Web Service Security

Foundations of Security

Security relies on the following elements:

- **Auditing**.
  - Effective auditing and logging is the key to non-repudiation.
  - Non-repudiation guarantees that a user cannot deny performing an operation or initiating a transaction.
- **Authentication**.
  - Authentication allows you to confidently identify the clients of your service. These might be end users, other services, processes, or computers.
  - WCF supports mutual authentication, which identifies both the client and the service in tandem, to help in preventing man-in-the-middle attacks.
- **Authorization**.
  - Authorization determines what system resources and operations can be accessed by the authenticated user.
  - This allows you to grant specific application and resource permissions for authenticated users.

Foundations of Security

- **Confidentiality**.
  - Confidentiality, also referred to as privacy, is the process of making sure that data remains private and confidential, and that it cannot be viewed by unauthorized users.
  - Encryption is frequently used to enforce confidentiality. Privacy is a key concern, particularly for data/messages passed across networks.
- **Integrity**.
  - Integrity is the guarantee that data is protected from accidental or deliberate modification. Like privacy, integrity is a key concern, particularly for data/messages passed across networks.
  - Integrity for data in transit is typically provided by using hashing techniques and message authentication codes.

WS-* Standards

The WS-* architecture is a set of standards-based protocols designed to secure Web service communication.

- **WS-Policy**.
  - Allows Web services to define policy requirements for endpoints.
  - These requirements include privacy rules, encryption rules, and security tokens.
- **WS-Security**.
  - WS-Security allows Web services to apply security to Simple Object Access Protocol (SOAP) messages through encryption and integrity checks on all or part of the message.
- **WS-Trust**.
  - WS-Trust allows Web services to use security tokens to establish trust in a brokered security environment.

WS-* Standards

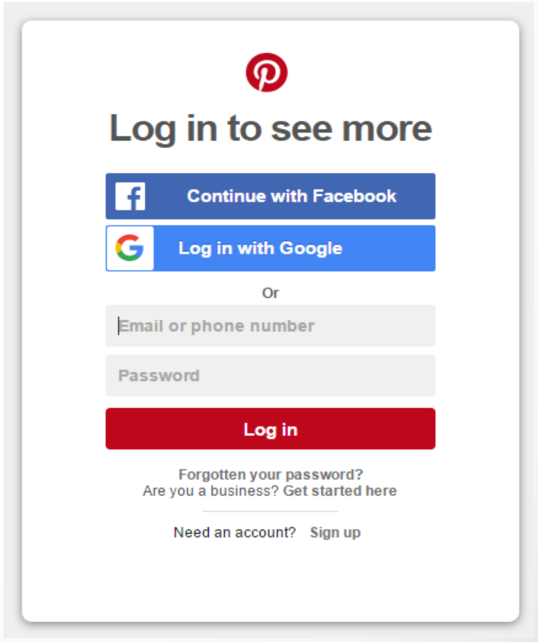
- **WS-SecureConversation**
  - WS-SecureConversation builds on top of WS-Policy, WS-Security, and WS-Trust to enable secure communications between client and service.
- **WS-ReliableMessaging**
  - WS-ReliableMessaging allows Web services and clients to trust that when a message is sent, it will be delivered to the intended party.
- **WS-AtomicTransactions**
  - WS-AtomicTransactions allows transaction-based Web services in which transactions can be rolled back in the event of a failure.

ReST Security – Best Practices

- **Validation** – Validate all inputs on the server. Protect your server against SQL or NoSQL injection attacks.
- **Session Based Authentication** – Use session based authentication to authenticate a user whenever a request is made to a Web Service method.
- **No Sensitive Data in the URL** – Never use username, password or session token in a URL, these values should be passed to Web Service via the POST method.
- **Restriction on Method Execution** – Allow restricted use of methods like GET, POST and DELETE methods. The GET method should not be able to delete data.
- **Validate Malformed XML/JSON** – Check for well-formed input passed to a web service method.
- **Throw generic Error Messages** – A web service method should use HTTP error messages like 403 to show access forbidden, etc

OAuth 2.0

- OAuth 2 is an authorization framework that enables applications to obtain limited access to user accounts on an HTTP service, such as Facebook, GitHub, and DigitalOcean.
- It works by delegating user authentication to the service that hosts the user account, and authorizing third-party applications to access the user account.
- OAuth 2 provides authorization flows for web and desktop applications, and mobile devices.


Log in to see more

OAuth Roles

- **Resource Owner: *User***

The resource owner is the *user* who authorizes an *application* to access their account. The application's access to the user's account is limited to the "scope" of the authorization granted (e.g. read or write access).

