OAuth 2.0

Amit Harsola

Wipro Technologies

Objective

The purpose of the presentation is to explain the need for an authorization standard and how does OAuth addresses requirements

Agenda

1. Social Web Integration
2. What is OAuth and Why?
3. History
4. Insight
5. Client Libraries
6. Provider Implementation
7. Use Cases
8. Challenges
9. References

Social Web Integration

The evolving world of Information Technology is changing the way people socialize, share information and access applications.

Applications over the web allows organizations ways to leverage their users to communicate or provide value added services using their offerings

This necessitates need for Applications to exchange protected information

Social Web Integration is about how web based applications integrate and share protected data which benefits both consumer and provider in some way

Such integration typically involves User Authorization before any sharing of data

Social Web Integration

In a typical web authentication model, user owning the resource shares credentials with other applications to provide them access to their protected data

The above approach presents significant challenges for the resource owners

1. It requires sharing of passwords in clear text which third party applications stores for any future use

2. Any compromise of third party application results in compromise of user's password and its data

3. Resource access revocation is only possible by user through password change

4. Third party applications have full access to user's data i.e. no data level or data scope access

Social Web Integration

Several proprietary authorization protocols have emerged to address the integration problem i.e. AuthSub (Google), OpenAuth (AOL), BBAuth (Yahoo), Amazon Web Services API, etc

These proprietary protocols burdens the API consumers with implementations, all serving the same purpose i.e. web integration for exchange of protected data.

Internet Engineering Task Force (IETF) have proposed an OAuth protocol which integrates the commonalities and adopts the best practices of the proprietary Web authorization protocols into a single open standard

What is OAuth?

OAuth stands for “Open Authorization”

An open standard protocol that provides simple and secure authorization for different types of applications

A simple and safe method for consumers to interact with protected data

Allows providers to give access to users without any exchange of credentials
Designed for use only with HTTP protocol. It does not support any other protocol

Why OAuth?

OAuth is created by studying each of the proprietary protocols and extracting the best practices to support new implementations as well as a smooth transition for existing services to support OAuth

It is flexible, compatible and designed to work with mobile devices and desktop applications

Provides a method for users to grant third-party access to their resources without sharing their credentials.

Provides a way to grant limited access in terms of scope and duration

Has support from big players in the industry

History

OAuth Core specification 1.0 was published in Dec 2007

To fix a security issue, a revised specification was published in June 2009

OAuth 2.0 is a completely new protocol which is not backwards compatible with previous versions. However, it retains the overall architecture and approach established by the previous versions.

Latest OAuth 2.0 daft is v30. Expected to get finalized by end of 2012

Various Services on the Web already support OAuth 2.0

- Facebook's Graph API
- Foursquare
- Github
- Google
- Salesforce
- Windows Live

Insight – 1.0 & 1.0a

OAuth 1.0 protocol mandates cryptographic signatures to be sent with each call to verify the identity and authorization of client

In OAuth 1.0, the API client and server share a token. Client generates a signature on every API call by encrypting information using the token. The server generates the same signature and grant access only if both signatures match

Does not specify authorization for non-web applications

OAuth 1.0a is a patch release that fixes the attack against the OAuth Request Token approval flow

Insight – Introduction

OAuth eliminates the need for cryptographic signatures in 2.0 through use of SSL. It mandates to use TLS with the protocol

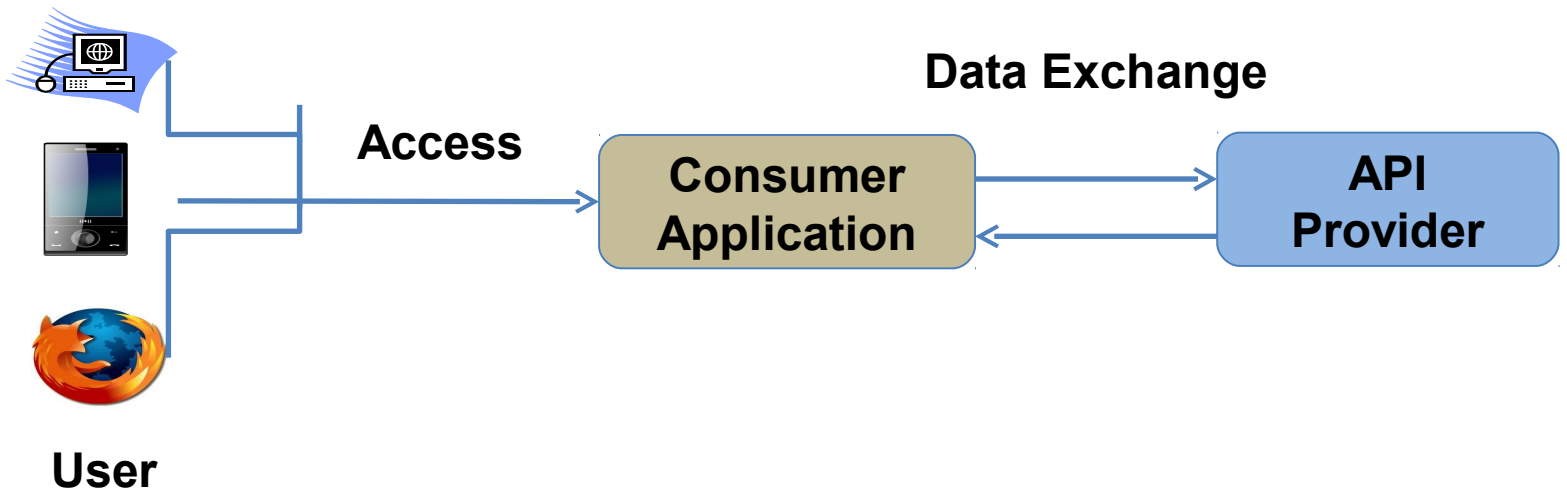
SSL is required for all the interactions required to generate the token, thus reducing complexity

Supports authorizations in mobile, browser and JavaScript applications

Clearly separates the roles of different actors and part of protocol they are responsible for

