Securing RESTful Resources with OAuth2

Rodrigo Cândido da Silva
@rcandidosilva

JavaOne 2014
CON4990

About Me

- Brazilian guy ;)
- Software Architect
  - Java Platform
- Work for Integritas Tech
  - http://integritastech.com
- JUG Leader of GUJavaSC
  - http://gujavasc.org
- Twitter
  - @rcandidosilva
- Personal
  - http://rodrigocandido.me

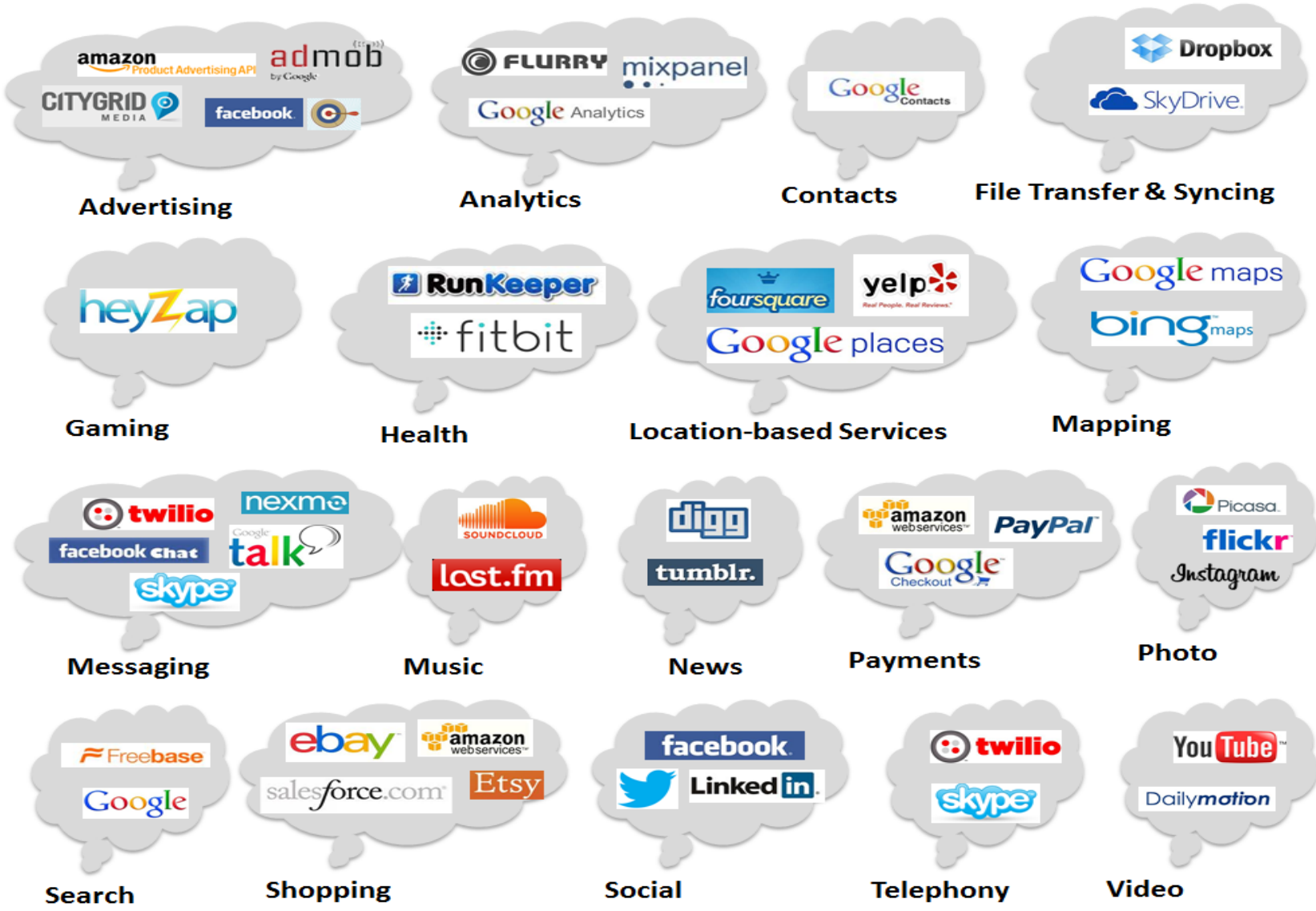
integritas

Agenda

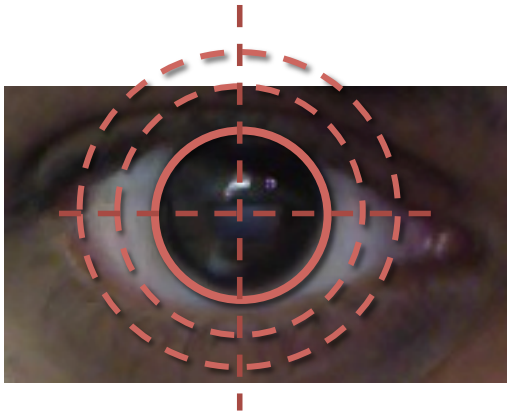
- Why use OAuth2?
- OAuth2 concepts
- Grant types
- OAuth2 Tokens
- Java Implementations
- Demo

