Presentation of Authentication Concepts of *Open Authorization 2*

Flows in OAuth 2

Timo Pagel

OWASP

About Me

- DevSecOps Consultant
- Lecturer for *Security in Web Applications* at *University of Applied Sciences Kiel/Wedel*
- Open Source / Open Knowledge Enthusiast

About Me

- DevSecOps Consultant
- Lecturer for *Security in Web Applications* at *University of Applied Sciences Kiel/Wedel*
- Open Source / Open Knowledge Enthusiast
  - OWASP Juice Shop
  - DevSecOps Maturity Model
  - OWASP Security Pins Project
  - Full University Module Security in Web App.
  - OWASP Software Assurance Maturity Model

Agenda

- Introduction
- Flows
- Conclusion

Agenda

- Introduction
- Flows
- Conclusion

Classic Username/Password

Username / Password

Login Successful / Unsuccessful

Blog A

Classic Username/Password

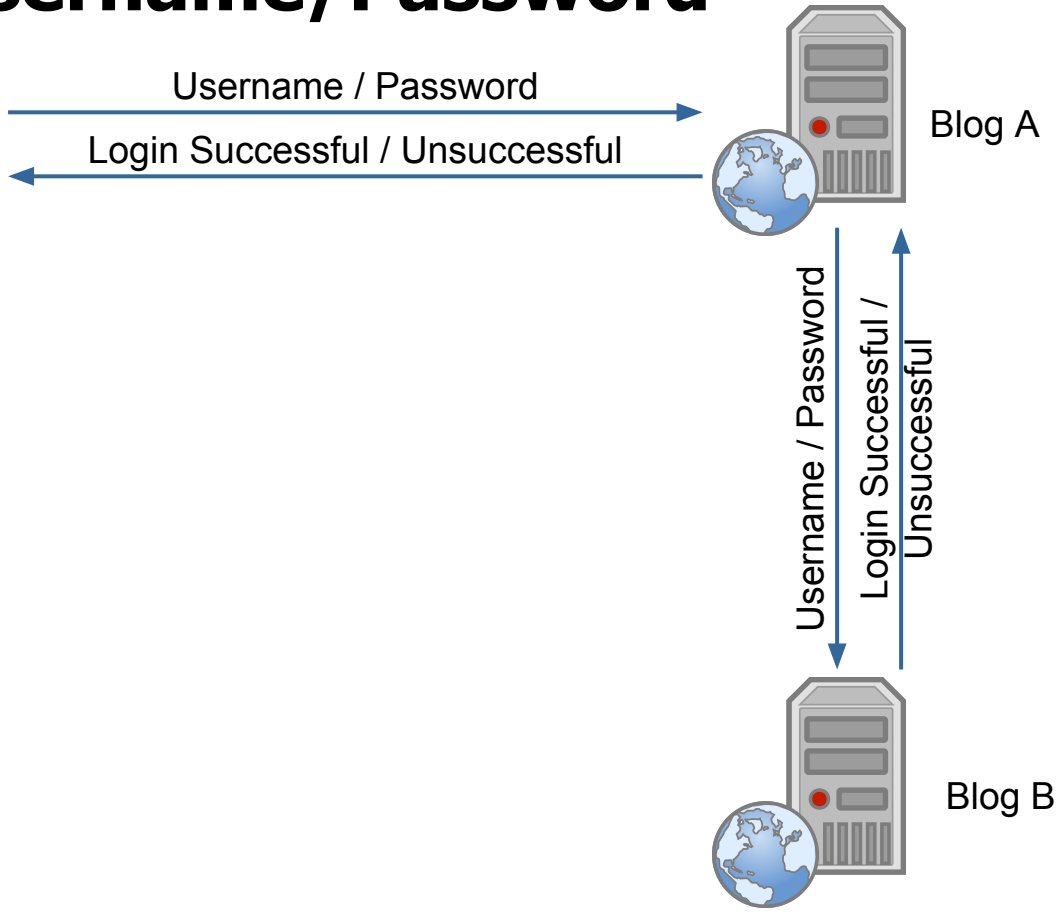
Username / Password

Login Successful / Unsuccessful

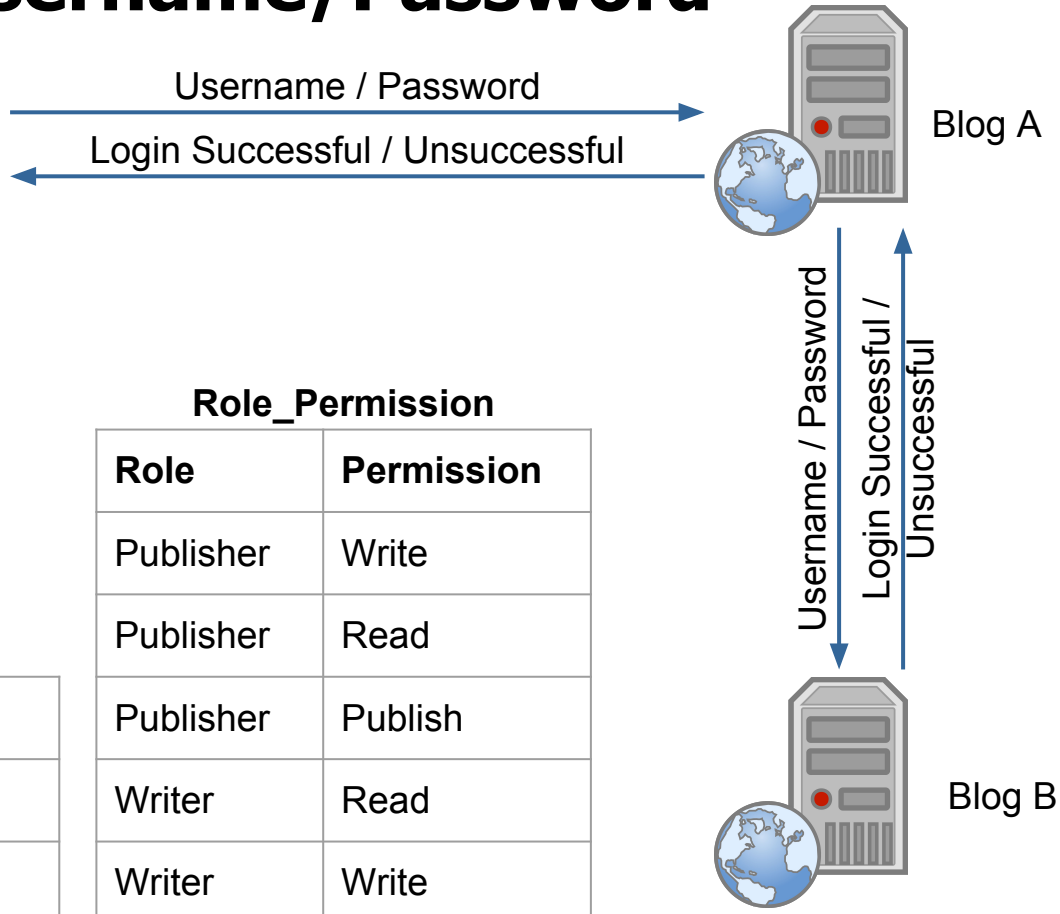
Blog A

Blog B

Classic Username/Password



Classic Username/Password



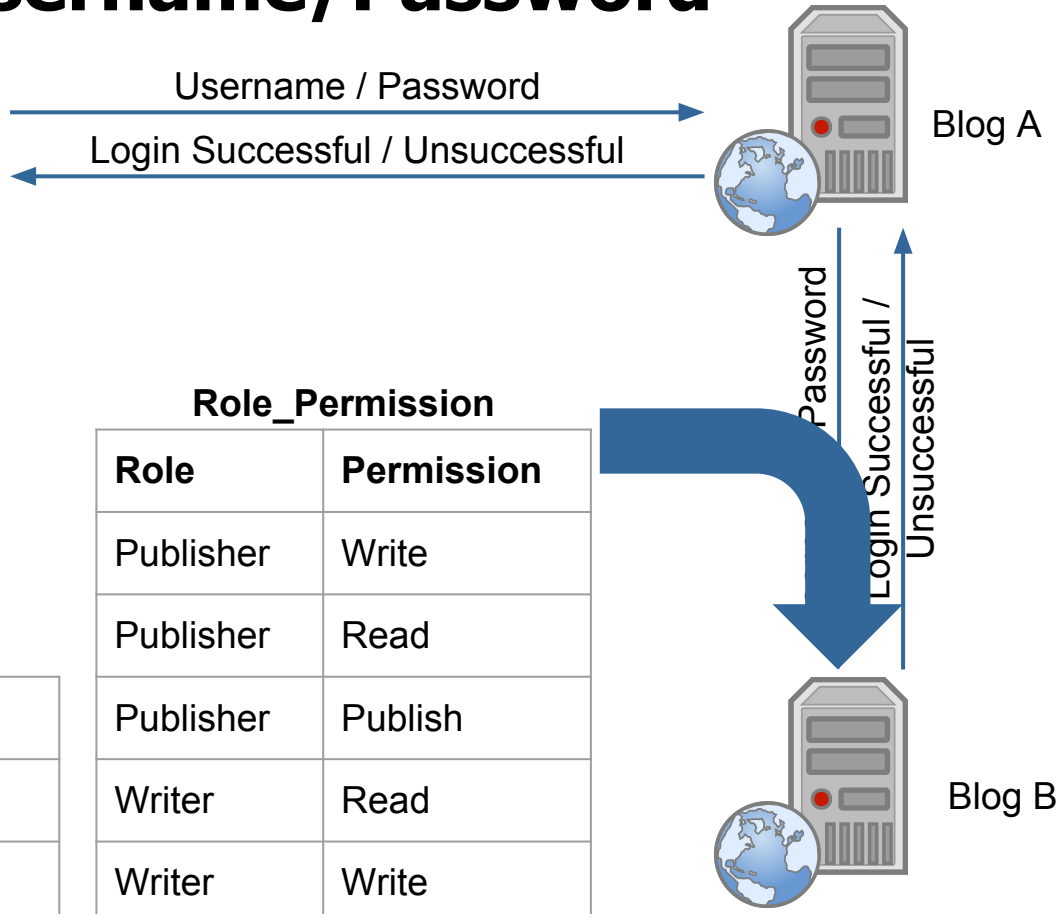
Role_Permission

Role	Permission
Publisher	Write
Publisher	Read
Publisher	Publish
Writer	Read
Writer	Write