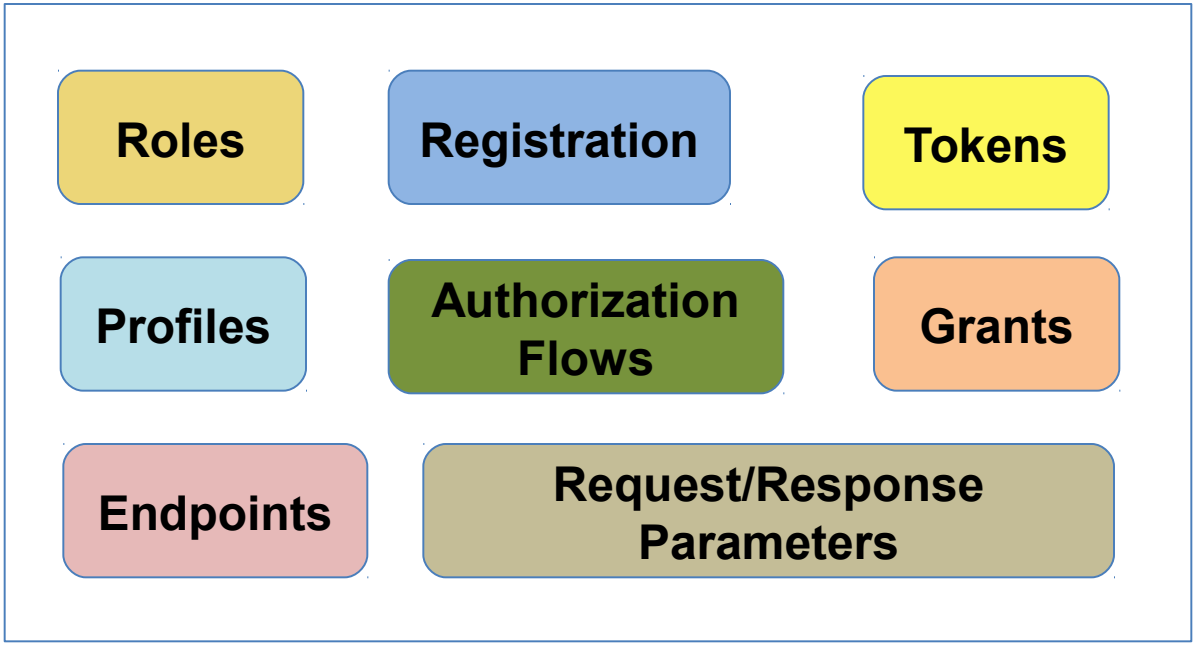
Insight – Introduction

- User accesses consumer application which first gets user authenticated through API Provider application for any exchange of information
- Once authenticated, consumer application and Provider exchange tokens for authorization
- After successful authorization, consumer application gets access to users resources on Provider application



Insight – Building Blocks

Building Blocks of OAuth



Insight – Roles

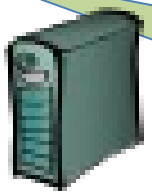
Client

An application that makes request for protected resources on behalf of Resource Owner with its authorization

Authorization Grant Request

Access Token Request

Protected Resource Access Request



Resource Owner

An entity referring to a system or a person (end-user) owning the resources

Authorization Server

Server responsible for issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization

Resource Server

An entity referring to a system or a person (end-user) owning the resources

The authorization server may be the same server as the resource server or a separate entity

Insight – Client Registration

OAuth requires Clients to register with the authorization server before making any API requests

Protocol leaves the registration mechanism open to API Provider

As part of registration, OAuth expects

- Client Type
- Redirections URI(s)
- Any other information required by API (data specific to the provider)

Example: Visit

- http://code.google.com/apis/console
- https://developers.facebook.com/apps

Successful registration results in issuance of credentials (Client ID & Client Secret) to client

Insight – Client Profiles

OAuth supports three client types

- **Server Side Web Application**
  - ✓ Client is a server hosting the client application
  - ✓ When accessed by Resource Owner, Client Application makes API calls using appropriate programming language.
  - ✓ Secret or access tokens are not exposed to the user
  - ✓ Client requires repetitive access to protected resources

- **User Agent based Application**
  - ✓ JavaScript, Flash plug-in, browser extension or any downloaded executing in browser could be OAuth client
  - ✓ Protocol data and credentials are easily accessible (and often visible) to the resource owner
  - ✓ There is no concern on token leakage to entrusted users or applications

- **Native Application**
  - ✓ OAuth client is installed and executed on the resource owner's device
  - ✓ Protocol data and credentials are accessible to resource owner
  - ✓ Possible limitations on use of web browser and its usage
  - ✓ Official applications released by API provider
  - ✓ Migration of existing clients using direct authentication schemes such as HTTP Basic or Digest authentication to OAuth by converting the stored credentials to an access token

Insight – Tokens

OAuth protocol supports bearer tokens. Once authorized, using bearer token client can request to access resource

What is Bearer Token?

A security token is something that any party in possession of the token (a "bearer") can use the token in any way that any other party in possession of it can which requiring bearer to present proof-of-possession

Protocol defines two types of tokens used in the process

- **Access Token**

  These are credentials required to access the protected resources. Represents the grant's scope, duration, and other attributes granted by the authorization grant

- **Refresh Token**

  Refresh tokens are credentials used to obtain access tokens. Refresh tokens are issued to the client by the authorization server and are used to obtain a new access token when the current access token becomes invalid or expires etc

Insight – Grant Types

Grant is a credential which represents owner's authorization and used by client to access owner's protected resources. Grant types defined by the specification

Authorization Code
Suited for Server Side Clients

Implicit
Optimized for User agent (browser) based clients

Resource Owner Password Credentials
Resource Owner's credentials (username/password) could be used to obtain access token. Could be used by trusted clients like mobile application

Client Credentials
Typically used by application to obtain access token for resources owned by the client or if there is some offline authorization agreed with an authorization server

Insight – End Points

Endpoints are the URIs which Client uses to make requests. Protocol supports following authorization server endpoints

Authorization Endpoint

Endpoint used by client to obtain resource authorization from resource owner

Token Endpoint

Used by client to retrieve access or refresh token

Redirection Endpoint

Authorization server uses the endpoint to redirect after authorization process

Insight – Authorization Flows

OAuth defines four authorization flows

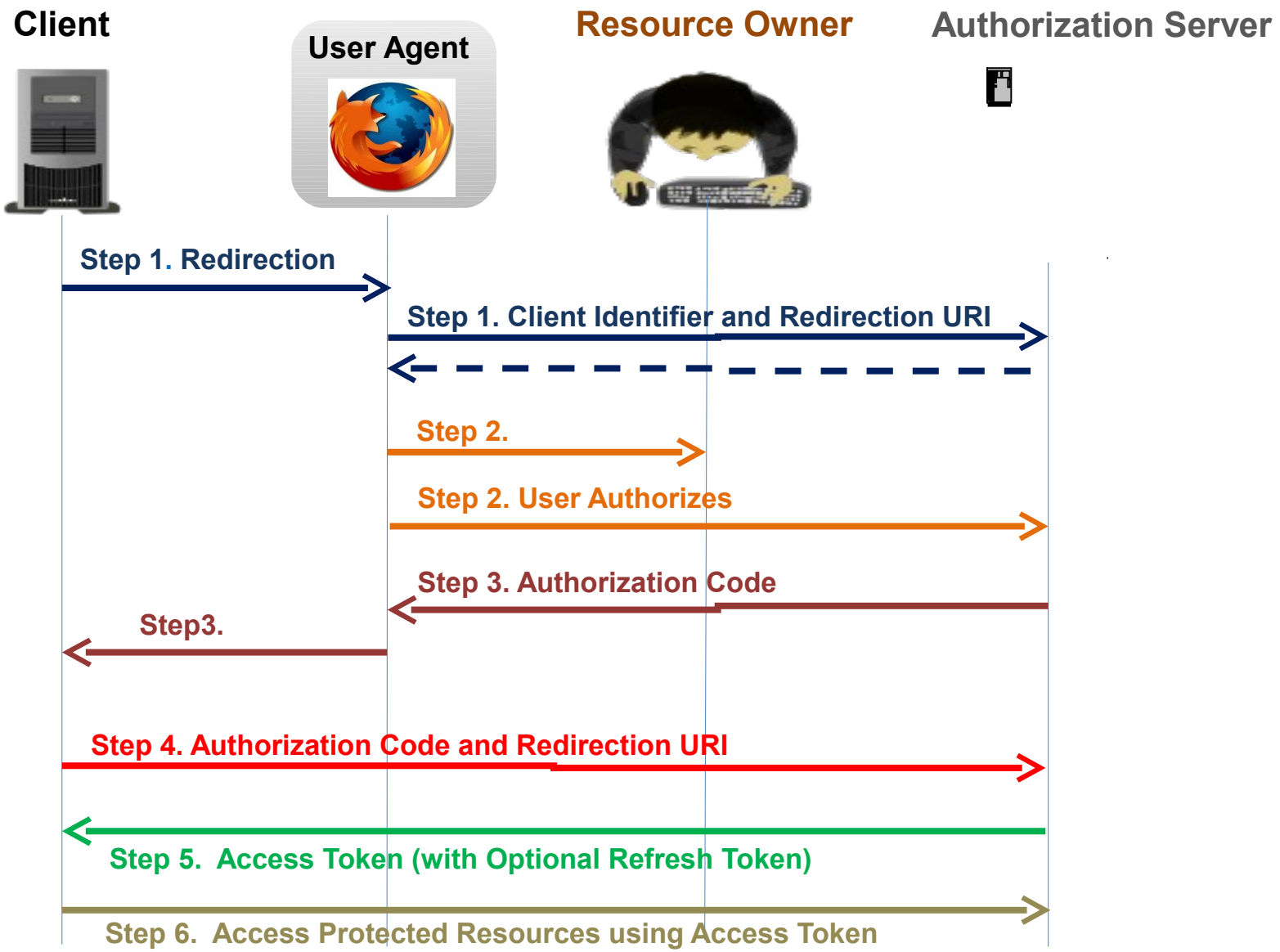
- Server Side Web Application Flow
- User Agent (Browser) Application Flow
- Resource Owner Password Flow
- Client Credentials Flow

