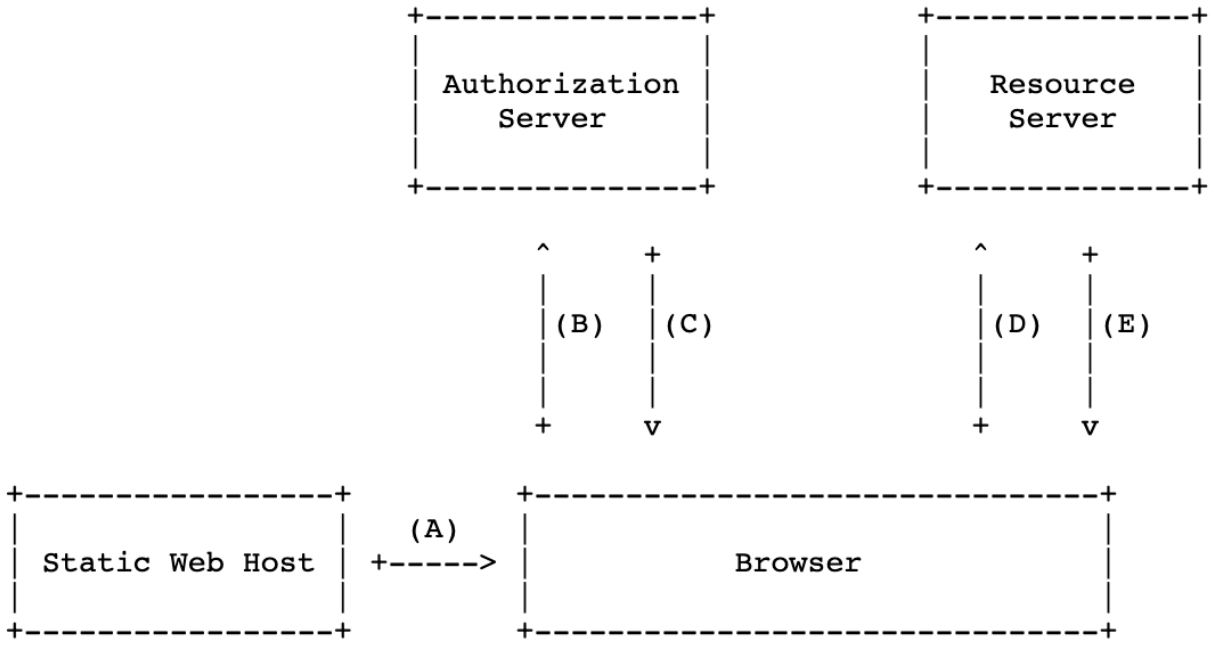
- Web Apps -> PKCE
- Native Mobile Apps -> PKCE
- SPA Apps-> PKCE