Username_Role

Username	Role
Tux	Publisher
Tuxine	Writer

Classic Username/Password



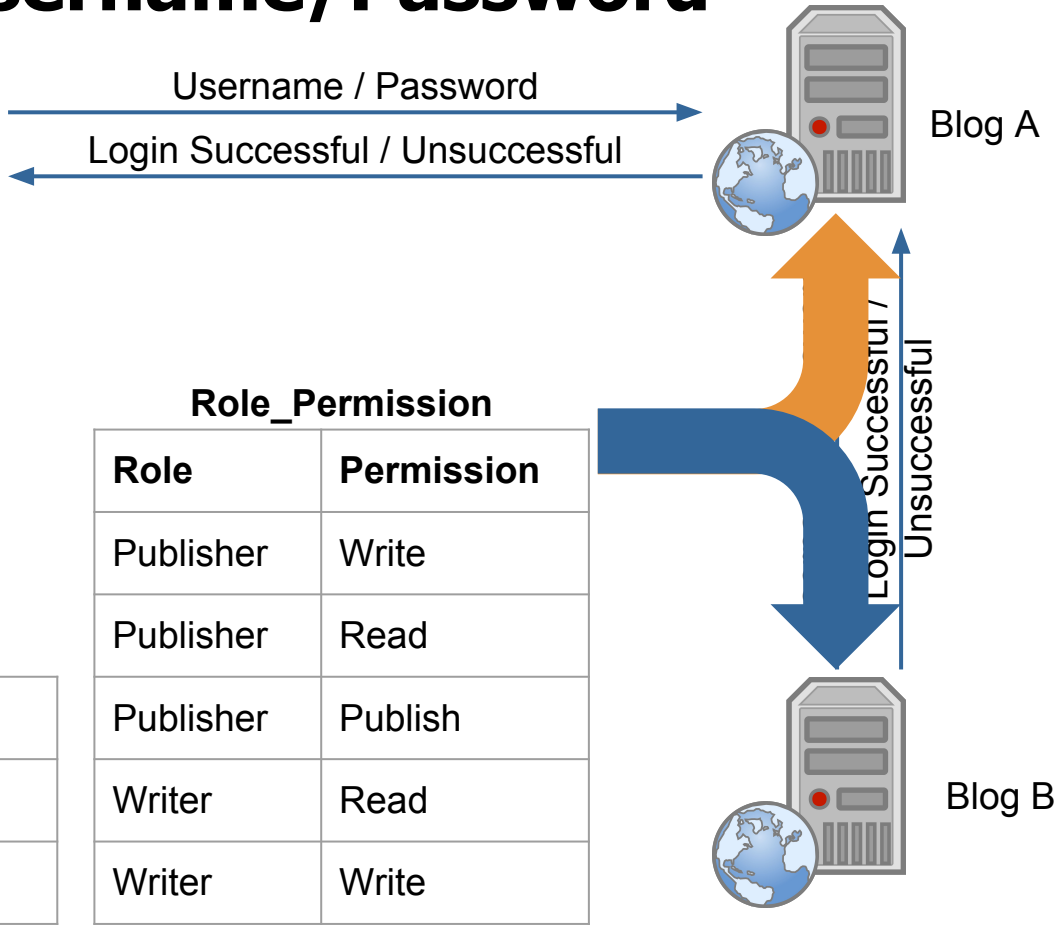
Role_Permission

Role	Permission
Publisher	Write
Publisher	Read
Publisher	Publish
Writer	Read
Writer	Write

Username_Role

Username	Role
Tux	Publisher
Tuxine	Writer

Classic Username/Password



Role_Permission

Role	Permission
Publisher	Write
Publisher	Read
Publisher	Publish
Writer	Read
Writer	Write

Username_Role

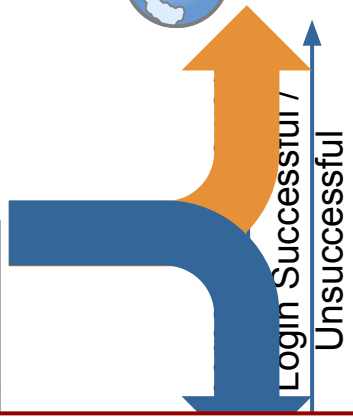
Username	Role
Tux	Publisher
Tuxine	Writer

Classic Username/Password

Username / Password

Login Successful / Unsuccessful

Blog A



Role_Permission

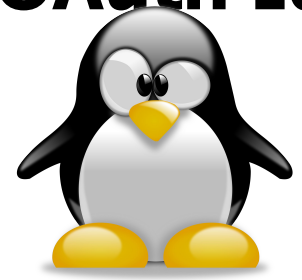
Role	Permission
Publisher	Write
Tuxine	Writer
Writer	Write

Usage of the UID/Password-Anti Pattern

How do we solve the UID-Password-Anti-Pattern?

-> Tokens

OAuth Idea



Blog A
(Client)

Username / Password

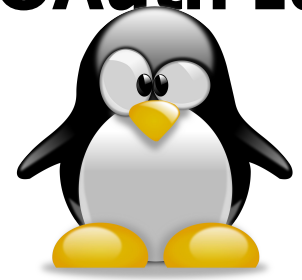
Token

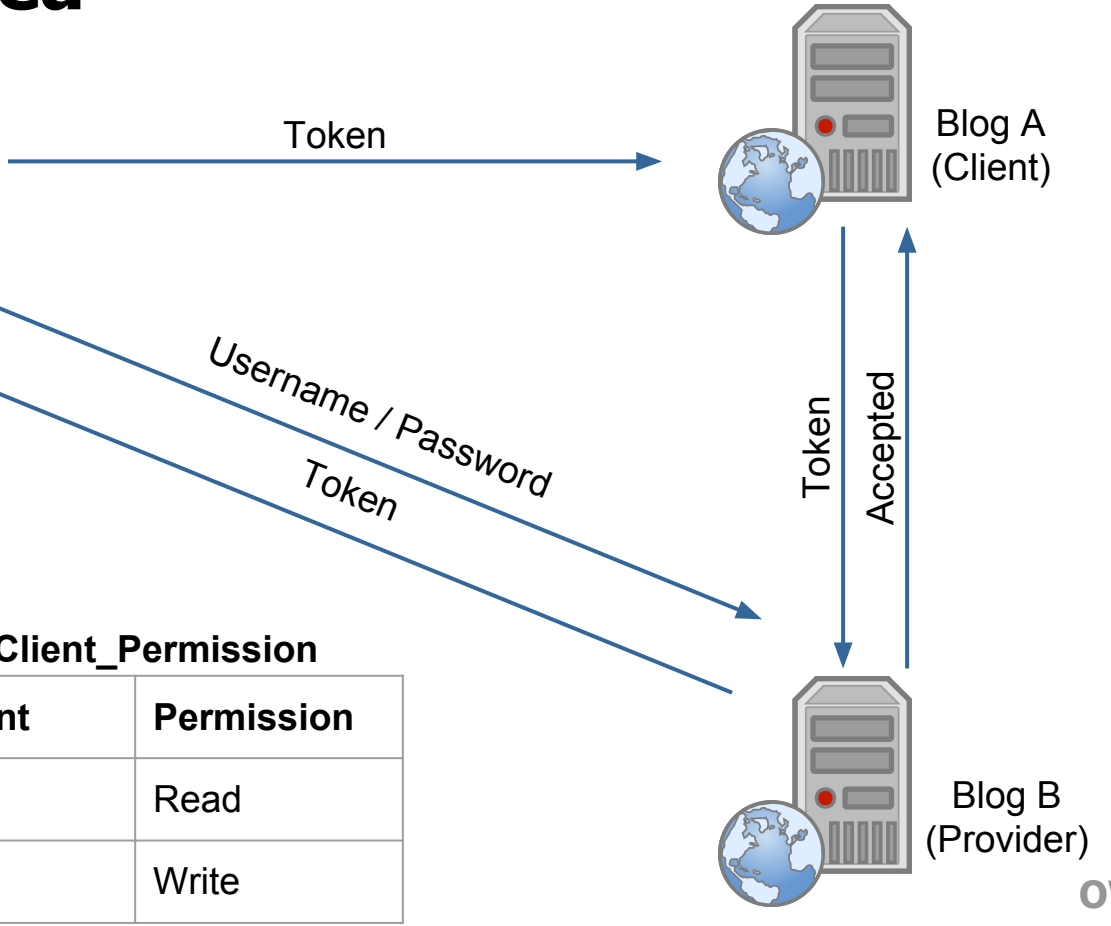
Client_Permission

Client	Permission
X…	Read
Y...	Write

Blog B
(Provider)

OAuth Idea





Client_Permission

Client	Permission
X…	Read
Y...	Write

Agenda

- Introduction
- Flows
- Conclusion

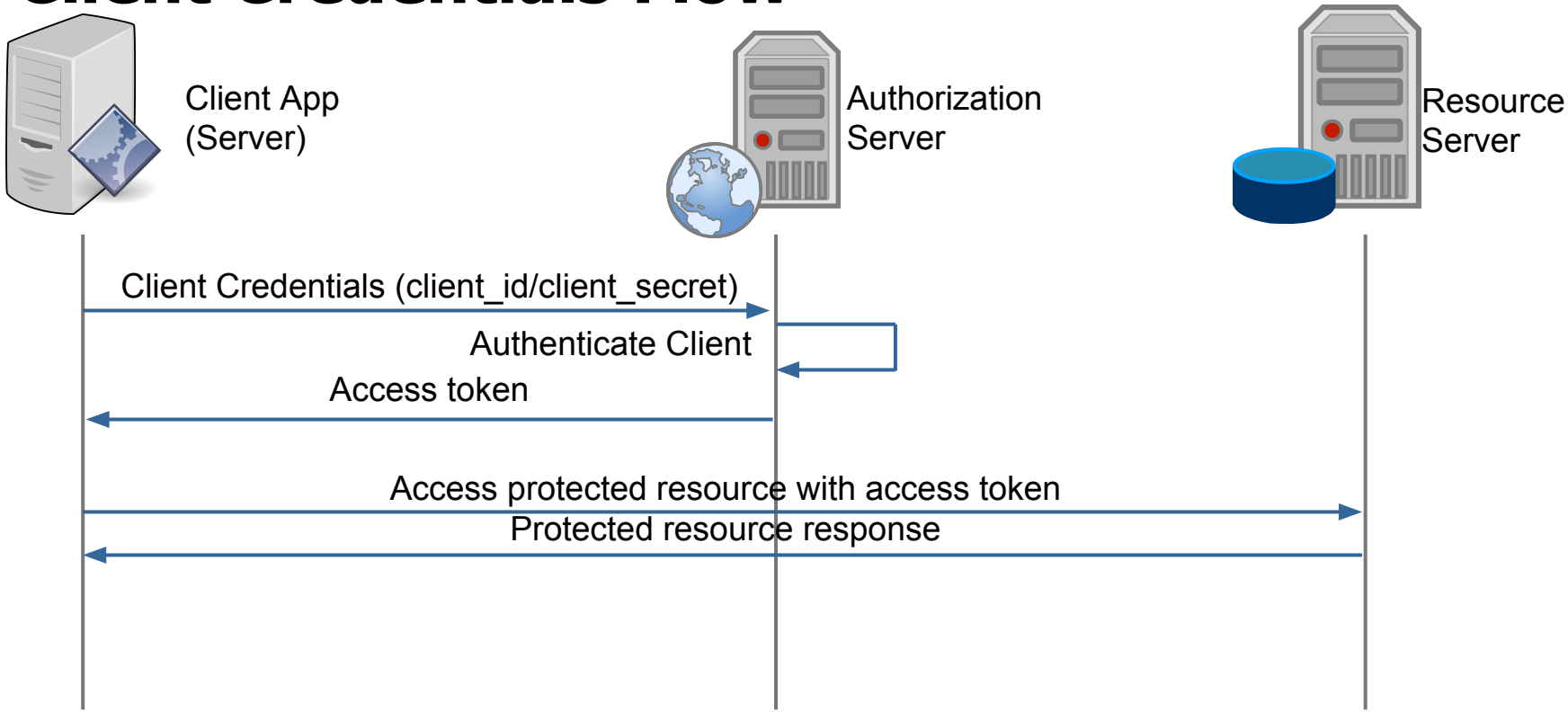
Client Credentials Flow

Client App
(Server)

Authorization
Server

Resource
Server

Client Credentials Flow



Overview Client Credentials Flow

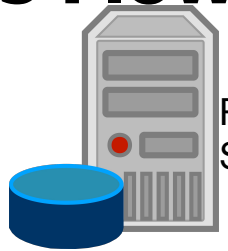
- No user-based Authentication
  Scope/Permissions: Bound to clients
- Usage: Intranet