Insight – Server Side Web Application Flow

Key Features

1. Two step protocol i.e. request for authorization code and access token request
2. Support for Refresh Tokens
3. Authorization code is used to obtain both access and refresh token

Insight – Server Side Application Flow



Insight – Server Side Web Application Flow

Authorization Flow Steps

1. The client initiates the flow by directing the resource owner's user-agent (browser) to the authorization endpoint. The client includes its client credentials , scope, state, and a redirection URI to which the authorization server will send the user-agent back once access is granted (or denied)

2. The authorization server authenticates the resource owner and determines whether the resource owner grants or denies the client's access request

3. The authorization server redirects the user-agent back to the client using the redirection URI provided earlier. The redirection URI includes an authorization code and state provided by the client earlier

4. The client requests an access token from the authorization server's token endpoint by including the authorization code. The client includes the redirection URI used to obtain the authorization code for verification

5. The authorization server authenticates the client, validates the authorization code, and ensures the redirection URI received matches the URI used to redirect the client in step (3). If valid, the authorization server responds back with an access token and optionally, a refresh token

6. Using access token, client access the resource owner’s protected resources

Insight – Server Side Web Application Flow

Authorization Request – Step 1 & 2

As part of the request URI (Authorization Endpoint), Client is required to pass following parameters

1. **response_type (required)**: Value set to '**code'**
2. **client_id (required) :** Value set to the client id received during registration
3. **redirect_uri (optional):** Not required, if the URI was provided during client registration
4. **scope (optional):** Represents application data scope. Usually contains a space-delimited strings. The valid values are provided by API Provider
5. **state (optional):** Optional but recommended to be used to prevent cross site request forgery. Contains some random string to maintain state between request and callback

Insight – Server Side Web Application Flow

Authorization Response – Step 3

The response generated by the authorization server contains the authorization code and the state present in the request URI

1. **code (required)**: The authorization code generated by the authorization server

2. **state (optional):** Required only if state parameter was part of the authorization request URI

Insight – Server Side Web Application Flow

Error Response – Step 3

If the authorization request fails due to any reason, the authorization server informs the client by adding parameters to the redirection URI

1. **error (required):** Value set to one of the following
   - invalid_request
   - unauthorized_client
   - access_denied
   - unsupported_response_type
   - invalid_scope
   - server_error
   - temporarily_unavailable

2. **error_description (optional):** Additional information about the error

3. **error_uri (optional):** A web page with information about the error

4. **state (optional):** Required only if state parameter was part of the authorization request URI

Insight – Server Side Web Application Flow

Access Token Request – Step 4

After successful authorization, client makes request for access token passing

1. **grant_type (required) :** Value set to **'authorization_code'**
2. **code (required) :** Value set to authorization code received during request authorization
3. **redirect_uri (required) :** Required, if the "redirect_uri" parameter was included in the authorization request and values are identical
4. **client_id (required) :** Value set to the client id received during registration

Insight – Server Side Web Application Flow

Access Token Response – Step 5

For a valid and authorized access token request, authorization server issues an access token and optional refresh token in JSON encoded format

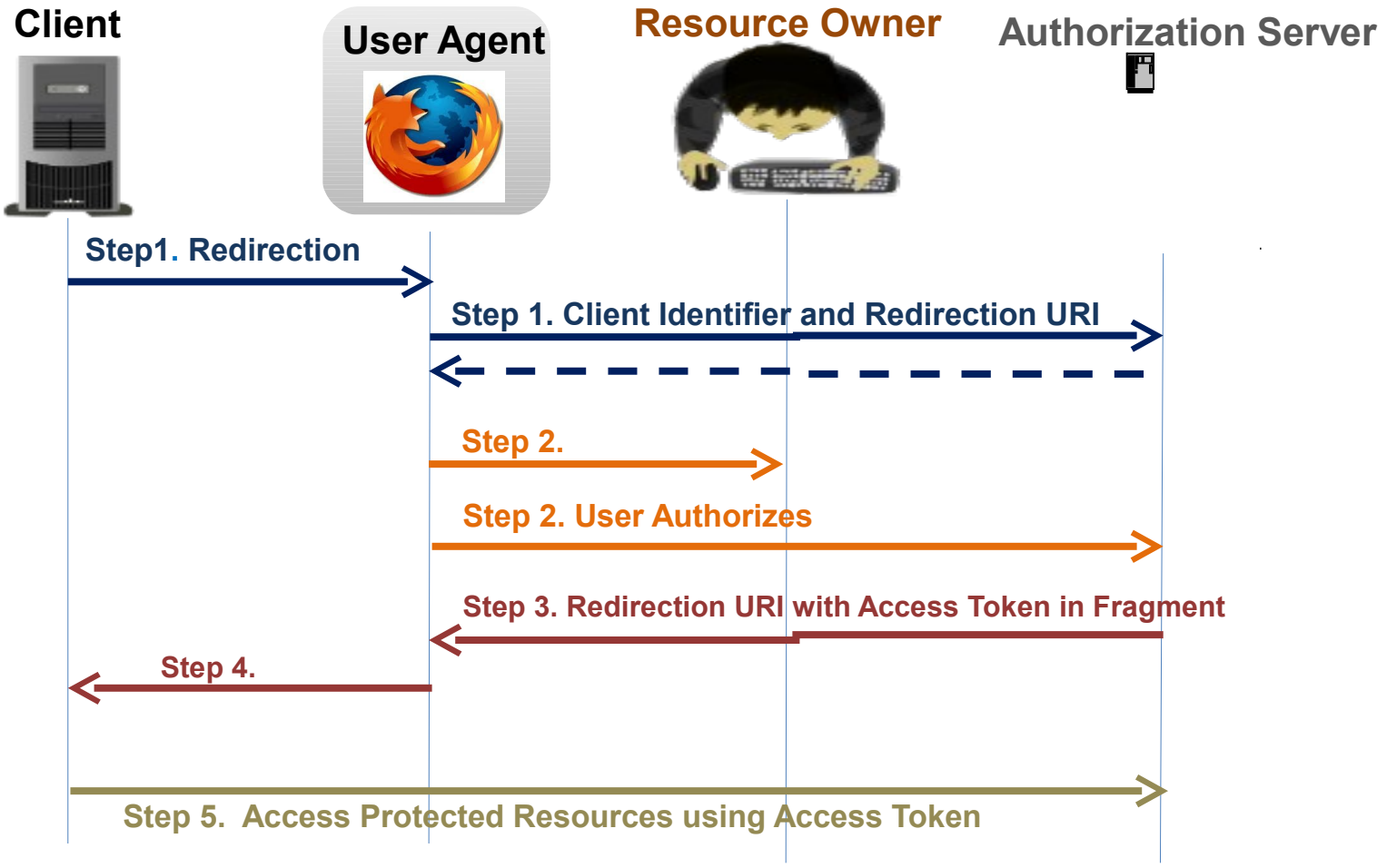
```
{
  "access_token":"2YotnFZFEjr1zCsicMWpAA",
  "token_type":“bearer",
  "expires_in":3600,
  "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
}
```