integritas

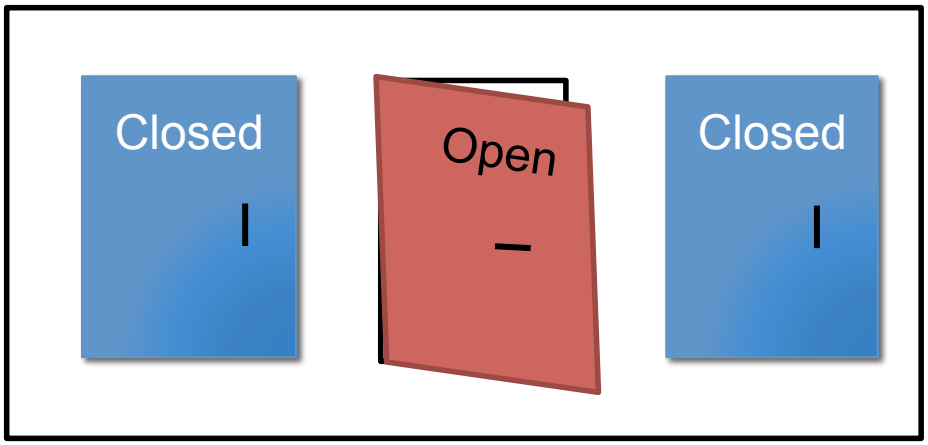
Public Web Service API's



Security



Authentication



Authorization

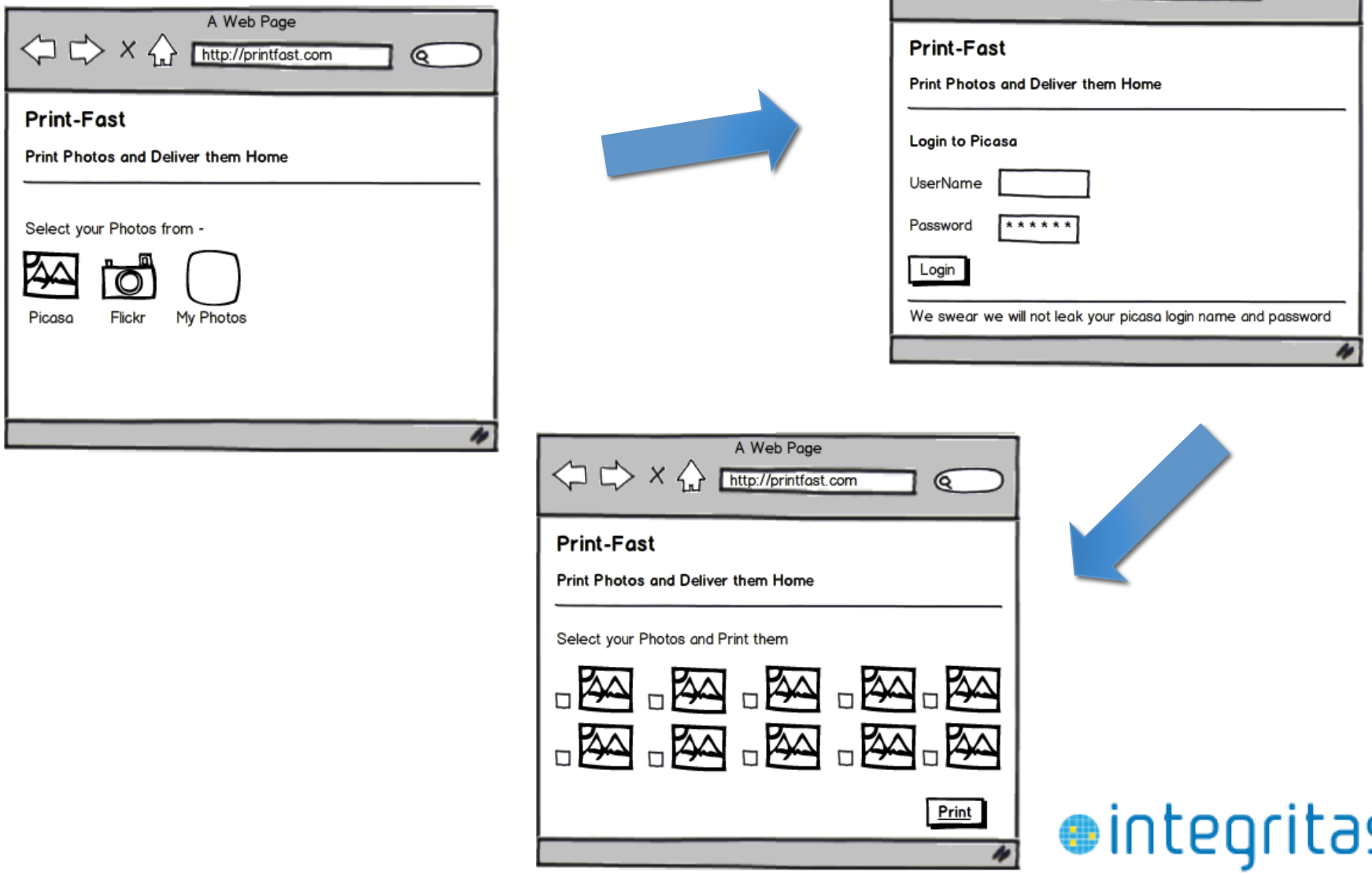
integritas

Securing APIs

- Securing resources strategies
  - Basic Auth (HTTP Basic)
    - Sending user credentials in http authentication header
  - Mutual Authentication (HTTP Digest)
    - Based on certificates, server authenticate to client, client to server
- RESTful architecture not defines security procedures
  - HTTP methods: GET, POST, PUT, DELETE
- REST API's are equal vulnerable as standard web apps
  - Injection attacks, replay attacks, cross-site scripting, etc.