Resource Owner Password Credentials Flow

Resource Owner

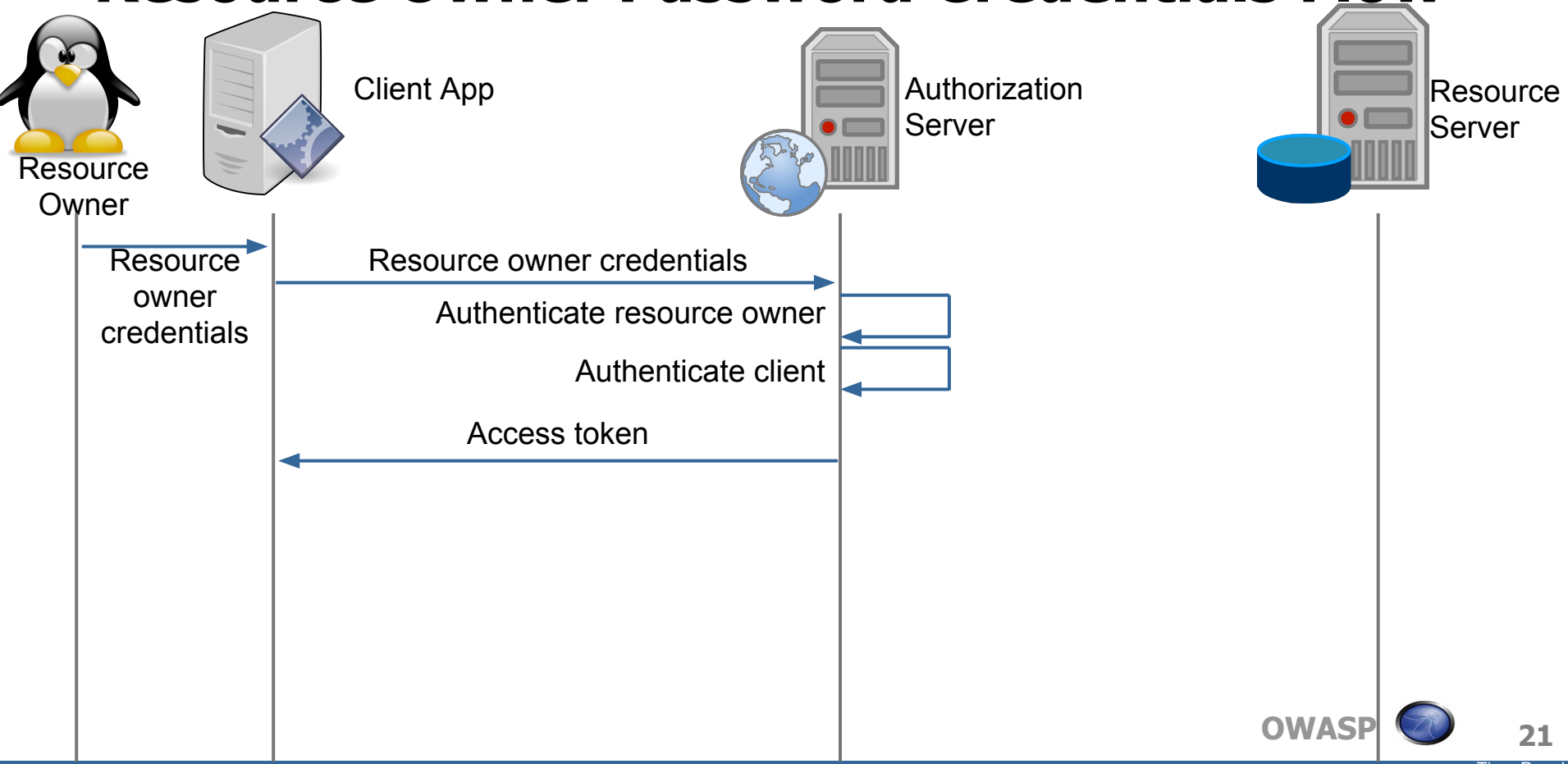
Client App

Authorization Server



Resource Server

Resource Owner Password Credentials Flow



Hochzeitsmühle an

https://www.muehle-hoffnung.de/admin

Joomla!®

Please enter your Google
Username and Google Password

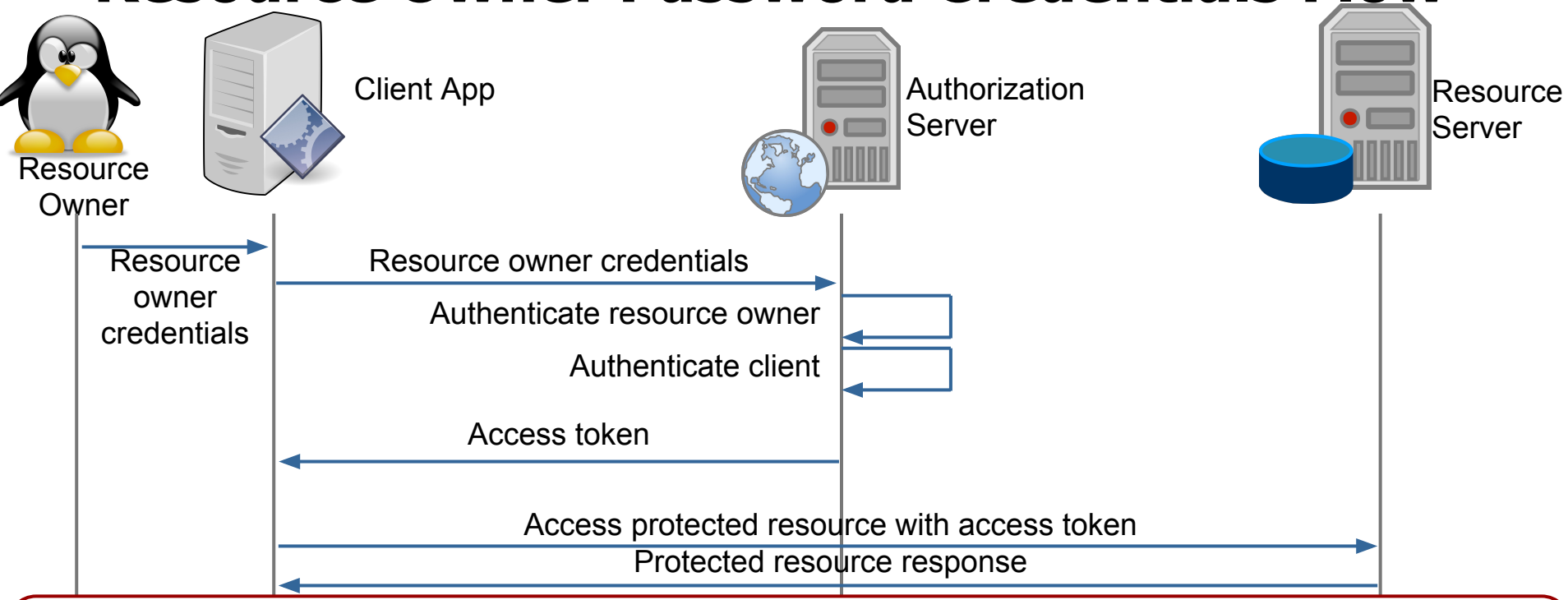
Username

Password

Language - Default

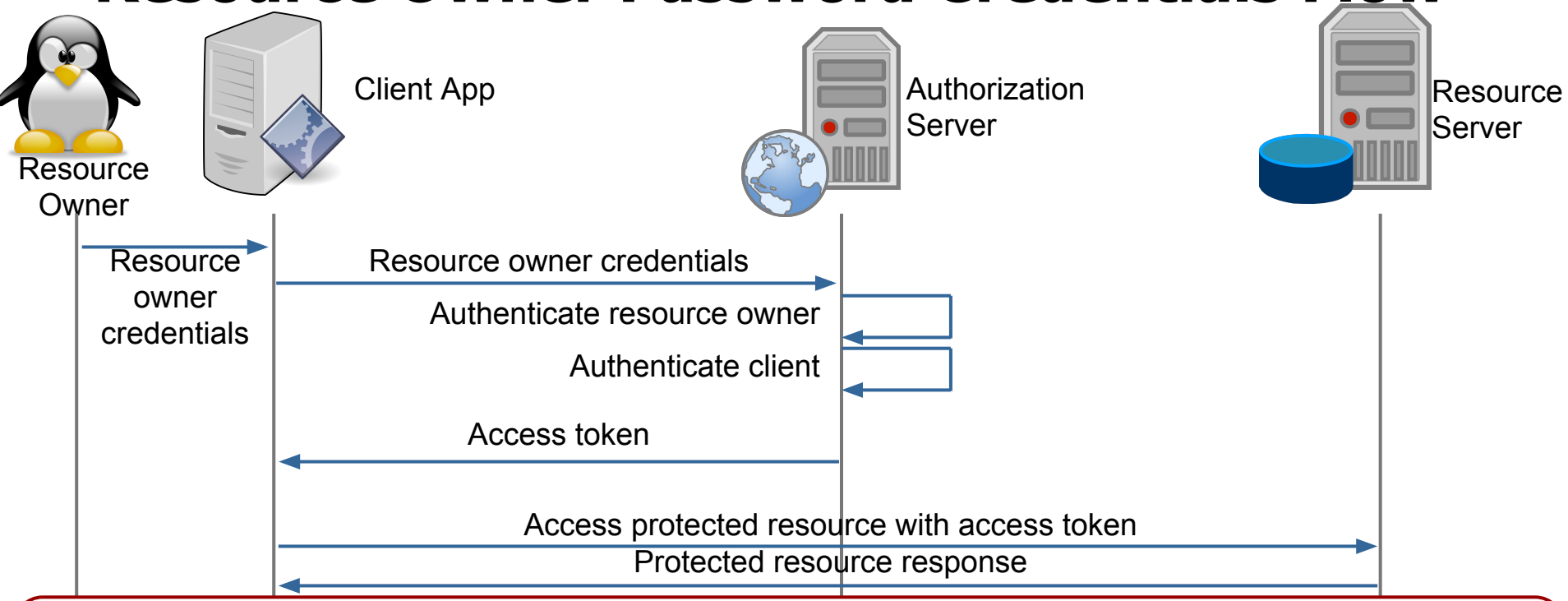
Log in

Resource Owner Password Credentials Flow



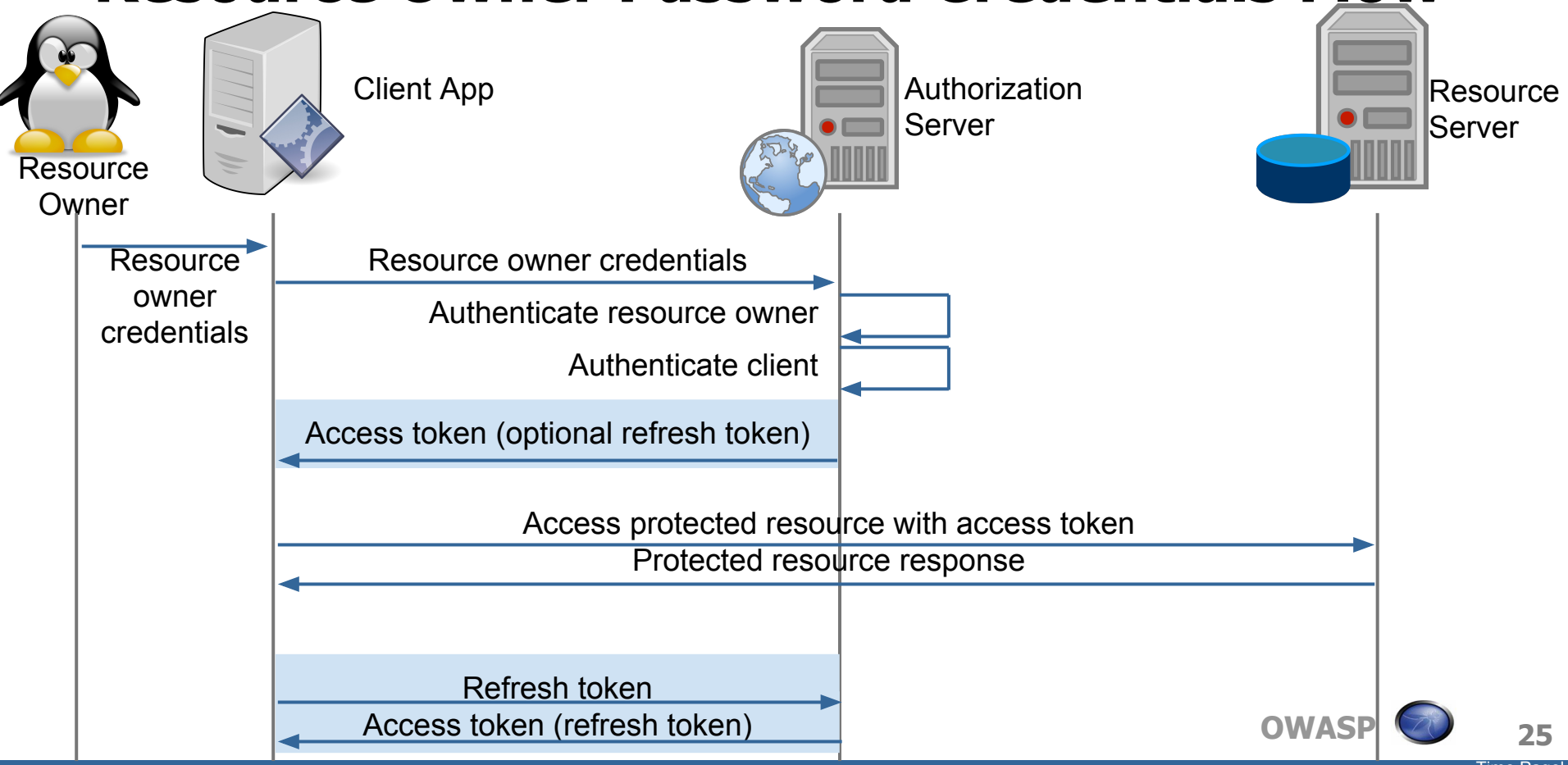
Usage of the UID/Password-Anti Pattern

Resource Owner Password Credentials Flow



What happens after the access token has expired?

Resource Owner Password Credentials Flow



OAuth2 ROPC-Specification

[...] The **resource owner password credentials** grant type is suitable in cases where the

resource owner has a trust relationship with the client,

such as the device operating system [...]

Source: RFC 6749 The OAuth 2.0 Authorization Framework - Section 4.3

Interpretation of OAuth ROPC-Specification

- The client and the device are completely under your control
- All other flows are not supported by the client

Interpretation of OAuth ROPC-Specification