Insight – User Agent Application Flow

Key Features

1. No separate authorization and access token request
2. Client receives the access token in response to authorization request
3. Does not support Refresh Tokens

Insight – Server Side Application Flow - I



Insight – User Agent Application Flow - I

Authorization Flow Steps

1. The client initiates the flow by directing the resource owner's user-agent (browser) to the authorization endpoint. The client includes its client credentials , scope, state, and a redirection URI to which the authorization server will send the user-agent back once access is granted (or denied)

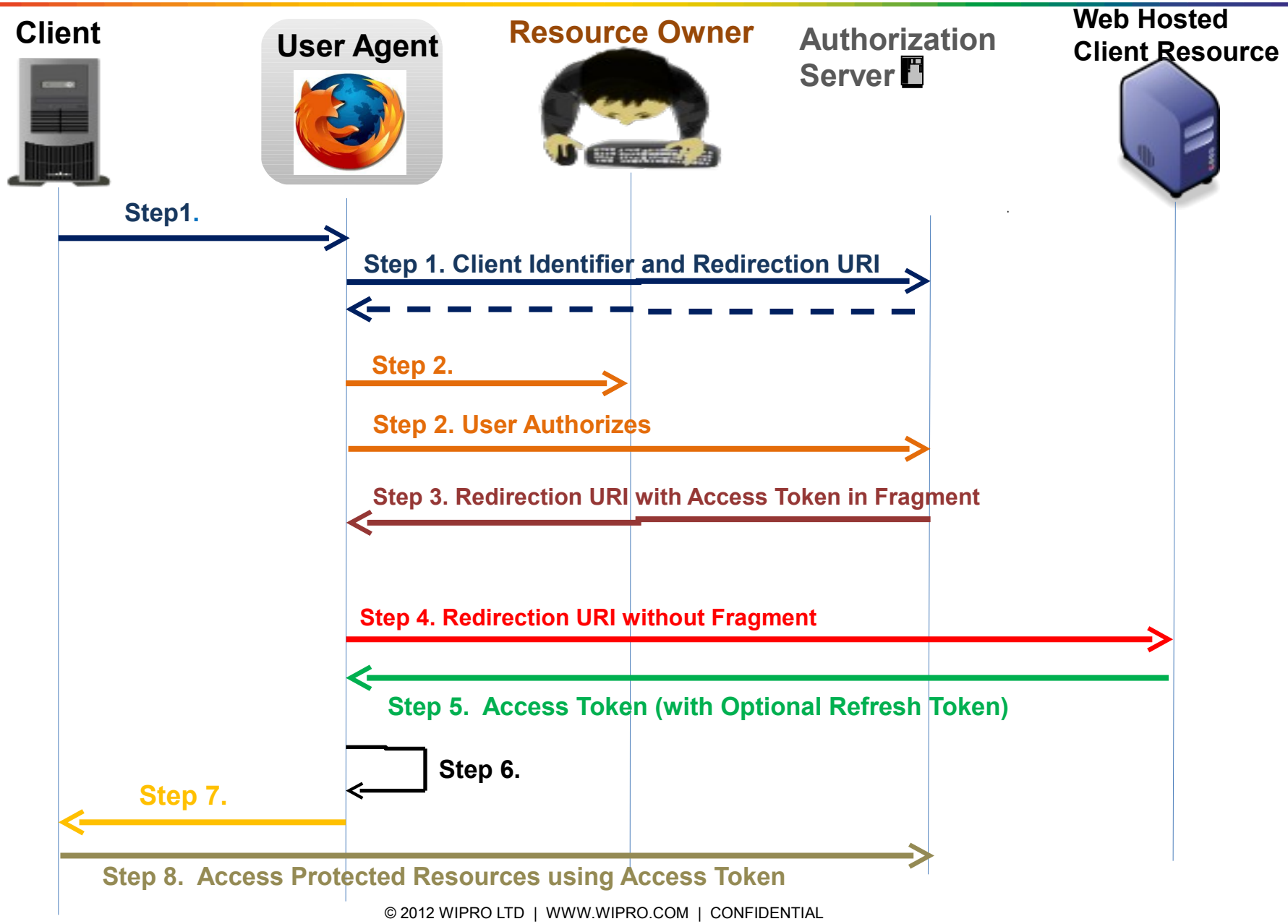
2. The authorization server authenticates the resource owner and establishes whether the resource owner grants or denies the client's access request

3. Assuming the resource owner grants access, the authorization server redirects the user-agent back to the client using the redirection URI provided earlier. The redirection URI includes the access token in the URI fragment

4. The user-agent extracts the access token and passes it to the client

5. Using access token, client access the resource owner’s protected resources

Insight – Server Side Application Flow - II



Insight – User Agent Application Flow - II

Authorization Flow Steps

1. The client initiates the flow by directing the resource owner's user-agent (browser) to the authorization endpoint. The client includes its client credentials , scope, state, and a redirection URI to which the authorization server will send the user-agent back once access is granted (or denied)

2. The authorization server authenticates the resource owner and establishes whether the resource owner grants or denies the client's access request

3. Assuming the resource owner grants access, the authorization server redirects the user-agent back to the client using the redirection URI provided earlier. The redirection URI includes the access token in the URI fragment

4. The user-agent follows the redirection instructions by making a request to the web-hosted client resource. The user-agent retains the fragment information locally

5. The web-hosted client resource returns a web page (typically an HTML document with an embedded script) capable of accessing the full redirection URI including the fragment retained by the user-agent, and extracting the access token (and other parameters) contained in the fragment

6. The user-agent executes the script provided by the web-hosted client resource locally, which extracts the access token and passes it to the client

7. Using access token, client access the resource owner’s protected resources

Insight – User Agent Application Flow

Authorization Request – Step 1 & 2

As part of the request URI (Authorization Endpoint URI), Client is required to pass same parameters as Server Side Web Application Flow

Insight – User Agent Application Flow

Access Token Response – Step 3

The authorization server issues an access token and delivers it to the client by adding the following parameters to the fragment component of the redirection URI