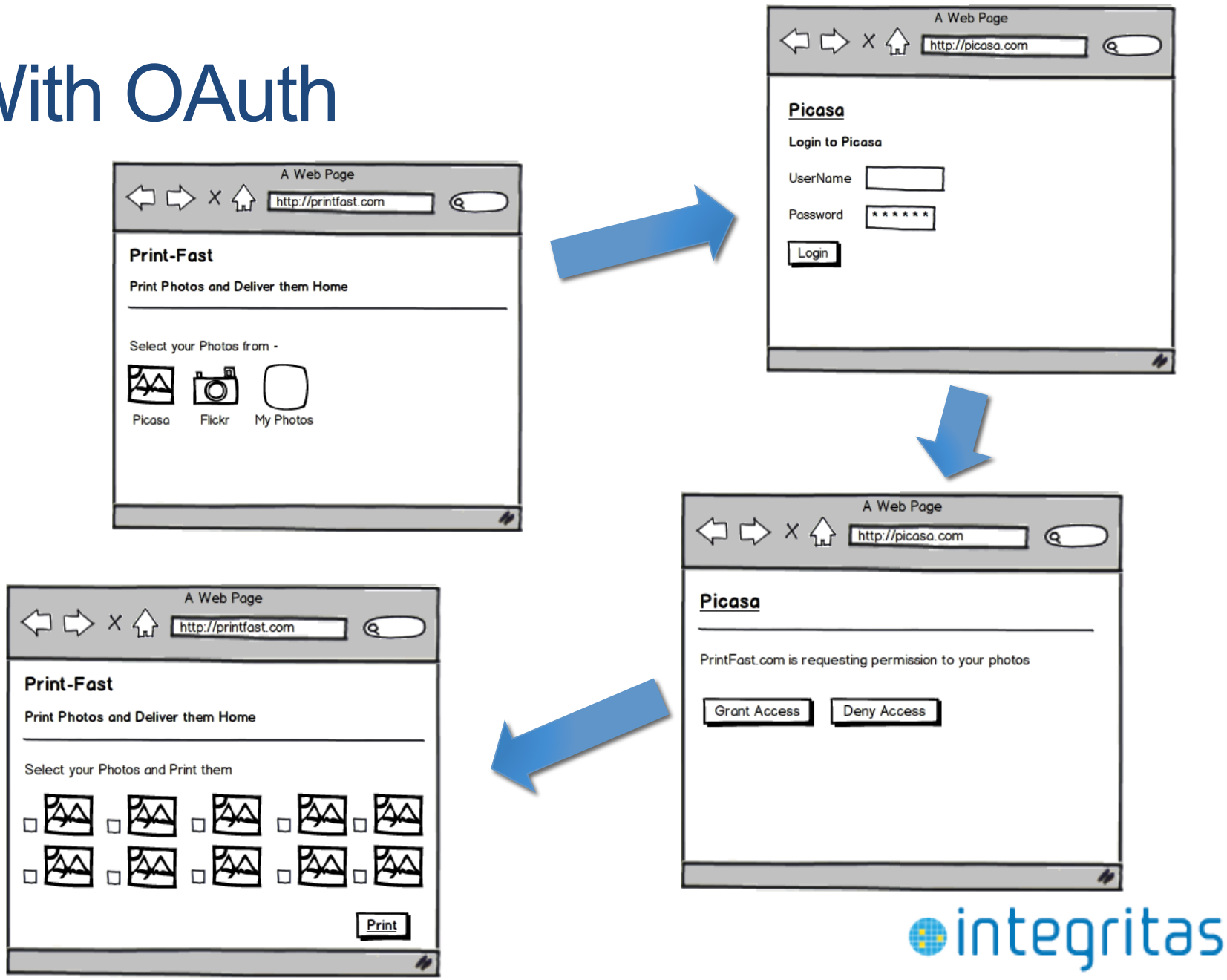
integritas

Without OAuth



integritas

With OAuth



integritas

Why OAuth

- Open standard protocol specification defined by IETF
- Enables applications to access each other's data without sharing credentials
- Avoid password issues
  - User and password authentication is fine, but what if your API needs to be used by other apps?
- Required for delegating access
  - Third party applications
  - For specified resource
  - For limited time
  - Can be selectively be revoked

integritas

Who is using OAuth

OAuth Timeline

- OAuth 1.0
  - Core specification published in Dec 2007
- OAuth 1.0a
  - Revised specification published in June 2009
  - Related to fix a security issue
- OAuth 2.0
  - Standardized since Oct-2012
  - Be more secure, simple, and standard
  - Additional RFCs are still being worked on

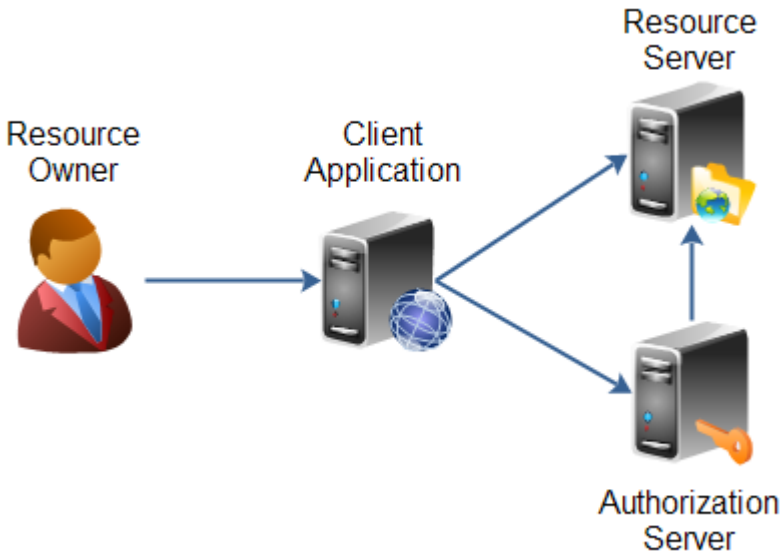
integritas

OAuth2

- No ~~username~~ or ~~passwords~~ (only tokens)
- Protocol for authorization – not authentication
- Delegated model
  - Fix the password anti-pattern
  - Trust relationship between resource, identity server and client app
- Goal was simplicity
- Relies heavily on TLS/SSL
- **Not backwards compatible**
- Easily to revoke