- Use Case: To move legacy application into the OAuth2-Universe
  - Scope
  - Expiration of tokens
  - …

ROPC Main Risks Overview

- UID/password anti-pattern

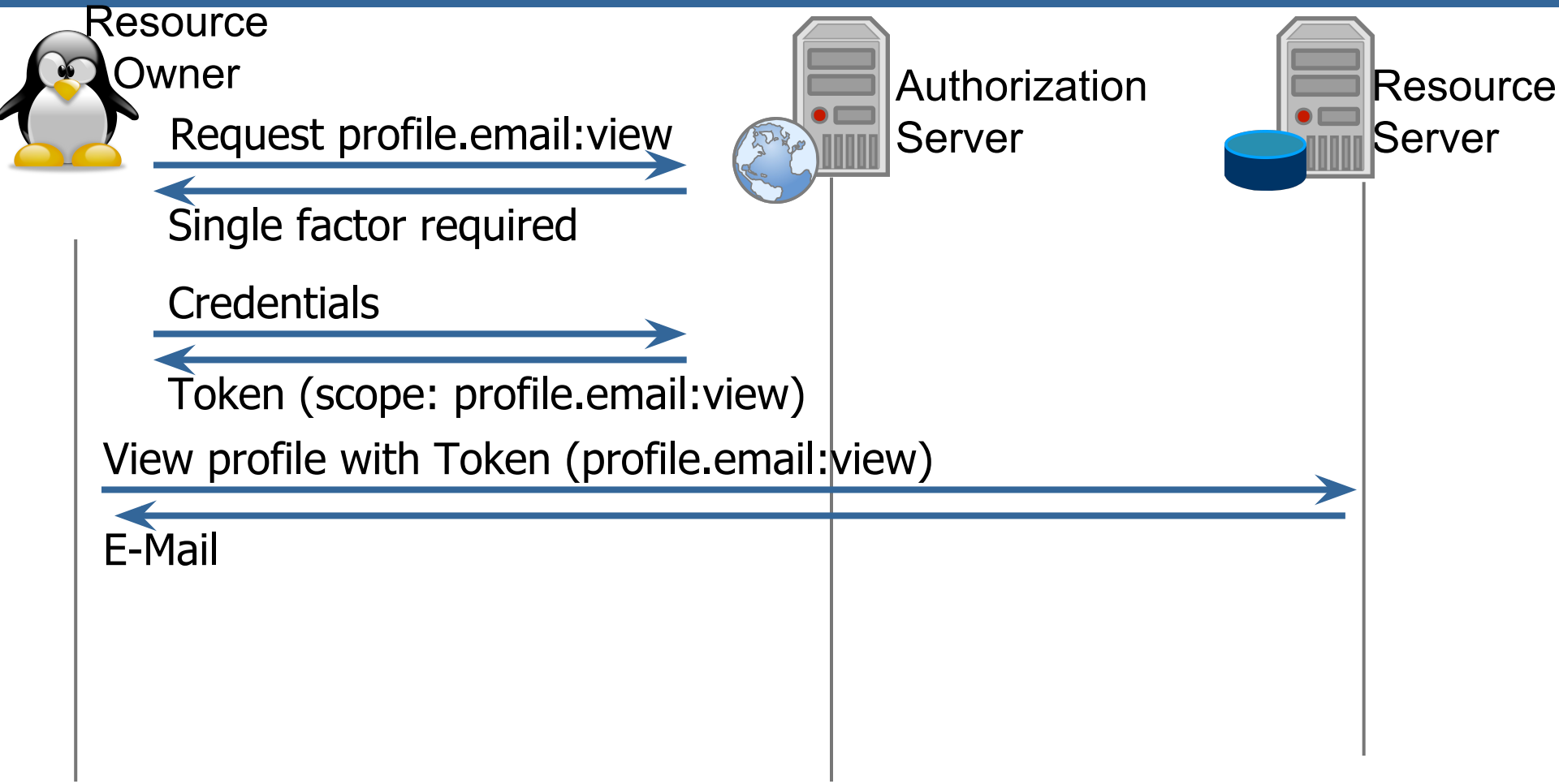
-> client, eavesdroppers, or endpoints could eavesdrop the user id and password

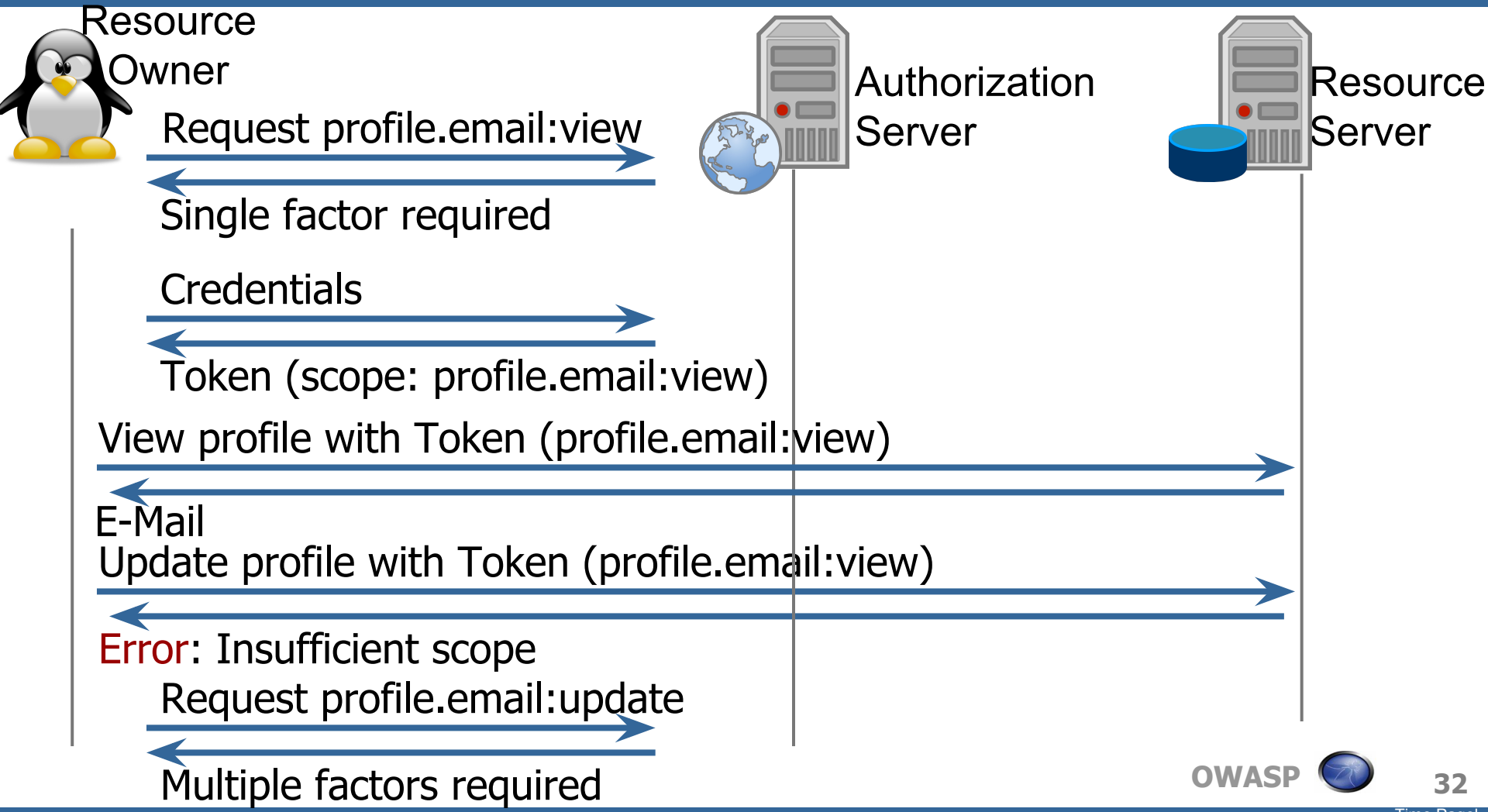
- Validation of the client's identity not possible
- Client app might issue a not needed scope

- Token revocation nearly useless

Scopes

Action	Scope
View own email	profile.email:view
Modify own email	profile.email:update
Delete own email	profile.email:delete

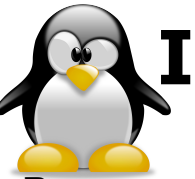

Resource Owner
Authorization Server
Resource Server
Request profile.email:view
Single factor required
Credentials
Token (scope: profile.email:view)
View profile with Token (profile.email:view)
E-Mail


Resource Owner
Authorization Server
Resource Server
Request profile.email:view
Single factor required
Credentials
Token (scope: profile.email:view)
View profile with Token (profile.email:view)
E-Mail
Update profile with Token (profile.email:view)
Error: Insufficient scope
Request profile.email:update
Multiple factors required