- APIs -> Client Authorization

SPA

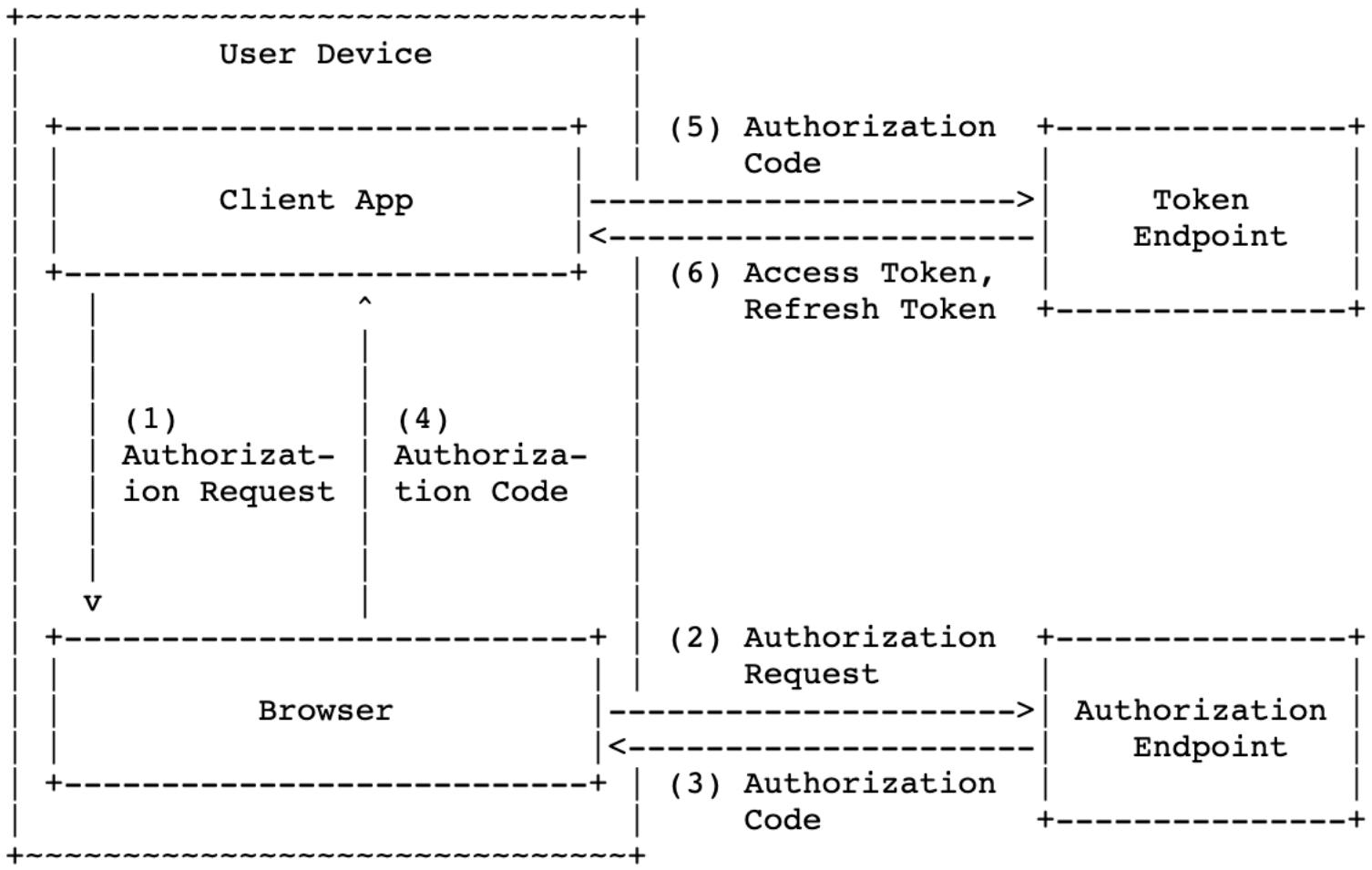


JavaScript Applications with a Backend



JavaScript Applications without a Backend

Native App:


User Device
Client App
Browser
(1) Authorization Request
(4) Authorization Code
(5) Authorization Code
(6) Access Token, Refresh Token
Token Endpoint
(2) Authorization Request
Authorization Endpoint
(3) Authorization Code

Demo

Future -> OAuth 2.1

- The authorization code grant is extended with the functionalit
- such that the only method of using the authorization code gra
- Redirect URIs must be compared using exact string matching
- The Implicit grant is omitted from the specification
- The Resource Owner Password Credentials grant is omitted fro
- Bearer token usage omits the use of bearer tokens in the quer
- Refresh tokens must either be sender-constrained or one-time

Future -> OAuth 2.1

- The authorization code grant is extended with PKCE
- Redirect URIs must be compared using exact string matching
- The Implicit grant is omitted
- The Password Credentials grant is omitted
- Bearer token usage omits the use in the query string
- Refresh tokens must either be sender-constrained or one-time use

Question?