integritas

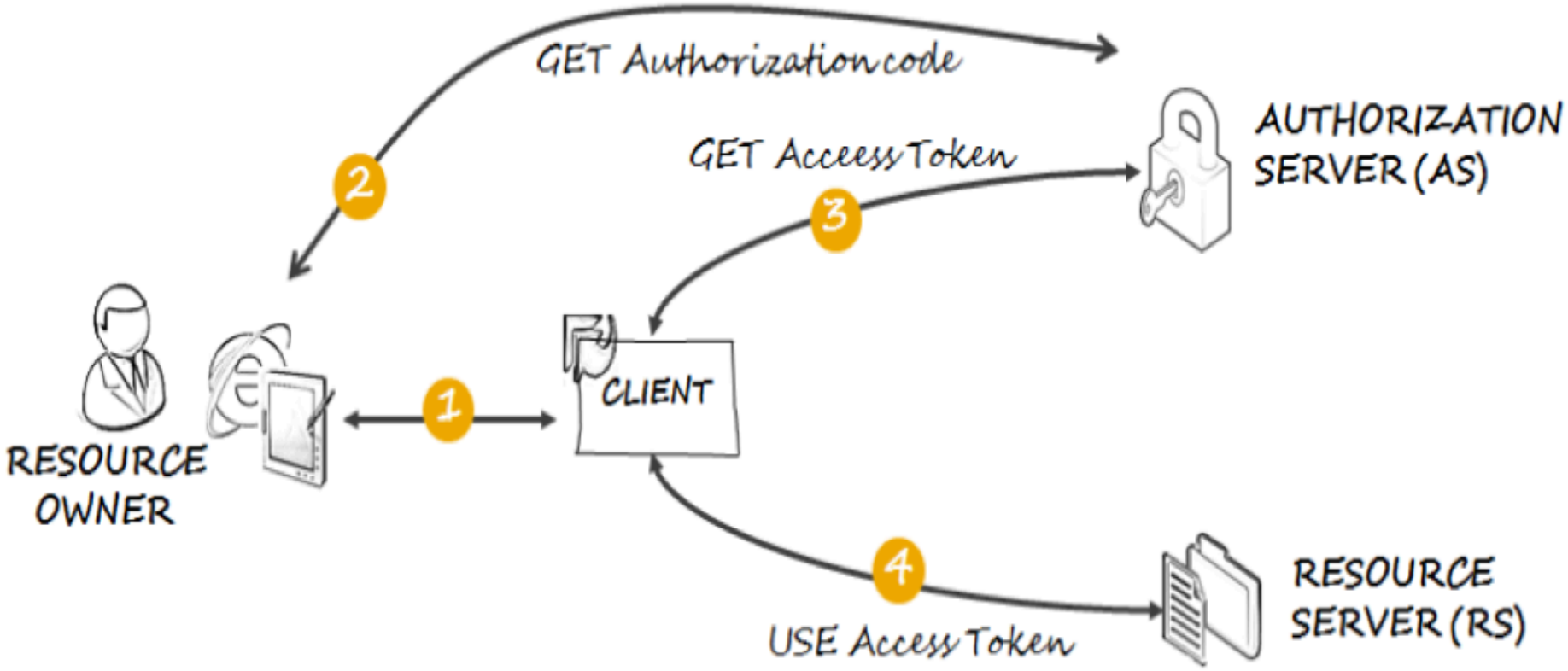
OAuth2 Roles



- Resource Owner
  - Entity capable of granting access to a protected resource
- Client Application
  - Application making protected resource requests on behalf of the resource owner
- Resource Server
  - The server hosting the protected resources
- Authorization Server
  - The server issuing access tokens to the clients