Implicit Flow

- Use Case: Browser
- Client Secret: Confidentiality can not be guaranteed

Implicit Flow



Resource Owner

Client/Browser

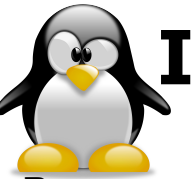
JavaScript

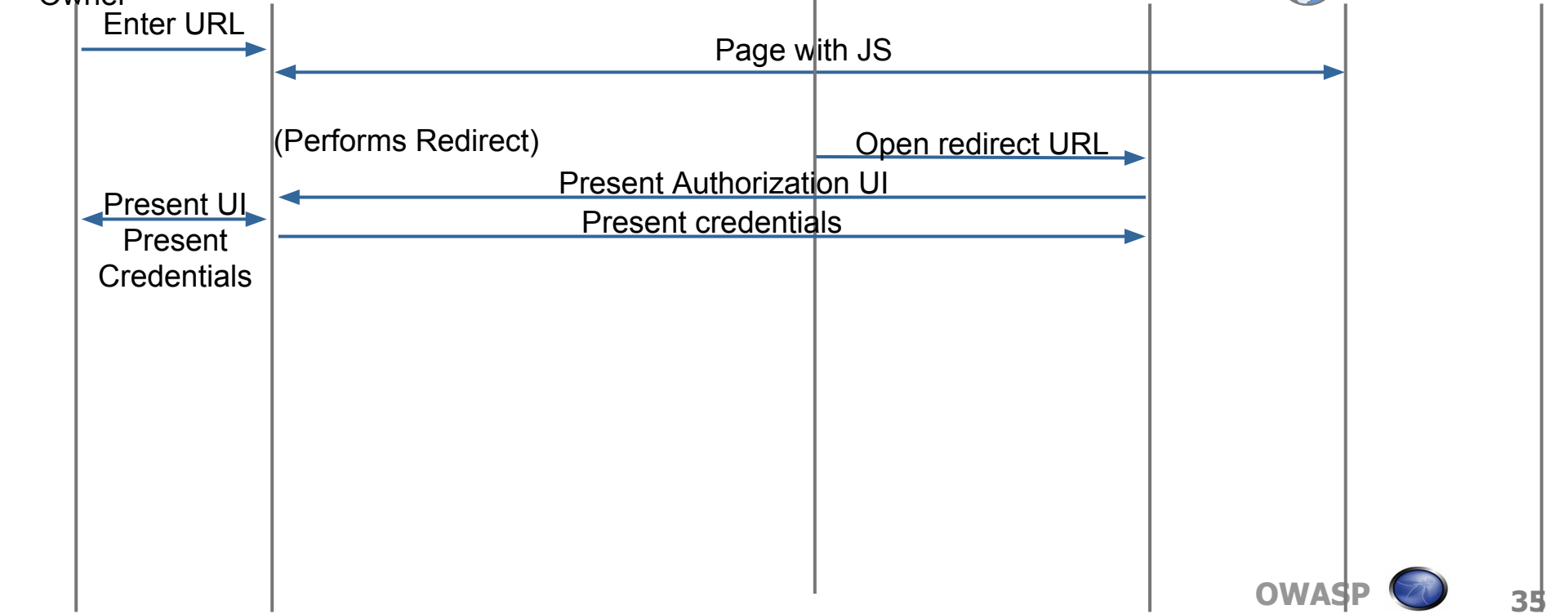
Authorization Server

Frontend Server

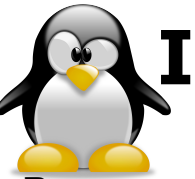
Resource Server

Implicit Flow





Implicit Flow



Resource Owner

Client/Browser

JavaScript

Authorization Server

Frontend Server

Resource Server

Enter URL

Page with JS

(Performs Redirect)

Open redirect URL

Present Authorization UI

Present UI

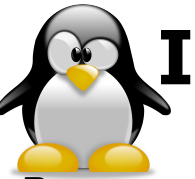
Present credentials

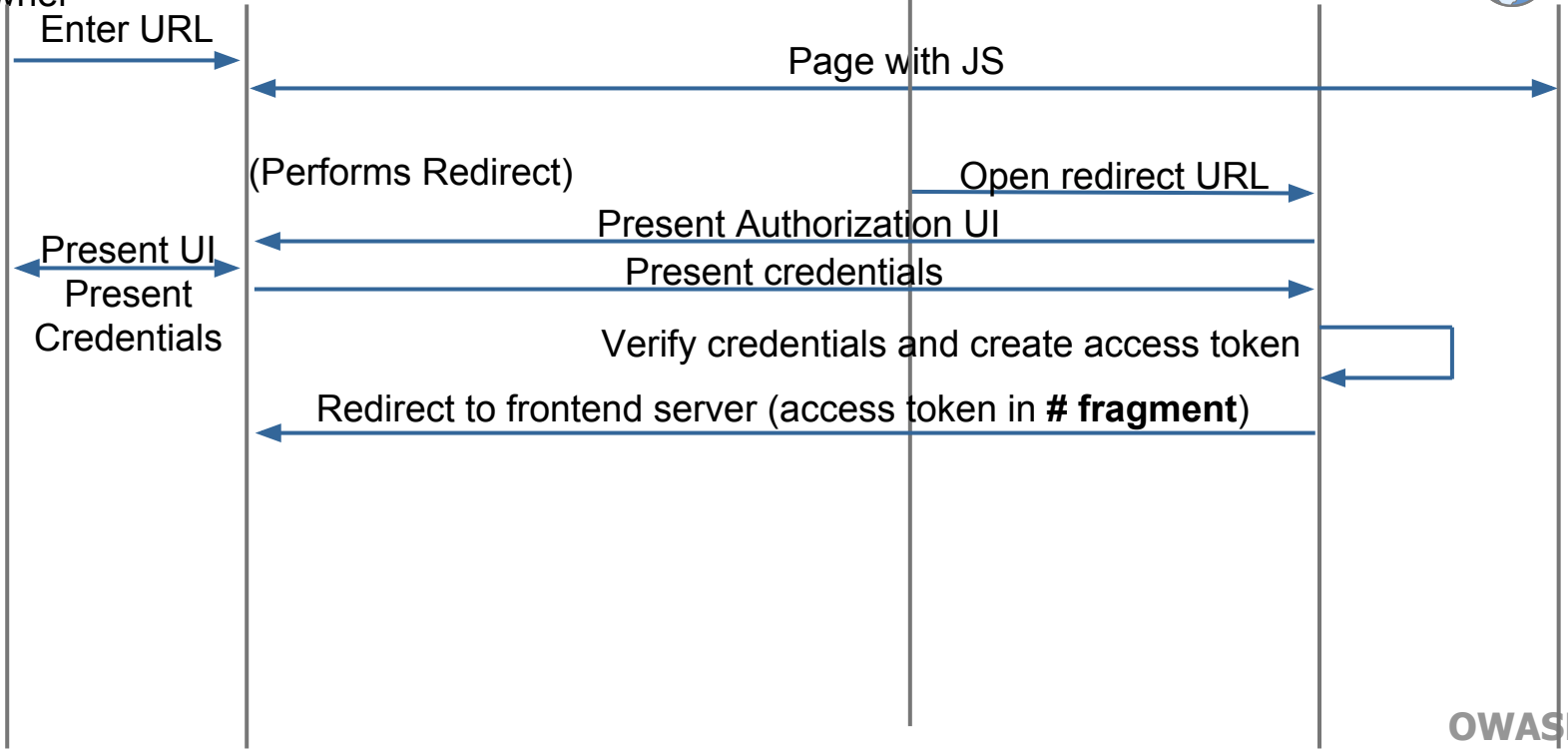
Present Credentials

Verify credentials and create access token

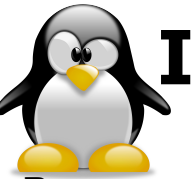
Redirect to frontend server (access token in # fragment)

Implicit Flow





Implicit Flow



Resource Owner

Client/Browser

JavaScript

Authorization Server

Frontend Server

Resource Server

Enter URL

Page with JS

(Performs Redirect)

Open redirect URL

Present Authorization UI

Present UI

Present credentials

Present Credentials

Verify credentials and create access token

Redirect to frontend server (access token in # fragment)

Follow redirect URL (without access token) and get page with JS

Extract and temp. store access token

Call protected resource with access token

Return protected resource

Threats Implicit Flow

- Resource owners might issue a token to a malicious client (e.g. via phishing)
- Attackers might steal token via other mechanisms

Source: RFC 6749 The OAuth 2.0 Authorization Framework - Section 10.16

- Main Risk: Whom is a token issued to?

Further Risks/Info

- Use Case: Browser-Applications
- Silent Refresh
- Disadvantages: Man-in-the-Middle can fetch tokens