1. **access_token (Required):** The access token issued by authorization server
2. **token_type (Required):** Value set to bearer
3. **expires_in (Optional):** Recommended to define the lifetime in seconds. If omitted, the authorization server should provide the expiration time via other means or document the default value
4. **scope (Optional):** Optional, if identical to the scope requested by the client, otherwise Required
5. **state (Optional):** Required only if state parameter was part of the authorization request URI

Insight – User Agent Application Flow

Parsing Access Token – Step 4, 5, 6 & 7 or 4

The access token received as part of the URL in the Access Token Response is extracted by the User Agent and send to the client

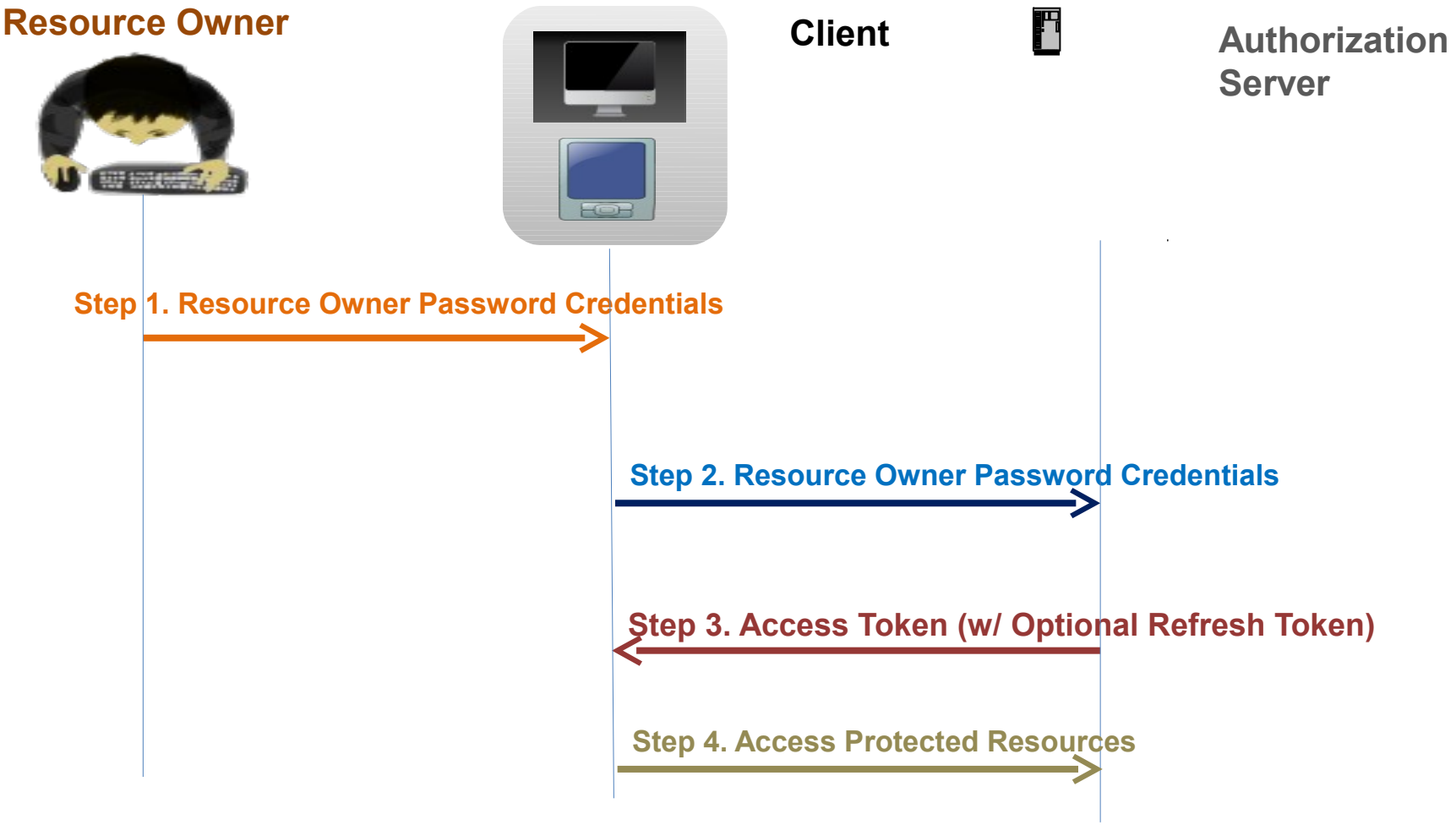
Insight - Resource Owner Credentials Grant Flow

Key Features

1. There is exchange of user's credential for tokens

2. Trusted Relationship between resource owner and the client

Insight - Resource Owner Credentials Grant Flow



Insight - Resource Owner Credentials Grant Flow

Authorization Flow Steps

1. The resource owner provides the client with its username and password.

2. The client requests an access token from the authorization server's token endpoint by including the credentials received from the resource owner. When making the request, the client authenticates with the authorization server.

3. The authorization server authenticates the client and validates the resource owner credentials, and if valid issues an access token

4. Using access token, client access the resource owner’s protected resources

Insight - Resource Owner Credentials Grant Flow

Access Token Request – Step 1 & 2

Client makes request for access token passing following parameters

1. **grant_type (required)**: Value set to "**password**".
2. **username (required):** Set to the resource owner username.
3. **password (required)**: Value set to the resource owner password.
4. **scope (optional)**: Represents application data scope. Usually contains a space-delimited strings. The valid values are provided by API Provider

Insight - Resource Owner Credentials Grant Flow

Access Token Response

For a valid and authorized access token request, authorization server issues an access token and optional refresh token in JSON encoded format