integritas

OAuth2 Basic Flow



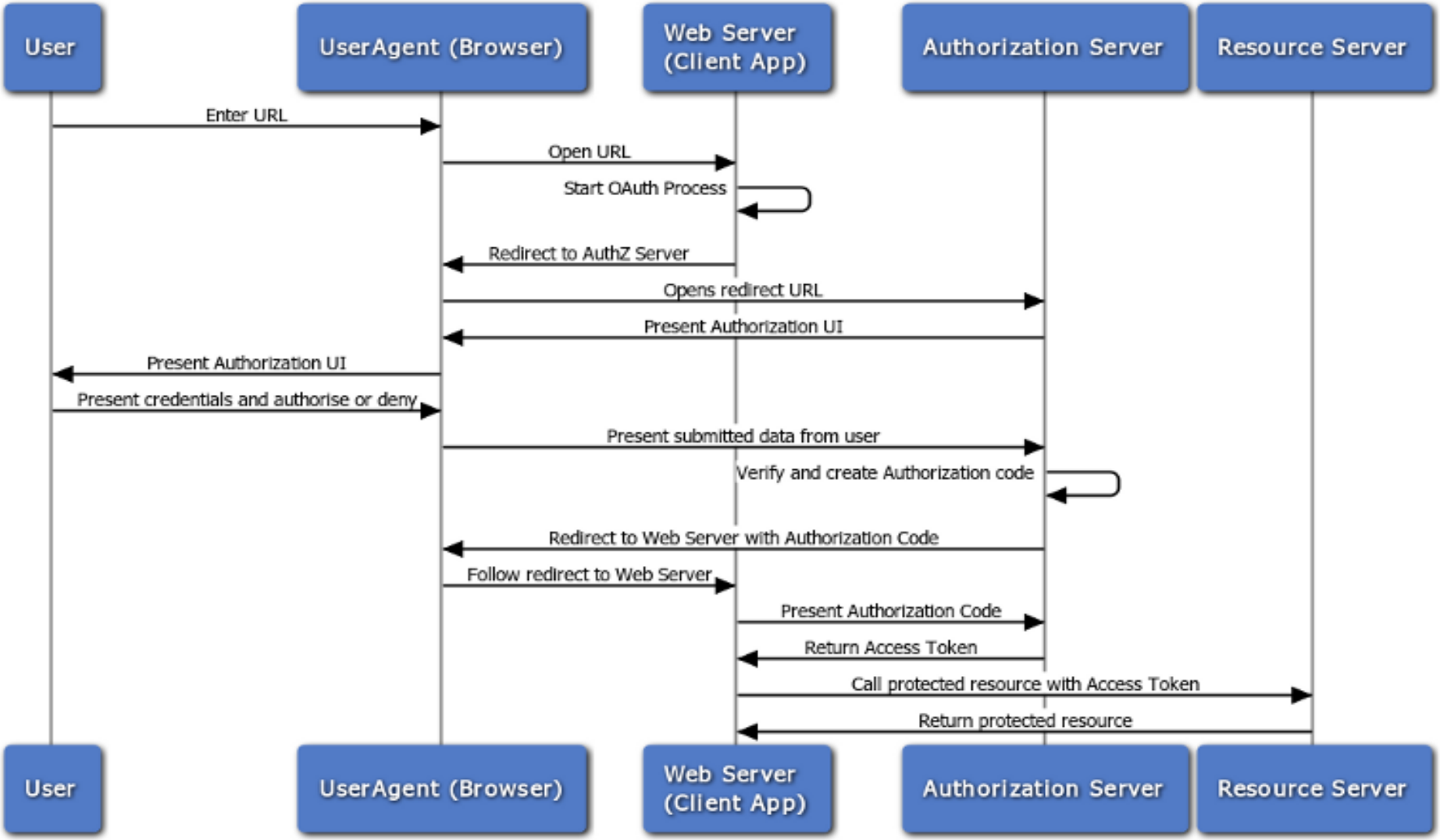
integritas

OAuth2 Grant Types

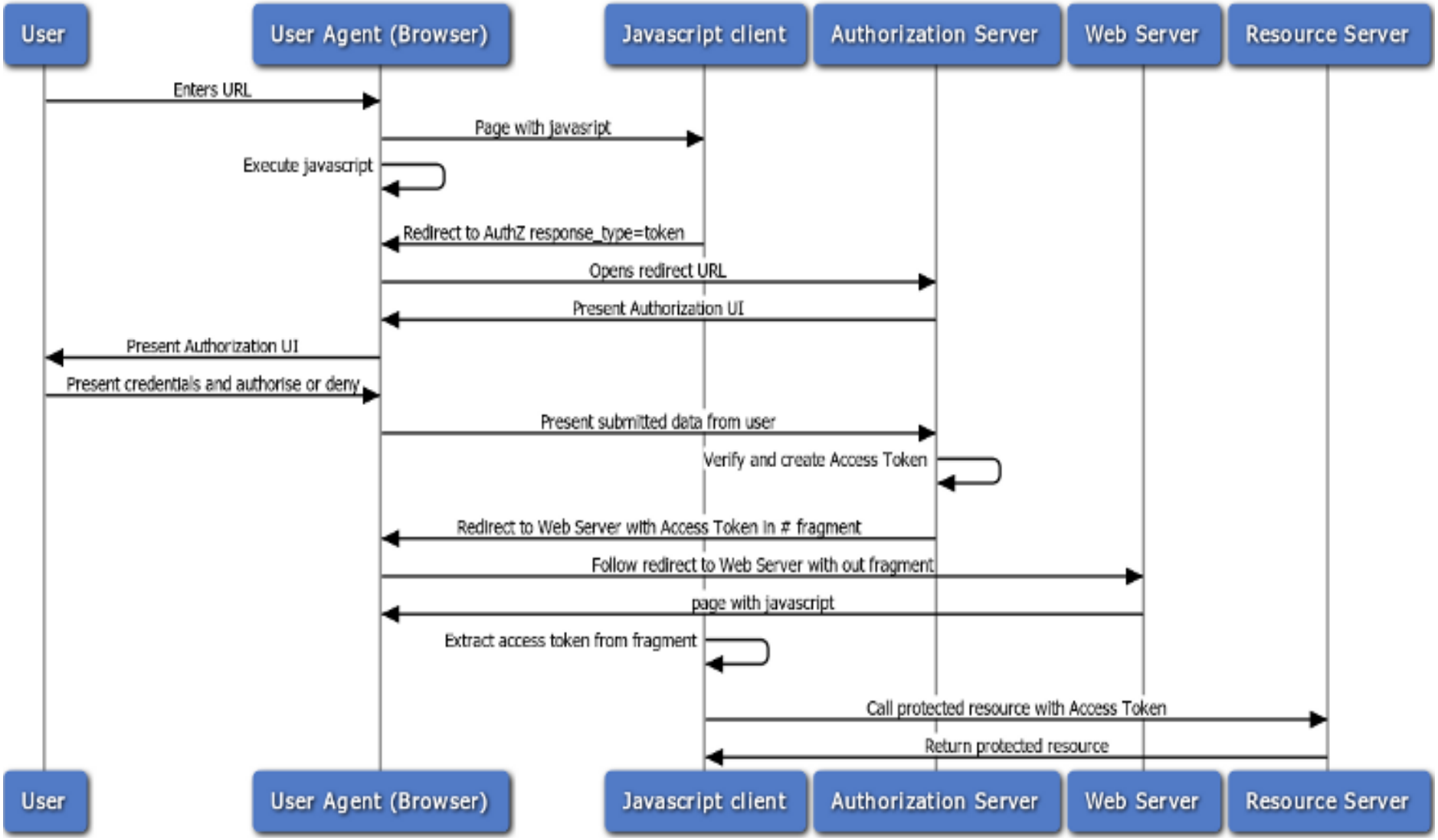
- Authorization Code **(web apps)**
  - Optimized for confidential clients
  - Uses a authorization code from the server
- Implicit **(browser-based and mobile apps)**
  - Optimized for script heavy web apps
  - User can see the access token
- Resource Owner Password Credentials **(user / password)**
  - Used in cases where the user trusts the client
  - Exposes user credentials to the client
- Client Credentials **(application)**
  - Clients gets an access token based on client credentials only