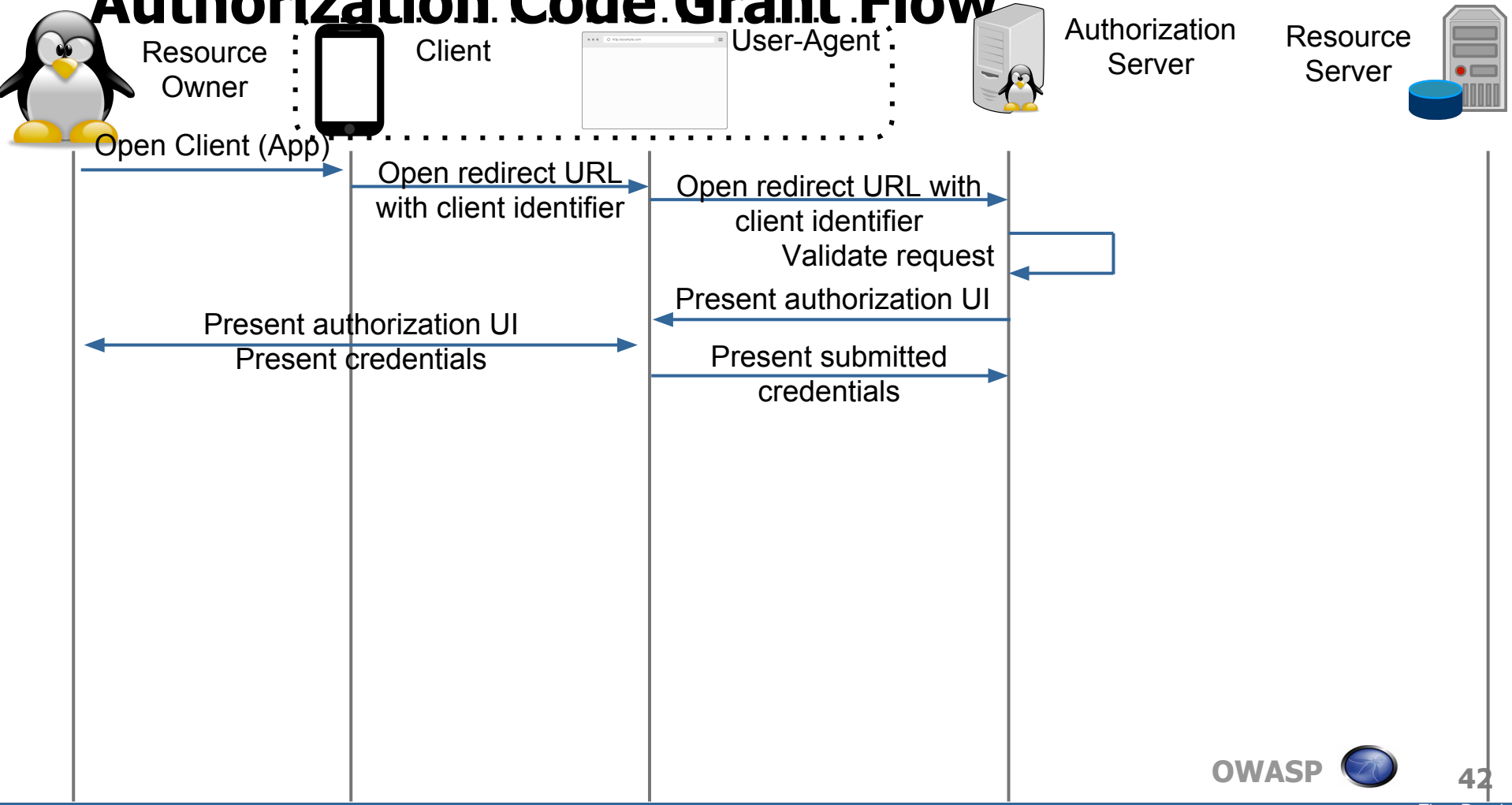
-> No refresh tokens

Authorization Code Grant

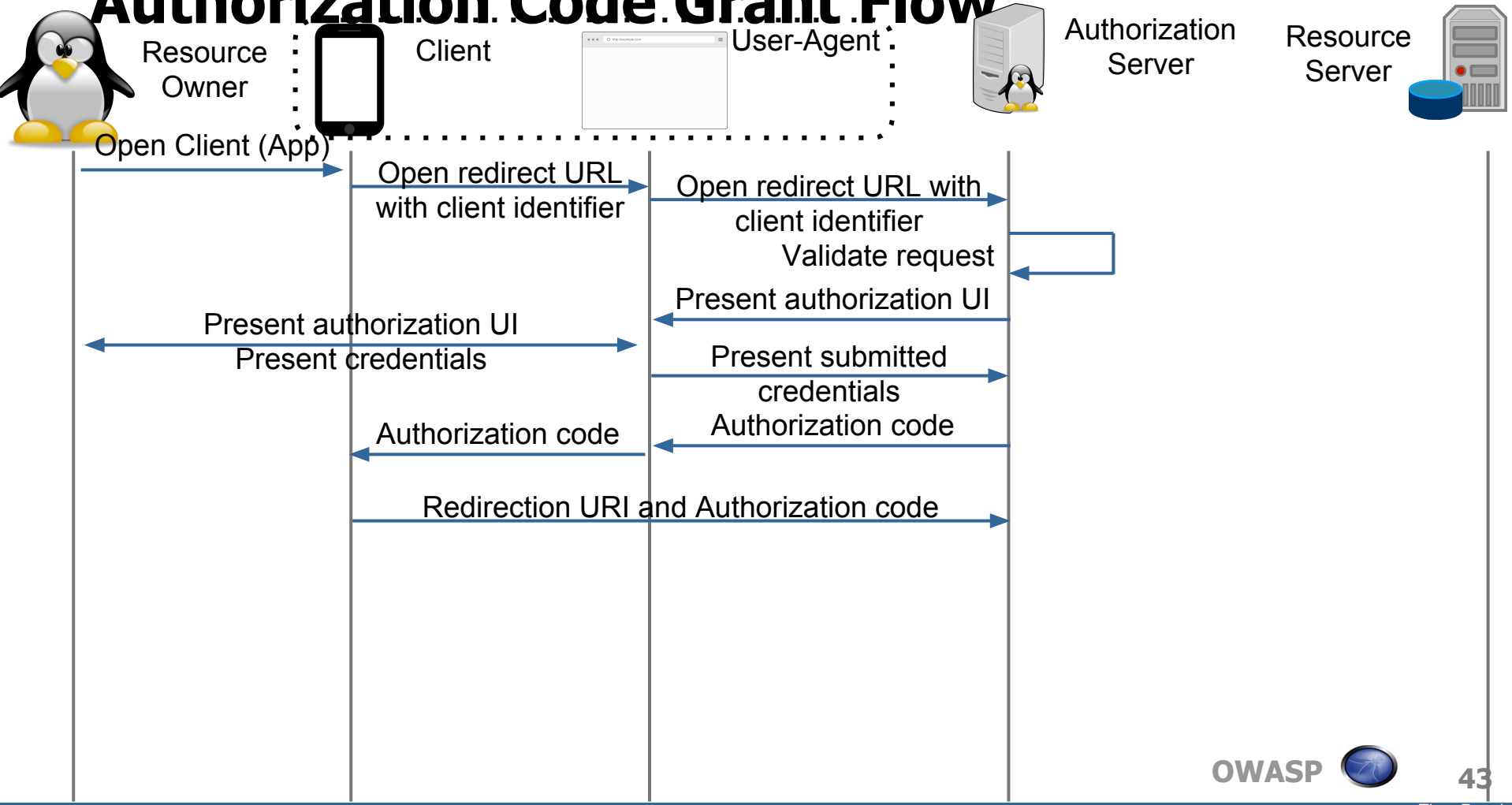
[...] the Authorization Code flow should only be used [...] where the Client Secret can be **safely stored**. [...]