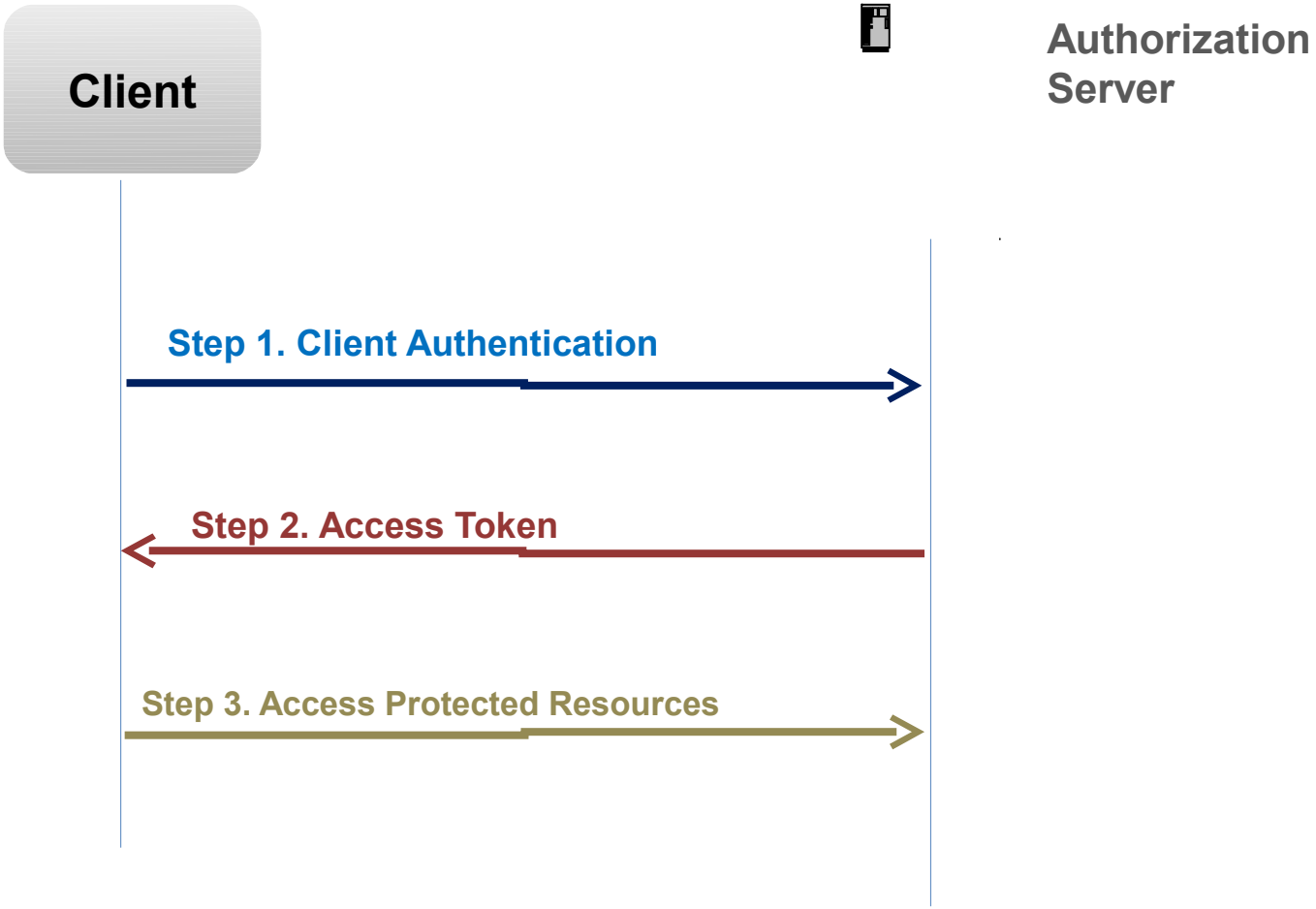

```
{
  "access_token":"2YotnFZFEjr1zCsicMWpAA",
  "token_type":“bearer",
  "expires_in":3600,
  "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
}
```

Insight – Client Credentials Flow

Key Features

1. No support for Refresh Token
2. No authorization grant

Insight - Client Credentials Flow



Insight - Client Credentials Flow

Authorization Flow Steps

1. The client authenticates with the authorization server and requests an access token from the token endpoint.

2. The authorization server authenticates the client, and if valid issues an access token

3. Using access token, client access the resource owner's protected resources

Insight - Client Credentials Flow

Authorization Request and Response

Since the client authentication is used as the authorization grant, no additional authorization request is needed

Insight - Client Credentials Flow

Access Token Request – Step 1

Client makes request for access token to the token endpoint passing

1. **grant_type (required):** Value set to "**client_credentials**".

2. **scope (optional):** Represents application data scope. Usually contains a space-delimited strings. The valid values are provided by API Provider

Insight - Client Credentials Flow

Access Token Response

For a valid and authorized access token request, authorization server issues an access token in JSON encoded format


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

Insight - Client Credentials Flow

When to use Client Credentials Flow

1. Client itself is the resource owner

2. Access to protected resources has been granted without use of OAuth protocol

Insight – Refresh Token

Tokens can have expiry. If authorization server has issued refresh token, clients can request to refresh the access token by passing

1. **grant_type (required) :** Set to "**refresh_token**“
2. **refresh_token (required):** Value set to the refresh token issued earlier
3. **scope (optional):** Identical to the scope sent in earlier request

Insight – SAML Assertion

SAML has become a de-facto standard for exchanging authentication and authorization data between security domains. The core specification does not mention the use of SAML assertion for authorization.

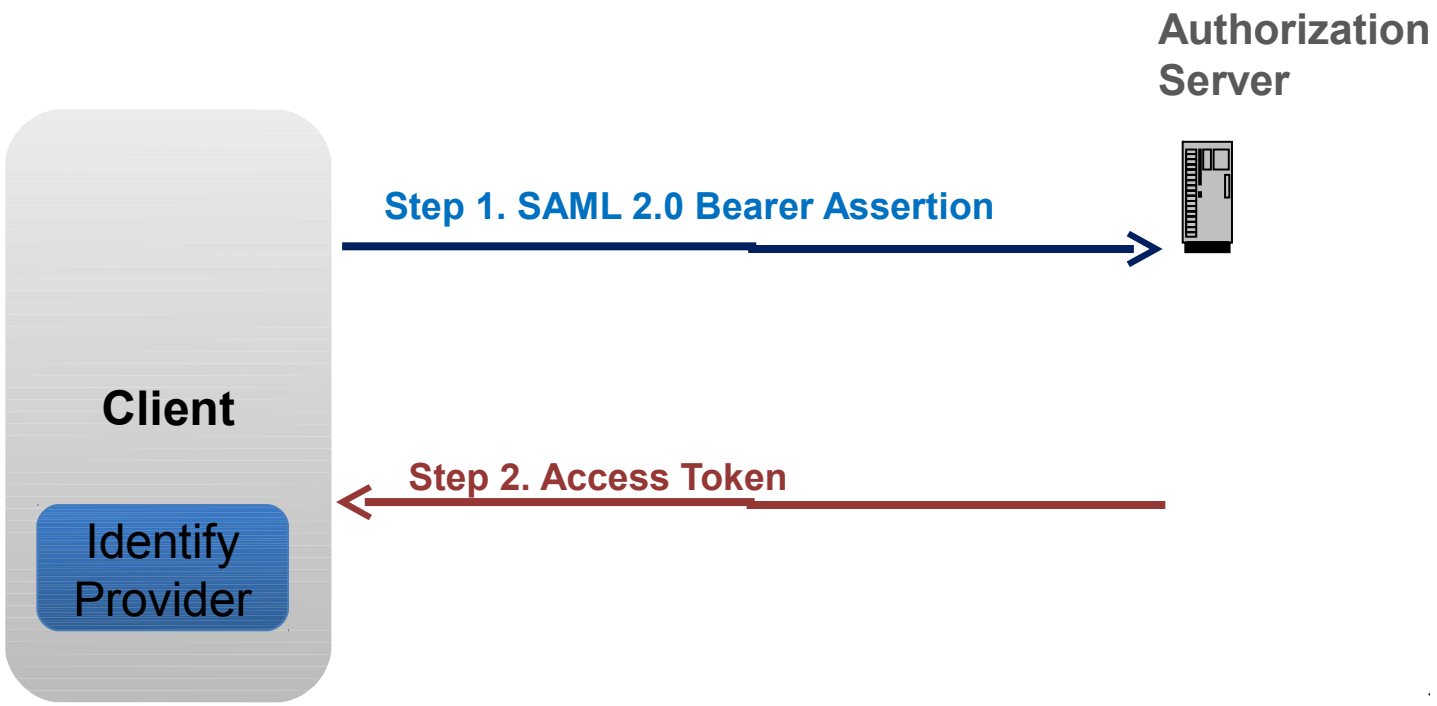
However, there is an extension to the core protocol specification that defines the use of SAML 2.0 Bearer Assertion grant type to request access token as well as client authentication

The OAuth 2.0 Assertion Profile is an abstract extension to OAuth 2.0 that provides a framework for the use of Assertions as client credentials and/or authorization grants with OAuth 2.0.