integritas

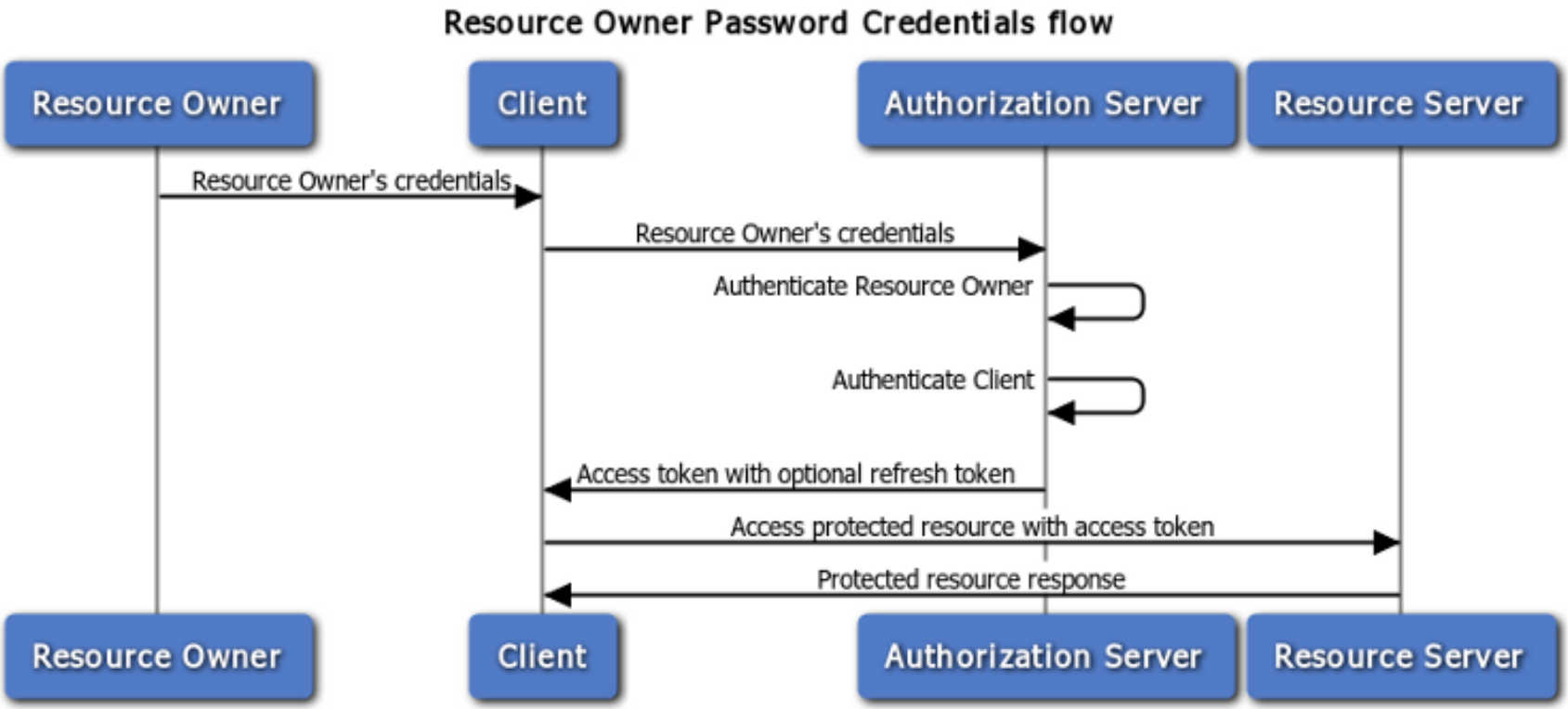
Authorization Code



Implicit

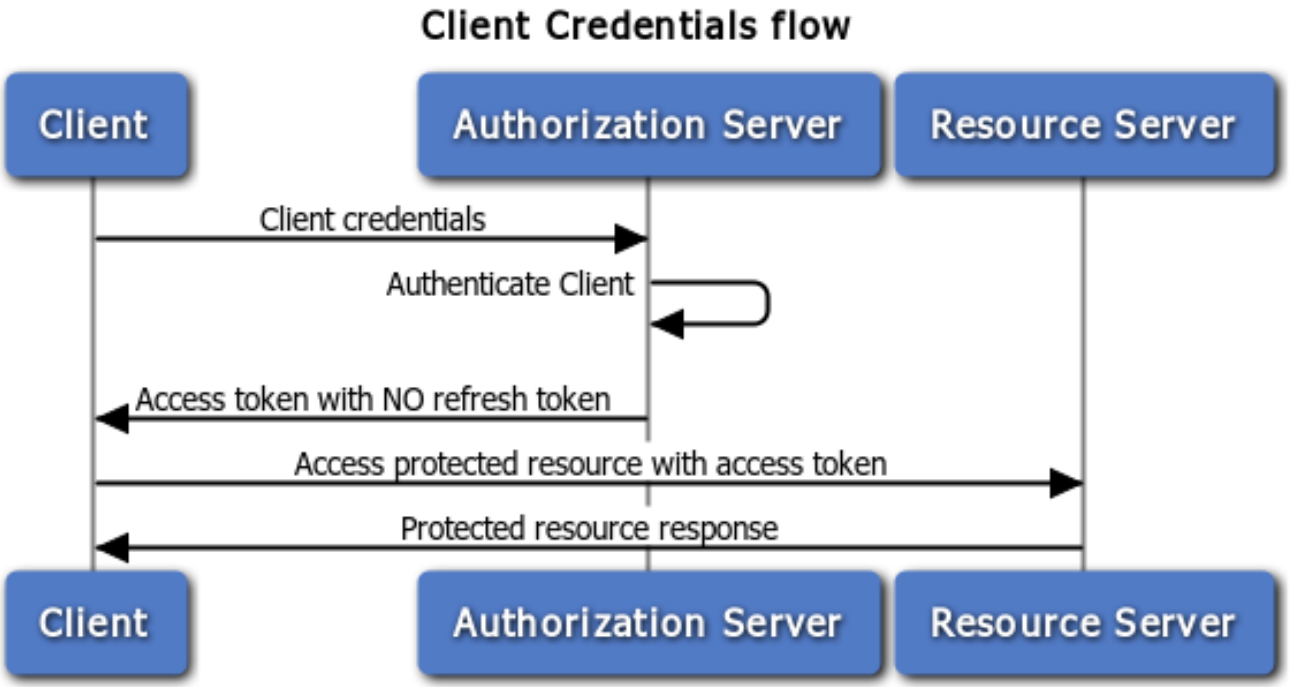

User
User Agent (Browser)
Javascript client
Authorization Server
Web Server
Resource Server
Enters URL
Page with javasript
Execute javascript
Redirect to AuthZ response_type=token
Opens redirect URL
Present Authorization UI
Present Authorization UI
Present credentials and authorise or deny
Present submitted data from user
Verify and create Access Token
Redirect to Web Server with Access Token in # fragment
Follow redirect to Web Server with out fragment
page with javascript
Extract access token from fragment
Call protected resource with Access Token
Return protected resource
User
User Agent (Browser)
Javascript client
Authorization Server
Web Server
Resource Server

Resource Owner Password Credentials



integritas

Client Credentials



integritas

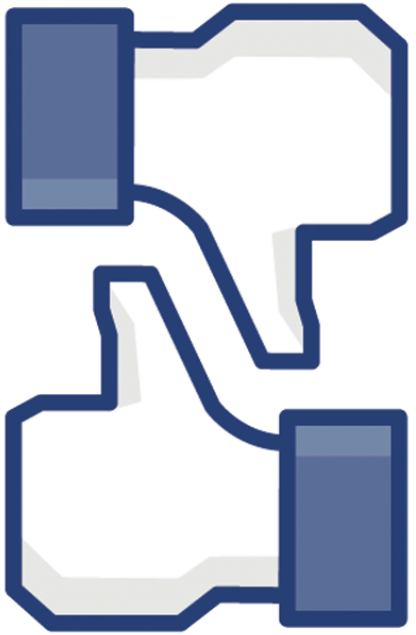
OAuth2 Tokens

- Types
  - Bearer
    - Large random token
    - Need SSL to protect it in transit
    - Server needs to store it securely hashed like a user password
  - Mac
    - Uses a nonce to prevent replay
    - Does not required SSL
    - OAuth 1.0 only supported
- Access Token
  - Short-lived token
- Refresh Token
  - Long-lived token

```
{
  "access_token":"2YotnFZFEjr1zCsicMWpAA",
  "token_type":"bearer",
  "expires_in":3600,
  "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
}
```

integritas

OAuth2 Pros & Cons



- Pros
  - Integration of third party apps to any sites
  - Access can be granted for limited scope or duration