https://auth0.com/docs/api-auth/tutorials/authorization-code-grant

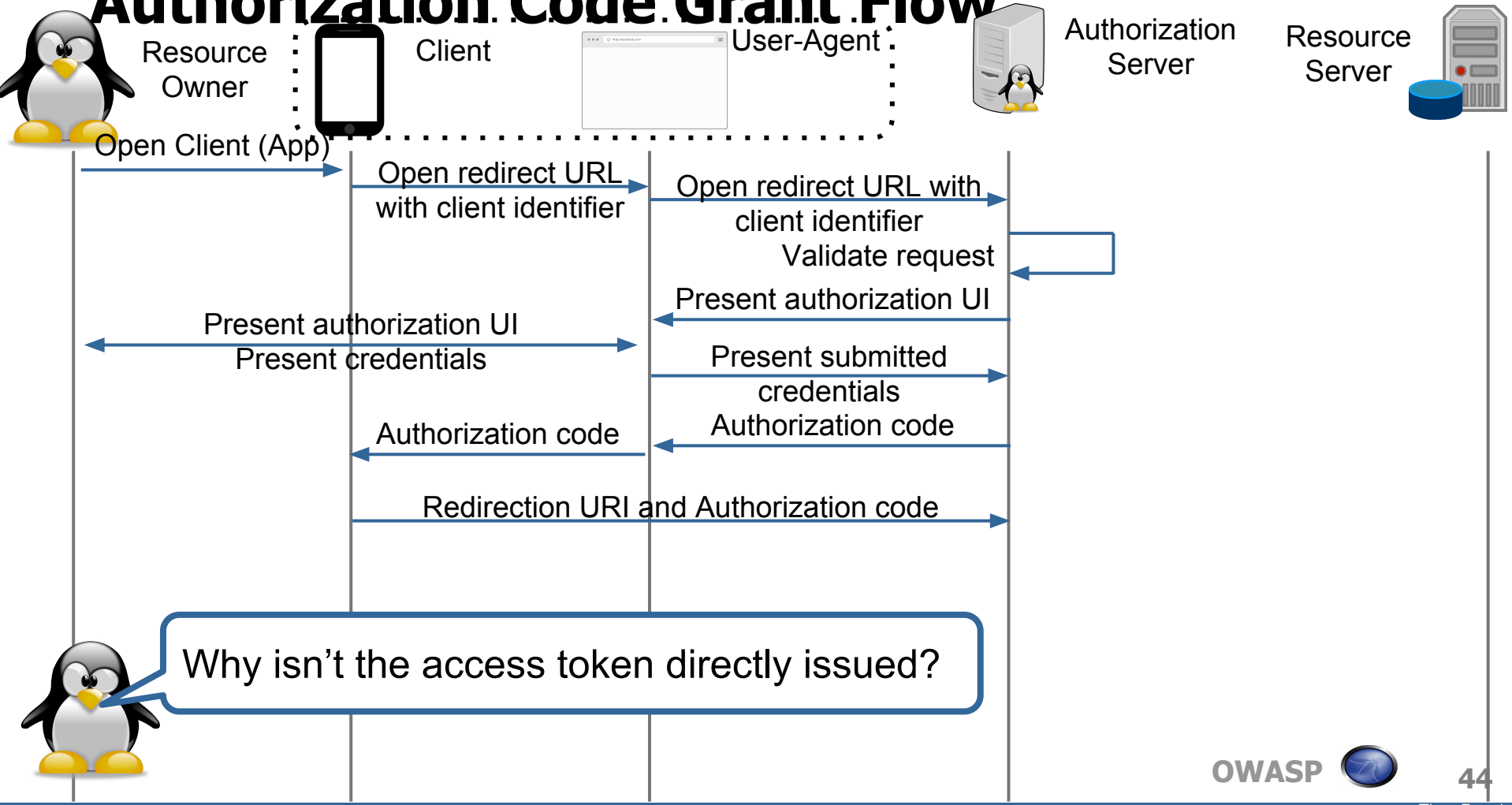
Authorization Code Grant Flow



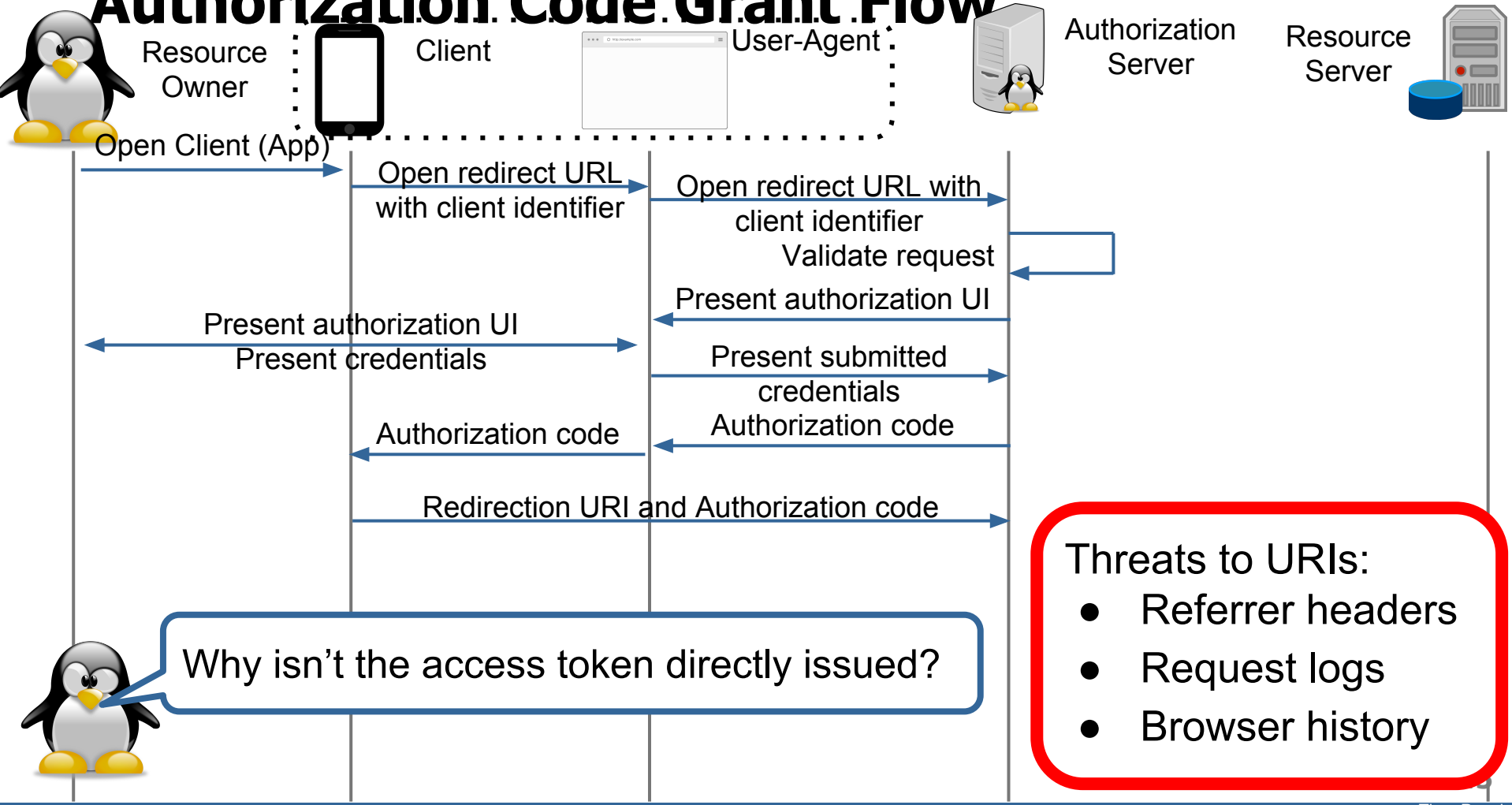
Authorization Code Grant Flow



Authorization Code Grant Flow



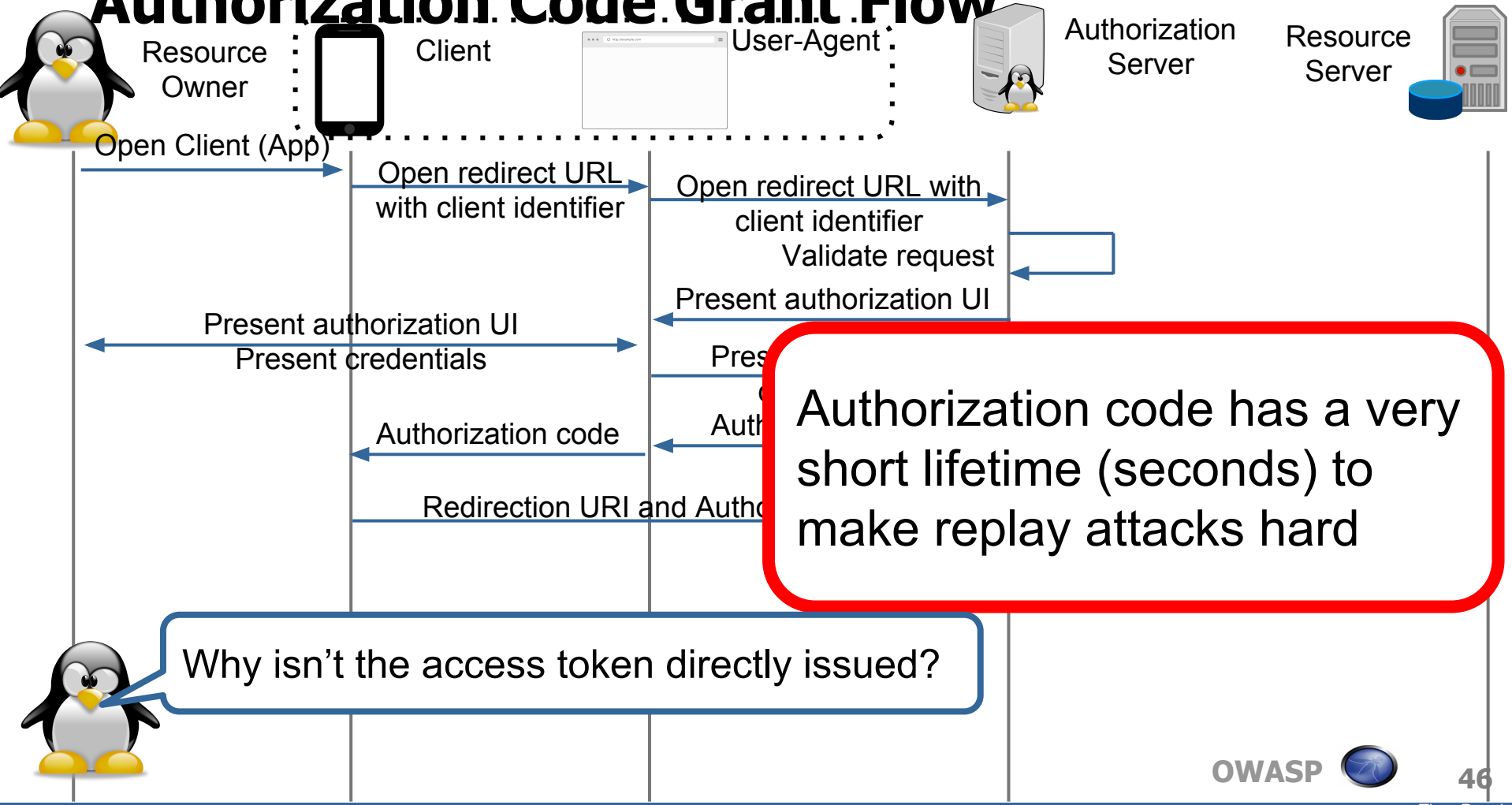
Authorization Code Grant Flow



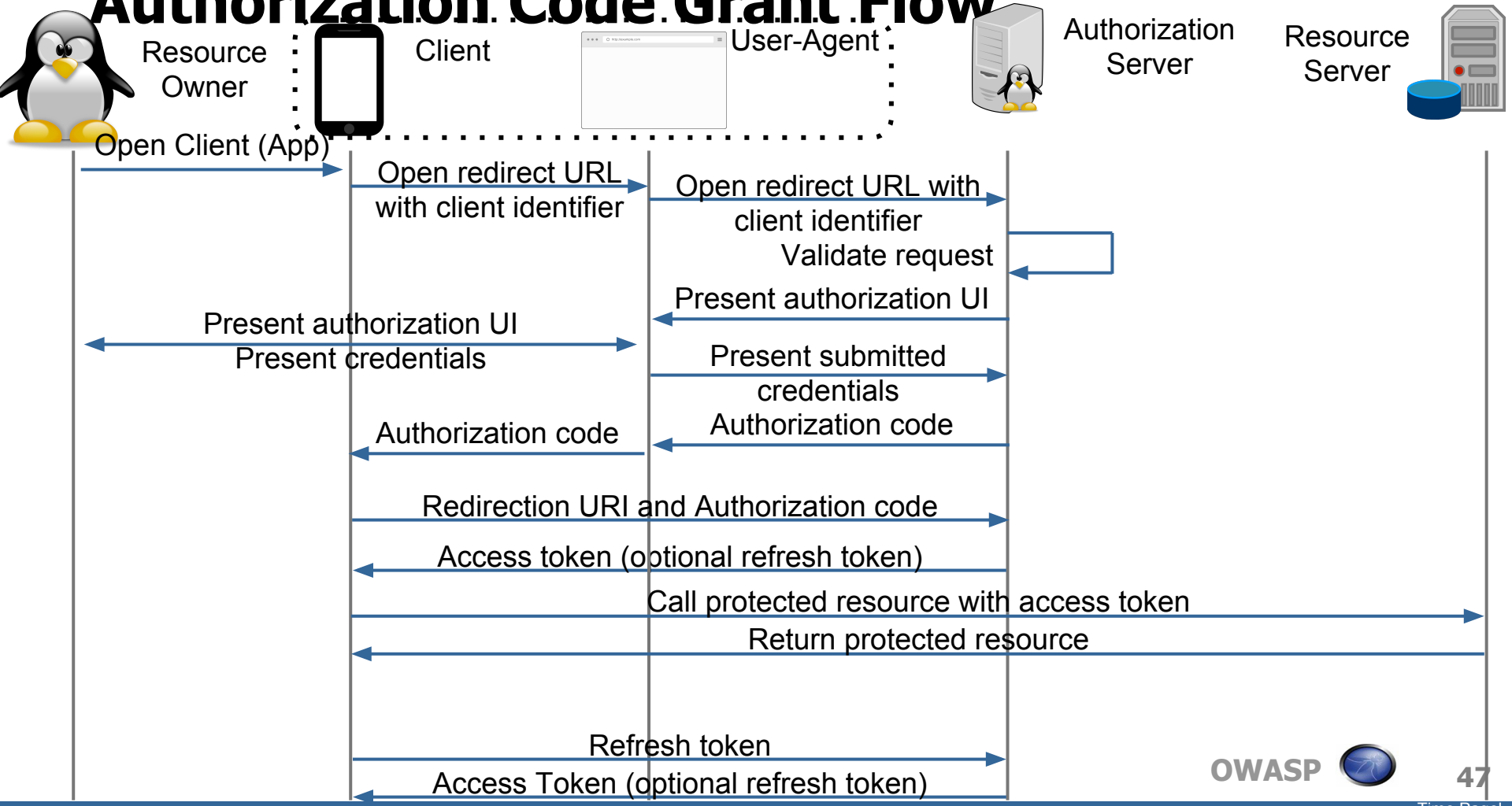
Threats to URIs:

- Referrer headers
- Request logs
- Browser history

Authorization Code Grant Flow



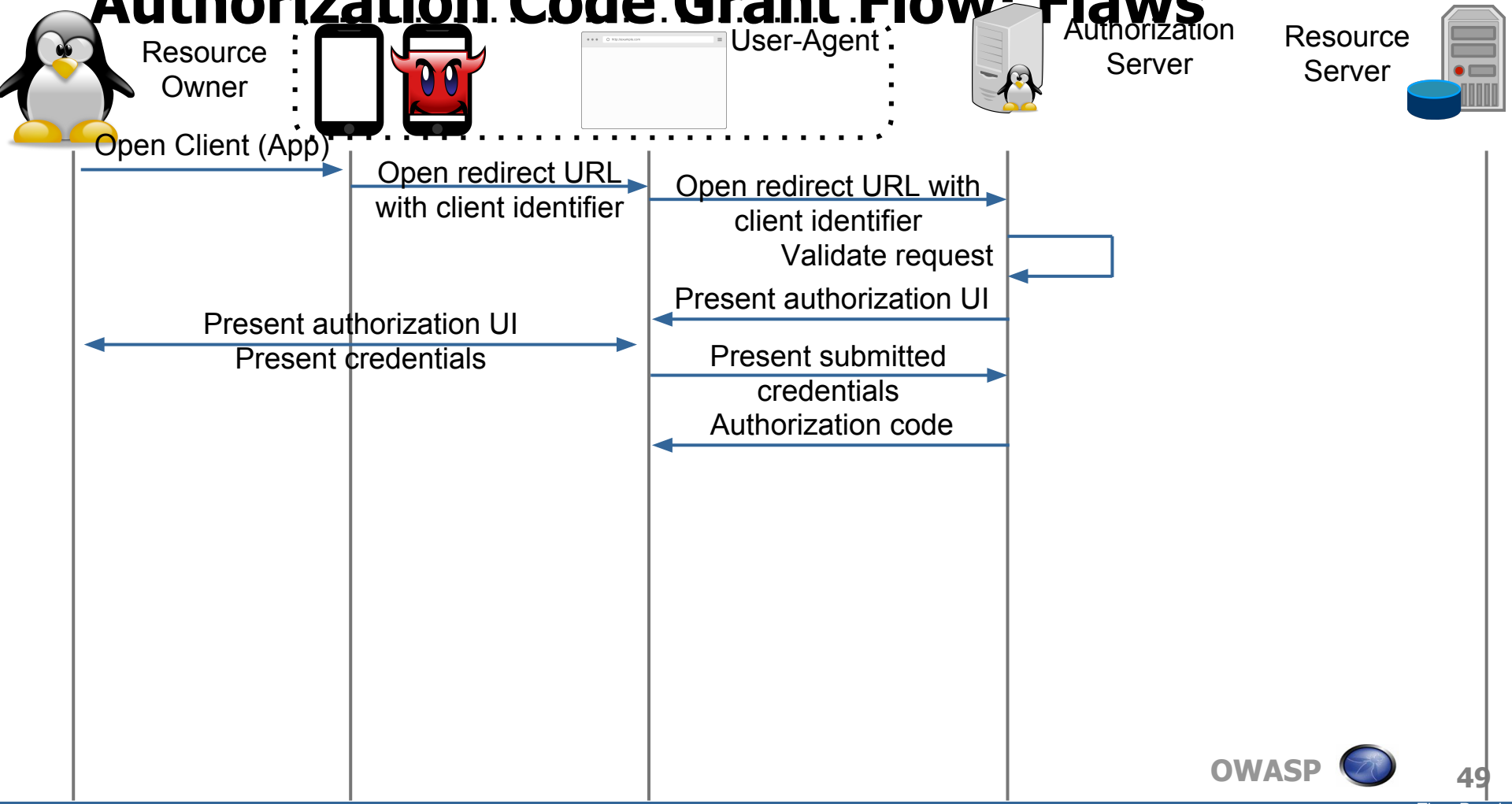
Authorization Code Grant Flow



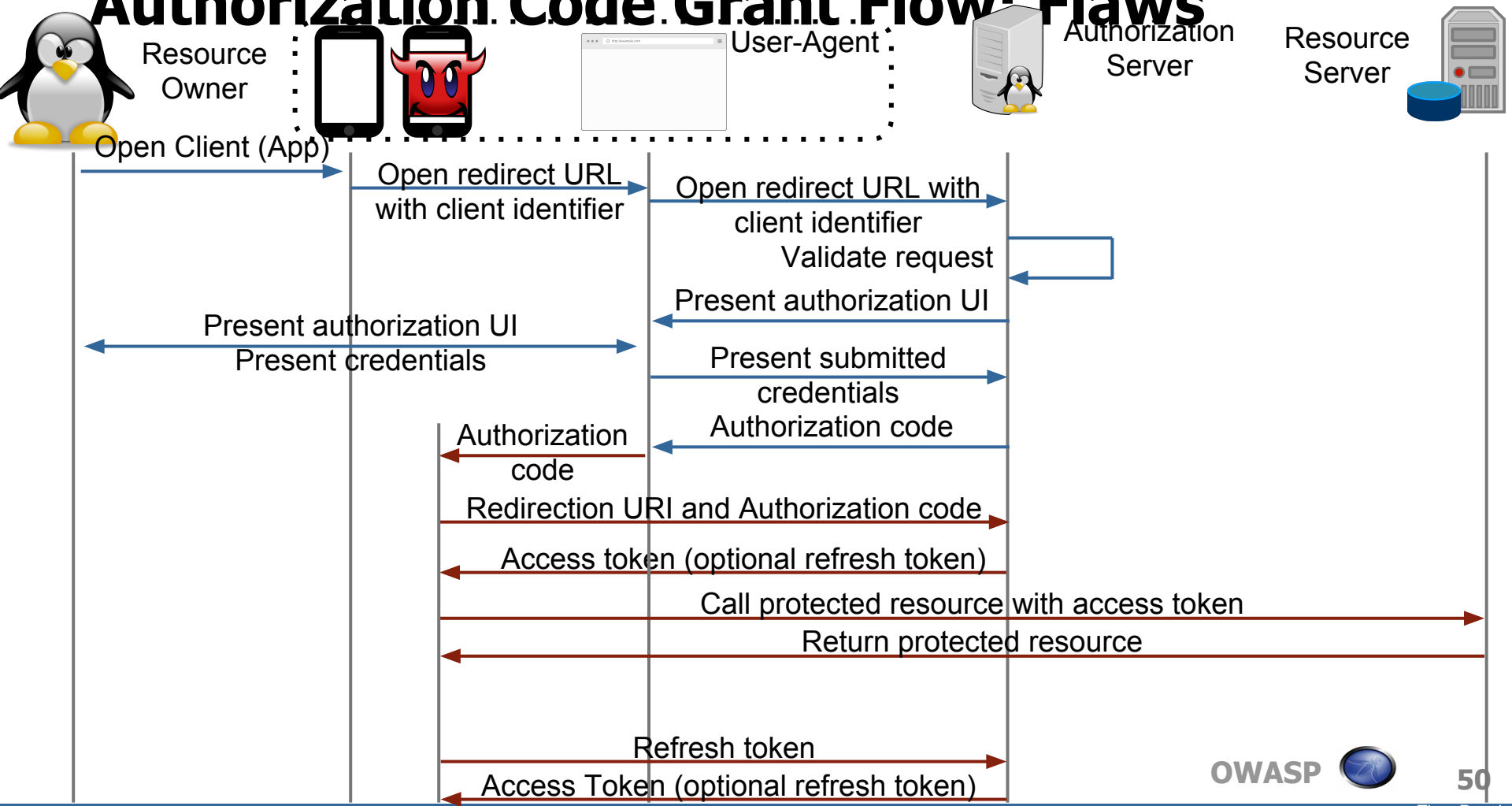
Native App Flow

Mainly: Proof Key for Code Exchange - PKCE (RFC 7636)

Authorization Code Grant Flow: Flaws



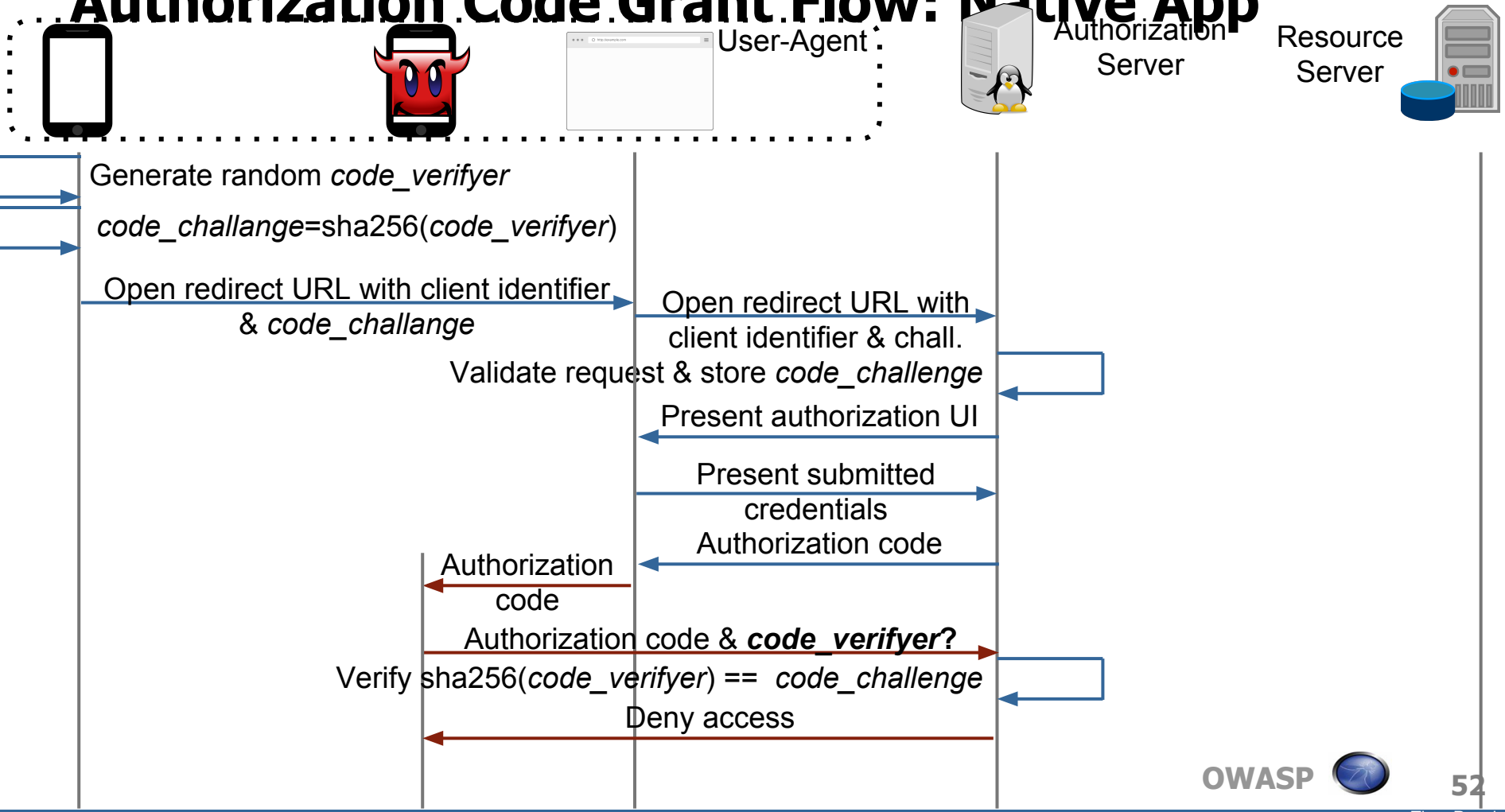
Authorization Code Grant Flow: Flaws



RFC 8252: OAuth 2.0 for Native Apps

- External User Agent:
  - External browser/app
  - In-App browser tab

Authorization Code Grant Flow: Native App



Further Security Considerations

- URI-Schema:
  - Domain-Related, e.g. *com.fhunii.eventmarketing*
  - Prevent DNS-Spoofing: Use 127.0.0.1 instead of localhost by performing redirection on localhost (Desktop)
- Defence against cross-app request forgery:
  - Usage of the *state* parameter with a random
- Embedded User Agent (Web-View):
  - Must open an external browser as the embedded user agent has full access to authorization grant

Agenda

- Introduction
- Flows
- Conclusion

Conclusion

- Choose the flow based on the use case
  - App: Auth. Code Grant + Native Apps
  - Web: Implicit Flow

Questions?

oauth2019@pagel.pro

Implementation Flaws

Store username and generate password in the client after authentication

Implementation Flaws

Storing the username/password locally

OWASP Juice Shop

localhost:3000/#/login

Login

Email

Password

Log in

Log in with Google

Remember me

Forgot your password? Not yet a customer?

Sign in - Google Acc

Secure | https://accounts.google.com/signin/oauth/oauthch...

Google

Choose an account

to continue to OWASP Juice Shop

T Timo Pagel
timo.pagel@owasp.org

T Timo Pagel
google@timo-pagel.de

Use another account

OWASP Juice Shop

localhost:3000/#/search?q=invalid'))%20UNION%20SELECT...

ct Us

cle

lain?

Us

ch Results

d')) UNION SELECT NULL,email,password,id,NULL,NULL,NULL,NULL FROM U

	Product	Description	Pri
[redacted]	admin@juice-sh.op	0192023a7bbd73250516f069df18b500	1
[redacted]	bender@juice-sh.op	0c36e517e3fa95aabf1bbffc6744a4ef	3
[redacted]	bjoern.kimminich@googlemail.com	448af65cf28e8adeab7ebb1ecff66f15	4
[redacted]	ciso@juice-sh.op	861917d5fa5f1172f931dc700d81a8fb	5
[redacted]	google@timo-pagel.de	421a487b68e3a4f057d84996968c5e2a	7
[redacted]	jim@juice-sh.op	e541ca7ecf72b8d1286474fc613e5e45	2
[redacted]	support@juice-sh.op	d57386e76107100a7d6c2782978b2e7b	6

Implications

- Endless Refresh?
- No Caching for shared proxies with Authentication-Header
- Logout -> Invalidation of Refresh/Access-Tokens
- Monitoring of unauthorized invalid Tokens usage attempts
- No-Algo Attack

Agenda

- Introduction
- Flows
- Implementation Flaws
- Conclusion

Conclusion

- OAuth2 is used to delegate access
- Choose the right flow for your use case
- OAuth2 does not prevent from thinking on your own! -> harden endpoints/processes

Risk Overview

https://auth0.com/docs/api-auth/which-oauth-flow-to-use

Flow	Client (Application)	Overall Risk
Resource Owner Password Credentials Flow	Browser / Mobile App	Critical (with public clients)
Authorization Code Flow	Confidential Client	Medium-High
Implicit Flow	Browser (JavaScript)	Medium-High
Authorization Code Flow (PKCE)	Mobile App	Medium

OAuth ROPC-Specification

*It is also used to **migrate** existing clients using direct authentication schemes such as HTTP Basic or Digest authentication to OAuth by converting the stored credentials to an access token.*

Source: RFC 6749 The OAuth 2.0 Authorization Framework - Section 4.3

Hardening Resource Owner Password Credentials Flow (not recommended) 1/2

- Harden Token Endpoint:
  - Do not allow cross-domain requests
  - Brute Force / “Token Brute Force”
  - Timing Attacks
  - Lack of security sensitive information
  - Throttling Policy
  - …
- Reduce Risk of Stolen Tokens:
  - TLS
  - Disable refresh tokens and use short lived access tokens
  - Reconsider lifetime of tokens

Hardening Resource Owner Password Credentials Flow (not recommended) 2/2

- Inform resource owners about password reuse
- Limit usage to org. where client/application and authorizing service are from the same org.
- The authorization server may generally restrict the scope of access tokens issued by this flow