The processing of assertion is similar to the web SSO

Insight – SAML Assertion

Using SAML Assertion as an Authorization Grant



Insight - SAML Assertion

Authorization Flow Steps

1. After fetching SAML assertion from its Identity Provider, the client requests an access token with the authorization server (It is assumed there is existing trust relationship

2. The authorization server authenticates the client, and if valid issues an access token

Insight - SAML Assertion

Access Token Request – Step 1

To use a SAML Bearer Assertion as an authorization grant, client passes the following parameters

1. **grant_type (required):** Value set to "**urn:ietf:params:oauth:grant-type:saml2-bearer**"
2. **assertion (required):** Value set to single SAML 2.0 Assertion XML data encoded using base64
3. **scope (optional):** Represents application data scope. Usually contains a space-delimited strings. The valid values are provided by API Provider

Insight - SAML Assertion

Access Token Response – Step 2

To use a SAML Bearer Assertion as an authorization grant, client passes the following parameters

1. **grant_type (required):** Value set to "**urn:ietf:params:oauth:grant-type:saml2-bearer**"
2. **assertion (required):** Value set to single SAML 2.0 Assertion XML data encoded using base64
3. **scope (optional):** Represents application data scope. Usually contains a space-delimited strings. The valid values are provided by API Provider

Insight – SAML Assertion

Access Token Response

For a valid and authorized access token request, authorization server issues an access token in JSON encoded format


```
{
  "access_token":"2YotnFZFEjr1zCsicMWpAA",
  "token_type":“urn:ietf:params:oauth:grant-type:saml2-bearer ",
  "expires_in":3600,
}
```

Insight – SAML Assertion

SAML Assertion can also be used as a client authentication mechanism. The use of an Assertion for client authentication is orthogonal and separable from using an Assertion as an authorization grant and the two can be used either in combination or in isolation. The process by which the client obtains the SAML Assertion, prior to exchanging it with the authorization server or using it for client authentication, is defined out of scope by specification

Libraries

OAuth flows are simple enough to be used with HTTP APIs by any API consumer. But client libraries are also available which makes using OAuth more simpler

Big API providers like Google, SaleForce, Yahoo, Facebook etc provide client libraries for integration with their APIs. These libraries abstract the OAuth 2.0 client implementation details and makes it easier for Clients

Google provides client libraries for all popular programming languages like Java, PHP, JavaScript, Ruby and Python

More details at @http://oauth.net/2/

Libraries

To implement the OAuth protocol, not enough matured libraries are available. However, there are few options

Apache provides an OAuth implementation in java which works with all API providers that support OAuth 2.0

http://incubator.apache.org/amber/

Scribe is another one that works with all APIs

https://github.com/fernandezpablo85/scribe-java

If there is a need for custom implementation, consider

1. Implementation of Authorization, Access and Refresh Token End Points
2. Generation, persistence and validation of Grants and Tokens
3. Use of JSON library for access token response generation
4. Client Registration application
5. Wrapper to abstract OAuth specific request and response processing