  - No need for users to give password on third party site
- Cons
  - Writing an authorization server is somewhat complex
  - Interoperability issues
  - Bad implementations can be security issues

integritas

OAuth2 Java Implementations

- Some Java implementations available
  - Jersey
  - Apache Oltu
  - Spring Security OAuth2
  - And others: CXF, Google OAuth2 API, etc
- Not available as Java EE standard yet

integritas

Jersey

- Open source RESTful Web services framework
- The JAX-RS reference implementation
- Integrates with the Java EE standard security
  - `@RolesAllowed`
  - `@PermitAll`
  - `@DenyAll`
- Supports entity filtering features
  - `@EntityFiltering`
- **Only supports OAuth2 at client side :/**

integritas

Jersey

Java EE security integration

```
@Path("restricted-resource")
@Produces("application/json")
public class RestrictedResource {

    @GET @Path(”denyAll")
    @DenyAll
    public RestrictedEntity denyAll() { ... }

    @GET @Path("rolesAllowed")
    @RolesAllowed({"manager"})
    public RestrictedEntity rolesAllowed() { ... }
}
```

integritas

Jersey

OAuth2 client support

```
OAuth2CodeGrantFlow.Builder builder =
    OAuth2ClientSupport
        .authorizationCodeGrantFlowBuilder(
            clientId,
            "https://example.com/oauth/authorization",
            "https://example.com/oauth/token");

OAuth2CodeGrantFlow flow = builder.property(
        OAuth2CodeGrantFlow.Phase.AUTHORIZATION,
            "readOnly", "true")
        .scope("contact")
        .build();

String authorizationUri = flow.start();
...
final TokenResult result = flow.finish(code, state);
...
```