Resource / Authorization Server: *API*

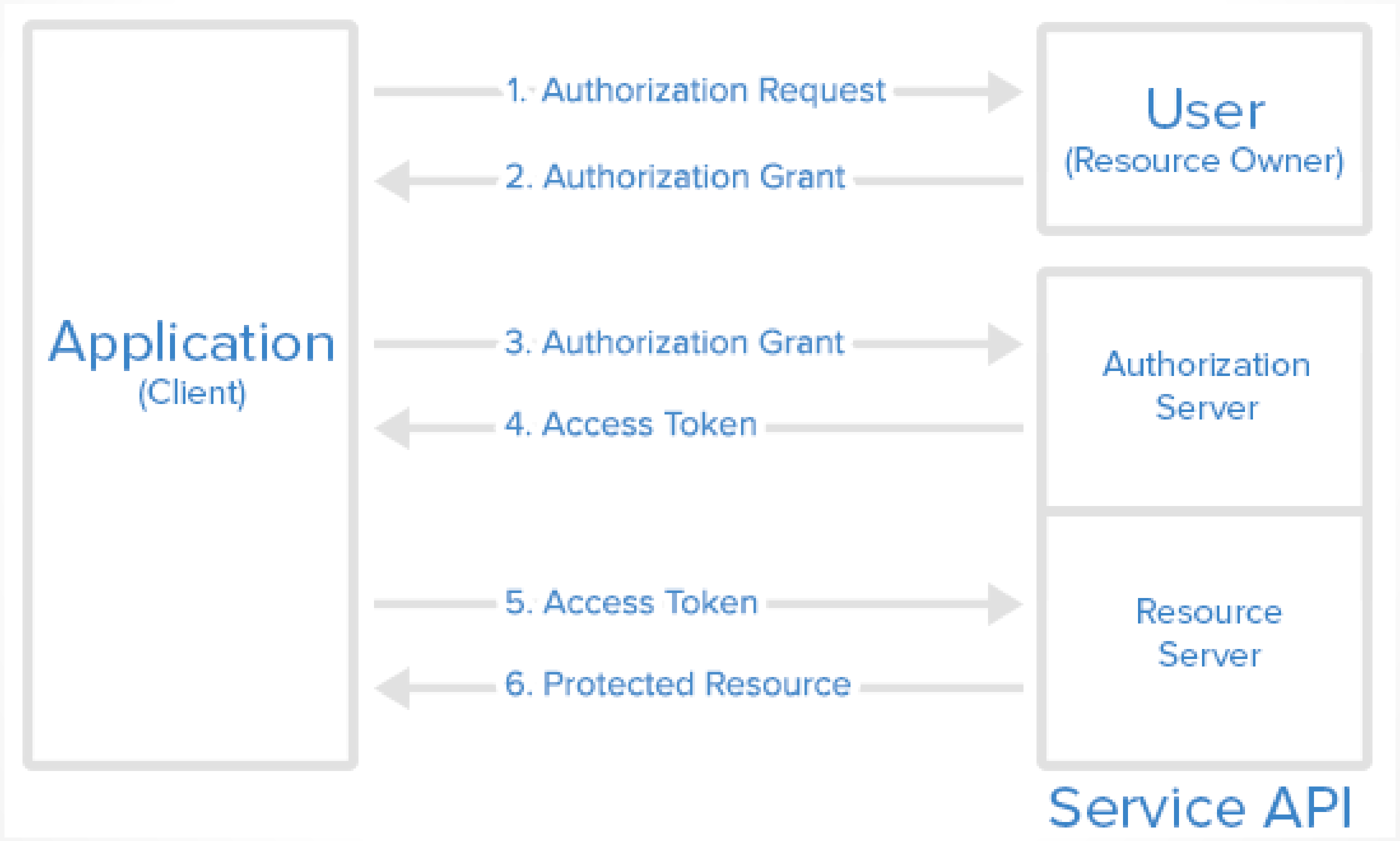
The resource server hosts the protected user accounts, and the authorization server verifies the identity of the *user* then issues access tokens to the *application*.

From an application developer's point of view, a service's **API** fulfills both the resource and authorization server roles.

- **Client: *Application***

The client is the *application* that wants to access the *user*'s account. Before it may do so, it must be authorized by the user, and the authorization must be validated by the API.

Abstract Protocol Flow



Abstract Protocol Flow

- The *application* requests authorization to access service resources from the *user*
- If the *user* authorized the request, the *application* receives an authorization grant
- The *application* requests an access token from the *authorization server* (API) by presenting authentication of its own identity, and the authorization grant
- If the application identity is authenticated and the authorization grant is valid, the *authorization server* (API) issues an access token to the application. Authorization is complete.
- The *application* requests the resource from the *resource server* (API) and presents the access token for authentication
- If the access token is valid, the *resource server* (API) serves the resource to the *application*

Example

- **SampleApp** has initially registered themselves as an "application" with **Linkedin**. In doing so, they're provided with a token set called "consumer key" and its paired "consumer key secret." These are used by SampleApp in their application code and as a part of the OAuth model in generating requests.
- From a user perspective, when you log in to SampleApp and click the "Login with Linkedin " button, SampleApp uses its consumer key to contact Linkedin and generate a "request token." You're then provided with a special URL that redirects you off to Linkedin's website.
- If you aren't already logged into Linkedin, you'll be prompted to, and then presented with a screen that asks if you'd like to provide SampleApp with access to your account.
- Clicking "Allow" tells Linkedin that SampleApp has requested access using its particular consumer key should have access to your Linkedin account. Linkedin then redirects you back to your SampleApp with an attached coded verification string.
- The SampleApp then reads the previously generated request token, and takes the returned verification to ask Linkedin to generate a final token set called "access token" and "access token secret.
- And amid all of this, your Linkedin username and password are never seen, let alone stored, by SampleApp.