Use Cases

Few OAuth use cases

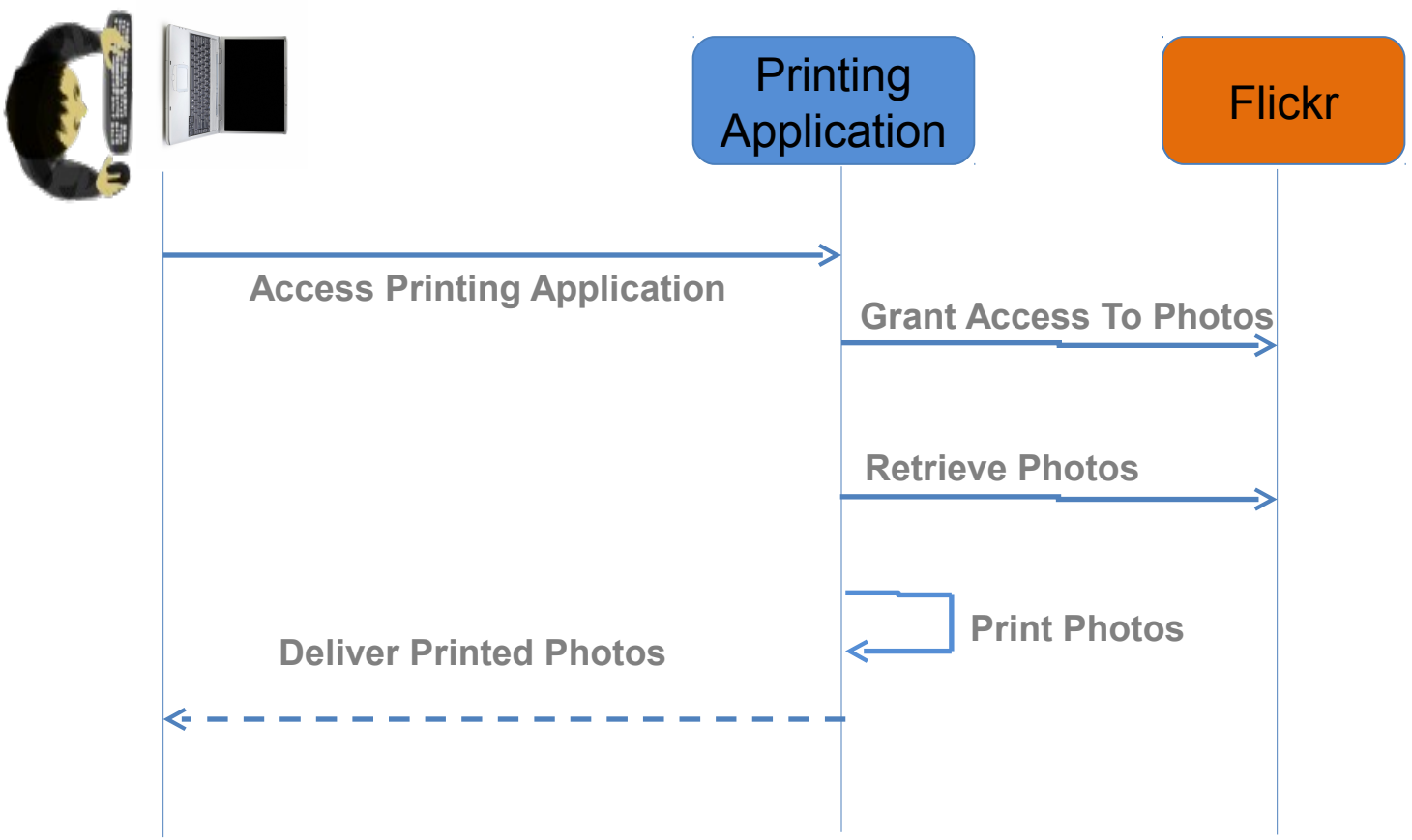
Social Portals
Financials
Cloud

Applicable across industry where open and restricted authorization is required

Use Cases – Social Portals

Social Portals

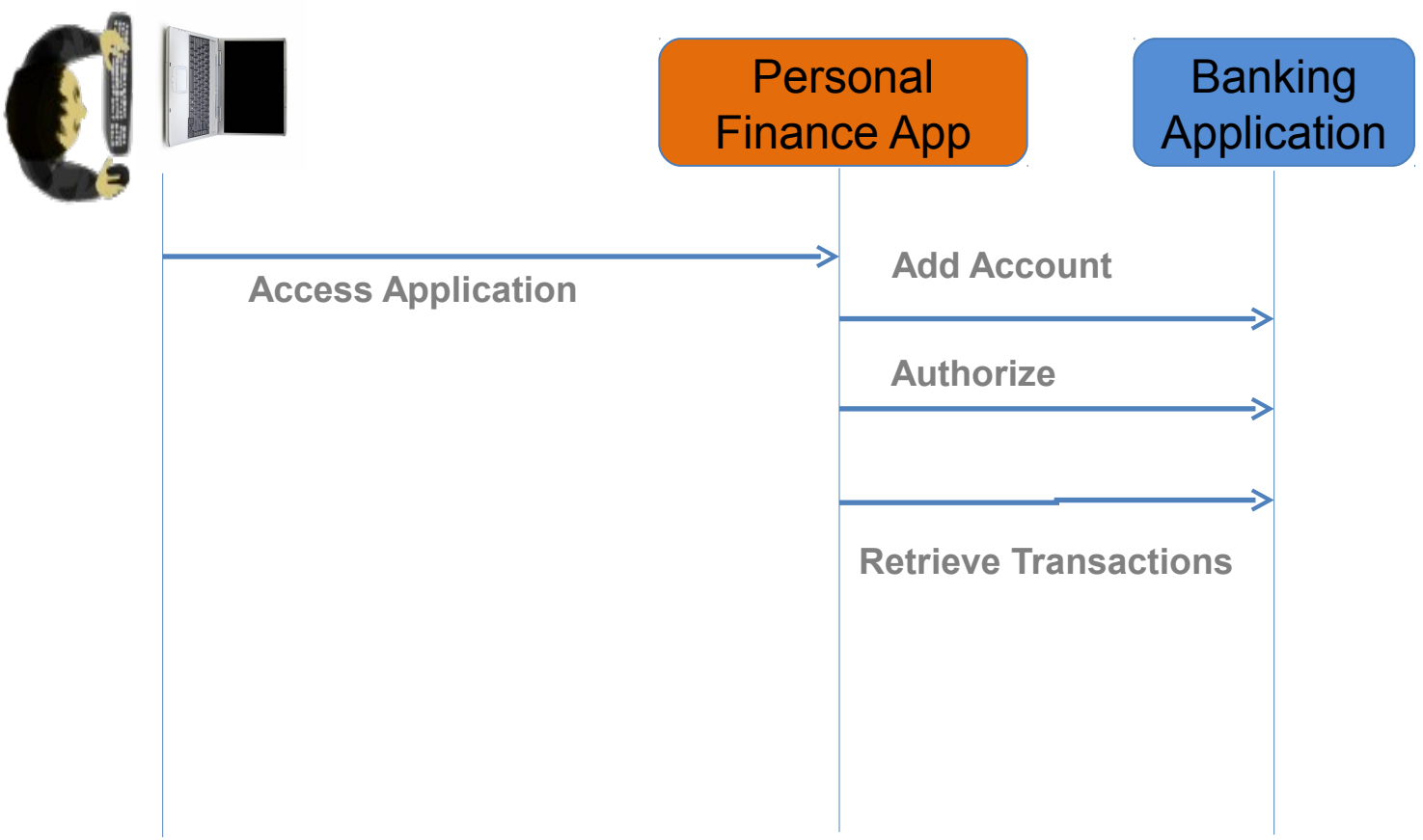
It is the most common use case where organization intends to leverage the users of Social Applications and APIs offered by them



Use Cases - Financial

Financial

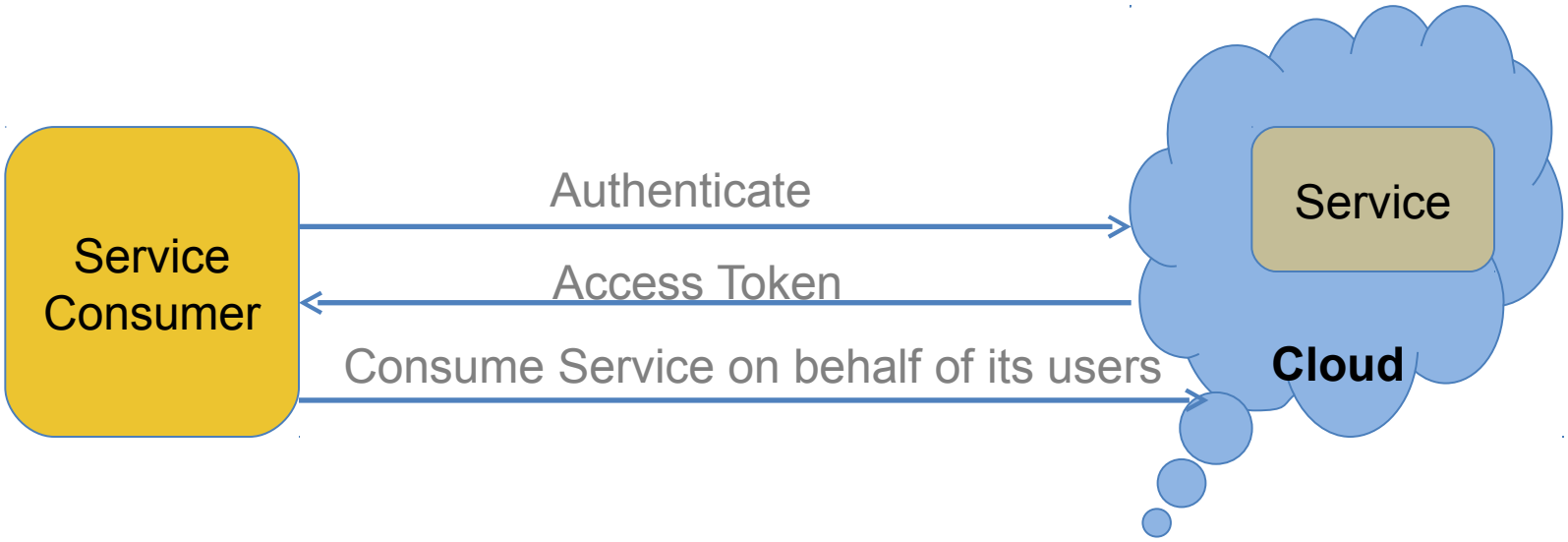
Another use case of OAuth could be Personal Finance Management Applications. Banking users could authorize bank to share all banking transactions with Money Finance Application which would provide an holistic view of user's financials



Use Cases - Cloud

Cloud

Protocol could be handy for Organizations wanting to utilize the services offered on the cloud for any background activity to be done on behalf of its users



Challenges

1. Protocol gaining visibility in the industry but not yet a standard
2. OAuth 2.0 Specification is in draft state and not yet approved
3. There are no standard libraries available. Also, available libraries implement different draft versions. Apache Amber is still in incubator stage
4. Most of the Security related concerns are left to specification implementations

References

Website

http://oauth.net/2

Specifications

http://tools.ietf.org/html/draft-ietf-oauth-v2-30 (main)

http://tools.ietf.org/html/draft-ietf-oauth-v2-bearer-23 (extension)

http://tools.ietf.org/html/draft-ietf-oauth-v2-http-mac-01 (extension)

http://tools.ietf.org/html/draft-ietf-oauth-saml2-bearer-13 (extension)

Books

Getting.Started.with.OAuth.2.0 (Oreilly Publication)

Not to forget Big Brother http://google.com

Amit Harsola

Senior Architect

amit.harsola@wipro.com