Apache Oltu

- Apache OAuth protocol implementation
- It also covers others related implementations
  - JSON Web Token (JWT)
  - JSON Web Signature (JWS)
  - OpenID Connect
- Supports the full OAuth2 features
  - Authorization Server
  - Resource Server
  - Client
- Provides predefined OAuth2 client types
  - Facebook, Foursquare, Github, Google, etc
- **Still being improved…**

integritas

Apache Oltu

Authorization endpoint

```
protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
                        throws ServletException, IOException {

   //dynamically recognize an OAuth profile and perform validation
   OAuthAuthzRequest oauthRequest = new OAuthAuthzRequest(request);

   validateRedirectionURI(oauthRequest)

   //build OAuth response
   OAuthResponse resp = OAuthASResponse
      .authorizationResponse(HttpServletResponse.SC_FOUND)
      .setCode(oauthIssuerImpl.authorizationCode())
      .location(ex.getRedirectUri())
      .buildQueryMessage();

   response.sendRedirect(resp.getLocationUri());
}
```

integritas

Apache Oltu

Token endpoint

```
protected void doPost(HttpServletRequest request,
   HttpServletResponse response)
      throws ServletException, IOException {

   OAuthIssuer oauthIssuerImpl =
       new OAuthIssuerImpl(new MD5Generator());

   OAuthTokenRequest oauthRequest =
       new OAuthTokenRequest(request);

   validateClient(oauthRequest);

   String authzCode = oauthRequest.getCode();
   String accessToken = oauthIssuerImpl.accessToken();
   String refreshToken = oauthIssuerImpl.refreshToken();

   OAuthResponse r = OAuthASResponse(...);
}
```

Apache Oltu

Protecting the resources

```
protected void doGet(HttpServletRequest request,
    HttpServletResponse response)
        throws ServletException, IOException {

    // Make the OAuth Request and validate it
    OAuthAccessResourceRequest oauthRequest = new
        OAuthAccessResourceRequest(request,
            ParameterStyle.BODY);

    // Get the access token
    String accessToken =
        oauthRequest.getAccessToken();

    //... validate access token
}
```

integritas

Apache Oltu

OAuth2 client

```
OAuthClientRequest request = OAuthClientRequest
   .tokenProvider(OAuthProviderType.FACEBOOK)
   .setGrantType(GrantType.AUTHORIZATION_CODE)
   .setClientId("your-facebook-application-client-id")
   .setClientSecret("your-facebook-application-client-secret")
   .setRedirectURI("http://www.example.com/redirect")
   .setCode(code)
   .buildQueryMessage();

//create OAuth client that uses custom http client under the hood
OAuthClient oAuthClient = new OAuthClient(new URLConnectionClient());

OAuthAccessTokenResponse oAuthResponse =
       oAuthClient.accessToken(request);

String accessToken = oAuthResponse.getAccessToken();
String expiresIn = oAuthResponse.getExpiresIn();
```

integritas

Spring Security OAuth

- Provides OAuth (1a) and OAuth2 support
- Implements 4 types of authorization grants
- Supports the OAuth2 full features
  - Authorization Server
  - Resources Server
  - Client
- Good integration with JAX-RS and Spring MVC
- Configuration using annotation support
- Integrates with the Spring ecosystem

integritas

Spring Authorization Server

- `@EnableAuthorizationServer`
  - Annotation used to configure OAuth2 authorization server
  - There is also XML configuration related `<authorization-server/>`
- `ClientDetailsServiceConfigurer`
  - Defines the client details service
  - In-memory or JDBC implementation
- `AuthorizationServerTokenServices`
  - Operations to manage OAuth2 tokens
  - Tokens in-memory, JDBC or JSON Web Token (JWT)
- `AuthorizationServerEndpointConfigurer`
  - Grant types supported by the server
  - All grant types are supported except password types

integritas

Spring Resource Server

- Can be the same as Authorization Server
  - Or deployed in a separate application
- Provides a authentication filter for web protection
- `@EnableResourceServer`
  - Annotation used to configure OAuth2 resource server
  - There is also XML configuration related `<resource-server/>`
- Supports expression-based access control
  - *`#oauth2.clientHasRole`*
  - *`#oauth2.clientHasAnyRole`*
  - *`#oauth2.denyClient`*

integritas

Spring OAuth2 Client

- Creates a filter to store the current request and context
- Manages the redirection to and from the OAuth authentication URI
- `@EnableOAuth2Client`
  - Annotation used to configure OAuth2 client
  - There is also XML configuration related `<client/>`
- `OAuth2RestTemplate`
  - Wrapper client object to access the resources

integritas

Demo

- OAuth2 Use Case
  - Conference application sharing resources with different clients
  - http://github.com/rcandidosilva/rest-oauth2-sample

integritas

Questions

integritas

References

- http://oauth.net/2/
- http://tools.ietf.org/html/rfc6749
- http://projects.spring.io/spring-security-oauth/
- https://github.com/spring-projects/spring-security-oauth
- http://cxf.apache.org/docs/jax-rs-oauth2.html
- https://jersey.java.net/documentation/latest/security.html#d0e10940
- https://oltu.apache.org

integritas

Thank you!

@rcandidosilva

rodrigocandido.me

integritas